Energy-Performance Design Space Exploration in SMT Architectures Exploiting Selective Load Value Predictions

A. Gellert[1] G. Palermo[2] V. Zaccaria[2] A. Florea[1] L. Vintan[1] C. Silvano[2]
[1]"Lucian Blaga" University of Sibiu - Computer Science and Engineering Department
[2]Politecnico di Milano - Dipartimento di Elettronica e Informazione
{arpad.gellert, adrian.florea, lucian.vintan}@ulbsibiu.ro {gpalermo, zaccaria, silvano}@elet.polimi.it

***Abstract*—This paper presents a design space exploration of a selective *load value prediction scheme* suitable for energy-aware Simultaneous Multi-Threaded (SMT) architectures. A load value predictor is an architectural enhancement which speculates over the results of a micro-processor *load* instruction to speed-up the execution of the following instructions. The proposed architectural enhancement differs from a classic predictor due to an improved selection scheme that allows to activate the predictor only when a *miss* occurs in the first level of cache. We analyze the effectiveness of the selective predictor in terms of overall energy reduction and performance improvement. To this end, we show how the proposed predictor can produce benefits (in terms of overall cost) when the cache size of the SMT architecture is reduced and we compare it with a classic non-selective load value prediction scheme. The experimental results have been gathered with a state-of-the-art SMT simulator running the SPEC2000 benchmark suite, both in SMT and non-SMT mode.[1]**

I. Introduction

The performance of current micro-processors is limited by the number of instructions that can be processed simultaneously [1]. The effectiveness of this process is limited by existing *read-after-write* (RAW) dependencies, since one instruction cannot be executed without reading the value of its source operands, which in turn are written by a previous instruction. Thus, whenever such a dependency exists, the involved instructions should be *serialized*.

This is true for a wide category of operations ranging from arithmetic instructions to branch and load/store instructions. To overcome such drawbacks, super-scalar and multi-threaded micro-architectures have been equipped with dynamic techniques such as *branch* and *value prediction* that allow to anticipate (predict) the value of the source operands and thus start instructions earlier than the *serialized* case. However, the additional energy consumption associated with such architectural enhancements can be significant and should be traded-off with the actual benefits in terms of performance to achieve a required energy budget.

In this paper, we focus on *load value prediction*. Let us introduce the concept of load value prediction by considering the following code segment:

```
1: ld.w 0($r2) → $r3
2: add $r4, $r3 → $r5
3: add $r3, $r6 → $r6
```

As can be seen, the execution of instructions 2 and 3 is strictly dependent on the result $r3 of the `ld.w` (load from memory) instruction 1. In fact, there is a RAW dependency between 1 and 2 and between 1 and 3. This means that, unless $r3 is known a-priori, both 2 and 3 should be serialized with respect to 1. A load value predictor allows to predict $r3 before instruction 1 has been completed to start both 2 and 3 earlier (and in parallel). This is done through a *prediction* whose outcome is checked before 2 and 3 are committed. The temporary buffer used for storing those intermediate results is typically called a *re-order buffer* (ROB). Whenever the prediction is verified to be wrong, a recovery mechanism is activated. In the previous case, this consist of *squashing* both instructions 2 and 3 from the ROB buffer and re-execute them with the actual value of $r3.

Traditional value prediction techniques have been increasingly challenged by the advent of mobile, battery-operated devices due to the significant amount of energy consumption. This is essentially due to the on-chip memory required for computing the prediction and the overall number of accesses to the predictor itself. In this work we introduce and analyze a **selective value predictor** which is triggered selectively only during specific cache miss events. In particular, we show that this kind of predictor allows to:

- Reduce the overall number of accesses and the energy consumption of the on-chip memory and logic reserved to the value speculation.
- Improve over traditional value predictors [2] in terms of performance and energy consumption.
- Create room for a reduction of the data-cache size by preserving performance, thus enabling a reduction of the system cost.

The paper is organized as follows. Section II introduces the state-of-the art of value prediction techniques. Section III introduces the target architecture and the proposed energy-aware load value predictor, while Section IV describes the experimental results obtained on an Alpha AXP 21264 architecture. Finally, Section V summarizes the relevant contributions of this work.

II. State-of-the-art Value Prediction

Lipasti et al. [2] originally introduced the Load Value Prediction as a new data-speculative micro-architectural technique exploiting the concept of value locality and the dynamic correlation between `ld.w` instruction address and its actual value.

An initial power analysis on value predictors has been presented in [3] and [4]. In both papers, the authors claimed that for a power efficient value prediction implementation designers have to limit the complexity of the predictor (such as the usage of a simple last value predictor), limiting also the number of access ports. In [3] the authors show that the

[1]This work was supported in part by the EC under grant MULTICUBE FP7-216693

dynamic classification done by the value predictor (predictable or unpredictable) can be used also to activate or deactivate architecture low-power modes. In [4] the authors have shown that a non-selective value prediction approach can cause in some case a noticeable performance and energy degradation. Other value predictors like the stride-, context- and perceptron-based, have been proposed in earlier work [5] for register-centric value prediction.

Different architectural support techniques for value prediction are presented in [6]–[9]. Calder et al. [6] proposed some selective techniques in order to reduce the pressure on the prediction tables, by filtering the instructions that accessed these resources. The ideal case is to dynamically select those instructions that belong to current longest data dependency chain as recognized in the instruction window (critical path).

In [7] a Checkpoint-Assisted VAlue prediction (CAVA) has been proposed to hide L2 cache misses through data value prediction. For each L2 miss, the architectural state is check-pointed before entering the speculative mode. When the `ld.w` is resolved, the prediction is checked and in case of misprediction the hardware rolls back the architectural state to the checkpoint.

Checkpoint-free load value prediction is presented in [8]. In this work, speculative instructions remain in the issue queue, since no check-point is made. When the actual data is received from memory, the speculative instructions are always discarded and re-executed. In this case, the speculative execution is essentially employed for pre-fetching.

Finally, a value prediction technique for SMT architectures is presented in [9]. The authors propose to create a checkpoint on a long latency `ld.w` instruction, as in [7], but instead of using the same thread context, a speculative thread using the predicted value is spawn in another thread context.

The main difference between the analysis presented in this paper and the previous ones is that we present a load value predictor method that selectively predicts only `ld.w` instructions that miss the level 1 data cache, attenuating the misprediction probability and reducing the hardware cost of the speculative micro-architecture. Moreover, the usage of a simple last value predictor as suggested in [3] requires less additional hardware enabling reduced energy consumption than traditional approaches.

III. Low-Energy Selective Load Value Prediction (SLVP)

The basic idea of the proposed predictor is to reduce the load *pressure* and thus the overall number of accesses to the predictor tables. In fact, the predictor is accessed only whenever a miss in the level 1 cache is detected. This action is performed in the initial evaluation stages of each `ld.w` instruction and it has a minimum delay which corresponds to the time needed for the tag check in the L1 cache.

Figure 1 shows the basic architecture of the predictor. It essentially consists of a small cache which is accessed by using the *program counter* of the `ld.w` instruction that had a miss in the L1 cache. Since we are using a 64 bit architecture, both the program counter and the size of the loads are 64 bit. The prediction policy is managed through a *2-bit saturating confidence counter* with two unpredictable and two predictable states [1] (CR).

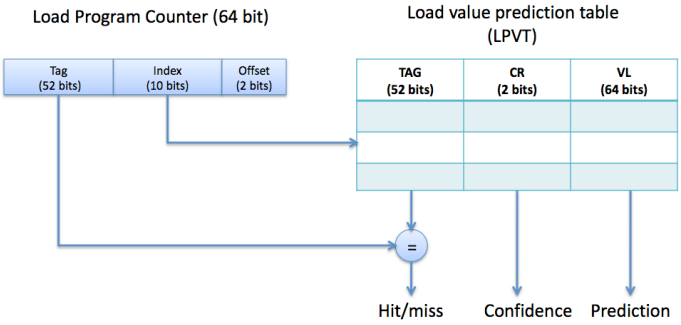


Fig. 1. The architecture of the *selective load value predictor* (SLVP).

Value prediction. In the case of a hit in the SLVP (checked by means of the TAG field), the corresponding CR is evaluated. If CR is in an *unpredictable* state, no prediction is performed and the dependent instructions are put on hold until the actual `ld.w` instruction has accessed the memory. Otherwise the data value (VL) from the selected SLVP entry is speculatively forwarded to the dependent instructions and the `ld.w` instruction is marked as *speculative* in the ROB.

Critical load commit. When a critical `ld.w` instruction has been executed (during the *commit* stage of the pipeline) the architecture checks if the predicted value VL is equal to the actual value of the `ld.w` instruction. In the case of miss-prediction, the dependent instructions are squashed from the ROB and re-executed. In the experimental section we consider an average recovery time of 7 cycles.

Load value predictor update. During the commit stage, every *critical* `ld.w` instruction updates the CR field. Whenever a misprediction has occurred, the VL field is also updated with the actual value of the `ld.w`. If an initial SLVP miss was detected, an entry is evicted from the SLVP (without any write-back) and allocated to the most recent `ld.w`. The selective approach enables the predictions only for loads with a miss in the L1 data cache. Therefore, the prediction latency consists in the L1 data cache tag check and the SLVP access latency. To compute the overall latency, we considered the architecture shown in Table I and a SLVP table of 1024 entries[2]. Cacti [11] reported a data cache tag check latency of 2 cycles and a SLVP access latency of 1 cycle. Thus, the overall latency of the proposed selective approach is 3 cycles.

IV. Design space exploration of the SLVP

In this section we present the results of the design space exploration of the selective load value predictor proposed in this paper. We first introduce the target architecture and the benchmark suite used for the exploration; we then present some experimental evidence on the benefits of using such kind of predictor in a super-scalar (plain and SMT) architecture.

The target architecture. The target architecture is a superscalar Alpha AXP 21264. We will consider a plain Alpha AXP and an SMT-enhanced Alpha AXP. To derive both

[2]A preliminary exploration presenting the impact of the table size on the prediction accuracy can be found in [10]

performance and energy consumption of the Alpha AXP, we used the M-SIM V2.0 simulation framework [12]. The simulator supports the execution of unmodified, statically linked Alpha AXP binaries and it exploits the Wattch models [13] to provide the energy consumption of the architectural components. For the performance estimation, M-SIM extends the SimpleScalar tool-set [14] with accurate models of the pipeline structures, including explicit register renaming, and support for the concurrent execution of multiple threads (SMT). In SMT mode, some processor structures (i.e. issue queue, physical register files, functional units, caches) are shared among the threads, while other structures (rename tables, ROBs, Load/Store Queues, branch and value predictors) are private to each thread.

Table I presents some important parameters of the simulated architecture. Memory and cache latencies have been computed by using [11]. Although the basic Alpha AXP architecture does not present any load value predictor, Table I shows also the architectural configuration of the SLVP. While the SLVP access latency is 1 cycle only, the prediction latency due to the selective value predictor is higher (3 cycles) as explained in the previous section.

Experimental setup. All the following simulation results are generated by running 1 billion instructions for each of the SPEC 2000 benchmarks [15], skipping the first 300 million instructions to avoid cold start and transient effects.

For the superscalar architecture we evaluated six floating-point benchmarks (`applu`, `equake`, `galgel`, `lucas`, `mesa`, `mgrid`) and seven integer benchmarks: *computation intensive* (`bzip`, `gcc`, `gzip`) and *memory intensive* (`mcf`, `parser`, `twolf`, `vpr`). The workload for the SMT mode has been generated combining randomly 2 applications from the same benchmark set.

TABLE I
ARCHITECTURE CONFIGURATION

Processor	
Frequency	1.2 GHz @ 80nm
Branch predictor	Bimodal predictor 2K entries
Fetch / Decode / Issue / Commit width	4 / 4 / 4 / 4
Register File size	128 INT + 128 FP
Fetch queue size	32 entries
ROB size	128 entries
Load/Store queue size	48 entries
Integer ALU #units / latency	4 / 1 cycle
Integer MUL-DIV #units / latency	1 / 3(MUL) - 20(DIV) cycles
Floating Point ALU #units / latency	4 / 2 cycles
Floating Point MUL-DIV #units / latency	1 / 4(MUL) - 12(DIV) cycles
Caches and Memory	
L1 I$ size/associativity/block	64KB / 2-way / 64B
L1 D$ size/associativity/block	64KB / 2-way/ 64B
L2$ size/associativity/block	4MB / 8-way / 64B
L1 I$ / L1 D$ / L2$ access latency	2 / 2 / 14 cycles
Memory access latency	270 cycles
Load Value Prediction Table	
SLVP size	1024 entries, direct mapped
SLVP access latency	1 cycle

Selective vs. Non-selective Load Value Predictor. Figure 2 and Figure 3 show, respectively, the IPC speed-up and the energy reduction with respect to a baseline Alpha AXP architecture (without any predictor) of the proposed selective predictor (SLVP) versus a non-selective, state-of-the-art predictor (LVPT) as presented in [2].

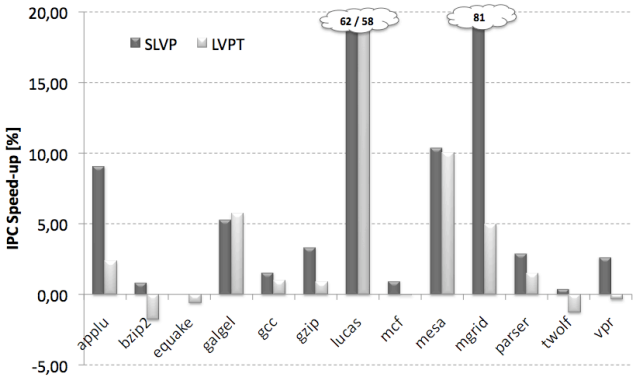


Fig. 2. Instruction-per-cycle speed-up of the SLVP and LVPT with respect to the baseline architecture.

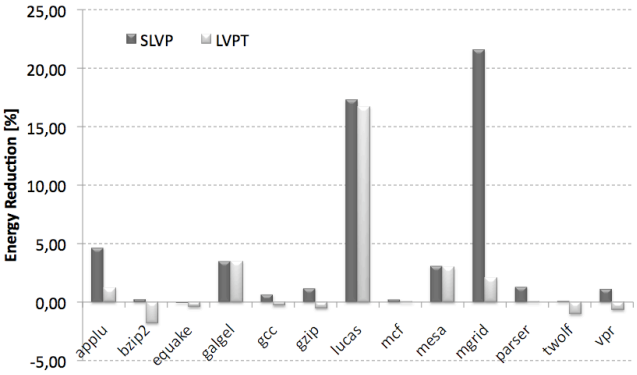


Fig. 3. Overall energy consumption improvement of the SLVP and LVPT with respect to the baseline architecture.

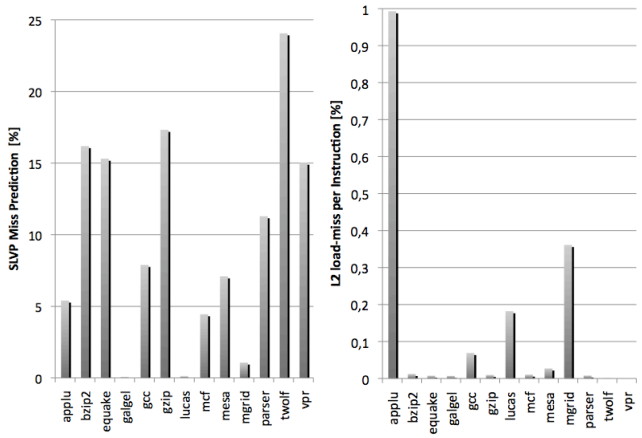
It can be seen that the SLVP has advantages both in terms of IPC and energy with respect to both the baseline architecture and the LVPT-enhanced one. This derives from the fact that SLVP is only triggered on `ld.w` instructions that miss the L1 cache while LVPT suffers from more load *pressure*, being it triggered at every `ld.w` instruction. We can also note that some specific benchmarks are more susceptible than others (e.g., `lucas` and `mgrid`) to performance improvements. This is due to two facts that are summarized in Figure 4:

1) they present a very high SLVP prediction accuracy.
2) they present a high number of `ld.w` instructions that miss both the first and the second level of cache.

Energy-Performance Design Space Exploration. In this section, we present a design space exploration analysis of the proposed SLVP predictor by analyzing the correlation with the size of the level 1 data cache. Here we focus on the SLVP since the previous analysis has shown the advantages of this technique with respect to the LVPT solution. In particular, we analyze the behavior of the SLVP by considering the Alpha AXP architecture in plain (superscalar) and SMT mode.

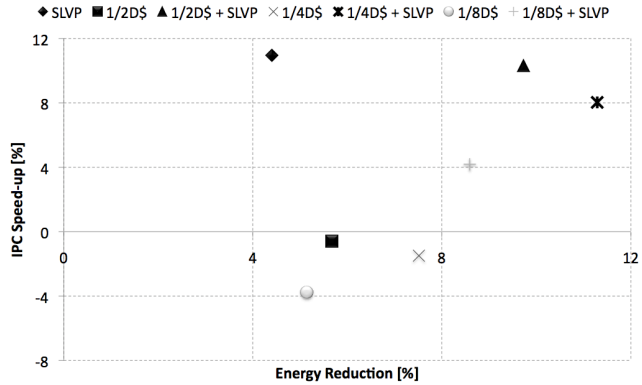
For the following exploration, we show the behavior of the SLVP when the size of the level 1 data cache is continuosly halved up to 1/8 of the original size, reduced to 32KB, 16KB and 8Kb (see Figure 5).

Figure 5(a) shows that, for both the baseline and the SLVP-

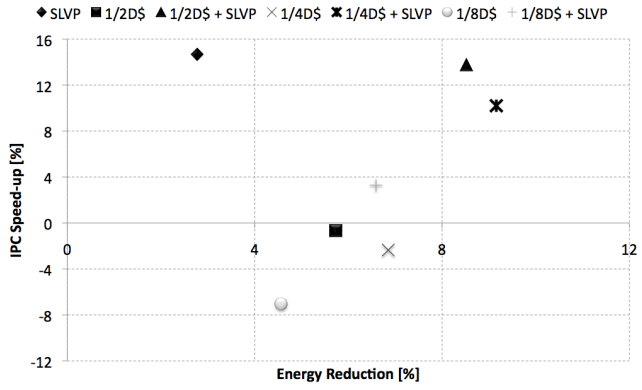


(a) SLVP misprediction rate (b) Level 2 cache miss rate

Fig. 4. SLVP misprediction rate and level 2 cache miss rate for the considered set of benchmarks.



(a) Superscalar case



(b) SMT case

Fig. 5. Scatter plot of energy and performance average improvements of the SLVP when the level 1 data cache is varied.

enhanced superscalar case, decreasing the cache size has a positive effect on the energy reduction up to quartering the cache (with and w/o the SLVP). In fact, the configuration obtained by quartering the cache results a limit for the Energy/Performance trade-off. However, SLVP helps maintaining a positive IPC and energy speed-up, while on the baseline architecture the IPC speed-up is negative.

Figure 5(b) shows a similar behavior for the SMT case with the exception of the entity of the negative effects when passing from 1/4D$ to 1/8D$ configuration, that is higher for the SMT case. This can be due to the fact that, in the SMT case, the data cache has a higher pressure; reducing the size of the cache means more cache misses and, thus, more pressure on the SLVP. This, in turn, further decreases the SLVP prediction accuracy and reduces the advantages in terms of IPC and energy.

V. Conclusions

This paper presented a design space exploration of a selective *load value prediction scheme* suitable for energy-aware Simultaneous Multi-Threaded (SMT) architectures. We have shown that a selective load value prediction can reduce the overall number of accesses and the energy consumption of the on-chip memory and create room for a reduction of the data-cache size by preserving performance, thus enabling a reduction of the system cost with respect to previous approaches.

References

[1] D. A. Patterson and J. L. Hennessy. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 2nd edition, 1996.

[2] Mikko H. Lipasti, Christopher B. Wilkerson, and John Paul Shen. Value locality and load value prediction. In *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 138–147, New York, NY, USA, 1996. ACM.

[3] Toshinori Sato and Itsujiro Arita. Reducing energy consumption via low-cost value prediction. In *PATMOS '02: Proceedings of the 12th International Workshop on Integrated Circuit Design. Power and Timing Modeling, Optimization and Simulation*, pages 380–389, London, UK, 2002. Springer-Verlag.

[4] Ravi Bhargava and Lizy K. John. Latency and energy aware value prediction for high-frequency processors. In *ICS '02: Proceedings of the 16th international conference on Supercomputing*, pages 45–56, New York, NY, USA, 2002. ACM.

[5] L.N. Vintan, A. Florea, and A. Gellert. Focalising dynamic value prediction to cpu's context. *IEE Proceedings - Computers and Digital Techniques*, 152(4):473–481, 2005.

[6] Brad Calder, Glenn Reinman, and Dean M. Tullsen. Selective value prediction. In *ISCA '99: Proceedings of the 26th annual international symposium on Computer architecture*, pages 64–74, Washington, DC, USA, 1999. IEEE Computer Society.

[7] Luis Ceze, Karin Strauss, James Tuck, Josep Torrellas, and Jose Renau. Cava: Using checkpoint-assisted value prediction to hide l2 misses. *ACM Transactions on Architecture and Code Optimization*, (3), 2006.

[8] Huiyang Zhou and Thomas M. Conte. Enhancing memory level parallelism via recovery-free value prediction. In *Proceedings of the 17th International Conference on Supercomputing*, pages 326–335, 2003.

[9] Nathan Tuck and Dean M. Tullsen. Multithreaded value prediction. In *HPCA '05: Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, pages 5–15, Washington, DC, USA, 2005. IEEE Computer Society.

[10] Arpad Gellert, Adrian Florea, and Lucian N. Vintan. Exploiting selective instruction reuse and value prediction in a superscalar architecture. *Journal of Systems Architecture*, 55(3):188–195, 2009.

[11] S. Wilton and N. Jouppi. CACTI:An Enhanced Cache Access and Cycle Time Model. volume 31, pages 677–688, 1996.

[12] J. Sharkey, D. Ponomarev, and K. Ghose. M-sim: a flexible, multi-threaded architectural simulation environment. Technical Report CS-TR-05-DP01, Department of Computer Science, State University of New York at Binghamton, 2005.

[13] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Proceedings ISCA 2000: International Symposium on Computer Architecture*, pages 83–94, 2000.

[14] Doug Burger, Todd M. Austin, and Steve Bennett. Evaluating future microprocessors: The simplescalar tool set. Technical Report CS-TR-1996-1308, University of Wisconsin, 1996.

[15] John L. Henning. Spec cpu2000: Measuring cpu performance in the new millennium. *Computer*, 33(7):28–35, 2000.