

# A Break-Even Formulation for Evaluating Branch Predictor Energy Efficiency

Michele Co, Dee A.B. Weikle, and Kevin Skadron  
Department of Computer Science  
University of Virginia

## Abstract

Recent work has demonstrated that a better branch predictor can increase the energy-efficiency of the system, even if the new predictor consumes more energy. Consequently, understanding the tradeoff between reduced mis-speculation, execution time, and increased power spent within a branch predictor is critical.

This paper proposes a simple, effective metric for evaluating the tradeoff between processor energy-efficiency and branch predictor energy. By calculating a *break-even* branch predictor energy budget for a given program and an energy-efficiency target, we are able to evaluate the energy-efficiency of several existing branch predictor designs and provide a simple way to think about energy-efficiency.

Furthermore, we develop a method for deriving a branch predictor energy budget without requiring a power model for the proposed branch predictor. We evaluate this approach by comparing the budgets we calculate with results from simulation. Average error in our estimates is less than 1.5% for all pipeline configurations with a confidence of  $\pm 0.02$  to  $\pm 0.06$  Joules (1.7%-2.1%) for the integer benchmarks and  $\pm 0.01$  to  $\pm 0.02$  Joules (1.1%-1.7%) for the floating point benchmarks for the evaluated configurations.

## 1 Introduction

The computer engineering community has focused a great deal of attention in recent years on energy-efficiency. This is important for almost all new chip designs today. Battery life is a concern for mobile devices, and even for systems running off wall power, utility costs are a concern—especially for large data centers.

Recent work by Parikh et al. [14] explored the energy-efficiency of branch predictors, and generally concluded that the better the predictor, the better the chip’s energy-efficiency, since better prediction reduces mis-speculated work and execution time, and hence reduces activity throughout the CPU. This means that spending *more* power in the predictor can still reduce power and improve energy-efficiency. Parikh et al. [14] and other subsequent

work [1, 3] have looked at ways to reduce power in the branch predictor.

We are not aware of any work that establishes a general framework for when a new branch predictor design is energy-efficient. Recent innovations in branch prediction that consume more power per prediction, coupled with continuing concerns about energy-efficiency, make a break-even analysis valuable. This is especially important for the very large predictors that have been proposed in recent years. For example, the piecewise-linear branch predictor proposed by Jimenez et al. [11] only confers significant benefits above 32 KB, consuming from 1% - 11% of total chip power. A number of other recent predictors could consume non-trivial amounts of power as well [9, 16, 19]. The contribution to total chip power of a few predictor organizations for four different pipeline configurations is categorized in Figure 1 (See Section 3 for details on processor configurations). These power estimates are based on Wattch [4] simulations.

This paper derives a simple metric for the break-even energy efficiency of a branch predictor. It captures the tradeoff between reduced mis-speculation and execution time, and increased power in the branch predictor. The metric is based on the  $ED^2$  metric [20] that has become popular because it is voltage independent. This technique gives the designer a way to predict how much energy is available to spend on a new branch predictor design relative to an existing processor design for a given performance of a particular program. This method for deriving break-even branch predictor energy budget can be used with or without a power model for the new design.

In this paper we:

- evaluate several existing branch predictor designs for energy-efficiency using our technique (1KB bimodal branch predictor as reference)
- observe that in general, for a single program, the power of the remainder of the processor excluding branch predictor is relatively constant regardless of branch predictor used
- demonstrate how to determine a break-even budget

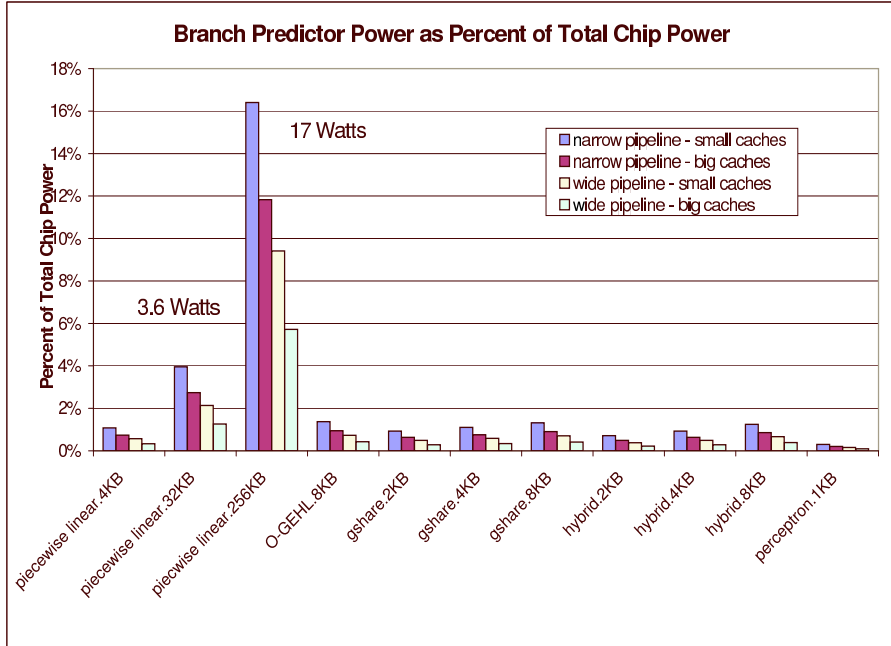


Figure 1: Branch predictor power as percent of total chip power in various processor configurations. (See Section 3.1 for configuration details.)

for a new design without a power model by estimating the new non-branch predictor energy and examine the validity of this estimation

- evaluate the independence of pipeline width, cache size, and leakage on the energy-efficiency trends between existing branch predictor designs
- show an approximate bound on branch predictor energy budget and on benefit that can be obtained through perfect branch prediction

This paper shows that pipeline width and cache size affect the energy budget of a predictor, but do not affect the relative performance of the branch predictors evaluated. It also shows that many branch predictors, including path perceptron and hybrid global/local predictors, use only a fraction of the energy they could to maintain energy-efficiency when compared to a 1 KB bimodal branch predictor reference. In some cases, the branch predictor uses only 8.4% of its energy budget.

We believe that our technique is general enough to apply to other processor structures as long as the assumption of constant energy per cycle for the remainder of the processor is adjusted to accurately describe the relationship of that structure’s behavior within the processor.

The rest of the paper is organized as follows: Section 2 presents an explanation of our energy-efficiency evaluation technique, Section 3 presents experimental methodology, Section 4 presents current experimental results, Sec-

tion 5 presents related work, and Section 6 presents conclusions and directions for future work.

## 2 Derivation of Break-Even Formula

This section describes the reasoning behind the technique for calculating the energy budget for a branch predictor design given a specific program, a reference processor configuration, and a reference branch predictor. The goal is two-fold. First, we want to determine if a new branch predictor is at least as successful as a previously designed predictor in terms of speed and power consumption. Second, we would like to determine an upper bound on the the benefits of a new branch predictor - specifically whether there is a sufficient benefit available to warrant further work toward an even better predictor.

### 2.1 Derivation of Energy Budget Formula

Assume for a program  $P$ , that the total energy of a processor ( $E_{total}$ ) can be described as the sum of the energy of the branch predictor ( $E_{bpred}$ ) plus the energy of the remainder of the processor ( $E_{remainder}$ ):

$$E_{total} = E_{bpred} + E_{remainder} \quad (1)$$

Given a reference processor configuration,  $Config_{ref}$ , and a new processor configuration,  $Config_{new}$ ,<sup>1</sup> we would like to have the same or better  $ED^2$  for  $Config_{new}$

<sup>1</sup> $Config_{new}$  differs from  $Config_{ref}$  by only the branch predictor design

when compared to  $Config_{ref}$ . We define this goal, the *break-even point*, based on the  $ED^2$  metric to be:

$$E_{total\_new} \times D_{total\_new}^2 = E_{total\_ref} \times D_{total\_ref}^2 \quad (2)$$

where  $E_{total\_new}$  is the total energy for running program P on  $Config_{new}$  and  $D_{total\_new}$  is the total new delay (execution time). While  $E_{total\_ref}$  is the total energy for running the same program P on  $Config_{ref}$ , and  $D_{total\_ref}$  is the total reference delay (execution time).

Expanding and rearranging (2) using (1), we obtain:

$$\frac{(E_{remainder\_new} + E_{bpred\_budget}) \times D_{total\_new}^2}{(E_{remainder\_ref} + E_{bpred\_ref}) \times D_{total\_ref}^2} = 1 \quad (3)$$

$E_{predicted\_remainder\_new}$  is the predicted energy that  $Config_{new}$  will consume not including the branch predictor energy.  $E_{bpred\_budget}$  is the amount of energy that the new branch predictor may consume and still maintain the same  $ED^2$  as  $Config_{ref}$ .  $E_{bpred\_ref}$  and  $E_{remainder\_ref}$  are the energy for the branch predictor of  $Config_{ref}$  and the energy for the remainder of  $Config_{ref}$  (not including branch predictor energy), respectively.  $D_{total\_ref}$  is the delay or total execution time (in seconds) of running program P on  $Config_{ref}$ .

We would like to know how much energy the new branch predictor is allowed to consume given a particular  $ED^2$ . Therefore, we solve for  $E_{bpred\_budget}$ . Rearranging the terms of (3) yields:

$$E_{bpred\_budget} = \frac{(E_{remainder\_ref} + E_{bpred\_ref}) \times D_{total\_ref}^2}{D_{total\_new}^2} - E_{remainder\_new} \quad (4)$$

All of the energy and delay values for the reference processor ( $E_{remainder\_ref}$ ,  $E_{bpred\_ref}$ , and  $D_{total\_ref}$ ) can be obtained from cycle-accurate simulation, or if available, from actual hardware measurement paired with the use of performance counters. The total execution time of  $Config_{new}$ ,  $D_{total\_new}$ , is gathered from cycle-accurate simulation. The energy for  $Config_{new}$  not including the branch predictor,  $E_{remainder\_new}$ , may be obtained from simulation results as described in Section 4.1 or may be estimated as described in Section 4.2. The branch predictor energy budget,  $E_{bpred\_budget}$ , is calculated from Equation (4).

The intent behind the derivation of this formula is to make explicit the tradeoffs designers make between performance and power consumption involved in branch prediction. This formula breaks the branch prediction budget down into individual components that can be filled in with the most accurate information the designer has on hand. In some cases, it may allow quick calculations based on

simple simulations or even estimated prediction rates to determine if a particular approach is worth pursuing further.

### 3 Experimental Methodology

All experiments in this work use SimpleScalar [5] and a modified Wattach [4] infrastructure with a power model based on the  $0.13\mu$  Alpha 21364 [18]. The microarchitecture model is summarized in Table 1.

To model leakage, when a port or unit is not in use, a fixed ratio of maximum power dissipation is charged: 10% in most of our experiments.

#### 3.1 Parameters

In our experiments, we evaluate variations of three factors in processor design:

- pipeline width: 4-wide issue (narrow) and 16-wide issue (wide)
- L1 and D1 cache sizes: 16KB (small) and 256KB (big)
- leakage ratio: 10% and 50%

We chose these parameters as a starting point for our study. In terms of pipeline width, we chose a pipeline that is 4-wide to consider today’s processors and a 16-wide to look forward to more aggressive processor issue widths. For caches, we chose a very small cache and a cache more representative of what might be seen in current processors. In addition, we modeled leakage ratios of 10% to reflect current technology and 50% leakage ratio to look forward to the future process technology trends.

#### 3.2 Branch Predictors Evaluated

The specific details of the branch predictor designs evaluated are listed in Table 2. We evaluate variations on path perceptron [19], gshare [13], hybrid [6, 12], O-GEHL [15], and piecewise linear [10] predictors. The same size BTB (2k-entry, 2-way set associative) and RAS (32-entry) are used for all branch predictors. We use the bimodal predictor as the reference model for all of the calculations described in Section 2.

#### 3.3 Benchmarks

We evaluate our results using the integer and floating point benchmarks from the SPEC CPU2000 suite. The benchmarks are compiled and statically linked for the Alpha instruction set using the Compaq Alpha compiler with SPEC *peak* settings and include all linked libraries but no operating-system or multiprogrammed behavior.

Simulations are fast-forwarded according to Sherwood and Calder’s SimPoint numbers [17], then run in full-detail cycle-accurate mode (without statistics-gathering)

Processor Core	
Active List	128 entries
Physical registers	80
LSQ	128 entries
Issue width	<b>wide:</b> 16 instructions per cycle <b>narrow:</b> 4 instructions per cycle
Functional Units	<b>wide:</b> 16 IntALU,4 IntMult/Div, 8 FPALU,4 FPMult/Div, 2 mem ports <b>narrow:</b> 4 IntALU,1 IntMult/Div, 2 FPALU,1 FPMult/Div, 2 mem ports
Memory Hierarchy	
IL1 & DL1	<b>small:</b> 16KB, 64-byte line, 2-way set associative, LRU <b>big:</b> 256KB, 64-byte line, 2-way set associative, LRU
L2	Unified Cache, 4 MB, 8-way LRU, 128B blocks, 12-cycle latency, writeback
Memory TLB Size	225 cycles (75ns) 128-entry, fully assoc., 30-cycle miss penalty
Branch Predictor	
BTB	2 K-entry, 2-way
RAS	32-entry

Table 1: Simulated processor microarchitecture.

Name	Area	Description
bimodal	1KB	4k-entry, 2-bit counters
path perceptron	10KB	64-bit global history register 10 1KB tables 10 bits of history per table
gshare	2KB	13-bit history, 8k-entry
gshare	4KB	14-bit history, 16k-entry
gshare	8KB	15-bit history, 32k-entry
hybrid	2KB	2k-entry meta, 2k-entry bimodal, 4k-entry 2lev
hybrid	4KB	4k-entry meta, 4k-entry bimodal, 8k-entry 2lev
hybrid	8KB	8k-entry meta, 8k-entry bimodal, 16k-entry 2lev
O-GEHL	8KB	6, 2k-entry, 4-bit tables 1k-entry, 5-bit counter table 2k-entry, 5-bit counter table 1k-entry, 1-bit tag table 48-bit global history register 48-entry, 8-bit global address register
piecewise linear	8KB	weight table: 8590-entry 7-bit counters bias table: 599-entry 7-bit counters global path history: 48 8-bit addresses 48-bit global history register local history table: 55 16-bit shift registers
piecewise linear	32KB	weight table: 34360-entry 7-bit counters bias table: 3496-entry 7-bit counters global path history: 26 8-bit addresses 26-bit global history register local history table: 220 16-bit shift registers

Table 2: Branch predictors evaluated

for 300 million instructions to train the processor structures—including the L2 cache—and the branch predictor before statistics gathering is started. This interval was found to be sufficient to yield representative results [8].

## 4 Results

The following sections describe the results of comparing the break-even branch predictor budgets of several current branch predictor designs with their actual energy consumption. Section 4.1 shows how budgets are calculated using cycle-accurate simulated values for  $E_{remainder\_new}$ . Section 4.2 then explains how to estimate  $E_{remainder\_new}$  and validates this estimate by comparing it to the simulated results. Section 4.3 describes an upper bound for branch predictor budgets based on perfect branch prediction. Section 4.4 then shows how changing the leakage ratio to 50% affects the results.

### 4.1 Branch Predictor Budgets from Simulation

$E_{bpred\_budget}$  can be calculated using  $E_{remainder\_new}$  gathered from cycle-accurate simulation. Figure 2 shows the calculated  $E_{bpred\_budget}$  values for each benchmark for each processor configuration. The actual values of the results differ based on processor configuration, but the overall relationship between the branch predictors evaluated is the same. Both the path perceptron and the O-GEHL predictor have a higher branch predictor energy budget on average than the other branch predictors evaluated. This indicates that both the path perceptron’s and O-GEHL’s prediction accuracy are much higher than the other predictors evaluated and the increased prediction accuracy boosts the energy budget available to be spent on their implementation and still satisfy the *break-even point*.

For all the graphs, the *gshare.2KB* predictor often displays negative branch predictor energy budget values. A negative budget on a particular benchmark indicates that the branch predictor’s performance does not improve the overall processor performance sufficiently to recoup the energy expended in the predictor itself. In other words, even if the branch predictor were to consume zero energy, the branch predictor would not be able to fulfill the  $ED^2$  *break-even point* and the reference predictor would be a better choice. On these specific benchmarks this is because the 2KB area causes destructive aliasing which cripples the *gshare* predictor’s accuracy.

Figure 3 shows the percent of the calculated branch predictor energy budget consumed by the evaluated branch predictors. Any predictor that uses less than 100% on a particular benchmark is worth considering for that benchmark. Branch predictors which use less than 100% of their energy budget are providing additional energy-efficiency beyond the break-even point. On average the trends of the branch predictor energy consumption amongst the benchmarks are the same. Several bench-

marks exhibit consumption in excess of the budget for one or more predictors, some in excess of 300%. For all these benchmarks, the particular branch predictor was unable to come close to the *break-even*  $ED^2$  point. More specifically, branch predictors which exhibit greater than 300% energy budget consumption can not fulfill the energy break-even point, and often have negative branch predictor budgets, which we represent in our graphs as infinitely large numbers above the 300% demarcation line.

The trend evident in the per benchmark graph of Figure 3 is still maintained when the average of the benchmarks is calculated as in Figure 4 which summarizes the overall results for both integer and floating point benchmarks. Once the average is plotted per predictor, though, it is clear that it is the *gshare.2KB* predictor that is not a benefit to power consumption when all benchmarks are considered. Note that on the floating point benchmarks few examples exist of predictors exceeding their budget. This is because the floating point benchmarks are very predictable, enabling even simple predictors to obtain a maximum benefit.

At this point one may ask why not just simulate the entire configuration and determine which predictors deliver better performance with less energy consumption. In the next section, we show an example of estimating one of the simulated components of the equation. We believe that this formula enables us to understand better and more quickly not only the benefits of a particular predictor, but the potential benefits of additional improvements as well.

### 4.2 Branch Predictor Energy Budgets without an Existing Power Model

It is desirable to estimate  $E_{predicted\_remainder\_new}$  when there is no pre-existing power model readily available for the candidate branch predictor.

If and only if it can be assumed that the average energy per cycle used by the non-predictor portion of the reference design is equal to the average energy per cycle of the non-predictor portion of the new design while running the same program, then<sup>2</sup>:

$$\frac{E_{remainder\_ref}}{D_{total\_ref}} = \frac{E_{predicted\_remainder\_new}}{D_{total\_new}} \quad (5)$$

Solving for  $E_{predicted\_remainder\_new}$ :

$$E_{predicted\_remainder\_new} = \frac{E_{remainder\_ref} \times D_{total\_new}}{D_{total\_ref}} \quad (6)$$

Equation (6) will then allow a designer to solve for  $E_{bpred\_budget}$ , which is the value of interest. Substitute

<sup>2</sup>The accuracy of this assumption is explored in Section 4.2

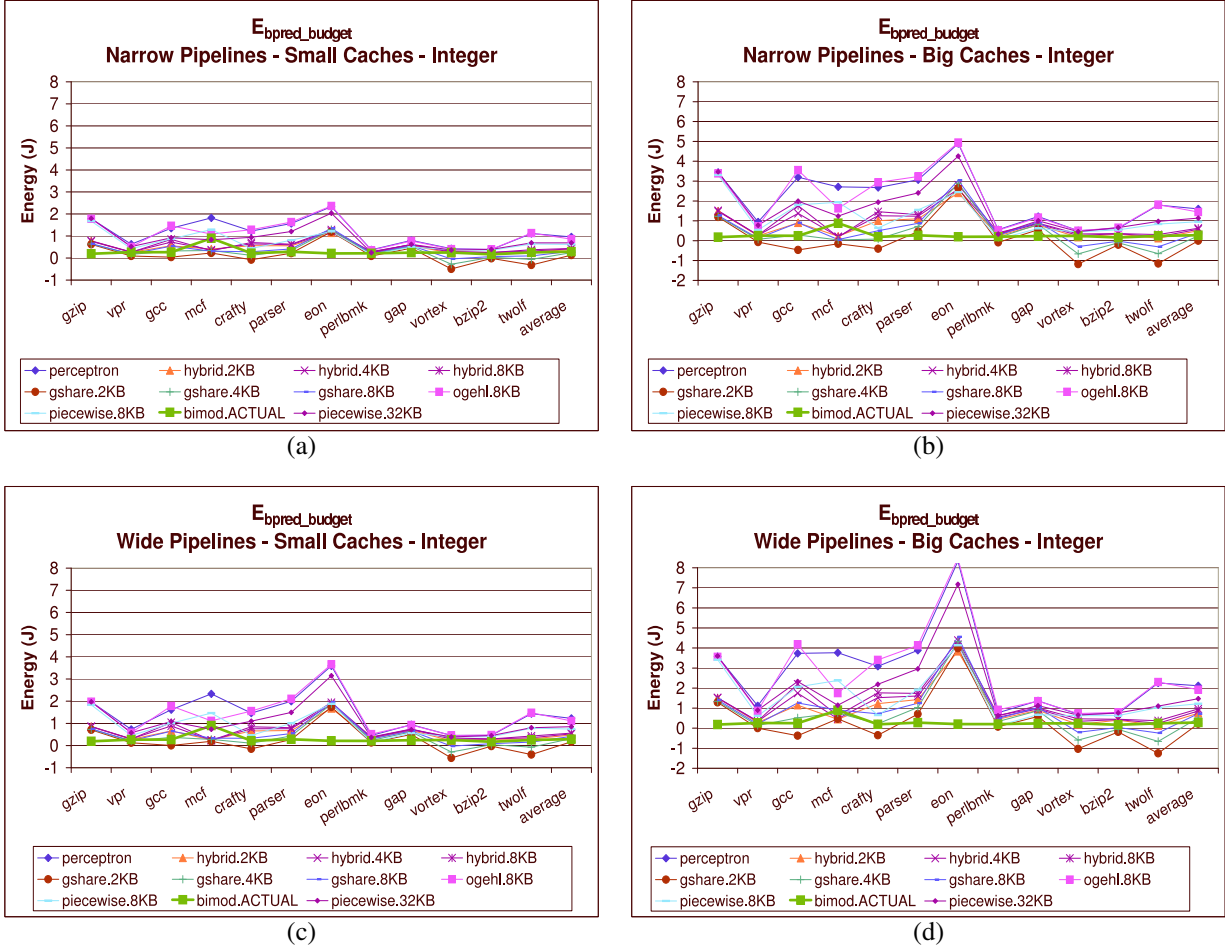


Figure 2: Calculated branch predictor energy budget ( $E_{bpred\_budget}$ ) for processor configurations. Note that in some cases that the energy budget is negative. This indicates that even at zero branch predictor energy consumption, the new predictor's performance is insufficient to recoup the energy (in excess of the reference predictor's energy) expended in the predictor itself.

(6) into (4):

$$\begin{aligned}
 E_{bpred\_budget} &= \frac{\text{Total energy budget for } Config_{new}}{\frac{(E_{remainder\_ref} + E_{bpred\_ref}) \times D_{total\_ref}^2}{D_{total\_new}^2}} \\
 &- \frac{E_{remainder\_ref} \times D_{total\_new}}{D_{total\_ref}} \\
 &E_{remainder\_new}
 \end{aligned} \tag{7}$$

Note that Equation (7) is simply an expanded version of Equation (1).

Figure 5 (a) shows the estimated  $E_{predicted\_remainder\_new}$  for the SPECint2000 benchmarks run on a processor configuration with a narrow pipeline and big caches. Figure 6 shows the raw  $E_{predicted\_remainder\_OGEHL.8KB}$  and  $E_{actual\_remainder\_OGEHL.8KB}$  results overlaid for the 8 KB O-GEHL predictor for both integer and floating point

benchmarks. Notice that there is little difference between the remainder energy predicted when compared to the remainder energy gathered from simulation data. We display the 8 KB O-GEHL predictor results because our technique exhibits the largest absolute difference on it of all the branch predictors evaluated. Raw results for the other processor configurations included in the study were similar in trend, so are omitted due to space constraints.

Figure 7 shows the absolute percent difference between  $E_{predicted\_remainder\_new}$  and  $E_{actual\_remainder\_new}$  for the four main processor configurations evaluated. The range of error is at most 11% and on average less than 1.5% for all benchmarks and for all processor configurations. Figure 8 summarizes the average percent deviation of  $E_{predicted\_remainder\_new}$  from  $E_{actual\_remainder\_new}$  for each of the configurations and clearly shows that the standard deviation from  $E_{actual\_remainder\_new}$  for both integer and floating point benchmarks is very small. The

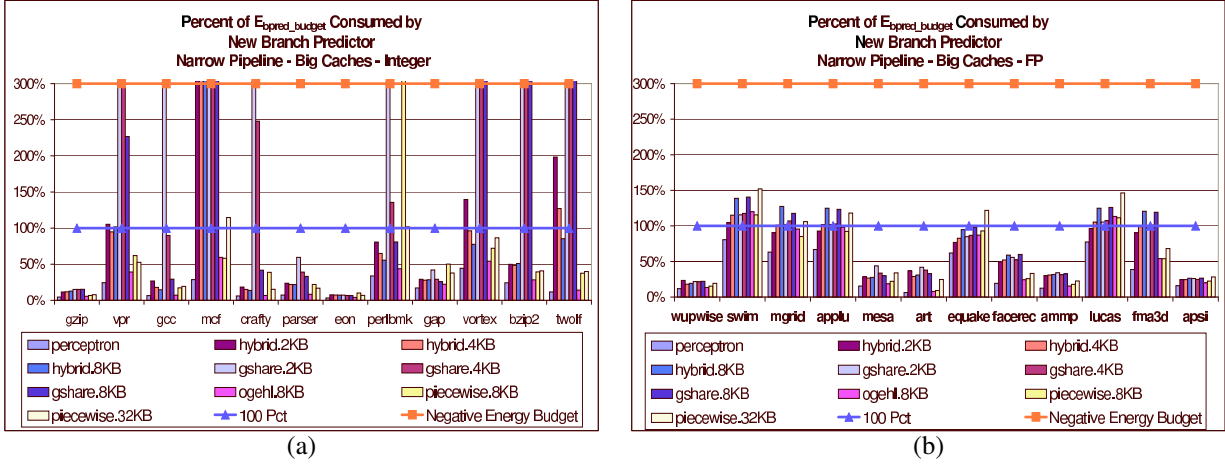


Figure 3: Percent of branch predictor energy budget ( $E_{bpred\_budget}$ ) actually consumed by branch predictor as measured by simulation for narrow pipeline with big caches processor configuration: (a) integer benchmarks, (b) floating point benchmarks. Note that lower values indicate branch predictor energy consumption lower than (better)  $E_{bpred\_budget}$  and higher values indicate energy consumption greater than  $E_{bpred\_budget}$  (worse). Note that branch predictor energy budget percentages below 100% line perform better than the energy break-even point. Configurations with negative branch predictor energy budgets are manually set to exceed the 300% threshold.

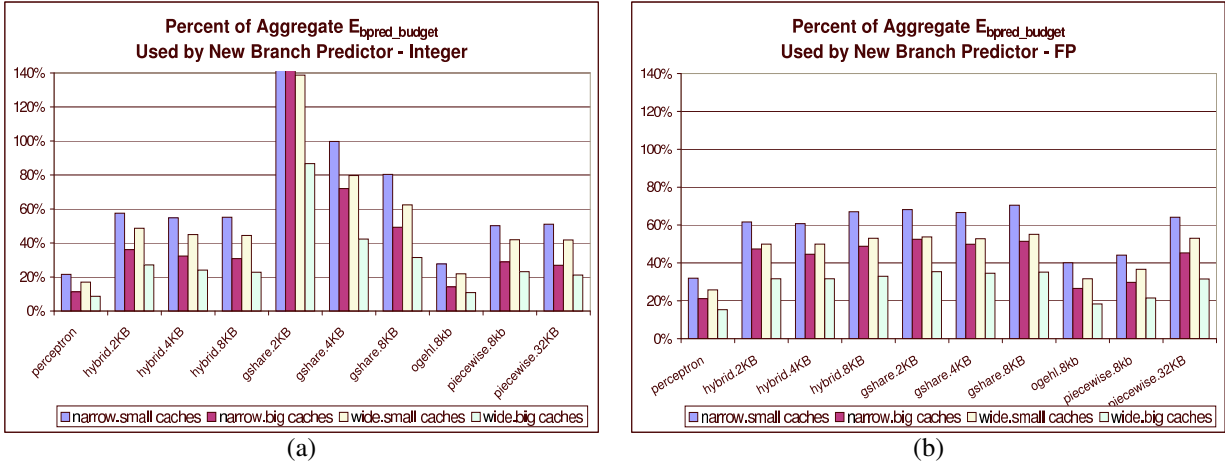


Figure 4: Percent of aggregate  $E_{bpred\_budget}$  consumed by new branch predictor. This is the ratio between the total  $E_{bpred\_actual}$  across the entire workload and the total  $E_{bpred\_budget}$  across the entire workload

error bars on the graph show  $\pm 1$  standard deviation.

The closeness of our estimated  $E_{predicted\_remainder\_new}$  to  $E_{actual\_remainder\_new}$  shows that the assumption made in Equation (5) is relatively accurate. Since the expectation is that the same amount of work is being performed in each case (executing a particular benchmark or program), one might think that the processor configuration with the shorter execution time would use a greater energy per cycle at the break-even point. The configuration with the slower execution time is actually carrying out additional work to recover from mis-speculation, rather than

performing useful work. Since this estimate of the  $E_{remainder\_new}$  is fairly accurate, we believe it is possible to determine a branch predictor energy budget without actually having a power model for the branch predictor. Section 4.4 shows that with increasing leakage ratio, our estimate only becomes more accurate.

### 4.3 Estimating an Upper Bound for Branch Predictor Energy Budgets

It would be interesting to evaluate the limits of branch predictor energy budgets. The best possible branch predictor energy budget comes from ideal or perfect branch prediction. We performed an experiment using an ap-

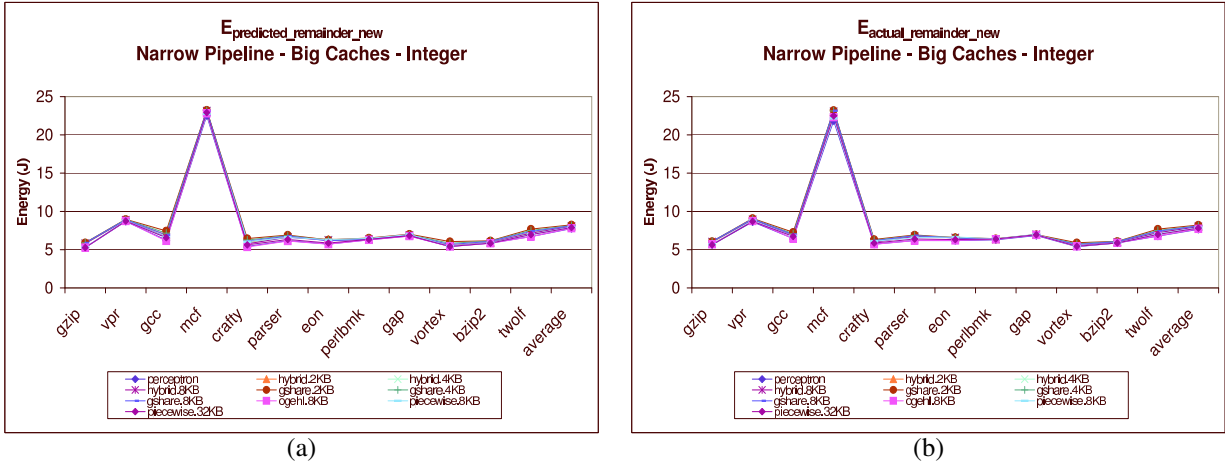


Figure 5: (a) predicted energy of remainder of  $Config_{new}$ ,  $E_{predicted\_remainder\_new}$  and (b) actual/simulated energy of remainder  $E_{actual\_remainder\_new}$ .

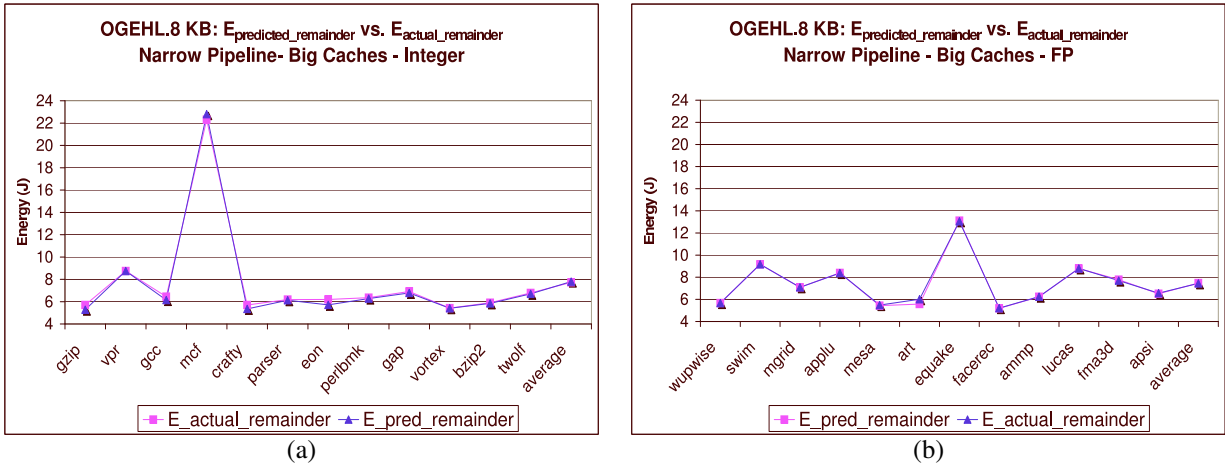


Figure 6: OGEHL.8KB  $E_{actual\_remainder}$  vs.  $E_{pred\_remainder}$  (a) integer and (b) floating point benchmarks.

proximated perfect branch prediction technique in which branch mispredictions are detected in the decode stage and then corrected. Misfetches are not avoidable with this technique. However, this method gives us an idea of where the upper bound of branch predictor energy budgets lies.

Figure 11 shows the results of our experiment for both integer and floating point benchmarks. Since the integer benchmarks are less predictable, the  $E_{bpred\_budget}$  with our pseudo-perfect technique is clearly greater (better) than that of the other predictors in our evaluation, with O-GEHL and piecewise linear branch predictors coming very close on *eon*. For the floating point benchmarks, one might be puzzled that the perfect prediction  $E_{bpred\_budget}$  is not the highest budget number on average, but rather O-GEHL and piecewise linear branch predictors have similar or higher budgets, especially for *art*. Both *eon* and *art*

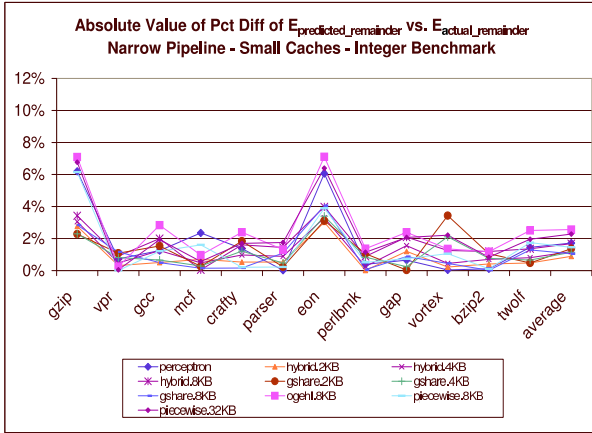
are benchmarks in which our pseudo-perfect prediction technique fails due to high numbers of instruction misfetches. For example, on *eon* our pseudo-perfect branch prediction technique misfetches 20 instructions per 1k instructions, and 8.6 instructions per 1k instructions for *art*. This is much higher than both O-GEHL's and piecewise linear branch predictors' misfetch rates which are effectively zero (0.05-0.08 misfetches per 1k instructions).

The graph shows that there is still some benefit to be gained by improving branch prediction accuracy and gives us an approximate idea of where this upper bound lies.

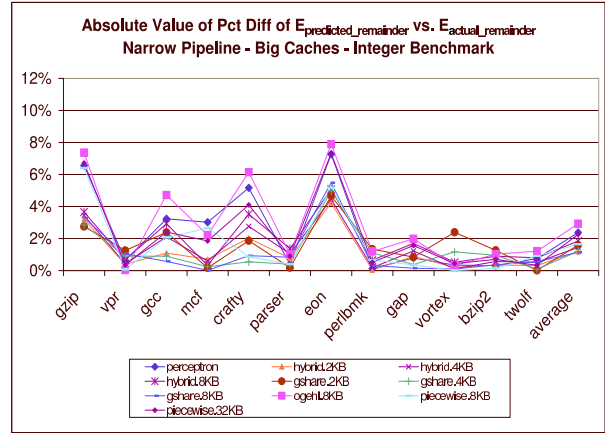
#### 4.4 Leakage Effects

All of the previous experiments were run using a value of 10% for leakage. To see the effects of leakage values more in line with future processor technology, we ran selected experiments using a 50% leakage ratio.

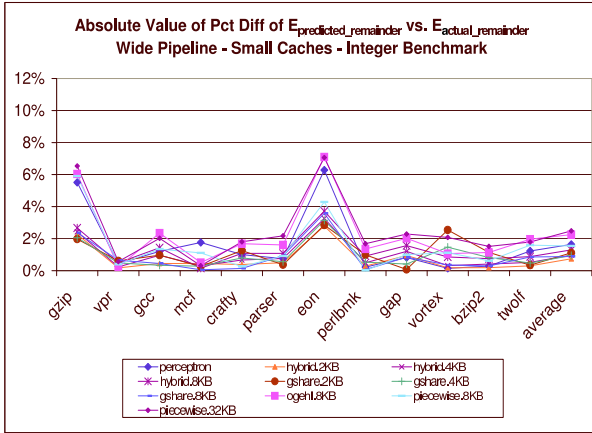
Figure 9 shows that the general trend for the branch



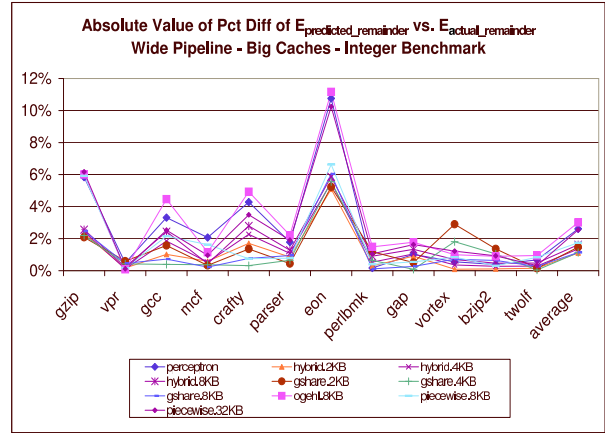
(a)



(b)

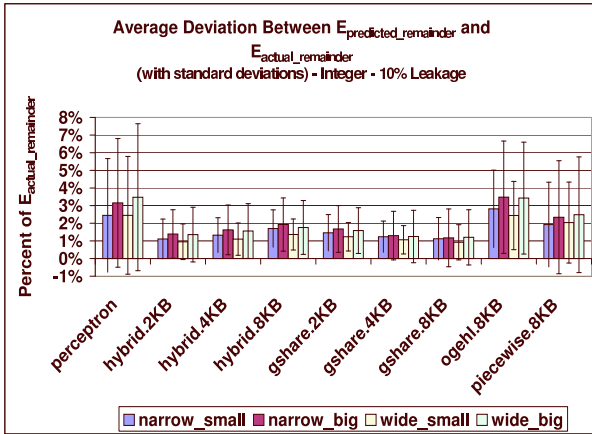


(c)

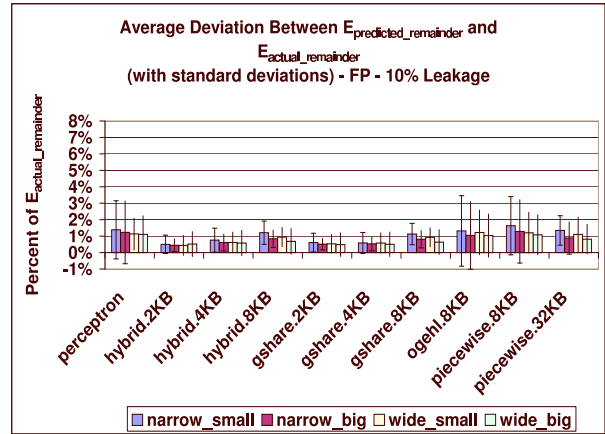


(d)

Figure 7: Absolute Value of Percent Difference between  $E_{predicted\_remainder\_new}$  and  $E_{actual\_remainder\_new}$  for (a) narrow pipeline with small caches, (b) narrow pipeline with big caches, (c) wide pipeline with small caches, (d) wide pipeline with big caches.



(a)



(b)

Figure 8: Average deviation of  $E_{predicted\_remainder\_new}$  from  $E_{actual\_remainder\_new}$  for 10% leakage ratio: (a) integer and (b) floating point benchmarks. Error bars illustrate  $\pm 1$  standard deviation.

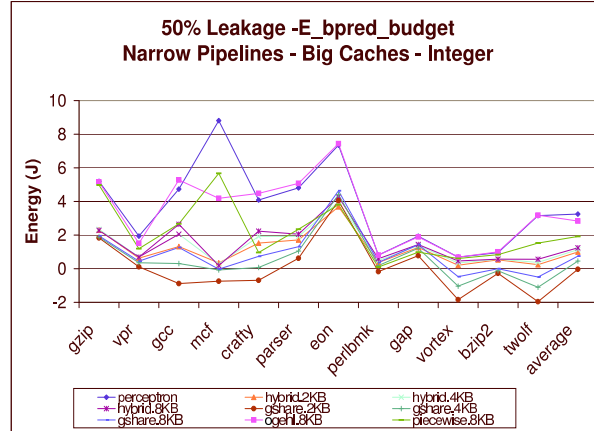


Figure 9: 50% Leakage Ratio:  $E_{bpred\_budget\_new}$  for narrow pipelines, big caches for integer benchmarks. Note that although the magnitude of the graph is amplified, the general shape of the graph is very similar to the shape of the graph in Figure 2(b).

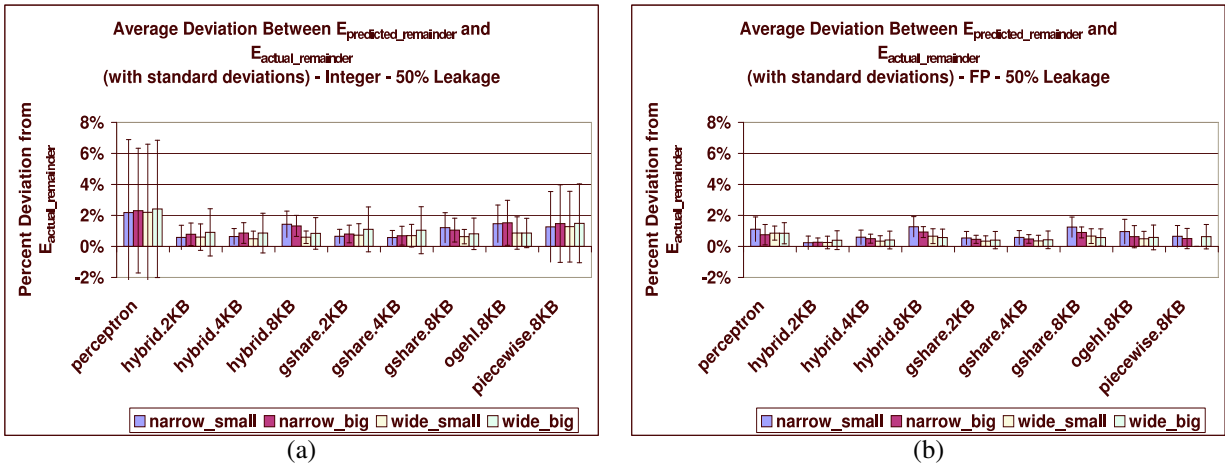


Figure 10: 50% Leakage Ratio: Average  $E_{predicted\_remainder\_new}$  and standard deviation from  $E_{actual\_remainder\_new}$  for (a) integer and (b) floating point benchmarks. Note that although the predicted energy goes up compared to 10% leakage results, the standard deviation does not increase proportionally.

predictor budget is the same between predictors and amongst benchmarks. Results for other configurations and for the floating point benchmarks were very similar and are not included due to space constraints.

We also see from the results that our estimation of  $E_{pred\_remainder\_new}$  is still accurate as the leakage ratio increases. In fact, the standard deviation does not increase proportionally with the leakage ratio as shown in Figure 10.<sup>3</sup> This is due to the fact that the power difference between activity and inactivity in the structures is less due to increased leakage. This demonstrates that our technique is still useful as leakage begins to dominate.

<sup>3</sup>Piecewise linear 32 KB results not included.

## 5 Related Work

Parikh et al. [14] explored the energy-efficiency of branch predictors. They concluded that better prediction accuracy led to better processor energy-efficiency. They also made the insight that spending additional power in the predictor can still reduce overall power and improve processor energy-efficiency. The work in this paper builds on this insight and develops a metric for determining a branch predictor energy budget for a new branch predictor design.

Aragon et al. [1] analyze the reasons for performance loss due to conditional branch mispredictions and develop a simple technique for fetching, decoding, and renaming along the alternate path for low confidence branches to reduce misprediction penalty and thus reduce overall energy

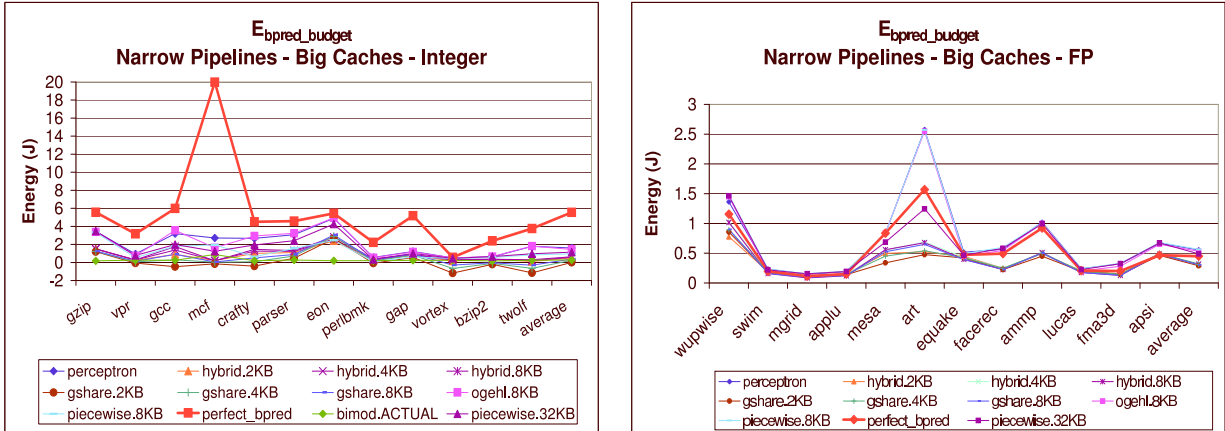


Figure 11: ( $E_{\text{bpred\_budget}}$ ) for perfect branch prediction for narrow pipeline, big caches configuration.

consumption. Baniasadi and Moshovos [2] examined reducing branch predictor power dissipation by selectively turning off tables in a combined branch predictor design. In additional work, Baniasadi and Moshovos exploit the insight that branches in steady state do not need to update the branch predictor to reduce the number of branch predictor accesses and therefore reduce branch predictor energy consumption. [3] Chaver et al. [7] proposed a method for using profiling to characterize branch prediction demand. They use this information to selectively disable portions of a hybrid branch predictor and resize the branch target buffer to reduce branch predictor energy consumption. Our work does not develop a specific technique for reducing energy consumption, but rather demonstrates a method for reasoning about the energy-efficiency of branch predictor designs.

## 6 Conclusions

This paper describes a general, systematic method for calculating the *break-even energy budget* for a branch predictor design. The method requires a cycle-accurate performance and power/energy model for a reference processor and a cycle-accurate simulation of the branch predictor design under consideration. An accurate estimate of the energy budget can then be made without a power/energy model for the candidate branch predictor. The techniques presented in this paper allow the comparison of the energy-efficiency of different branch predictor designs without having to equalize branch predictor area or branch prediction accuracy rates. It further gives a branch predictor designer a technique with which to easily determine the energy available to achieve an energy-efficient branch predictor, given the performance for a set of programs and an upper bound on the energy available for an ideal predictor.

This paper also evaluates the branch predictor energy budgets for several existing branch predictor designs on

the SPECcpu2000 benchmarks and evaluates the energy-efficiency of these designs. We also put forth the notion that average energy per cycle consumption of the remainder of the pipeline varies little between different branch predictor designs. We further find that the branch predictor performance and energy trends are fairly independent relative to pipeline width and cache size, thus reducing the design space exploration needed during future branch predictor research. Finally, these results were determined to hold even when the leakage was increased to 50%.

Overall, our results suggest that even the very aggressive branch predictors recently proposed in ISCA 2005 do not yet violate energy efficiency bounds, at least not when aggregating across SPEC overall as a workload. This indicates that research on further improvements in branch prediction is warranted.

## 7 Future Work

There are many directions in which this work may be extended. The study could easily be expanded to include all configurations of larger, more aggressive, and more complex branch predictor designs as well as no-predictor configurations. branch predictors, which are improved versions of the designs presented in [10, 15].

An upper bound for branch predictor energy budget can be demonstrated and used to estimate the bound on benefit that can be obtained from achieving perfect branch prediction.

This work also has the potential to lead to a technique to estimate a branch predictor energy budget without requiring either cycle-accurate simulator or power model for the future branch predictor design. We envision that in the future, given a functional simulator with which to derive branch prediction accuracy on a particular program, designers will be able to derive the branch predictor break-even energy budget much earlier in the design process, allowing them to narrow the design space search

much more quickly.

This technique could also be combined with program phase detection techniques and adaptive hardware techniques to develop a method to improve energy-efficiency by adapting the branch predictor hardware based on program characteristics and break-even energy information.

Another interesting factor to explore is the impact of training time due to operating system context switches on branch predictor energy budget.

We believe that the method described in this paper can be refined to allow simpler estimates for the power/performance tradeoffs associated with branch predictor design. In addition, the technique described in this paper could be applied as presented to determine the energy budgets of other processor structures such as caches, register files, buffers, etc.

This paper shows how these results are independent of cache size, pipeline width, and leakage ratio. Future work could determine the effect of register file size and buffers as well.

## Acknowledgments

This work is supported in part by the National Science Foundation under grant nos. NSF CAREER award CCR-0133634, and CNS-0340813, EIA-0224434, and a grant from Intel MRL. We would also like to thank Jason D. Hiser for his helpful input.

## References

- [1] J. L. Aragon, J. Gonzalez, A. Gonzalez, and J. E. Smith. Dual path instruction processing. In *Proceedings of the 2002 International Conference on Supercomputing*, pages 220–229, New York, NY, USA, 2002. ACM Press.
- [2] A. Baniasadi and A. Moshovos. Branch predictor prediction: A power-aware branch predictor for high-performance processors. In *Proceedings of the 2002 International Conference on Computer Design*, pages 458–461, 2002.
- [3] A. Baniasadi and A. Moshovos. Sepas: a highly accurate energy-efficient branch predictor. In *Proceedings of the 2004 International Symposium on Low Power Electronics and Design*, pages 38–43, New York, NY, USA, 2004. ACM Press.
- [4] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 83–94, June 2000.
- [5] D. C. Burger and T. M. Austin. The SimpleScalar tool set, version 2.0. *Computer Architecture News*, 25(3):13–25, June 1997.
- [6] P.-Y. Chang, E. Hao, T.-Y. Yeh, and Y. Patt. Branch classification: a new mechanism for improving branch predictor performance. In *MICRO27*, pages 22–31, New York, NY, USA, 1994. ACM Press.
- [7] D. Chaver, L. Piñuel, M. Prieto, F. Tirado, and M. C. Huang. Branch prediction on demand: an energy-efficient solution. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pages 390–395, New York, NY, USA, 2003. ACM Press.
- [8] J. W. Haskins, Jr. and K. Skadron. Memory reference reuse latency: Accelerated sampled microarchitecture simulation. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software*, pages 195–203, Mar. 2003.
- [9] Q. Jacobson, E. Rotenberg, and J. E. Smith. Path-based next trace prediction. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 14–23, 1997.
- [10] D. Jimenez. Idealized piecewise linear branch prediction. In *Proceedings of the First Workshop Championship Branch Prediction in conjunction with MICRO-37*, December 2004.
- [11] D. A. Jiménez. Piecewise linear branch prediction. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, page TBD. IEEE Computer Society, 2005.
- [12] S. McFarling. Combining Branch Predictors. Technical Report TN-36, June 1993.
- [13] S. McFarling and J. Hennessey. Reducing the cost of branches. In *Proceedings of the 13th Annual International Symposium on Computer Architecture*, pages 396–403, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.
- [14] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In *Proceedings of the Eighth International Symposium on High-Performance Computer Architecture*, pages 233–44, Feb. 2002.
- [15] A. Seznec. The O-GEHL branch predictor. In *Proceedings of the First Workshop Championship Branch Prediction in conjunction with MICRO-37*, December 2004.
- [16] A. Seznec. Analysis of the O-Geometric History Length branch predictor. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, page TBD. IEEE Computer Society, 2005.
- [17] T. Sherwood, E. Perelman, and B. Calder. Basic block distribution analysis to find periodic behavior and simulation points in applications. In *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, Sept. 2001.
- [18] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. Temperature-aware microarchitecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 2–13, June 2003.
- [19] D. Tarjan, K. Skadron, and M. Stan. An ahead pipelined alloyed perceptron with single cycle access time. In *Proceedings of the 5th Workshop on Complexity-Effective Design*, 2004.
- [20] V. Zyuban and P. Strenski. Unified methodology for resolving power-performance tradeoffs at the microarchitectural and circuit levels. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design*, pages 166–171. ACM Press, 2002.