On the Energy-Efficiency of Speculative Hardware

Nana B. Sam
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, USA
+1-607-255-7488
besema@csl.cornell.edu

Martin Burtscher
Computer Systems Laboratory
Cornell University
Ithaca, NY 14853, USA
+1-607-254-6580
burtscher@csl.cornell.edu

ABSTRACT

Microprocessor trends are moving towards wider architectures and more aggressive speculation. With the increasing transistor budgets, energy consumption has become a critical design constraint. To address this problem, several researchers have proposed and evaluated energy-efficient variants of speculation mechanisms. However, such hardware is typically evaluated in isolation and its impact on the energy consumption of the rest of the processor, for example, due to wrong-path executions, is ignored. Moreover, the available metrics that would provide a thorough evaluation of an architectural optimization employ somewhat complicated formulas with hard-to-measure parameters.

In this paper, we introduce a simple method to accurately compare the energy-efficiency of speculative architectures. Our metric is based on runtime analysis of the entire processor chip and thus captures the energy consumption due to the positive as well as the negative activities that arise from the speculation activities. We demonstrate the usefulness of our metric on the example of value speculation, where we found some proposed value predictors, including low-power designs, not to be energy-efficient.

Categories and Subject Descriptors

C.1.1 [**Computer Systems Organization**]: Processor Architectures – *pipeline processors.*

General Terms

Design, Measurement, Performance.

Keywords

Energy-Efficiency, Energy-Performance Metric, Speculation.

1. INTRODUCTION

Today's high-end microprocessors try to extract and exploit more instruction-level parallelism than ever before. However, the traditional emphasis on performance often leads to designs that waste energy. For instance, the rapid increase in the complexity and speed of each new processor generation cannot be compensated for by reducing the supply voltage. Consequently, organizational choices and tradeoffs need to be made with energy in mind, and designers are increasingly being challenged to come up with novel ways to reduce energy while trying to meet all other constraints imposed on the design.

Most research on energy optimization or estimation has focused on single components of the system, such as the on-chip memory, the processor core or the branch predictor. While this approach is good for optimizing individual units, it is even more important to evaluate the impact of hardware and software optimizations on the whole chip. After all, the introduction of new components may cause interactions that change the power activity in the rest of the system in significant ways, which is especially true for speculative hardware.

Control speculation, data dependence speculation, hardware prefetching, and other speculative mechanisms allow the processor to make forward progress without waiting for long-latency operations to complete. However, even though speculation can greatly improve performance, it also increases power dissipation and possibly energy consumption. This increase is caused not only by the speculative hardware, but also by useless activities in other components that are performed by instructions that are later discarded due to a misspeculation. A useless instruction contributes to the dynamic power consumption through data path switching activity until it is removed from the pipeline. Hence, when designing an energy-efficient speculative optimization, it is necessary to consider the impact of the speculation activities on the whole chip.

This paper makes the following contributions. First, we introduce a simple, processor-wide energy-efficiency metric that is based on cycle-accurate energy estimation and static supply voltage scaling. We developed this metric out of a need for an accurate energy-performance metric for speculative optimizations in one of our research projects. The general relation derived by Zyuban et al. [27] for the optimal balance between the architectural complexity, hardware intensity and power supply was the closest for our purposes. Unfortunately, it was difficult to accurately measure some of the formula's parameters, such as the architectural complexity. Our metric, on the other hand, has well-defined and measurable parameters and has proven useful in our research. It only needs the targeted supply and threshold voltages as well as the expected average speedup and energy increase due to the new speculative hardware to determine whether it is worthwhile adding this hardware to the processor. Second, we use our metric to examine the energy-efficiency of several architectural optimizations in the domain of value speculation. We chose this domain because even though many different value predictors have been proposed, to date none have been implemented in hardware. Moreover, the analysis of the energy consumption of value speculation is relatively new and to our knowledge, no prior study has compared multiple predictors using an energy-efficiency metric that includes the effect of speculation activities in the entire microprocessor. In fact, we were surprised to find that some supposedly energy-efficient designs turned out not to be. While this paper assumes an aggressive, dynamically scheduled, wide, superscalar processor, we believe many of our findings apply to other processors as well.

The rest of the paper is organized as follows. Section 2 discusses energy-performance considerations in speculative architectures. Section 3 describes our energy-efficiency metric. Section 4 presents the simulation framework. In Section 5, we discuss our evaluations and results. Section 6 summarizes our conclusions.

2. SPECULATION AND ENERGY-PERFORMANCE TRADEOFFS

2.1 Energy Consumption in Speculative Architectures

The increasing density of on-die transistors has enabled designers and researchers to explore novel ways to improve instruction throughput. One strategy for using these transistors is to increase execution resources and to use aggressive speculation techniques. Examples include branch prediction, which removes control dependencies, and memory dependency prediction, which removes false dependencies between store and load instructions.

In the past years, value speculation has been proposed to eliminate true data dependencies between instructions. Load instructions have been shown to fetch predictable sequences [8], [14] and several predictors have been proposed including single-level [8], [9], [14], [21], multi-level [11], [20], [26] and hybrid [7], [16], [18], [25] predictors. Even though value speculation shows potential for increasing the performance of future microprocessors, the extensive hardware budgets and high energy consumption of many of these predictors cannot be ignored. To improve the prediction accuracy, these structures are made as large as possible, which increases their energy consumption. Interestingly, making them too small can also waste energy due to an increased number of misspeculations. For example, branch mispredictions are responsible for about 28% of the power dissipated in a typical processor [1].

The power limitation of high-performance microprocessors is already critical to their design [3], [10]. To address the energy consumption problem some researchers have suggested value predictors that take space and power limitations into consideration [2], [7], [13], [16], [19].

Unfortunately, evaluations of these recommended energy-saving techniques have only focused on the predictors themselves without considering other sources of energy consumption introduced by the speculation activities. Moreno et al. [15] observed that recovery from mispredictions has a significant negative impact on the energy consumption of the microprocessor. Figures 1 and 2 show the energy consumption of some of the major hardware structures for a processor with and without value prediction, for the two sample SPECcpu2000 programs *gcc* and *mcf*, respectively. The simulation parameters used are specified in Section 4. In both figures we see that adding value prediction to the processor significantly increases the energy consumption of the register file, the result bus, the instruction window and the global clock. Note that the energy consumption of the value prediction unit is substantially less than the total increase in energy consumption of the other units.

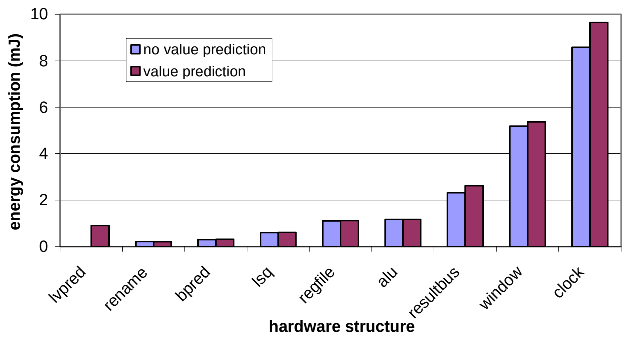

Figure 1. Energy consumption of common hardware structures in microprocessors with and without value prediction for *gcc*

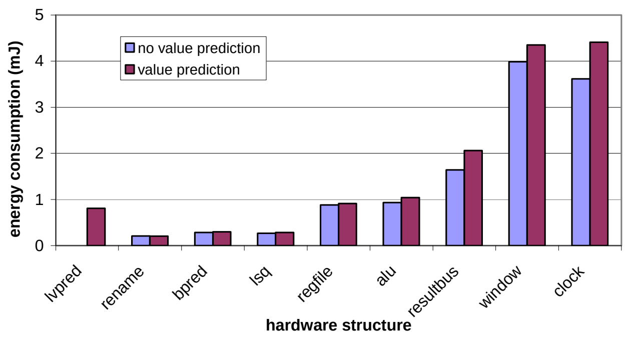

Figure 2. Energy consumption of common hardware structures in microprocessors with and without value prediction for *mcf*

On average, across the ten SPECcpu2000 programs we used for this study, we found that when value speculation was incorporated into the processor, the total increase in energy consumption of the rename unit, register file, load/store queue, functional units, result bus, instruction window, branch predictor, global clock and caches was 3.95 times that of the value predictor. Thus, while it is important to design energy-efficient processor units, it is even more important that their evaluation involve the whole speculative system.

2.2 Energy-Performance Tradeoffs and Metrics

The tradeoff between performance and energy consumption has received much attention in recent years. Designing energy-efficient microprocessors requires consideration of the energy consumption at early stages in the development where the opportunity for making energy-performance tradeoffs is the highest. Several parameters are involved in a given architecture, and different combinations of architectural parameters result in design points with different performance and energy efficiencies. A reliable metric should make knowledgeable energy-performance tradeoffs in this multi-dimensional space.

A number of energy-performance metrics have been proposed, some of which have been used to compare different products on the market. The 'MIPS per Watt' metric has been used to compare low-end products and to trade-off throughput and energy consumption [6]. It has also been employed to analyze high-performance processors, whose energy at maximum speed exceeds the power-dissipation capabilities of the package. The energy-delay product has been shown to be a more reasonable metric than 'MIPS per Watt' [12] for evaluating the energy efficiency at the microarchitectural level [10]. Formulas placing more emphasis on performance by raising the exponent of 'MIPS' have also been used to compare high-end server microprocessors [3].

To evaluate the energy efficiency of architectural features at early design stages, Zyuban et al. [27] derived a metric that combines relative changes in the architectural speed, dynamic instruction count, average energy dissipated per executed instruction, and maximum clocking rate of the processor that result from design modifications at the architectural and microarchitectural levels. Their formula subsumes previously used energy-performance metrics [6], [10] as special cases of a more general equation. Unfortunately, it is difficult to use this metric to evaluate significant architectural changes, such as the addition of value speculation to a processor, because some of the parameters are almost impossible to obtain and it is unclear how to account for the unpredictable behavior associated with speculation.

3. AN ENERGY-EFFICIENCY METRIC FOR SPECULATIVE HARDWARE

It is practically impossible to determine the optimal design point for an architecture because the optimality criteria depend on the type of processor. There are configurations targeted at achieving the maximum performance and others targeted at achieving the minimum energy dissipation per instruction. Between these two extremes are configurations with a reasonably high performance and reasonably low energy. To better understand the energy-performance tradeoff, it is helpful to define an energy-efficient configuration as one that delivers the highest performance among all the configurations consuming the same amount of energy. An alternative definition is that it is the one that consumes the least energy among all configurations that deliver the same performance. We consider these definitions equivalent since either one suffices to define an energy-efficient configuration.

Since our focus is to analyze speculative architectures, we need a metric that provides a dynamic processor-wide energy-performance analysis. To measure the energy-efficiency of a design, we start with an initial architectural configuration, which we call CPU_{orig}. We enhance CPU_{orig} with the speculative optimization under investigation and call this configuration CPU_{enh}. We then measure the performance and the energy consumed by CPU_{orig} and CPU_{enh}. We use SimpleScalar [5] as our basic cycle-accurate simulation engine because of its wide adoption in the microprocessor research community and because it models a typical modern microprocessor architecture. We integrate Wattch [4] into this simulator to obtain detailed energy-consumption information.

We then perform a post-simulation analysis, which involves scaling the supply voltage V_{dd} of CPU_{enh}, which is treated as the independent variable in the optimization process. To achieve the desired energy and performance characteristics, we assume the supply voltage can be set to any value in the range for which the technology is qualified. We define ΔT as the speedup of CPU_{enh}, and ΔE as the corresponding energy increase. Given the threshold voltage V_{th}, let

$$\Delta E = xV_{dd}^{\ 2} \qquad \Rightarrow \qquad x = \frac{\Delta E}{V_{dd}^{\ 2}}$$

$$\frac{1}{\Delta T} = \frac{yV_{dd}}{(V_{dd} - V_{th})^2} \qquad \Rightarrow \qquad y = \frac{(V_{dd} - V_{th})^2}{\Delta T V_{dd}}$$

'x' is used to establish a direct relation between power and supply voltage (the variable in our metric). This simple representation of the well-known CMOS power dissipation equation allows us to better demonstrate how we arrive at our final relation. Likewise, the use of 'y' establishes a direct relation between the execution time (delay) and the supply voltage. The relations can be expanded to show that

$$x = (pCf_{clk}) + S$$

$$y = kC$$

where p is the switching probability, C is the load capacitance (wiring and device capacitance), f_{clk} is the clock frequency, and S is a factor that expresses static power as a function of V_{dd}^2. k is a proportionality constant specific to a given technology. We assume the carrier velocity saturation to range between 1 and 2.

To equalize the energy consumption of CPU_{enh} with that of CPU_{orig}, the supply voltage V_e of CPU_{enh} has to be

$$xV_e^{\ 2} = 1 \qquad \Rightarrow \qquad V_e = \frac{1}{\sqrt{x}}$$

Similarly, to equalize the performance of CPU_{enh} with that of CPU_{orig}, the supply voltage V_t of CPU_{enh} has to be

$$\frac{yV_t}{(V_t - V_{th})^2} = 1 \;\Rightarrow\; V_t = V_{th} + \frac{y}{2} + \frac{\sqrt{(4yV_{th} + y^2)}}{2}$$

We define

$$E_{norm} = xV_t^{\ 2} \quad \text{and} \quad T_{norm} = \frac{yV_e}{(V_e - V_{th})^2}$$

Then, the

$$\textbf{normalized speedup} = \frac{1}{T_{norm}} - 1$$

is the performance gain of CPU_{enh} when the supply voltage is scaled such that its energy consumption matches that of CPU_{orig}, and the

$$\textbf{normalized energy savings} = (1 - E_{norm})$$

is the energy saved by CPU_{enh} when the supply voltage is scaled such that its performance matches that of CPU_{orig}. We define an optimization as energy-efficient if the normalized speedup and the normalized energy savings are positive.

We chose this method because after optimizing a processor, programs hopefully run faster than on the original processor. However, as we are not interested in increasing both the performance and the energy consumption, we scale down the supply voltage such that the average execution time on the enhanced CPU is the same as the execution time of the original CPU. Since reducing the supply voltage has a quadratic effect on the dynamic energy consumption, the enhanced CPU (with voltage scaling) often has lower energy consumption than the original CPU. A similar approach is used to obtain the speedup of the enhanced CPU given the same energy budget as the original one. Note that we scale only the supply voltage and not the threshold voltage.

The benefit of our approach is that all variables (supply voltage, threshold voltage, the speedup and the change in energy consumption) in the relation are well understood and easily obtained. Furthermore, our approach provides a simple way to determine the true energy-efficiency of an optimization. Finally, since we use a cycle-accurate performance/energy simulation model, we capture the positive as well as the negative impacts of speculation activity on the performance and energy consumption of the whole chip.

4. METHODOLOGY

We obtain cycle-accurate performance data with the SimpleScalar/Alpha 3.0 tool set [5]. We integrated this simulator with the Wattch power model [4] to obtain the energy data. Wattch provides switching capacitance modeling for structures like ALUs, caches, arrays and buses in a processor. We incorporated value prediction into the simulator.

4.1 Simulation Framework

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 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 ALU units, four floating-point adders and two floating-point MULT/DIV units. There are two load/store 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. We enable 'no store alias' dependence prediction to predict aliases between load and store instructions [18].

We use Wattch's linear scaling to obtain energy results for 0.13µm technology, V_{dd} = 1.3V and a clock speed of 2.0 GHz. The V_{th} is 0.38V. The cache and predictor latencies are obtained with Cacti 3.2 [22]. We estimate static power as 25% of dynamic power.

4.2 The Predictors

We modeled a range of predictors, which are briefly described below. All predictors, including hybrid components, have 1024 entries. The predictors include a bimodal confidence estimator (CE) [14], [17], [18] with three-bit saturating counters with a threshold of five, a penalty of three and an award of one. A prediction is made only when the confidence is above the threshold value. The CE value is increased by the award when the value is predictable and decreased by the penalty when it is not. We use the same CE configuration 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 considered implementing one of the replay schemes used in the Alpha 21264 and the Pentium 4. In the squashing replay scheme used in the Alpha 21264, all dependent and independent instructions issued after the load scheduling miss are invalidated and replayed. The selective replay scheme used in the Pentium 4 reschedules only instructions dependent on misscheduled loads. However, complexity may significantly increase for precise dependence tracking. It was apparent to us that these replay schemes were not practical for load-value speculation as instructions dependent on the misses need to be searched across all in-flight instructions and it can take hundreds of cycles to discover the misspeculation. Hence, we use the re-fetch misprediction recovery 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 were overwritten with the true load value before they were first used.

LV: The last value predictor [8], [14] predicts that a load instruction will load the same value it did the previous time it executed.

ST2D: The stride 2-delta predictor [21] remembers the last value for each load but also maintains a stride, i.e., the difference between the last two loaded values. ST2D can predict sequences with zero (like LV) or non-zero strides.

DFCM3: The third-order differential finite context method predictor [11] computes a hash value [17], [18], [20] out of the difference between the last three load values to index the predictor's second-level table. This table stores strides between consecutive values that follow every seen sequence of three strides. After observing a sequence of load values, DFCM3 can predict any load that fetches the same sequence or a different sequence with the same strides.

wp-LV: To reduce the energy consumption, Loh [13] replaced the traditional predictor with multiple smaller tables and proposed a data-width predictor to choose among the tables. We implement Loh's width-partitioned last-value predictor (wp-LV). Since our base predictors have 1024 entries, we pick the configuration that Loh compared to the traditional 1024-entry LV. Specifically, we use a 4096-entry last width predictor, a 512-entry VPT_8, a 256-entry VPT_{16}, a 1024-entry VPT_{33} and a 256-entry VPT_{64}.

wp-ST2D, wp-DFCM3: We extended Loh's energy-reducing (width-partitioning) idea to ST2D (wp-ST2D) and DFCM3 (wp-DFCM3).

LSD-hybrid: Hybrid predictors have been shown to be more effective than unit predictors [18]. We combined LV, ST2D and DFCM3 into a hybrid and the component with the highest confidence makes the prediction.

shared-LSD-hybrid: Some predictors are in themselves extensions of smaller predictors. For instance, the ST2D has an LV component, and so does the DFCM3. To avoid replicating the common components, it has been proposed to share tables in the hybrid [7], [15], [16] to save space and power. We extend this idea to the LSD-hybrid by sharing the LV table between all components. Also, we maintain only one decoder for each predictor level, i.e., we increase the size of each line to encompass the information required by all components to further reduce the energy consumption.

wp-shared-LSD-hybrid: To minimize the energy of the shared-LSD-hybrid, we replace the LV component with Loh's wp-LV, which is described above.

4.3 Benchmarks

Table 1 describes the ten C programs (six integer and four floating-point) from the SPECcpu2000 benchmark suite [23] that we use for our measurements. They were compiled on a DEC Alpha 21264A processor using the DEC C compiler under the OSF/1 v5.1 operating system using the "-O3 -arch host" optimization flags. We employed the reference inputs provided with these programs. We utilize SimPoint [24] 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.

The following SPECcpu2000 programs are not used. Compiling *crafty, gap, parser* and *perlbmk* with our optimization flags makes the resulting binaries incompatible with the simulator, and simulating *vpr* is very slow since we would have to 'fast-forward' over 1600 billion instructions. Also, we do not use the C++ and Fortran programs because we do not have a good compiler for them.

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	1.532
art	6.5	162.5	32.5	1.407
bzip2	19.5	145.9	29.2	1.314
equake	131.5	235.1	47.0	0.330
gcc	4.0	228.2	45.6	1.111
gzip	3.0	121.8	24.4	1.284
mcf	23.0	209.7	41.9	0.488
mesa	67.5	129.5	25.9	1.742
twolf	247.0	142.6	28.5	1.209
vortex	106.5	127.2	25.4	1.843
geo. mean	57.8	148.8	29.8	1.115

5. EXPERIMENTAL RESULTS

As discussed in the previous sections, to evaluate the energy-performance benefit of a speculative design, it is necessary to take the entire chip into account. In this section, we compare the energy-efficiency of frequently used load-value predictors from the literature, including some energy-aware alternatives, using our metric described in Section 3. We start with the initial configuration described in Section 4 and evaluate the benefit of incorporating the different speculative optimizations to the processor. Note that no prior processor-wide runtime energy-performance analysis has been done for all of these load-value predictors and that some of the presented predictors are our own extensions of previously proposed predictors

5.1 Load-Value Predictors

In this sub-section we evaluate the LV, ST2D and DFCM3 predictors as well as Loh's energy-aware width-partitioned wp-LV. We also assess wp-ST2D and wp-DFCM3, our enhancement of the ST2D and DFCM3, respectively, with Loh's width-partitioning scheme. These six predictors are described in detail in Section 4.2.

We first examine the energy-efficiency of these six predictors for *gcc* and *bzip2*, and then study the overall behavior and trend across all the programs.

Figure 3 shows the IPCs of the processor with and without the above-mentioned predictors for *gcc*. *gcc* is of particular interest to us since it is one of the hardest programs to optimize in the SPECcpu2000 benchmark suite. We see that ST2D outperforms both LV and DFCM3. Context predictors such as DFCM3 have been shown to be the best performing load-value predictors. However, previous work has evaluated such predictors with large predictor tables, i.e., larger than 1024 entries. In our study, we found context predictors to be quite sensitive to the predictor size. Using realistically sized tables (1024 entries in our case) makes DFCM3 perform worse than its counterparts. This is because all loads share the second-level table in DFCM3, which increases aliasing and possibly negative interference.

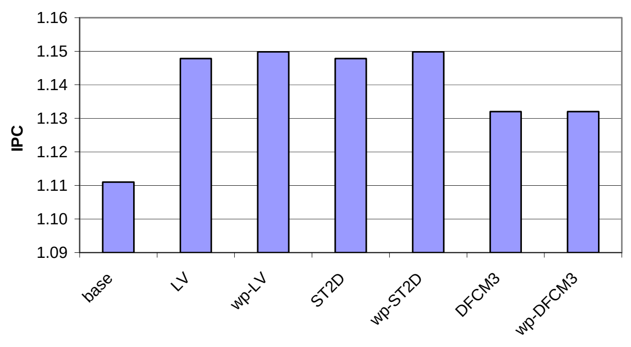

Figure 3. IPCs of a microprocessor with and without load-value predictors for *gcc*

Figure 4 shows the corresponding energy consumption (relative to the base CPU, which does not include any load-value predictor). Here, we observe that the CPU with DFCM3 consumes significantly less energy than those with LV and ST2D even though DFCM3 is larger in overall size. The decision on whether to make a prediction is determined by the confidence estimator associated with each predictor entry. As fewer and fewer correct predictions are made, the confidence of the DFCM3 falls to such low levels that relatively fewer predictions are made, which results in less speculation activity and explains its low performance and energy consumption.

We also find that although the CPUs with wp-LV and wp-ST2D consume less energy than LV and ST2D, respectively, they do not perform worse. On the other hand, extending the width-partitioning technique to DFCM3 increases the chip's energy consumption without a significant change in performance.

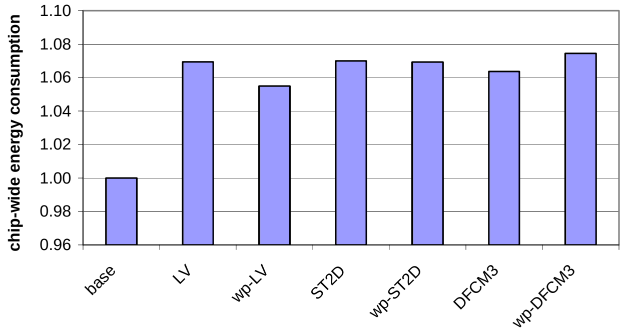

Figure 4. Processor-wide energy consumption with load-value predictors relative to the base CPU for *gcc*

To better see which of these designs are really energy-efficient, we derive the speedup of each speculative optimization given the same energy budget as the base processor, or alternatively the energy savings of each optimization given the same performance as the base processor. This process is described in Section 3. We define an optimization as energy-efficient if the resulting normalized speedup, or alternatively the normalized energy savings, is positive. In other words, a negative outcome indicates that it is not worth including the optimization.

Table 2. Normalized speedup and energy savings for microprocessors with load-value predictors for *gcc*

(Section 3 details our normalization process.)

	normalized speedup (%)	normalized energy savings (%)
LV	-2.9	-3.2
wp-LV	-1.5	-1.6
ST2D	-2.9	-3.3
wp-ST2D	-2.7	-3.0
DFCM3	-3.7	-4.2
wp-DFCM3	-4.7	-5.3

Surprisingly, we observe in Table 2 that for *gcc* none of the optimizations is energy-efficient despite the fact that no optimization slowed down the CPU (Figure 3). Note that the normalized speedup and normalized energy savings are derived from our metric described in Section 3 and should not be confused with Figure 3 or Figure 4.

We identified a couple of sources of inefficiency for *gcc*. Due to the finite table size of load-value predictors, different loads can map to the same predictor line, which often results in detrimental aliasing. Figure 5 shows the degree of aliasing for *gcc* in a 1024-entry predictor table. The minimum number of aliasing loads, i.e., static loads that access the same line, is 19 and the maximum is 55. 78 of the predictor entries have 37 aliasing loads.

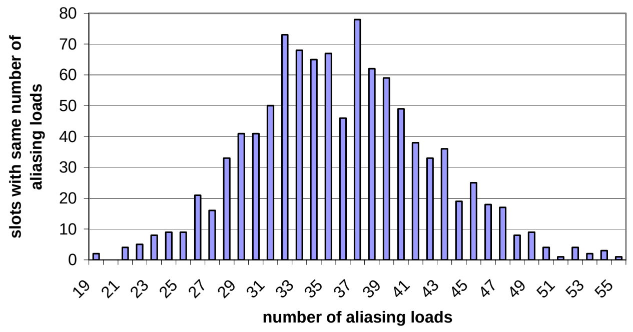

Figure 5. Aliasing static loads in 1024-entry table for *gcc*

Thus, *gcc* could benefit from an adaptive scheme that identifies the critical and predictable loads, and limits prediction resources to these, while keeping negative interferences to a minimum. However, this was not the major problem with *gcc*, as we explain next.

We find that, on average, almost 98% of the data cache accesses were hits and only 3.9% of the L1 misses had to access main memory. This means that no matter how accurate the value predictor is, the gains will be minimal for gcc. As can be observed in Figure 6, for the six predictor configurations, less than 1% of the loads are mispredicted. This is important because it is better not to predict than to do so incorrectly due to high cost of mispredictions for performance and especially for energy consumption. In fact, we find that for *gcc* employing a perfect predictor provides only 13% speedup.

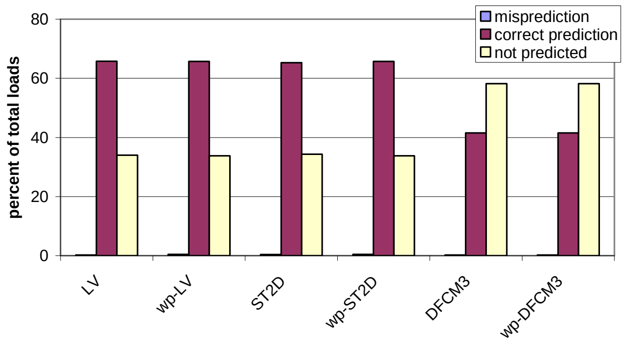

Figure 6. Percent of total loads that are not predicted, correctly predicted and mispredicted for *gcc*

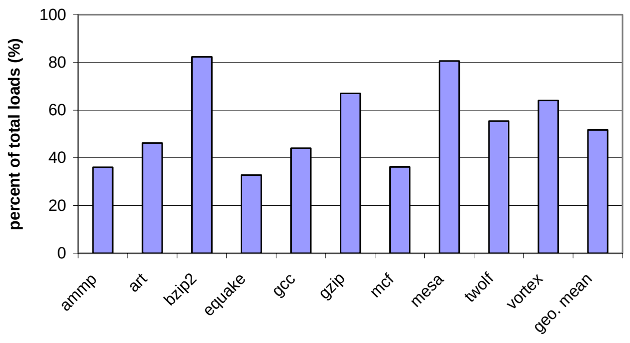

Figure 7. Percent of total loads with a load-to-use latency of fewer than 20 cycles

We also varied the number of pipeline stages in our simulations (Table 3). This experiment shows that the pipeline length has a minimal impact on *gcc*'s energy-efficiency. There appears to be a slight improvement in energy-efficiency in the predictors especially for DFCM3, except for wp-DFCM3, as the number of stages is reduced. We believe this trend can be attributed to the decrease in the cost of mispredictions in a processor with a shorter pipeline depth.

Table 3. Normalized speedup for varying pipeline depths for *gcc*

	10 stages	15 stages	20 stages
LV	-2.6	-2.8	-2.9
wp-LV	-1.3	-1.5	-1.5
ST2D	-2.7	-2.9	-2.9
wp-ST2D	-2.6	-2.7	-2.7
DFCM3	-1.7	-1.8	-3.7
wp-DFCM3	-4.6	-5.8	-2.7

A high prediction accuracy does not always translate into high performance. This is due to the fact that a correct prediction is useless if a dependent instruction does not consume the prediction before the memory provides the result. Thus one of the factors that influence the performance of a predictor is the load-to-use latency, i.e., the time between the issue of a load to when a dependent instruction is ready to execute.

Figure 7 shows the percent of total loads with a load-to-use latency of less than 20 cycles, i.e., the time it takes to access the L2 cache. This graph shows that on average 51.7% of the loads will potentially cause a stall if the L1 cache results in a miss. As is shown, *bzip2* has the highest potential for such stalls, and thus we expect it to benefit significantly from value prediction. We evaluate *bzip2*'s energy-efficiency next.

Figures 8 and 9 show the IPCs and the corresponding energy consumption, respectively, for *bzip2*. As expected, *bzip2* appears to benefit substantially from value prediction. Here, we observe that the width-partitioned optimizations do not enhance the performance but reduce the energy consumption, and significantly so for DFCM3.

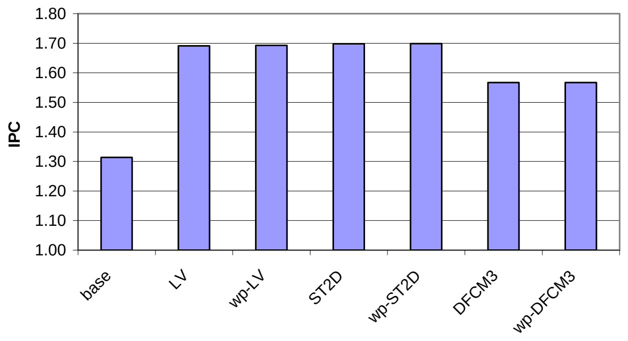

Figure 8. IPCs of a microprocessor with and without load-value prediction for *bzip2*

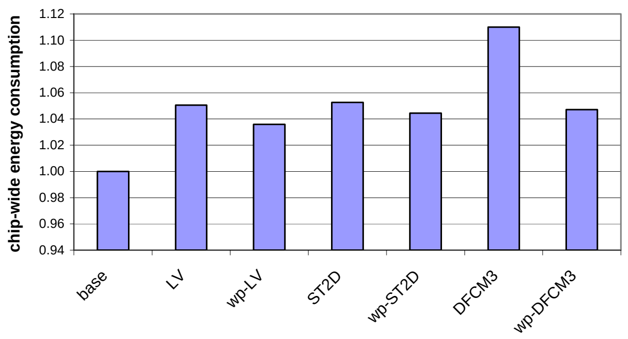

Figure 9. Processor-wide energy consumption with load-value prediction relative to the base CPU for *bzip2*

When we evaluate the performance and energy savings using our metric, we find that all six optimizations are energy-efficient for *bzip2*. Note that unlike with *gcc*, wp-DFCM3 is more energy-efficient than DFCM3. It is also interesting that the best design for *bzip2* is wp-LV (Table 4), which is one of the simpler predictors.

We have presented the detailed analyses for *gcc* and *bzip2* as examples of programs on opposite ends of energy-efficiency spectrum. We have presented a few possible sources of efficiency/inefficiency, namely prediction accuracy, cache miss rate, aliasing in predictor tables and the load-to-use latency. Given the wide disparity in program behavior, it may be worthwhile to adopt adaptive schemes to make value predictors more energy-efficient. An effective scheme would be one that makes predictions based on the energy-efficiency of a potential prediction. This is left for future work.

Table 4. Normalized speedup and energy savings for microprocessors with load-value predictors for *bzip2*

	normalized speedup (%)	normalized energy savings (%)
LV	23.0	19.3
wp-LV	24.7	20.5
ST2D	23.2	19.5
wp-ST2D	24.2	20.2
DFCM3	8.2	7.9
wp-DFCM3	14.3	13.1

Table 5 shows that DFCM3 and wp-DFCM3 are the only optimizations that are not energy-efficient across all the programs. The best predictor is wp-ST2D. Again, this is a fairly simple predictor compared to DFCM3 and wp-DFCM3. Research has shown that the most accurate load-value predictors tend to be large and complex. However, our findings demonstrate that designers can expect better energy-efficiency from seemingly simple predictors and future research in this field may need to focus on using simpler predictors more effectively.

Table 5. Normalized speedup and energy savings for microprocessors with load-value predictors across all programs

	normalized speedup (%)	normalized energy savings (%)
LV	0.9	0.9
wp-LV	0.9	1.0
ST2D	1.0	1.1
wp-ST2D	1.1	1.2
DFCM3	-1.9	-2.0
wp-DFCM3	-1.4	-1.6

5.2 Hybrid Load-Value Predictors

Different value predictors have been designed to exploit different predictable sequences. To enhance predictability, hybrid predictors combine dissociated predictors and employ a selection mechanism. The selector is responsible for choosing the most suitable component for predicting each load instruction. In the LSD-hybrid (described in Section 4.2), a confidence estimator is associated with each predictor and the component with the highest confidence makes the prediction. We compare this predictor to a space-saving version, i.e., one in which tables common to all components are shared [7], [15], [16]. We call this the shared-LSD-hybrid. We extend the width-partitioning technique to this predictor to obtain the wp-shared-LSD-hybrid.

Table 6. Normalized speedup for microprocessors with hybrid value predictors

	normalized speedup (%)		
program	LSD-hybrid	shared-LSD-hybrid	wp-shared-LSD-hybrid
ammp	-1.5	-0.9	-1.9
art	-2.7	-2.2	-1.7
bzip2	20.9	21.8	8.8
equake	11.2	11.9	0.7
gcc	-4.6	-4.0	-3.3
gzip	-0.3	0.6	-1.3
mcf	-8.4	-7.4	-3.6
mesa	-2.9	-2.2	-1.3
twolf	-10.0	-9.2	-7.6
vortex	-5.3	-4.7	-2.9
geo. mean	-0.7	0.0	-1.4

Tables 6 and 7 show the normalized speedup and normalized energy savings, respectively, for these three predictors and for the programs we simulate. Sharing tables makes the LSD hybrid more energy-efficient. This not unusual since this enhancement does not change the speculation activity of the processor but decreases the size and therefore the energy consumption of the predictor. With the exception of *ammp*, *bzip2*, *equake* and *gzip*, wp-shared-LSD-hybrid further improves the energy-efficiency of shared-LSD-hybrid.

What we found astonishing is the fact that except for *bzip2* and *equake*, none of the programs benefit from adding any of the hybrid predictors to the processor. This is primarily due to the fact that the added complexity increases the energy consumption of the processor at a higher rate than it increases performance. This observation further indicates that when energy consumption is taken into consideration, complex predictors do not provide a good energy-performance tradeoff.

Table 7. Normalized energy savings for microprocessors with hybrid value predictors

	normalized energy savings (%)		
program	LSD-hybrid	shared-LSD-hybrid	wp-shared-LSD-hybrid
ammp	-1.6	-1.0	-2.1
art	-3.0	-2.4	-1.9
bzip2	17.8	18.5	8.4
equake	10.8	11.3	0.8
gcc	-5.2	-4.5	-3.7
gzip	-0.3	0.6	-1.4
mcf	-9.9	-8.6	-4.0
mesa	-3.3	-2.4	-1.4
twolf	-11.9	-10.9	-9.0
vortex	-6.0	-5.3	-3.2
geo. mean	-0.8	0.0	-1.5

Note that we do not draw conclusions on the overall performance of these predictors and their energy-saving variants. Our observations are within the confines of our architectural configuration outlined in Section 4. It may well be possible to derive benefit from these same predictors for a different processor. The purpose of this study is to demonstrate how our metric can be used to measure the energy-efficiency of a speculative optimization and to answer the question 'Is it worth it to add the speculative optimization to the base processor?' A natural progression of our work is to evaluate what could be done to make the predictors more energy-efficient. This is left for future work.

6. CONCLUSIONS

Aggressive speculation is increasingly being used to exploit the growing transistor budgets. However, designers are being challenged to come up with novel ways to improve performance within current power and energy constraints. Several speculative hardware components have been optimized to reduce energy consumption. However, many of these optimizations have not taken into consideration the energy consumption introduced in other parts of the chip due to useless speculation activities. To better evaluate speculative architectures for their energy-efficiency, we have described a simple metric that is based on a cycle-accurate energy-performance analysis.

Unlike previously proposed metrics with hard-to-measure parameters, ours only relies on parameters that can easily be obtained (the supply and threshold voltages) and that can be measured through simulation (the speedup and increase in energy consumption of the optimization under investigation). We have illustrated the use of our metric on several value speculation optimizations and found that some designs that would have been considered energy-efficient are not. For instance, our metric revealed that applying an energy-saving technique such as width-partitioning to the DFCM3 predictor can hurt the energy efficiency of the processor as a whole.

7. ACKNOWLEDGMENTS

We thank the anonymous reviewers for useful feedback. This work was supported in part by a grant from Intel Corporation.

8. REFERENCES

[1] J. Aragon, J. Gonzalez, A. Gonzalez. Power-Aware Control Speculation through Selective Throttling. *Ninth International Symposium on High-Performance Computer Architecture*, 2003, pp. 103-112.

[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] D.M. Brooks, P. Bose, S.E. Schuster, H. Jacobson, P.N. Kudva, A. Buyuktosunoglu, J-D. Wellman, V. Zyuban, M. Gupta, P.W. Cook. Power-Aware Microarchitecture: Design and Modeling Challenges for Next-Generation Microprocessors. *IEEE Micro, v.20 n.6*, 2000, pp. 26-44.

[4] D. Brooks, V. Tiwari, M. Martonosi. Wattch: A Framework for High-Performance Microprocessors. *Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 171-1 2.

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

[6] T. Burd, R. Brodersen. Energy Efficient CMOS Microprocessor Design. *28th Annual Hawaii International Conference on System Sciences*, 1995, pp. 288-297.

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

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

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

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

[11] 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.

[12] M. Horowitz, T. Indermaur, R. Gonzalez. Low-power Digital Design. *IEEE Symposium on Low Power Electronics*, 1994, pp. 8-11.

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

[14] 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.

[15] 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.

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

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

[18] 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.

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

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

[21] Y. Sazeides, J. E. Smith. The Predictability of Data Values. *30th Annual IEEE/ACM International Symposium on Microarchitecture*, 1997, pp. 248-258.

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

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

[24] 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.

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

[26] 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.

[27] 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.