Improving Memory System Performance with Energy-Efficient Value Speculation

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

ABSTRACT

Microprocessor speeds have been improving much faster than memory speeds, resulting in the CPU spending a larger and larger amount of time waiting for data. Processor designers have employed several ways to improve memory performance, including hierarchical caching, prefetching, and faster memory chips. Yet, memory accesses still represent a major performance bottleneck and much remains to be done to tolerate the increasing memory latencies. Load-value prediction has been shown to effectively hide some of this latency. However, the hardware required to achieve good performance is substantial, making load-value prediction unappealing in light of increasing power constraints. In this paper, we present a novel predictor that significantly increases CPU performance while at the same time decreasing the energy consumption of the entire processor relative to a baseline with a well-performing hybrid load-value predictor.

1. INTRODUCTION

While processor speeds have been increasing by about a factor of two every 18 months in keeping with Moore's Law, memory speeds have only been improving by 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 a significant portion of the time idle because the memory cannot serve data at a sufficient speed. Finding solutions to bridge this gap has kept designers engaged for over a decade now.

Techniques to help resolve the imbalance between fast processors and slow memory include large and deep caching, non-blocking caches, cache-conscious load scheduling, hardware and software prefetching, and data speculation. Conventional cache systems rely heavily on the temporal and spatial locality of programs. However, programs also exhibit value locality [12]. Hence, value prediction has the potential to be a powerful supplement to the cache system's effectiveness. Value locality describes the correlation of previously seen values with future values in registers and memory cells. It allows the classical dataflow limit to be exceeded by executing instructions before their operands have been computed. Several studies on exploiting different types of value locality have shown that most instructions generate predictable results. However, load instructions are by far the most critical to overall performance [7].

To exploit as much load-value locality as possible, many predictors have been proposed, including hybrids that combine several component predictors [5], [17], [26] and incorporate a selection mechanism to determine the best component for predicting each dynamic instruction. The selection mechanisms include using the confidence of a component [17], the type of value sequence [15], or the characteristic of the load [18]. Poor selectors, however, can nullify the performance benefit of a hybrid predictor. Moreover, hybrids require sizeable hardware structures that consume large amounts of energy. With the power dissipation of modern processors becoming one of the most critical issues facing designers [11], it is imperative that predictors provide as much performance for as little power as possible. In this paper, we show that the selector in a hybrid can be a significant source of inefficiency and propose the novel *cycling hybrid predictor*, which is designed to improve the selection mechanism and the energy-efficiency. We find that this novel predictor can simultaneously increase the CPU performance and reduce the energy consumption of the processor.

The remainder of this paper is organized as follows. Section 2 describes our energy-efficient cycling hybrid predictor. Section 3 presents the simulation framework. Section 4 provides performance results and analyses. Section 5 gives an overview of related work and Section 6 summarizes the contributions of our study.

2. HIGH-PERFORMANCE, ENERGY-EFFICIENT VALUE SPECULATION

2.1 High-Performance Value Prediction

Load instructions represent a major performance bottleneck but they have been shown to fetch predictable sequences [12]. In fact, load-value prediction, especially using hybrid predictors [6], [17], [26], has been demonstrated to effectively hide some of the memory latency and thus enhance performance. Hybrid predictors employ a selection mechanism responsible for choosing the most suitable component for each prediction. The three most often used predictor components are described next.

The last value predictor (LV) [8], [12], [22] predicts that a load instruction will load the same value it did the previous time it executed.

The stride 2-delta predictor (ST2D) [22] stores the last value for each load and also maintains a stride, i.e., the difference between the last two loaded values. It can predict sequences with zero or non-zero differences between consecutive values. When a load completes, ST2D updates the last value but updates the stride only if it encounters the same stride twice in a row. This update hysterisis has been shown to significantly increase performance [22].

The third-order differential finite context method predictor (DFCM3) [9] computes a hash value [16], [17], [22] out of the last three strides to index the predictor's second-level table, which is shared by all loads. This table stores the strides that followed all previously seen sequences of three strides (modulo the table size). After encountering a sequence of load values for the first time, DFCM3 can predict any load that loads the same sequence or a different sequence with the same strides.

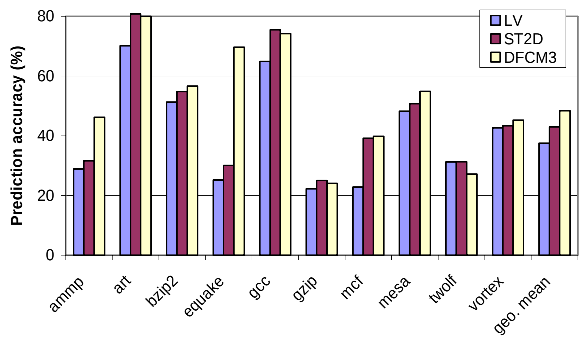

Figure 1. Performance of predictors when all loads are predicted

Figure 1 shows the accuracy that can be expected from these predictors for the ten SPECcpu2000 programs we evaluated. On average fewer than 50% of the load values can be predicted correctly with these predictors.

Because making no prediction and waiting for the memory access to complete is faster than making an incorrect prediction and having to recover from it, most predictors from the literature include a confidence estimator. Confidence estimators inhibit predictions that are likely to be incorrect [4] and thus reduce the number of mispredictions and the associated recovery cost, which improves the predictor's overall performance. The frequently-used bimodal confidence estimator [12], [16], [17] is based on saturating up/down counters with four parameters: a maximum, a threshold, a penalty and an award. The maximum is the upper bound of the counter (the minimum is always zero). A value 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 the counter is incremented by the award.

Rychlik et al. [17] introduced a hybrid predictor that combines a stride and a finite context method (FCM) predictor. The component with the highest confidence makes the prediction. In case of a tie, the FCM is given priority since it provides the best accuracy. The authors showed that the hybrid predictor was more effective than either of the component predictors. We incorporate their selection scheme in a hybrid of a last value, a stride-2-delta and a third-order differential finite context method predictor. In the event of a tie in confidence, the DFCM3 is given priority over the ST2D, which has priority over the LV.

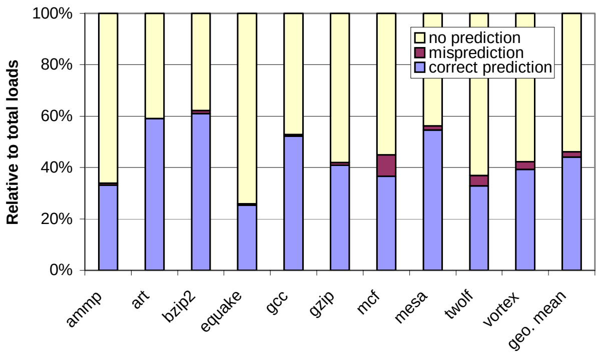

Figure 2. Fraction of loads not predicted, mispredicted and correctly predicted by a conventional hybrid with a confidence estimator

Figure 2 shows that on average, 44.1% of the loads are predicted by the conventional hybrid with 98% accuracy. However, a large percentage (over 55% on average) of the loads is not predicted. This is especially true for *equake* where fewer than 30% of the loads are predicted.

Figure 3 provides a closer examination of which components were selected for making the predictions. For most of the programs the LV component is used rarely. This is partly due to the fact that the LV component has the lowest priority in the event of a confidence tie and partly because LV is the weakest component as Figure 1 shows. Nevertheless, the majority of the predictions could be made by any of the three components.

The goal of putting different predictors in a hybrid is to maximize the loads that are predicted correctly. If a component in the hybrid is hardly used, it consumes energy needlessly. Moreover, DFCM3 is the largest and most complex component in the hybrid and therefore the component that consumes the most energy on each access. Because power consumption is becoming a major design constraint, it is imperative that each unit in the predictor be used efficiently.

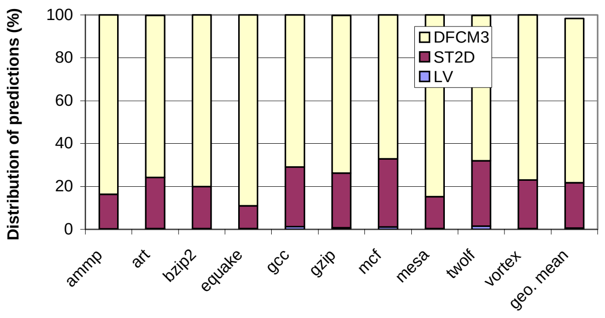

Figure 3. Percent of loads directed to each component in the conventional hybrid predictor

2.2 The Cycling Hybrid Predictor

In order to efficiently use the components in the hybrid predictor, we designed the cycling hybrid predictor. This predictor also reduces selection-related losses and decreases energy consumption.

The counters in confidence estimators are incremented in small steps when a predictable value is seen and decremented in large steps when the value is unpredictable. As mentioned earlier, the conventional hybrid uses a priority scheme to determine which component to use for a prediction when there is a tie in the confidence. However, most loads are best predicted by a particular component. In addition, different loads may be mapped to the same predictor entry, resulting in interference that often leads to the selector switching back and forth between components. To offset these weaknesses, we devised the *cycling selector*.

How the *cycling selector* works: It comprises a saturating up/down counter and a component pointer. Each predictor line has its own component pointer and counter (Figure 4). The component pointer indicates which component to use for the next prediction. When the value is predictable, the counter is set to the maximum, otherwise it is decremented by one. We found these parameters to result in the best performance enhancement. Note that this is opposite of how confidence estimators work. For as long as the counter remains above zero, the component pointed to is not altered. Just like in the conventional predictor, a prediction is only made if the confidence of the particular component is above a pre-defined threshold. When the counter reaches zero, the component pointer moves to a new component and the counter is reset to the maximum. The components are traversed in a round robin fashion, hence the name “cycling selector”.

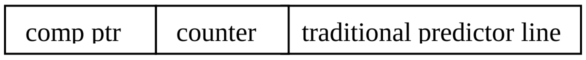

Figure 4. Each predictor line is extended with the cycling selector (component pointer and counter)

By cycling through the hybrid’s components, each load gets to try a component for some time without polluting the remaining components. If the current component turns out to be ineffective, the selector advances to the next component. This continues until the load settles on a good component.

Because multiple loads can map to the same line, the confidence of that line is affected by the predictability of all loads accessing that line. Thus, unpredictable loads can pollute the confidence of predictable loads. Since the conventional selector relies on this confidence, it can easily select the wrong component. Using the cycling selector, a load is forced to stay with a component at least for a while. This way, the effects of negative aliasing in the confidence estimator are drastically reduced. Additionally, since only the selected component is updated, the predictor tables also experience less pollution.

We use 4-bit counters in the cycling selector and initially set each one to the maximum, i.e., 15. If a value is unpredictable, the counter is decremented by one. Otherwise, it is reset to 15. The pointers are initialized with different components, i.e., the component pointer associated with the first predictor entry is initialized with LV, the second with ST2D, the third with DFCM3, the fourth with LV, etc. This scheme is termed *cycling_hybrid_4*.

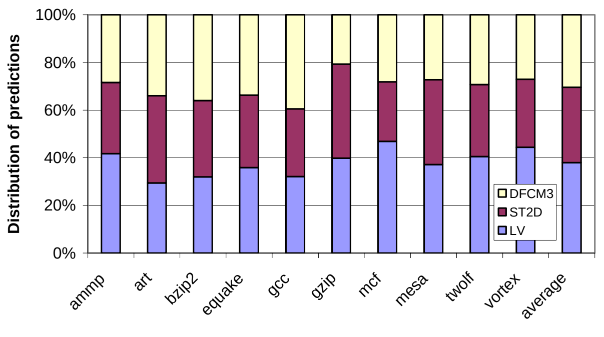

Figure 5. Percent of dynamic loads directed to each component in the *cycling_hybrid_4* predictor

The effect of the cycling nature of the selector is evident in Figure 5. Because the components are traversed in round-robin fashion, their usage is nicely balanced. This is the effect we had hoped for. The lower-power components are now used more frequently, the pollution of the confidence estimator and prediction tables is reduced, and each load is still predicted by a good component. As a result, the cycling hybrid outperforms the already high-performing conventional hybrid predictor and reduces the energy consumption of the processor. To the best of our knowledge, no previous work has described a hybrid selection mechanism that improves both performance and energy consumption simultaneously.

3. EXPERIMENTAL FRAMEWORK

Our experiments were conducted using a detailed, cycle-accurate simulator derived from the SimpleScalar/Alpha 3.0 tool set [3]. We incorporated value prediction into this simulator and integrated it with the Wattch power model [2] to obtain the energy data. Wattch provides switching capacitance modeling for structures like ALUs, caches, arrays and busses in a processor.

3.1 Processor Configuration

Our baseline architecture is an 8-way superscalar out-of-order CPU with 20 pipeline stages, a 128-entry instruction window, a 64-entry load/store queue, 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, two load/store units, six integer ALU units, 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. All functional units except the divide unit are pipelined to allow a new instruction to initiate execution each cycle. It takes 300 cycles to access main memory. ‘No store alias’ dependence prediction is enabled to predict aliases between load and store instructions [17].

Wattch’s linear scaling is used to obtain energy results for 0.13μm technology, V_{dd} = 1.3V and a clock speed of 2.0 GHz. V_{th} is 0.38V. The cache and predictor latencies are obtained with Cacti 3.2 [24]. Static power is estimated as 25% of dynamic power.

This baseline processor is augmented with a hybrid predictor with an LV, an ST2D and a DFCM3 component. Each component has 1024 entries in its tables. DFCM3 has two such tables. The predictors include a bimodal confidence estimator (CE) with three-bit saturating counters with a threshold of five, a penalty of three and an award of one. The same CE configuration is used for all predictors. Predictions are made after decode, 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 scheme [8]. It is identical to that used for recovering from branch mispredictions. As an energy-saving optimization, we do not recover from wrong predictions that are overwritten with the true load value before they were first used.

3.2 Benchmark Programs

Ten C programs (six integer and four floating-point) from the SPECcpu2000 benchmark suite [25], together with the provided reference inputs, are used in our evaluation. Each program was compiled on a DEC Alpha 21264 processor using the “–O3 –arch host” optimization flags. We employed SimPoint [23] to select a representative subset (500 million instructions in length) of each benchmark trace. Table 1 shows the number of instructions (in billions) that are skipped before beginning the cycle-accurate simulations, the number of simulated load instructions (in millions), the percentage of simulated instructions that are loads and the IPC on the baseline CPU, for each program.

Table 1. Information about the simulated segments of the benchmark programs

program	skipped insts (B)	simulated loads (M)	% loads	base IPC
ammp	27.5	134.1	26.8	1.565
art	6.5	162.5	32.5	1.429
bzip2	19.5	145.9	29.2	1.691
equake	131.5	235.1	47.0	0.369
gcc	4.0	228.2	45.6	1.144
gzip	3.0	121.8	24.4	1.378
mcf	23.0	209.7	41.9	0.501
mesa	67.5	129.5	25.9	1.795
twolf	247.0	142.6	28.5	1.219
vortex	106.5	127.2	25.4	1.858
geo. mean	57.8	148.8	29.8	1.162

4. RESULTS AND ANALYSES

This section presents the performance evaluation of the cycling hybrid predictor. Unlike most previous work in value speculation that considered energy, we take the energy consumption of the entire processor into account, not just the predictor. This is essential because adding value prediction increases the energy consumption in several parts of the processor due to the increase in speculative activity, whether useful or not [14], [19].

Figure 6 shows the IPCs for each program when utilizing our technique compared to the conventional hybrid predictor. It is worth mentioning that the conventional method provides substantial performance improvement, up to 28% in the ten programs we study, over no prediction at all. Figure 6 demonstrates that our approach outperforms the conventional hybrid, especially for *mcf* and *twolf*. In fact, for *mcf* it results in a 30% increase in performance. *twolf*'s loads are better predicted by LV and ST2D than by DFCM3. However, the conventional hybrid makes only 32% of predictions with these two components (Figure 3). On the other hand, the cycling hybrid makes 71% of the predictions for *twolf* with the LV and ST2D components (Figure 5). Note that *equake* is the only program whose performance suffers with our method. This is because *equake*'s loads are much better predicted by DFCM3 than LV or ST2D (Figure 1). Therefore, *equake* benefits when the conventional hybrid makes 89% of prediction with the DFCM3 component. On the other hand, the load-balancing feature in the cycling hybrid uses the DFCM3 for only 34% of the predictions.

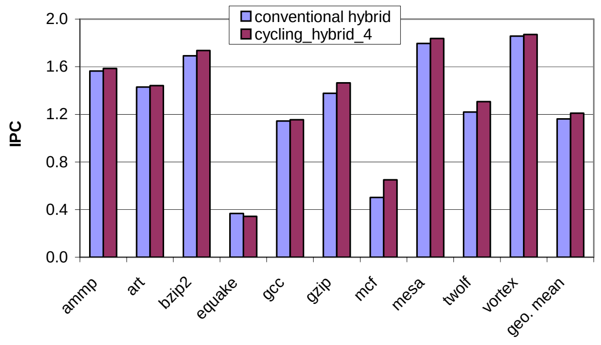

Figure 6. IPCs using the cycling hybrid compared to using the conventional hybrid

As demonstrated next, the cycling hybrid not only enhances performance but also reduces the energy consumption. Figure 7 shows how much energy the processor consumes when running each program with the two hybrid predictors. Again, with the exception of *equake*, the cycling hybrid consistently provides energy savings over the conventional hybrid. *mcf* benefits the most; the processor with the cycling hybrid consumes about 42% less energy than that with the conventional hybrid. Note that this is a processor-wide energy reduction of 42%, not just in the predictor. It is worth noting that *mcf* is a memory-bound program, i.e., it spends a substantial amount of time waiting for data, during which time the processor is expending energy needlessly. By hiding more of the memory latency with our hybrid, the processor is able to make progress and save energy.

In general, the higher performance-to-energy ratio obtained with the cycling hybrid is primarily due to the fact that predictor pollution is reduced, which results in more correct predictions and more correct confidence estimations. This in turn speeds up the processor and reduces the overall energy requirement.

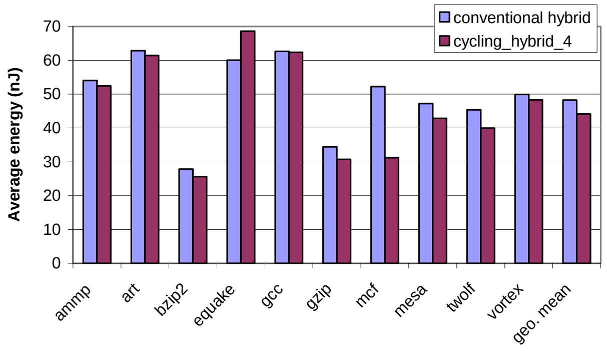

Figure 7. Processor-wide energy consumption of the cycling hybrid compared to the conventional hybrid

Furthermore, the fewer mispredictions lower the energy expenditure due to useless speculation activities. Finally, by having the simpler predictor components make more predictions than in the conventional hybrid, less energy is consumed.

Sensitivity to predictor size: Figure 8 presents the IPCs for each program for different numbers of entries in the hybrid components. When the size of the cycling hybrid is halved (cycling_hybrid_4[512]), i.e., reduced from 1024 to 512 predictor entries, the performance is still within 99% of that of cycling_hybrid_4[1024]. When we further reduce the number of predictor entries to 256, the performance reduction is less than 5%, with a concomitant savings in energy. In fact, *mesa* and *vortex* show a slight improvement in performance when less state is used. This indicates a potential for reducing energy consumption even further with little loss in speedup. Note that cycling_hybrid_4[256] still outperforms the conventional hybrid even though it has an almost four times smaller die-area requirement and can be accessed faster.

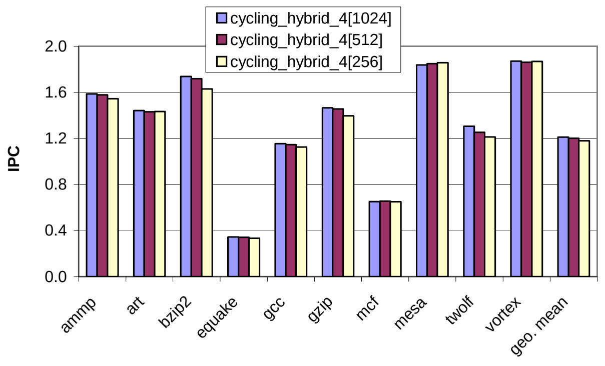

Figure 8. IPCs of cycling hybrids with varying number of predictor entries

Sensitivity to the counter size: So far, a 4-bit counter per predictor entry has been employed as part of our selection mechanism (see Section 2.2). To determine how much hysterisis is required for good performance, we varied the size of these counters, and consequently the maximum value of the counter. The number of predictor entries is fixed at 1024 for this study.

Figure 9 shows the IPCs for each program when the counter size is varied between 2 and 6 bits. We observe that the predictor is largely insensitive to the maximum count. For most of the programs, we find that using a 5-bit counter provides the best performance to energy ratio. When the count size is too small, e.g., 2 bits, there is not enough hysteresis and the selector switches from one component to the next too quickly. When the counter size is larger than 5 bits the performance benefit begins to diminish.

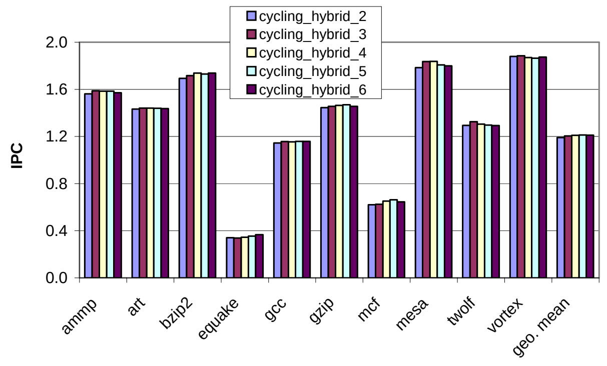

Figure 9. IPCs of cycling hybrids for varying counter sizes

5. RELATED WORK

Improving memory system performance with load-value speculation has been studied extensively [8], [9], [10], [12], [17], [22], [26]. However, in recent years the energy requirements of the predictors have started to receive some attention. Techniques proposed to reduce energy consumption include sharing common tables in hybrid predictors [5], predicting only frequently occurring values [21], partitioning tables into smaller ones [13], [20], and predicting instructions selectively [1], [7]. Unlike in these prior studies, we focus not only on the energy consumption of the predictor but on the entire processor. This is important because some power-saving techniques increase the number of mispredictions and recovering from mispredictions has a significant negative impact on the energy consumption of the processor as a whole [14], [19].

The most commonly used selection mechanism in the literature was introduced by Rychlik et al. [17]. It chooses the component with the highest confidence value and gives priority to the better performing component in the case of a tie. We compare our selector to theirs in this paper. Even though their mechanism provides good performance, our approach utilizes the hybrid components in a more balanced way while increasing the performance and energy-efficiency of the predictor, and reducing the energy consumption of the entire processor.

Pinuel et al. [15] proposed a hybrid predictor that combines a last value, a stride and a finite context method predictor. They suggested a finite state machine that selects which component to use for the next prediction based on the classification of value sequences. For each sequence the component with the lowest hardware cost is used to make the prediction. Our approach provides a simpler alternative to tracking and classifying sequences.

In a previous publication, the authors proposed a hybrid selector that exploited the fact that microprocessors support different types of loads and that each load type is best predicted by a particular component. That approach is dependent on profile information, while the cycling hybrid predictor is dynamically adaptive.

6. CONCLUSIONS

Current microprocessors execute instructions very fast provided that long latency memory operations are not involved. Thus, the increasing memory latencies represent a major setback in exploiting instruction-level parallelism. Fortunately, load-value prediction has been shown to be an effective latency tolerating technique. By correctly predicting the value of a load instruction, dependent instructions can avoid stalling while the memory is being accessed. However, value prediction has remained undesirable because high-performing predictors cause the processor to consume large amounts of energy.

In this paper, we propose the novel cycling hybrid predictor that outperforms the already well-performing conventional hybrid while significantly reducing the energy consumption of the processor. We describe a novel selector that uses simple counters to dynamically ‘cycle’ through the hybrid components and assign the best component to each load instruction. By employing our selector, the components of a hybrid are put to use in a much more balanced way than in a conventional hybrid. Consequently, the less complex components are used more often and more efficiently, saving on overall energy. Additionally, by updating only the relevant components identified by the cycling selector, predictor pollution is greatly reduced and performance is increased over the conventional hybrid. We also show that our cycling hybrid can be made much smaller, while still outperforming a larger conventional hybrid.

Memory-bound applications suffer the most in execution time and energy consumption from the increasing memory latencies. In this work, we have demonstrated that our simple cycling hybrid predictor can effectively offset this behavior by hiding the latency, thus improving performance and reducing energy consumption at the same time.

7. ACKNOWLEDGEMENT

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

8. REFERENCES

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

[2] 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.

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

[4] M. Burtscher, B. G. Zorn. Prediction Outcome History-based Confidence Estimation for Load Value Prediction. *Journal of Instruction-Level Parallelism*, 1999.

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

[6] M. Burtscher, B. G. Zorn. Hybrid Load-Value Predictors. *IEEE Transactions on Computers*, 2002, pp. 759-774.

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

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

[9] 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.

[10] J. Gonzalez, A. Gonzalez. The Potential of Data Value Speculation to Boost ILP. *12th International Conference on Supercomputing*, 1998, pp. 21-28.

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

[12] 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.

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

[14] 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.

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

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

[17] 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.

[18] N. B. Sam, M. Burtscher. Exploiting Type Information in Load-Value Predictors. *Second Value-Prediction and Value-Based Optimization Workshop*, 2004, pp. 32-39.

[19] N. B. Sam, M. Burtscher. On the Energy-Efficiency of Speculative Hardware. To appear in *2005 ACM International Conference on Computing Frontiers*, 2005.

[20] 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.

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

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

[23] 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.

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

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

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