Complex Load-Value Predictors: Why We Need Not Bother

Nana B. Sam and Martin Burtscher
Computer Systems Laboratory
Cornell University, Ithaca, NY 14853
{besema, burtscher}@csl.cornell.edu

Abstract

Memory accesses continue to represent a major performance bottleneck and much remains to be done to tolerate their latencies. A large body of work exists that presents load-value prediction as an effective means to hide some of the memory latency. To increase the prediction accuracy and hence the performance, researchers have proposed more complex and larger predictor designs.

This paper re-evaluates load-value predictors and examines the tradeoffs between predictor size, latency, energy consumption and performance. We demonstrate that even though complex predictors deliver the highest accuracy, they are unlikely to be implemented in hardware because they provide a much worse energy-performance tradeoff than simpler predictors with moderate sizes.

1. Introduction

While processor speeds have been increasing by approximately a factor of two every 18 months in keeping with Moore's Law, memory speeds have only been improving by about 5% a year. This has led to a speed gap between the memory and the processor that doubles every 21 months or so. Thus processors spend more and more time idling because the memory cannot serve data sufficiently quickly.

Many superscalar, out-of-order processors can tolerate the latencies for first- and second-level cache hits [29] if enough independent instructions are available. However, main memory access latencies, which are on the order of hundreds of cycles, cannot be hidden and cause significant performance degradations. For the SPECcpu2000 benchmark programs running on a modern, high-performance microprocessor, over half the runtime is spent stalling for loads that miss in the second-level cache [16].

To hide some of the negative effects of these stalling loads, load-value prediction has been proposed, which allows the dependent instructions to execute concurrently with the memory access. This proved possible because loads fetch predictable sequences, i.e., they exhibit value locality [10], [17]. However, exploiting value locality requires a significant investment in hardware to achieve modest prediction accuracies [17], [25], [30], [31]. Since power/energy consumption constitutes one of the primary design constraints of future microprocessors [12], [32], it is no longer practical to keep pushing the performance envelope by proposing to add predictors without taking energy into consideration.

Even though several predictors have been proposed, it is not immediately obvious which design offers the best energy-performance tradeoff. For example, a simple predictor, such as the last-value predictor [10], [17], consumes only a fraction of the energy of a hybrid predictor [21] of similar size because the former accesses only one data array while the latter accesses multiple data arrays per prediction. However, the simpler predictor exhibits worse accuracy than the hybrid. This is a problem because when a value is predicted incorrectly, the speculation hardware has to perform recovery actions that slow down the processor and waste energy. On the other hand, the more complex hybrid predictor, which delivers better prediction accuracy and coverage, consumes large amounts of energy and has a longer access latency. Long access latencies are undesirable since the predictor is on the critical path and must deliver its predictions quickly.

Deciding on a predictor's size involves a similar dilemma. A small predictor consumes less energy per access and has a shorter access latency, but a larger size increases the prediction accuracy. However, as pipelines deepen and clock rates increase, access delay significantly decreases the maximum size of on-chip SRAM arrays, such as value predictors, that can be accessed in one cycle [1].

In this paper, we assess the impact of five frequently used value predictors, each with six different sizes, on the performance and energy consumption of the entire microprocessor as opposed to just the predic-

tors themselves. We were surprised to find that when area, processor-wide energy consumption, and table access latency are taken into account, simpler predictors represent a better solution than complex predictors of similar size. We also find that making the predictors larger than about 20kB is unwise as beyond this point the energy consumption increases at a higher rate than the performance.

2. Background And Related Work

2.1 Load-Value Predictors

The increasing density of on-die transistors has enabled designers and researchers to use aggressive speculation techniques to improve instruction throughput. One such technique is load-value prediction, which breaks true data dependencies between instructions. In this section, we describe the five predictors that we evaluate in this paper. They are the most frequently used predictors in the literature.

The last value predictor (LV) [10], [17], [25] predicts that a load instruction will load the same value it did the previous time it executed.

The stride 2-delta predictor (ST2D) [25] remembers the last value for each load (like LV) but also maintains a stride, i.e., the difference between the last two loaded values. To make a prediction, ST2D adds the stride to the last value of the load. When a load is completed, ST2D updates the last value but only updates the stride if it encounters the same stride twice in a row. ST2D can predict sequences with zero (like LV) and non-zero strides.

The third-order finite context method predictor (FCM3) [24], [25] computes a hash value [20], [21], [24] out of the last three loaded values to index the predictor's second-level table. This table stores the values that follow every seen sequence of three values (modulo the table size). Since this table is shared, loads can communicate information to one another in this predictor. Hence, after observing a sequence of load values, FCM3 can predict any load that loads the same sequence.

The third-order differential finite context method predictor (DFCM3) [13] improves on FCM3 by retaining strides instead of absolute values. This reduces aliasing in the second-level table and enables DFCM3 to predict values it has never seen before. Thus DFCM3 combines the strengths of an FCM and a stride predictor at the cost of more elaborate and slower hardware.

Rychlik et al. [21] introduced a hybrid predictor that combines a stride and a finite context method predictor. The component with the highest confidence (see next paragraph) makes the prediction. In case of a tie, the FCM is given priority. The authors showed that this hybrid is more effective than either of the component predictors.

Due to the high cost of recovering from mispredictions, confidence estimators are used to dynamically inhibit predictions that are likely to be incorrect [8], [9], [11], [17], [20], [21], [24], 30]. The bimodal confidence estimator [17], [20], [21], which is frequently used, is based on saturating up/down counters with four parameters: a maximum, a threshold, a penalty and an award. A prediction is made only if the count is above the threshold. When an unpredictable value is encountered, the counter is decremented by the penalty, and on a predictable value, it is incremented by the award.

2.2 Sources of Predictor Complexity

The complexity of value predictors stems from predictor organizations that place multiple levels of logic on the critical path to making a prediction.

Large PC-indexed tables are one such source of complexity. Larger tables take longer to read. Accessing a table with a high latency may cause a prediction to arrive too late for it to have a positive impact on performance. Bhargava and John [2] evaluated the effect of varying numbers of access ports and varying predictor sizes on the table access latency for a hybrid predictor. They found that to maintain high performance the hybrid has to have at least 4096 entries. We demonstrate in this paper that an ST2D-FCM3 hybrid predictor with 4096 entries provides an energy-performance tradeoff that is no better than a ST2D predictor of similar size. Moreover, Bhargava et al. only considered the energy consumption of the predictors themselves, whereas we consider the energy consumption of the entire processor and thus capture the negative impact of incorrect predictions on other parts of the processor, which can be significant. Finally our study is not limited to a hybrid predictor. Rather, the objective of this paper is to show that complex predictors such as hybrids provide a worse energy-performance tradeoff than their simpler counterparts.

Computation is another source of complexity. Multi-level predictors, including FCM3 and DFCM3, perform complex computations to determine the index into the second level. Hybrid predictors, as well as FCM3 and DFCM3, need to extract data from multiple tables and ST2D and DFCM3 have to perform additions to compute the final prediction. These add more delay and energy use to the process of making a prediction. For example, Jimenez et al. showed that a 100% accurate branch predictor with a two-cycle latency performs worse than a relatively inaccurate

branch predictor with single-cycle latency [14]. Unlike in much of the previous work, where the computation latency has been ignored, we use spice simulations to estimate these extra latencies and take them into account in our simulations. With this we find that, for instance, the simple LV can outperform the complex FCM3.

2.3 Sources of Energy Expenditure

Even though load-value prediction does improve performance, it contributes to the energy consumption. One source is the speculative hardware itself. To boost the prediction accuracy, these hardware structures are made as large as possible, which increases their energy consumption. Another significant source of energy consumption is the useless activities that are performed in other processor components due to mispredicted instructions that are later discarded. A mispredicted instruction contributes to the dynamic energy consumption through datapath switching activity until it is removed from the pipeline. A third source is the increased (useful) activity introduced by the speculation. Increased speculation reduces the runtime and thus may actually lower the energy use.

Several techniques have been proposed to save space and energy, including sharing common predictor components in hybrid predictors [7], [19], partitioning large predictors into multiple smaller tables [15], [22], [23], and using static and dynamic approaches to filter out loads that are likely to be predicted incorrectly [3], [6], [9].

Unfortunately, most energy-saving techniques recommended for value predictors have only focused on one source of energy consumption, the predictor itself. Moreno et al. [18] analyzed power-related issues with value speculation and note that recovery from mispredictions has a significant negative impact on the energy consumption. In our model, we found that when value speculation is incorporated into the processor, the total increase in energy expenditure of the rename unit, register file, load/store queue, functional units, result busses, instruction window, branch predictor, global clock and caches is 3.95 times that of the value predictor. Unlike Moreno et al., who advocate a low cost hybrid predictor to reduce the energy consumption, we show that hybrids and other complex predictors are not a good idea in light of energy-performance considerations.

Our simulations account for the negative effect of mispredictions on performance and energy consumption, and report the energy use of the entire processor chip, including the first- and second-level caches. They also account for any extra computation latency required by each predictor.

3. Simulation Methodology

The execution-driven simulator we use is derived from the SimpleScalar/Alpha 3.0 tool set [5]. Specifically, we extended *sim-outorder* to perform load-value prediction and we integrated the simulator with Wattch [4] to obtain the energy data.

We simulate an 8-way superscalar, out-of-order CPU with 20 pipeline stages, a 128-entry instruction window, a 64-entry load/store buffer, a 32-entry 8-way instruction TLB, a 64-entry 8-way data TLB, both with a 30-cycle miss penalty, a 64kB, 2-way 2-cycle L1 instruction cache, a 128kB, 2-way 3-cycle L1 data cache, a unified 4MB, 4-way 20-cycle L2 cache, an 8K-entry hybrid gshare-bimodal branch predictor, six integer ALUs, four floating-point adders and two floating-point MULT/DIV units. The data cache is write-back and non-blocking with two ports. The caches have a block size of 64 bytes. It takes 300 cycles to access main memory. We use Wattch's linear scaling to obtain energy results for 0.13µm technology, V_{dd} = 1.3V and a clock speed of 2GHz. Wattch treats leakage power as 10% of the dynamic power. The cache and predictor access latencies are obtained with Cacti 3.2 [26] and the predictor computation latencies (i.e., addition) are estimated using Spice simulations, (see Section 2.2).

Each predictor has two ports and includes a bimodal confidence estimator (CE) with three-bit saturating counters with a threshold of five, a penalty of three and an award of one. Predictions are made after decode, and the predictors are updated as soon as the true load value is available, there are no speculative updates, and an out-of-date prediction is made as long as there are pending updates to the same predictor line. We use the re-fetch misprediction recovery scheme [10]. It is identical to that used for recovering from branch mispredictions. We chose this mechanism because we believe a first value-prediction implementation is likely to use the same recovery scheme as branch prediction. As an energy-saving optimization, we do not recover from wrong predictions that were overwritten with the true load value before they were consumed.

The predictor configurations we investigated are shown in Table 1. For each of the six predictor sizes (2kB to 80kB), the table gives the number of entries (lines) in the predictor and the access latency in cycles.

Table 1. Predictor configurations.

(L = LV, S = ST2D, F = FCM3, D = DFCM3, S+F = ST2D+FCM3 hybrid)

Predictor name	Predictor entries	Access latency (cycles)	Predictor name	Predictor entries	Access latency (cycles)	Predictor name	Predictor entries	Access latency (cycles)
total predictor size ~ 2kB			**total predictor size ~ 5kB**			**total predictor size ~ 10kB**		
L_2	256	2	L_5	512	2	L_10	1024	2
S_2	256	2	S_5	512	2	S_10	1024	2
F_2	256	3	F_5	512	3	F_10	1024	3
D_2	128	3	D_5	256	3	D_10	512	3
S+F_2	128	3	S+F_5	256	3	S+F_10	512	3
total predictor size ~ 20kB			**total predictor size ~ 40kB**			**total predictor size ~ 80kB**		
L_20	2048	2	L_40	4096	3	L_80	8192	3
S_20	2048	3	S_40	4096	3	S_80	8192	4
F_20	2048	4	F_40	4096	5	F_80	8192	6
D_20	1024	4	D_40	2048	5	D_80	4096	5
S+F_20	1024	4	S+F_40	2048	5	S+F_80	4096	6

Six integer (*bzip2, gcc, gzip, mcf, twolf, vortex*) and four floating point (*ammp, art, equake, mesa*) C programs from the SPECcpu2000 benchmark suite [27] are used for the measurements in this paper. They were compiled on a DEC Alpha 21264A processor with the DEC C compiler under the OSF/1 v5.1 operating system using the "-O3 -arch host" optimization flags. We utilize the reference inputs provided with these programs and SimPoint [28] to select a representative subset (500 million instructions in length) for each benchmark program.

4. Experimental Results

4.1 Impact of Predictor Access Latency on Performance

Figure 1 shows the performance of the five predictors, each with sizes ranging from 2kB to 80kB. We report the geometric-mean results across all simulated programs. Ideally, a prediction should be available as soon as it is needed. In our 20-stage pipeline, there are four cycles from the time a load prediction is requested to the time its result may first be needed by a dependent instruction. As indicated in Table 1, the actual access latencies depend on the size and type of the predictor and can be as high as six cycles.

For the single-level predictors (LV and ST2D), the access latencies are within four cycles. Thus, the time it takes to read a predictor line has no negative impact on the performance. This is also true for the complex, multi-level predictors (FCM3, DFCM3 and ST2D-FCM3 hybrid) that are 20kB or less in size. However, above 20kB the higher access latencies significantly impact the performance of the simulated processor.

When not modeling the access latency, we observe a monotonic increase in performance for each predictor type as the predictor size increases, which is in line with findings in previous research. However, it should be noted that the rate of increase in performance is much lower than the rate of increase in predictor size. Both context-method predictors (FCM3 and DFCM3) outperform the simpler predictors (LV and ST2D) at sizes above 20kB. Moreover, the ST2D-FCM3 hybrid outperforms its unit predictors, albeit only minimally.

In reality, increasing the number of lines in the predictor tables lengthens the decode time. Accounting for the extra computation time of the complex predictors further increases their access latencies. When the predictor access latency exceeds the number of cycles (four in our case) required to keep a dependent instruction from stalling, i.e., waiting for a value to make forward progress, performance begins to taper off (DFCM3 above 20kB) or is diminished (FCM3 and hybrid above 20kB). With realistic latencies, the 80kB FCM3 performs worse than its 10kB counterpart. The same is true for the ST2D-FCM3 hybrid.

We expect these observations to hold for other processors as well. Even though they may have different pipeline lengths than our simulated CPU, a similar effect will take place above some predictor latency, and the large and complex predictors that are the most likely to exceed this threshold.

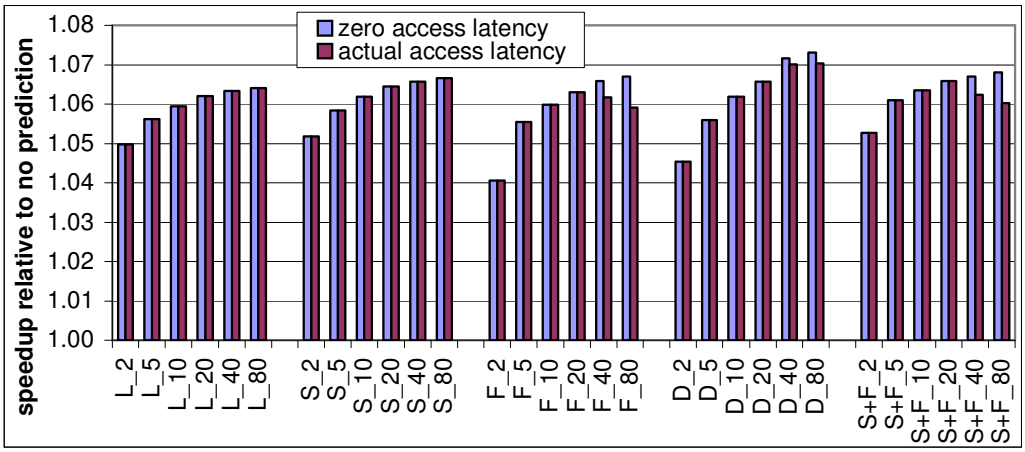

Figure 1. Geometric-mean performance using zero and actual access latencies.

4.2 Impact of Predictor Size on Performance

Figure 2 shows the performance (speedup over no prediction) of the five predictors sorted by speedup in each predictor size category. Despite the complexity of the FCM3 and DFCM3, these predictors do not outperform the much simpler predictors in the 2kB and 5kB categories. The ST2D delivers better performance than the more complex FCM3 when at most 20kB of state are allocated to the predictors.

The hybrid performs best for size categories up to 20kB. Interestingly, above 20kB even LV (the simplest predictor) outperforms the hybrid. This is due to the negative effect of the large access latency of the hybrid. Also, we observe that above 20kB the DFCM3 performs the best, revealing this predictor's superiority for large sizes. Unfortunately, due to energy constraints such large tables may not be practical for implementation in hardware. Note that the relatively simple ST2D performs well in all categories.

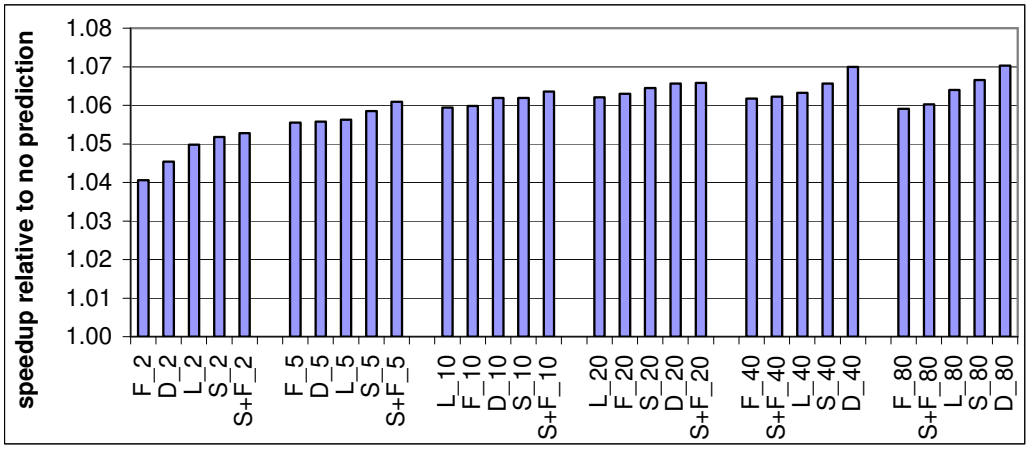

Figure 2. Geometric-mean performance with actual access latencies.

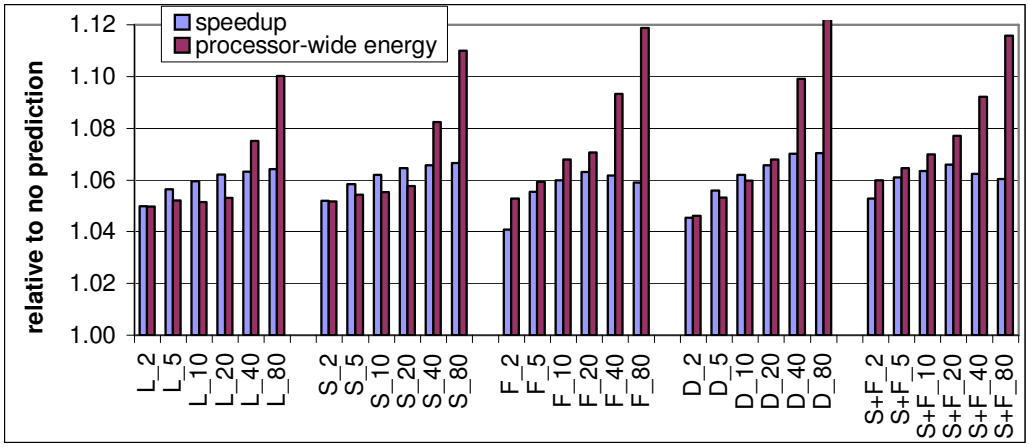

Figure 3. Geometric-mean speedup and energy consumption with actual access latencies.

4.3 Energy Consumption and Performance

With the increasing concern over energy consumption, the addition of any new hardware can only be justified by a significant performance improvement. Figure 3 shows the performance gain and extra processor-wide energy consumption for each predictor configuration across all the programs. Note that we show energy consumption for the entire processor and not just the predictors. We chose to show performance and energy expenditure separately instead of using metrics such as MIPS/Watt, the energy-delay product or the energy-delay-squared product for the simple reason that different conclusions can be inferred from each of these metrics and architects do not seem to agree on which metric conveys the best information [33]. Results for each individual program are provided in Appendix A.

For 5kB, 10kB and 20kB, LV and ST2D provide more performance gain than energy increase. This also holds true for the 5kB and 10kB DFCM3. For all other configurations, relatively speaking more energy is consumed than performance gained. Other processors would likely result in different percentages but we believe the trends would be the same. Beyond 20kB, we observe a significant increase in energy consumption for all predictor configurations. This is due to the combined effect that above 20kB, the extra performance gain is marginal while the predictor sizes are multiple times larger and consume correspondingly more energy.

Note that the extra energy use of the processor with load-value prediction includes the energy consumed by the predictors as well as that from the resulting speculation activities elsewhere on the chip. Hence, predictors that are too small (and thus result in many mispredictions) can also waste energy. This is especially obvious in the 2kB FCM3. Increasing the predictor size reduces the mispredictions and the corresponding waste in energy. However, the energy consumed by the predictor itself increases.

In general, we find that increasing the predictor size increases the performance only as long as the access latency remains reasonably low. Even then, a predictor cannot be made too large because above some optimal size the cost of energy consumption far exceeds the performance benefit. Adding complexity, with the goal of improving performance, increases the access latency and energy use of the predictor and ultimately may well eliminate the potential performance gains. Designing the predictor too small increases mispredictions, which negatively impacts performance. Thus, to maximize the energy-performance tradeoff, load-value predictors need not be too large, too complex or too small.

5. Conclusions

With increasing transistor budgets, aggressive speculation has proven to be a viable method to increase throughput and drive performance in high-end microprocessors. These speculation mechanisms usually require predictors that tend to be large and complex to obtain good performance. However, with power dissipation and energy consumption becoming a first-order design constraint, implementation of such complex hardware is unappealing.

In this paper, we take a closer look at load-value speculation and show that moderate-sized, simpler predictors often provide a better energy-performance tradeoff than more complex ones of similar table size.

We demonstrate that designers can expect good performance from seemingly simple predictors. We also show that a good predictor does not have to be overly large. For instance, we find that a 20kB ST2D provides a better energy-performance tradeoff than more complex and larger predictors. Adding complexity to predictors to increase their performance may have the opposite effect. Thus, in light of energy constraints, future research into load-value prediction should step away from the trend of increasing predictor complexity and size to improve performance. Rather, we believe the focus should be on using simpler predictors and enhancing the prediction algorithms.

6. Acknowledgment

This work has been supported in part by the National Science Foundation (NSF) under Award #0208567 and by a grant from Intel Corporation.

7. References

[1] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, D. Burger. Clock Rate versus IPC: The End of the Road for Conventional Microarchitectures. *27th Annual International Symposium on Computer Architecture*, 2000, pp. 248-259.

[2] R. Bhargava, L. K. John. Latency and Energy Aware Value Prediction for High-Frequency Processors. *16th International Conference on Supercomputing,* 2002, pp. 45-56.

[3] R. Bhargava, L. K. John. Performance and Energy Impact of Instruction-Level Value Predictor Filtering. *First Value-Prediction Workshop,* 2003, pp. 71-78.

[4] D. Brooks, V. Tiwari, M. Martonosi. Wattch: A Framework for High-Performance Microprocessors. *Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 171-182.

[5] D. Burger, T. M. Austin. The SimpleScalar Tool Set, version 2.0. *ACM SIGARCH Computer Architecture News,* 1997. http://www.simplescalar.com

[6] M. Burtscher, A. Diwan, M. Hauswirth. Static Load Classification for Improving the Value Predictability of Data-Cache Misses. *ACM SIGPLAN 2002 Conference on Programming Language Design and Implementation*, 2002, pp. 222-233.

[7] M. Burtscher, B. G. Zorn. Hybridizing and Coalescing Load Value Predictors. *International Conference on Computer Design*, 2000, pp. 81-92.

[8] M. Burtscher, B. G. Zorn. Prediction Outcome History-based Confidence Estimation for Load Value Prediction. *Journal of Instruction-Level Parallelism*, 1999. http://jilp.org/vol1/v1paper3.ps

[9] B. Calder, G. Reinman, D. M. Tullsen. Selective Value Prediction. *26th Annual International Symposium on Computer Architecture*, 1999, pp. 64-74.

[10] F. Gabbay. Speculative Execution Based on Value Prediction. *Technical Report 1080, Department of Electrical Engineering, Technion-Israel Institue of Technology*, 1996.

[11] F. Gabbay, A. Mendelson. Can Program Profiling Support Value Prediction? *30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 270-280.

[12] R. Gonzalez, M. Horowitz. Energy Dissipation in General Purpose Microprocessors. *IEEE Journal of Solid-State Circuits,* 1996, pp. 1227-1284.

[13] B. Goeman, H. Vandierendonck, K. de Bosschere. Differential FCM: Increasing Value Prediction Accuracy by Improving Table Usage Efficiency. *Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 207-216.

[14] D. A. Jimenez, S. W. Keckler, C. Lin. The impact of delay on the design of branch predictors. *33rd Annual International Symposium on Microarchitecture*, 2000, pp. 67-76.

[15] G. H. Loh. Width-Partitioned Load Value Predictors. *Journal of Instruction-Level Parallelism*, 2003, pp. 1-23.

[16] W. Lin, S. K. Reinhardt, D. Burger. Reducing DRAM latencies with an integrated memory hierarchy design. *Seventh International Symposium on High Performance Computer Architecture*, 2001, pp. 301-312.

[17] M. H. Lipasti, C. B. Wilkerson, J. P. Shen. Value Locality and Load Value Prediction. *Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996, pp. 138-147.

[18] R. Moreno, L. Pinuel, S. del Pino, F. Tirado. A Power Perspective of Value Speculation for Superscalar Microprocessors. *International Conference on Computer Design,* 2000, pp. 147-154.

[19] L. Pinuel, R. A. Moreno, F. Tirado. Implementation of Hybrid Context Based Value Predictors Using Value Sequence Classification. *Euro-Par*, 1999, pp. 1291-1295.

[20] G. Reinman, B. Calder. Predictive Techniques for Aggressive Load Speculation. *31st IEEE/ACM International Symposium on Microarchitecture*, 1998, pp. 127-137.

[21] B. Rychlik, J. Faistl, B. Krug, J. P. Shen. Efficacy and Performance Impact of Value Prediction. *International Conference on Parallel Architectures and Compilation Techniques*, 1998, pp. 148-154.

[22] T. Sato, I. Arita. Table Size Reduction for Data Value Predictors by Exploiting Narrow Width Values. *14th International Conference on Supercomputing,* 2000, pp. 196-205.

[23] T. Sato, I. Arita. Low-Cost Value Prediction Using Frequent Value Locality. *Fourth International Symposium on High Performance Computing,* 2002, pp. 106-119.

[24] Y. Sazeides, J. E. Smith. Implementations of Context Based Value Predictors. *Technical Report ECE-97-8, University of Wisconsin, Madison, Wisconsin*, 1997.

[25] Y. Sazeides, J. E. Smith. The Predictability of Data Values. *Thirteenth IEEE/ACM International Symposium on Microarchitecture,* 1997, pp. 248-258.

[26] P. Shivakumar, N. P. Jouppi. CACTI 3.0: An Integrated Cache Timing, Power and Area Model. Technical Report 2001/2. *Compaq Western Research Laboratory*, 2001.

[27] SPECcpu2000 benchmarks. http://www.spec.org/osg/cpu2000.

[28] T. Sherwood, E. Perelman, G. Hamerly, B. Calder. Automatically Characterizing Large Scale Program Behavior. *Tenth International Conference on Architectural Support for Programming Languages and Operating Systems,* 2002, pp. 45-57.

[29] S. T. Srinivisan, A. R. Lebeck. Load latency tolerance in dynamically scheduled processors. *Journal of Instruction Level Parallelism*, pp. 1-24, 1999.

[30] K. Wang, M. Franklin. Highly Accurate Data Value Prediction using Hybrid Predictors. *30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 358-363.

[31] H. Zhou, J. Flanagan, T. M. Conte. Detecting Global Stride Locality in Value Streams. *30th Annual International Symposium on Computer Architecture*, 2003, pp. 324-335.

[32] V. Zyuban, P. M. Kogge. Optimization of High-Performance Superscalar Architectures for Energy Efficiency. *2000 International Symposium on Low Power Electronics and Design,* 2000, pp. 196-205.

[33] V. Zyuban, P. Strenski. Unified Methodology for Resolving Power-Performance Tradeoffs at the Microarchitectural and Circuit Levels. *International Symposium on Low Power Electronics and Design*, 2002, pp. 166-171.

Appendix A – Per-benchmark speedup and processor-wide energy consumption

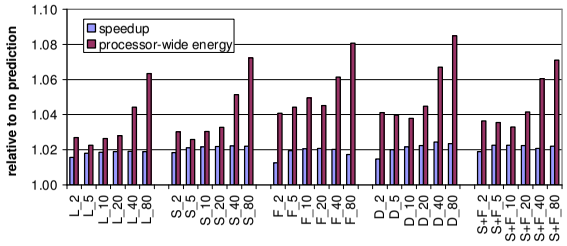

Fig A1. ammp

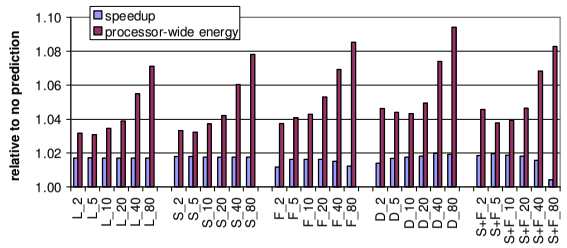

Fig A2. art

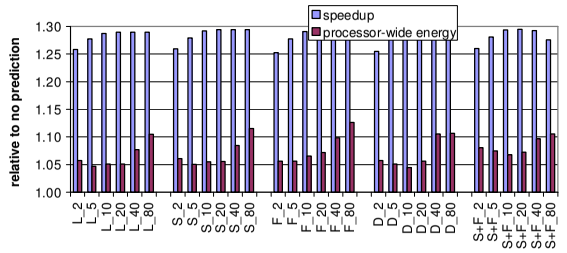

Fig A3. bzip2

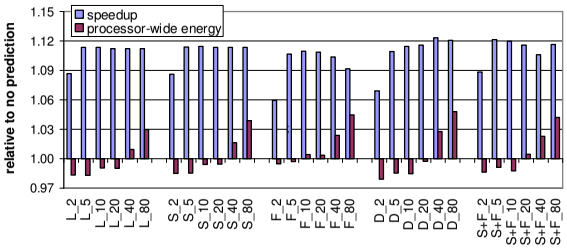

Fig A4. equake

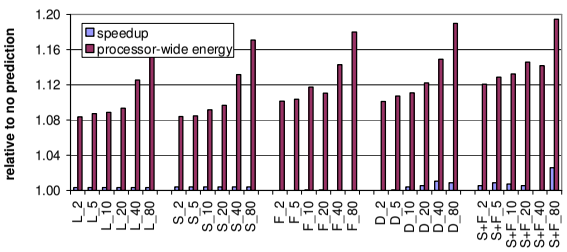

Fig A5. gcc

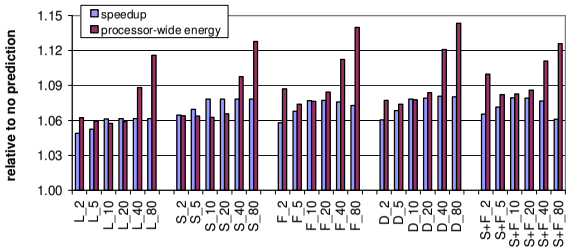

Fig A6. gzip

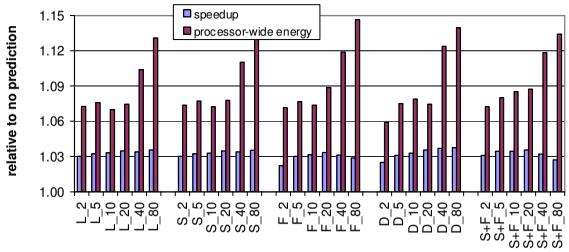

Fig A7. mcf

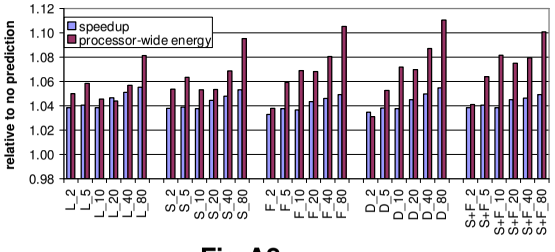

Fig A8. mesa

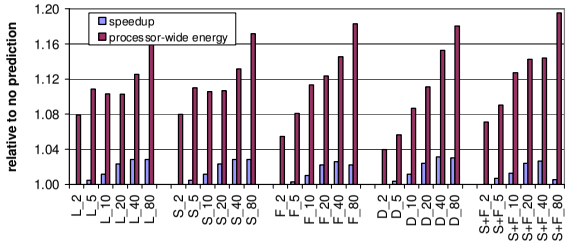

Fig A9. twolf

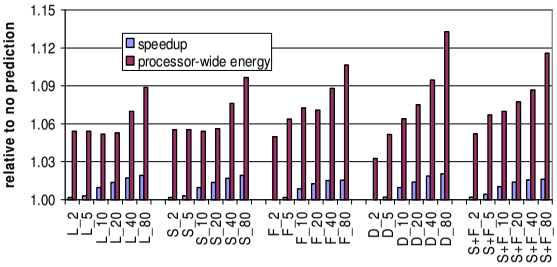

Fig A10. vortex