Integrated I-cache Way Predictor and Branch Target Buffer to Reduce Energy Consumption

Weiyu Tang Alexander V. Veidenbaum Alexandru Nicolau
Rajesh Gupta
Department of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
{wtang, alexv, nicolau, rgupta}@ics.uci.edu

Abstract

In this paper, we present a Branch Target Buffer (BTB) design for energy savings in set-associative instruction caches. We extend the functionality of a BTB by caching way predictions in addition to branch target addresses. Way prediction and branch target prediction are done in parallel. Instruction cache energy savings are achieved by accessing one cache way if the way prediction for a fetch is available.

To increase the number of way predictions for higher energy savings, we modify the BTB management policy to allocate entries for non-branch instructions. Furthermore, we propose to partition a BTB into ways for branch instructions and ways for non-branch instructions to reduce the BTB energy as well.

We evaluate the effectiveness of our BTB design and management policies with SPEC95 benchmarks. The best BTB configuration shows a 74% energy savings on average in a 4-way set-associative instruction cache and the performance degradation is only 0.1%. When the instruction cache energy and the BTB energy are considered together, the average energy-delay product reduction is 65%.

1 Introduction

To exploit instruction level parallelism, modern processors support out of order execution engines. This requires instruction fetch across basic block boundaries to keep the execution units busy. Many instructions will be fetched and issued before a branch target address can be resolved.

A BTB [16] is used for branch target address prediction. There are two flavors of BTB design, coupled and decoupled where the tradeoff in performance and energy is fixed at the design time.

The coupled BTB design is used in Alpha 21264 [12] and UltraSparc-II[1], where the BTB is integrated with the instruction cache. Each cache line has two additional fields, a "line-prediction" field and a "way-prediction" field, to predict a cache line for the next fetch.

The decoupled design is used in Pentium 4 [8], Athlon [2] and G5 [7], where a BTB is organized as a separate cache. The number of BTB entries in Pentium 4, Athlon and G5 is 4K, 2K and 4K, respectively. The branch instruction address is used to index a BTB. If a matching entry is found and the branch is predicted taken, the BTB will provide the target address for the next fetch.

In a processor with a coupled BTB design, one instruction cache way is accessed when the way prediction is available. Therefore the instruction cache energy is lower than that in a processor with a decoupled BTB design where all the ways are accessed in parallel for high performance.

However, the accuracy of target address prediction in a coupled BTB is often lower than that in a decoupled design with same number of entries and associativity. "Way-prediction" and "line-prediction" fields in a coupled BTB can only point to one cache line. Prediction misses often occur if there are multiple branches in a cache line or branches change direction. A decoupled BTB can still provide accurate prediction in such cases. In modern processors with long pipelines, high target address prediction accuracy by the BTB is crucial for high performance.

In this paper, we propose a **W**ay-**P**redicting **BTB** (WP-BTB) for energy savings in set-associative instruction caches. A WP-BTB makes modifications to

a conventional decoupled BTB design. When a branch is allocated an entry in the BTB, in addition to the branch target address, way predictions for both the cache line with the target address and the cache line with the fall-through address are saved in the WP-BTB. When a branch accesses the WP-BTB for next fetch address and the branch has an entry in the WP-BTB, both the next fetch address and the corresponding way prediction are provided. Hence the next fetch needs only to access one cache way.

To enable more way predictions for higher instruction cache energy savings, we also modify the WP-BTB allocation policy. Non-branch instructions are also allocated entries in the WP-BTB. This increases the number of BTB accesses and hence overall BTB energy consumption. To keep the BTB energy low, a set-associative WP-BTB can be partitioned into ways for branch instructions and ways for non-branch instructions. As a consequence, the overall BTB energy is reduced due to lower per BTB access energy as not all the ways are accessed.

The rest of this paper is organized as follows. In Section 2, we briefly describe related work on cache and BTB energy savings. We present in Section 3 the design of WP-BTB. The experimental results are given in Section 4. This paper is concluded with future work in Section 5.

2 Related Work

2.1 Cache Energy Savings

High utilization of the instruction memory hierarchy is needed to exploit instruction level parallelism. As a consequence, energy consumption by on-chip instruction caches can comprise as high as 27% of the CPU energy [11].

Cache way partitioning is often used for energy savings in set-associative caches. The energy consumption by one cache way in a n-way set-associative cache is approximately $1/n$ the total cache energy consumption when all the ways are accessed.

Way prediction technique as used in Alpha 21264 can reduce instruction cache energy as only one cache way is accessed when the prediction is correct. To minimize way prediction miss penalty in a coupled BTB design, tags for all the ways are accessed in parallel with predicted way data access. The percentage of energy consumption by tag access increases with the number of cache ways. Based on energy parameters generated by Cacti [17], the relative energy consumption by a 4-way tag access and a one-way data access in a 32KB 4-way set-associative cache is 44% and 56% respectively. In an 8-way cache, the energy consumption by an 8-way tag access and a one-way data access is 66% and 34% respectively.

Another method of way prediction is proposed in [10]. A table is used to save for each cache set the most recently used way. On a cache access, the table is accessed first to retrieve the way prediction. Then the predicted way is speculatively accessed. On a way prediction miss, the remaining ways are accessed in the next cycle. This approach suffers from serialization, which may affect the processor cycle time if cache access is on a critical path of the processor.

Alpha 21164 uses another kind of access serialization for L2 cache energy savings. Tags for all ways are accessed first to determine which way holds the data. Then that particular data way is accessed. This kind of access serialization nearly doubles cache access time and cannot apply to the L1 cache where short cache access time is critical to performance.

[3] recognizes that not all the cache ways are needed in an application and proposes to turn off some ways for energy savings based on application demands. All the remaining ways are accessed in parallel. Hence energy savings are much smaller than those in way prediction based techniques where only one way is accessed most of the time. This technique also cannot apply to applications with large working set because the number of energy expensive L1 cache misses may increase dramatically.

Cache subbanking and line buffers [9] are used for cache energy savings. These techniques are orthogonal to way prediction based energy saving techniques. For example, when cache subbanking is used in a n-way set-associative cache, n subbanks, one from each cache way, have to be accessed in parallel. If the cache way holding the data is known before the access, then only one subbank needs to be accessed. As a consequence, the percentage of energy savings by way prediction based techniques is unchanged.

Filter cache [13] and L-cache [4] place a small and hence low-energy cache in front of the L1 cache. This design has high performance penalty and is only effective for applications with a small working set.

2.2 BTB Energy Savings

A large BTB is required for high performance. The number of BTB accesses is large and its energy consumption is also high. Rather small 512-entry BTB of the Pentium Pro consumes about 5% of the processor energy [14].

BTB is often organized as a set-associative cache and several ways can be turned off for energy savings if

application demands are low [3]. [15] uses a BTB predictor for BTB energy savings. The assumption is that BTB misses occur in bursts. If a miss in the BTB is predicted for the next access, then BTB is not accessed for the next branch instruction.

Decreasing the BTB entry size can also reduce the BTB energy. In G5 [7], instruction size is 32 bits. Instead of using 32 bits each for branch address and target address, the number of bits for each of them is 13 and 19 respectively. This approach trades branch target address prediction accuracy for energy savings.

3 WP-BTB Design

3.1 Way Prediction for Branch Instructions

Each entry of the WP-BTB is augmented with two fields for way prediction:

- *twp*: **T**arget **W**ay **P**rediction for the cache line with the branch target address;
- *fwp*: **F**all-through **W**ay **P**rediction for the cache line with the fall-through address.

Each entry in the WP-BTB is in the following format:

(branch_addr, target_addr, lru, twp, fwp).

Branch_addr and *target_addr* are used for target address prediction and *lru* is used to manage the replacement in a set-associative BTB.

When a branch address hits in the WP-BTB, the predicted target address and the corresponding way prediction are provided simultaneously. Using way prediction, the next fetch needs only to access one cache way.

The following hardware support is needed for way prediction in the WP-BTB:

- A queue, called **W**ay **Q**ueue (WQ), holds recently hit ways;
- A queue, called **B**TB **U**pdate **R**equest **Q**ueue (BURQ), holds pending WP-BTB update requests;
- Additional fields for each branch instruction in the pipeline:
  - *wq_ptr*: pointer to the next entry in the WQ;
  - *fwp, twp*: way predictions of the hit WP-BTB entry of this branch.

For a branch, the next fetch line is either the fall-through cache line or the target cache line depending on the direction of the branch. WQ and *wq_ptr* are used to track the way for either the target cache line or the fall-through cache line. BURQ is needed because the WP-BTB update requests cannot be processed immediately if the next fetch address misses from the cache or the target address prediction is not correct. Fields *fwp* and *twp* are needed for way prediction update.

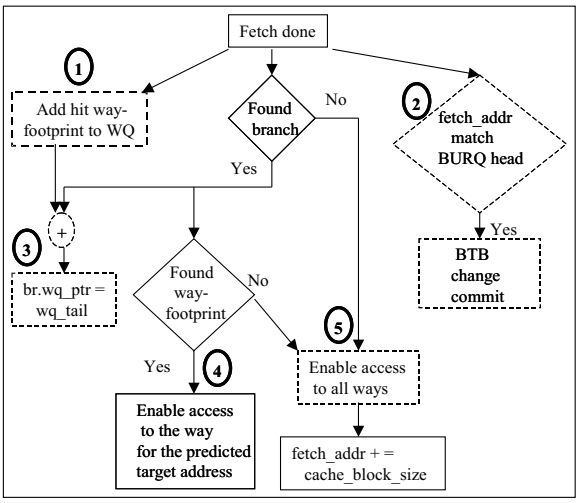

Figure 1. Modifications when instruction fetch finishes.

Figure 1 shows modifications (labels "1", "2", "3", "4" and "5") to the conventional design when an instruction fetch finishes. First, the hit way is added to the WQ. Second, if the head entry in the BURQ is waiting for the finish of this fetch, then that entry is ready to commit changes to the WP-BTB. Third, if a branch instruction is found, the *wq_ptr* of this branch is set to the next entry in the WQ. When a branch instruction commits and the branch prediction is correct, *wq_ptr* is used to retrieve the way for either the fall-through cache line or the target cache line. Fourth, if the way prediction for the next fetch is found in the WP-BTB, the next fetch needs only to access one way. Otherwise, all cache ways are enabled for the next fetch as indicated by label "5".

Modifications to the branch instruction commit are shown in Figure 2. In a conventional BTB, if a branch has an entry in the BTB, changes to the BTB can be updated immediately because the target address is already resolved. In the WP-BTB, the way prediction is also needed. Thus changes to the WP-BTB can be updated only after the way prediction is available. There

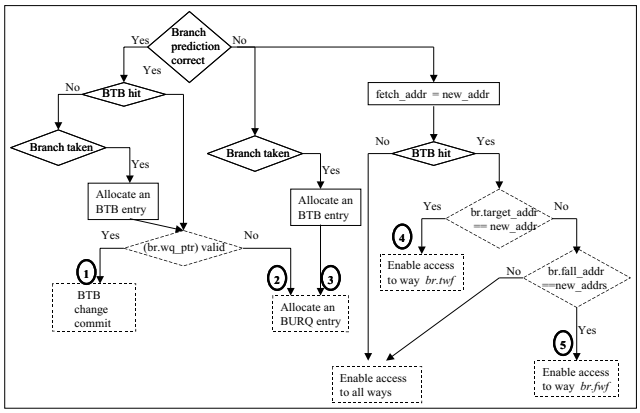

Figure 2. Modifications for branch instruction commit.

are two scenarios when way prediction is not available (label "2" and "3"). In the first scenario, the branch target prediction is correct but the next fetch address is missing from the cache. In the second scenario, the predicted address is not correct and the cache line with the correct address is to be fetched in the next cycle. In either scenario, the way prediction will be available once next fetch finishes. Therefore the WP-BTB update request will be added to BURQ first. Then once the next fetch finishes, the head entry of the BURQ will be ready to update the WP-BTB (label "2" in Figure 1). On branch prediction misses, if the way prediction for the cache line with the correct fetch address is found in the WP-BTB (label "4" and "5"), then only one way needs to be accessed for the next fetch.

3.2 Way Prediction for Non-branch Instructions

As only branch instructions access the WP-BTB, way prediction is limited to speculative fetch addresses. As a consequence, instruction cache energy savings are only available to speculative instruction fetches. To enable more instruction cache energy savings, non-branch instructions are allowed to access the WP-BTB for way prediction. For this purpose, every non-branch instruction in the pipeline needs the following additional fields, which are similar to the fields added for branch instructions:

- *wq_ptr*: pointer to the next entry in the WQ;
- *fwp* : way prediction for the fall-through cache line.

Wq_ptr is needed to track the way prediction for the next fetch line and fwp is needed for WP-BTB way prediction update.

If there is no branch in a cache line that is under fetch, the following changes are needed:

- the address of the first instruction is used to index the WP-BTB to find the way prediction for the fall-through cache line;
- the first instruction in the cache line will set wq_ptr to the next entry in the WQ;
- if the way prediction is found, then only one way will be enabled for the next fetch.

When a non-branch instruction commits, if its wq_ptr is valid and it doesn't have an entry in the WP-BTB, an entry will be allocated. Then, if the way prediction is available at this time, the WP-BTB can be updated immediately. In case the way prediction is not available, an entry will be added to the WQ to wait for the finish of the next fetch.

3.3 WP-BTB Partitioning

As a WP-BTB is organized as a cache, way-based cache energy saving techniques can also apply to a set-associative WP-BTB. A WP-BTB can be partitioned into ways for branch instructions and ways for non-branch instructions. We assume that there is predecoding or other mechanisms to determine whether a WP-BTB access is by a branch instruction or by a non-branch instruction. Therefore only the corresponding WP-BTB ways need to be enabled. Reduced switching activities per WP-BTB access result in BTB energy savings.

4 Experimental Results

4.1 Simulation Setup

We use the SimpleScalar toolset [6] to model an out-of-order superscalar processor. The processor parameters shown in Table 1 roughly correspond to those in a current high-end microprocessor. Wattch [5] is used for energy estimation. A set of SPEC95 benchmarks are simulated. The test input set is used.

We have evaluated the performance and energy of five WP-BTB configurations:

- *base*: only branch instructions access the WP-BTB;
- *share*: both branch and non-branch instructions can access all ways in the WP-BTB;

Parameter	Value
branch pred.	combined, 4K 2-bit chooser, 4k-entry bimodal, 12-bit, 4K-entry global
BTB	2K-entry, 4-way
RUU	64
LSQ	32
fetch queue	16
fetch width	4
int. ALUs	4
flt. ALUs	2
int. Mult/Div	2
flt. Mult/Div	2
L1 Icache	64KB, 4-way, 32B block
L1 Dcache	64KB, 4-way, 32B block
L2 cache	512KB, 4-way, 64B block

Table 1. System configuration.

- 1_3: the WP-BTB is partitioned into 1 way for branch instructions and 3 ways for non-branch instructions;
- 2_2: the WP-BTB is partitioned into 2 ways for branch instructions and 2 ways for non-branch instructions;
- 3_1: the WP-BTB is partitioned into 3 ways for branch instructions and 1 way for non-branch instructions.

4.2 Results

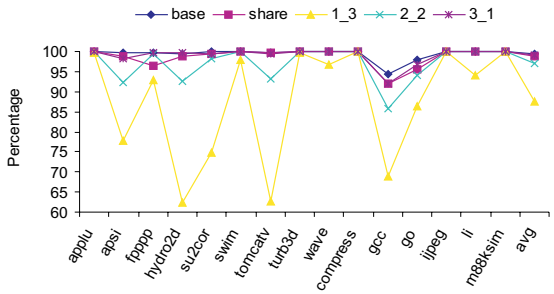

Figure 3. Branch hit rate in the WP-BTB.

Figure 3 shows branch hit rate in the WP-BTB. As branch prediction is not available for a branch missing from the WP-BTB, low hit rate means low prediction accuracy. The hit rate decreases with effective BTB capacity, the number of entries available for branch instructions. *Share* almost always has the highest hit rate. The rate by 3_1 is almost same as that by *share*. For some benchmarks, the difference in hit rate by different configurations is very small. These benchmarks either don't have many branches like *applu*, or have many branches with high access locality like *ijpeg* and *li*. BTB capacity decrease won't impact hit rate much for these benchmark. On the other hand, *hydro2d*, *su2cor*, *tomcatv* and *gcc* are affected the most by the decrease in BTB capacity and associativity. They have many branches that cannot fit in a small and low associativity BTB.

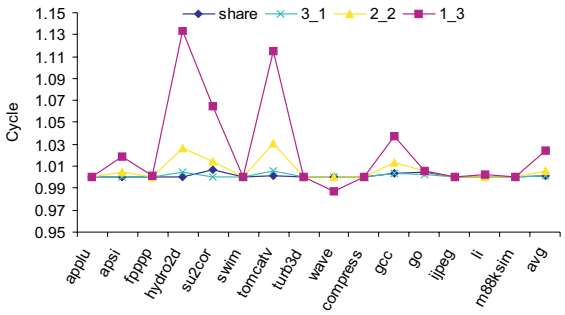

Figure 4. Normalized execution time.

Figure 4 shows normalized execution time with regard to *base*. For 2_2 and 1_3, five benchmarks–*apsi*, *hydro2d*, *su2cor*, *tomcatv* and *gcc* show high performance degradation. Referring to Figure 3, we notice that execution time increases whenever the branch hit rate in BTB decreases. Hit rate decrease results in performance degradation. There is virtually no performance degradation for *share* and 3_1 as the branch hit in the BTB is unchanged. The average performance degradation for *share*, 3_1, 2_2 and 1_3 is 0.1%, 0.1%, 0.6% and 2.4% respectively.

Figure 5 shows the non-branch access hit rate in the WP-BTB. *Share* has the highest hit rate, followed in order by 1_3, 2_2 and 3_1 as the number of entries available for non-branch access decreases. Most benchmarks have small working set and 1K entries for non-branch instructions are enough for way prediction by most instructions. Hence the difference in hit rate among 2_2 , 1_3 and *share* is very small and the hit rate for *share* and 1_3 is close to 100%.

Noticeable differences in hit rate can be found in half of the benchmarks between 2_2 and 3_1. For these benchmarks, 512 entries for non-branch instructions are not enough for way prediction by instructions in

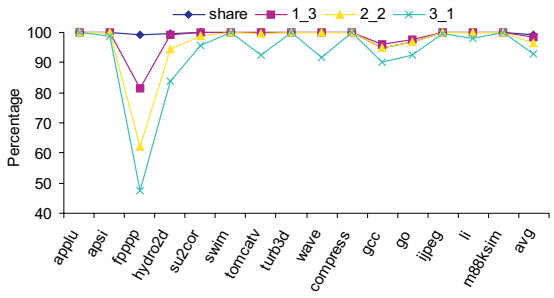

Figure 5. Non-branch access hit rate in the WP-BTB.

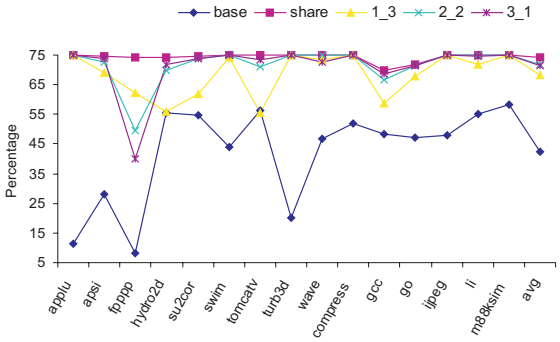

Figure 7. Instruction cache energy savings.

the working set. For example, *fpppp* has a large working set, which is evident as instruction cache miss rate in *fpppp* increases from 0.5% to 7.5% when the cache size decreases from 64KB to 32KB. Therefore the differences in hit rate among different configurations are large.

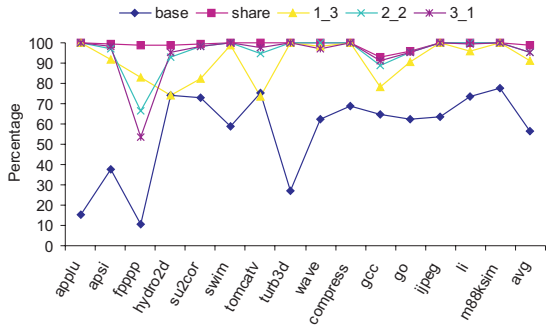

Figure 6. BTB hit rate for branch and non-branch accesses.

Figure 6 shows BTB hit rate for branch and non-branch accesses. High hit rate indicates more number of way predictions. *Share* has the highest hit rate and *base* has the lowest. The hit rate for partitioned BTB configurations varies. 1_3 generally has lower hit rate than 2_2 and 3_1. For all benchmarks except *fpppp*, there is at least one partitioned configuration that has a hit rate as high as *share*.

Figure 7 shows instruction cache energy savings. The percentage in savings follows the same trend as the BTB hit rate shown in Figure 6. When there is a hit in the BTB, the way prediction for the next fetch line can be retrieved and the next fetch needs only to access one cache way. High hit rate results in high instruction cache energy savings. The average percentage in energy savings for *share*, 1_3, 2_2 and 3_1 is 74.2%, 68.3%, 71.6% and 71.3% respectively. *Share* results in the best instruction cache energy savings.

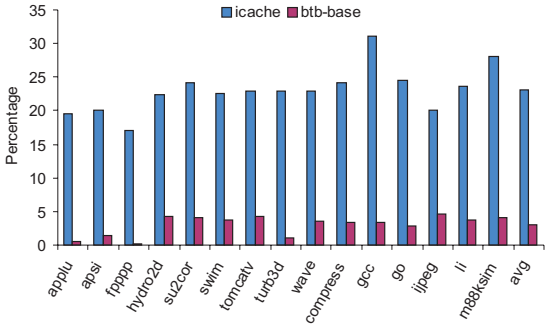

Figure 8. % processor energy contribution by the instruction cache and the BTB for configuration *base*.

Figure 8 shows processor energy contribution (clocking energy excluded) by the instruction cache and the BTB for configuration *base*. The instruction cache contributes a much larger percentage to the processor energy than the BTB. The contribution by the instruction cache ranges from 17% in *fpppp* to 31% in *gcc* with an average of 23%. The contribution by the BTB ranges from 0.25% in *fpppp* to 4.55% in *ijpeg* with an average of 2.97%.

Figure 9 shows processor energy contribution by the instruction cache and the BTB for configuration *share*. Comparing with Figure 8, we notice that the instruction cache energy decreases dramatically. On the other hand, average BTB energy consumption nearly doubles

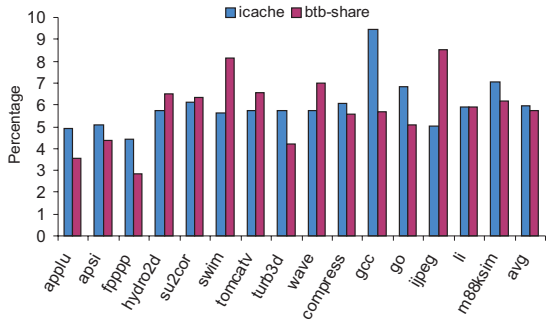

Figure 9. % processor energy contribution by the instruction cache and the BTB for configuration *share*.

with additional BTB activity by non-branch instruction accesses. For some benchmarks such as *swim* and *ijpeg*, the energy consumption by the BTB is larger than that by the instruction cache. The average BTB energy is roughly equal to the average instruction cache energy. This figure shows the need to keep BTB energy under control as well.

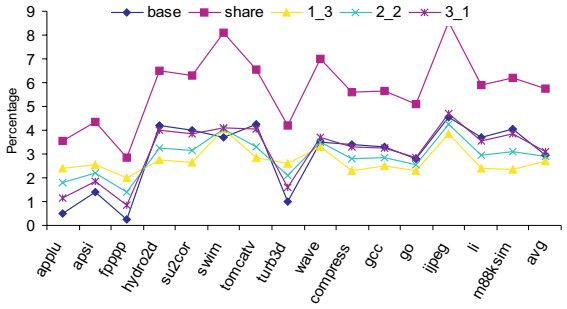

Figure 10. % processor energy contribution by the BTB with different WP-BTB configurations.

Figure 10 shows processor energy contribution by the BTB with different WP-BTB configurations. The BTB energy by *share* is much higher than that by other configurations because the number of BTB accesses increases dramatically. Although the number of accesses in partitioned BTB configurations also increases, per access energy decreases because not all the ways are accessed. For 5 benchmarks, *base* achieves the minimal BTB energy. For the remaining 10 benchmarks, 1_3 achieves the minimal BTB energy. 1_3 has the lowest average BTB energy, followed in order by 2_2, 3_1 and *base*. The differences among them are small. We conclude that partitioned BTB configurations don't increase BTB energy much even with more number of accesses.

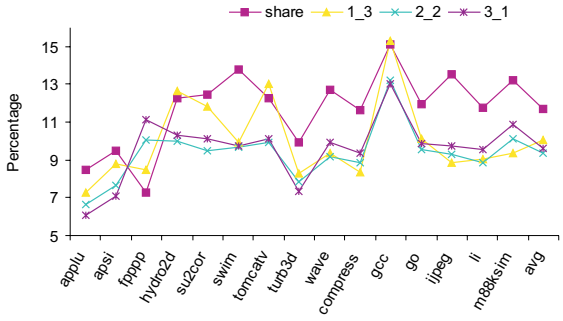

Figure 11. % processor energy contribution by the instruction cache and the BTB with different WP-BTB configurations.

Figure 11 shows total energy contribution by the instruction cache and the BTB. Partitioned BTB configurations achieve the minimal energy for all benchmarks except *fpppp*. Which partitioned configuration is the best for a given benchmark depends on the working set size and the ratio of branch instruction accesses to the non-branch instruction accesses. For example, 3_1 is the best for *applu* and *turb3d*. In these benchmarks, the working set size is small and most of the BTB accesses are by non-branch instructions. Single BTB way for non-branch instruction accesses results in minimal overall energy. As either compiler or profiling techniques may determine the branch to non-branch access ratio and the working set size, the optimal BTB partition may be determined statically based on application demands for the best tradeoff in performance and energy.

Figure 12 shows energy-delay product for the instruction cache and the BTB. For each benchmark, point *min* represents the minimal energy-delay product for this benchmark among all the configurations. The minimal energy-delay product ranges from 0.29 for *m88ksim* to 0.42 for *fpppp* and the average is 0.35. An average 65 percent energy-delay product reduction for the instruction cache and the BTB is achieved by the WP-BTB.

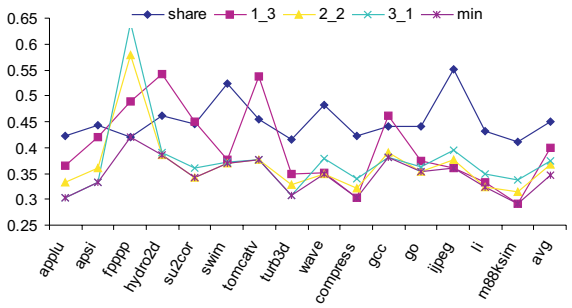

Figure 12. Energy-delay product for the instruction cache and the BTB.

5 Conclusion

In this paper, we have presented a way-predicting BTB design. It enables way prediction by both branch and non-branch instructions for energy savings in set-associative instruction caches. In addition, a set-associative WP-BTB is partitioned into ways for branch instruction and ways for non-branch instructions to keep the WP-BTB energy under control.

Configuration *share* is the best in terms of instruction cache energy. It achieves an average 74% of energy saving in a 4-way set-associative instruction cache with only 0.1% performance degradation. When the BTB energy and the instruction cache energy are considered together, partitioned BTB configurations are better than the *share* configuration and can achieve 65% reduction in energy-delay product.

We are currently investigating other techniques such as compiler/profiling techniques and dynamic mechanisms to determine application-specific BTB configuration with the best tradeoff in energy and performance.

Acknowledgments

This work is supported by DARPA/ITO under PACC and DIS programs.

References

[1] K.B. Normoyle et al. UltraSparc-IIi: expanding the boundaries of system on a chip. *IEEE Trans. Micro*, 18(2):14–24, 1998.

[2] Advanced Micro Devices, Inc. *AMD athlon processor architecture*, 2000. White paper.

[3] D. H. Albonesi. Selective cache ways: on-demand cache resource allocation. In *Int'l Symp. Microarchitecture*, pages 248–259, 1999.

[4] N. Bellas, I. Hajj, and C. Polychronopoulos. Using dynamic cache management techniques to reduce energy in a high-performance processor. In *Int'l Symp. on Low Power Electronics and Design*, pages 64–69, 1999.

[5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a framework for architectural-level power analysis and optimizations. In *Int'l Symp. Computer Architecture*, pages 83–94, 2000.

[6] D. Burger and T.Austin. The simplescalar toolset, version 2.0. Technical Report TR-97-1342, University of Wisconsin-Madison, 1997.

[7] M. Check and T. Slegel. Custom S/390 G5 and G6 microprocessors. *IBM Journal of Research and Development*, 43(5/6):671–680, 1999.

[8] G. Hinton et al. The microarchitecture of the pentium 4 processor. *Intel Technology Journal*, Q1, 2001.

[9] K. Ghose and M. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *Int'l Symp. on Low Power Electronics and Design*, pages 70–75, 1999.

[10] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In *Int'l Symp. on Low Power Electronics and Design*, pages 273–275, 1999.

[11] J. Montanaro et al. A 160-MHz, 32-b, 0.5-W CMOS RISC microprocessor. *IEEE Journal of Solid-State Circuits*, 32(11):1703–14, 1996.

[12] R. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.

[13] J. Kin, M. Gupta, and W. Mangione-Smith. The filter cache: An energy efficient memory structure. In *Int'l Symp. Microarchitecture*, pages 184–193, 1997.

[14] S. Manne, A. Klauser, and D. Grunwald. Pipeline gating: speculation control for energy reduction. In *Int'l Symp. Computer Architecture*, pages 132–141, 1998.

[15] E. Musoll. Predicting the usefulness of a block result: a micro-architectural technique for high-performance low-power processors. In *Int'l Symp. Microarchitecture*, pages 238–247, 1999.

[16] C. Perleberg and A. Smith. Branch target buffer design and optimization. *IEEE Trans. Computers*, 42(4):396–412, 1993.

[17] S. Wilton and N. Jouppi. An enhanced access and cycle time model for on-chip caches. Technical Report 93/5, Digital Western Research Laboratory, 1994.