Practical Way Halting by Speculatively Accessing Halt Tags

Daniel Moreau[†], Alen Bardizbanyan[†], Magnus Själander[‡], David Whalley[§], and Per Larsson-Edefors[†]
[†]Chalmers University of Technology, Gothenburg, Sweden
[‡]Uppsala University, Uppsala, Sweden
[§]Florida State University, Tallahassee, USA
nd.moreau@gmail.com, alenb@chalmers.se, magnus.sjalander@it.uu.se, whalley@cs.fsu.edu, perla@chalmers.se

***Abstract*—Conventional set-associative data cache accesses waste energy since tag and data arrays of several ways are simultaneously accessed to sustain pipeline speed. Different access techniques to avoid activating all cache ways have been previously proposed in an effort to reduce energy usage. However, a problem that many of these access techniques have in common is that they need to access different cache memory portions in a sequential manner, which is difficult to support with standard synchronous SRAM memory. We propose the speculative halt-tag access (SHA) approach, which accesses low-order tag bits, i.e., the halt tag, in the address generation stage instead of the SRAM access stage to eliminate accesses to cache ways that cannot possibly contain the data. The key feature of our SHA approach is that it determines which tag and data arrays need to be accessed early enough for conventional SRAMs to be used. We evaluate the SHA approach using a 65-nm processor implementation running MiBench benchmarks and find that it on average reduces data access energy by 25.6%.**

I. Introduction

Memory hierarchies have long been designed with performance and cost as key metrics. In recent years, energy efficiency has become more important as we aim to extend the battery life of mobile computing devices, to provide a competitive clock rate under present thermal constraints, and to reduce the electricity used. It has been shown that the level-one (L1) data cache (DC) is responsible for a significant portion of a processor's energy usage [1]. The dilemma when reducing L1 DC energy, however, is that load accesses are performance critical. Thus, the challenge which we address in this paper is how to significantly improve the energy efficiency of L1 DC accesses without degrading performance.

A data access in a conventional set-associative L1 DC causes all ways to be accessed in parallel. This practice achieves good performance but wastes energy as the sought memory location can at most reside in one of the ways. Several proposed techniques, e.g., way prediction [2], attempt to reduce the energy by limiting the number of ways that are accessed. However, the L1 DC is timing sensitive and any increase in its critical path is likely to have a negative effect on a CPU core's clock rate and energy dissipation. Furthermore, L1 DC timing issues can not easily be mitigated by increasing the pipeline depth as the delay between loads and their use would increase, which has a significant negative impact on performance [3]. Thus, it is crucial that any technique that attempts to improve the energy efficiency of the L1 DC does not adversely affect its timing.

One proposed technique for reducing L1 DC energy is the way-halting cache (WHC) which halts the L1 DC access to a way at the point it is known that the way within a set cannot possibly contain the data [4]. Here, the tags for the L1 DC are split into two parts; a few low-order bits called the halt tag and the remaining higher-order bits of the tag. The halt tags are first compared with the low-order bits of the base address tag and the L1 DC tag and data arrays are only accessed when there is a halt-tag match. The key insight here is that the low-order bits of the tag are the bits that are the ones most likely to differ [5].

We have previously proposed the speculative tag access (STA) technique to reduce L1 DC energy. Here, the L1 DC tag arrays are accessed in the address generation stage when the magnitude of the displacement field does not exceed half the line size of the L1 DC [6]. In this paper, we propose to combine the advantages of the WHC and our STA technique to create a practical and energy-efficient implementation of low complexity. We propose to use way-halt tag arrays, but to access them in the address generation stage instead of the SRAM access stage, as shown in Fig. 1. Thus, our speculative halt-tag access (SHA) approach determines which L1 DC tag and data arrays need to be accessed by the beginning of the SRAM access stage, which relaxes the timing requirement on the data access stage and enables a conventional SRAM implementation to be used for the L1 DC. By accessing the halt tags in the address generation stage, the SHA approach enables the halt-tag array access and the halt-tag comparison to be done during a whole clock cycle. Should the speculation fail, the cache is accessed conventionally the next cycle with no impact on performance.

II. Related Work

The way-halting cache (WHC) technique was proposed to be performed in parallel with the decoding of the L1 DC index field (see Fig. 2) so that a fully associative halt-tag check is required [4]. Another technique just suggests to access the L1 DC tag and data arrays after the partial tag comparison (PTC) is completed [5]. However, there are several aspects of these proposed techniques that may make them impractical to implement. The WHC technique was evaluated by detailed comparisons of the access time for a fully associative halt array and an address decoder, but the study neglects to account for potential wiring costs. To reduce word line lengths, memories are often banked and the tag and data are stored in individual

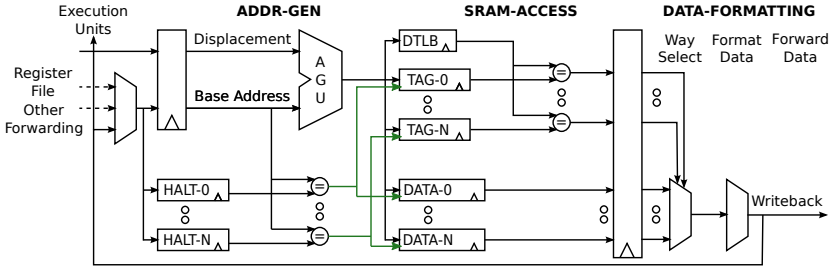

Fig. 1. Three-stage load pipeline with speculative halt-tag arrays and N-way cache.

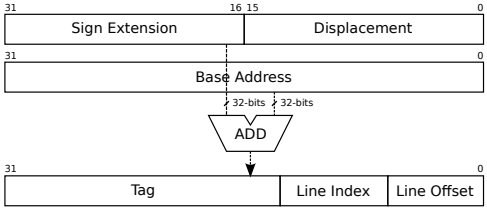

Fig. 2. Data access address calculation.

memories. The fully associative halt-tag structure would have to be either duplicated (i.e., incurring a higher energy and area overhead) for each bank in both the tag and data memories or the halt-tag hit signals would have to be routed across the chip to the different banks and memories (i.e., incurring delay penalties). Likewise, the PTC technique compares the partial tags after the index is decoded and have the same issues as WHC of applying the halt-tag hit signal to the sense amplifiers across all banks and memories. Furthermore, conventional SRAMs are synchronous for performance reasons and can only be controlled at the start of the clock cycle. Both approaches would require custom SRAM implementations, which are costly and not readily available to the majority of the semiconductor industry. Both approaches also assume page coloring for which the operating system (OS) enforces that the least significant bits (LSBs) of the virtual tags are the same as the LSBs of the physical tags.

Shafiee et al. proposed to use partial tag comparison to reduce energy dissipation in snoop-based chip multiprocessors [7]. For each snoop request, which requires a comparison for all the tag ways in a cluster of processors, the LSBs of all tag memories from different processors are checked. Only the tag ways that generate a partial hit signal are further probed for a full tag comparison, which reduces energy dissipation at the expense of an extra cycle. Their technique utilizes partial tag comparison in a very different context than our proposed SHA approach.

We recently proposed the STA technique: L1 DC tag arrays are accessed during the address generation stage and a single L1 DC data array is accessed in the SRAM access stage if the speculation succeeds and there is a hit in the cache [6]. While STA reduces the number of accessed data arrays, it will always access all tag arrays in parallel. Thus, STA has no effect on store energy, as stores always access a single data array. The STA approach also requires a more complex design than a conventional L1 DC. The data translation lookaside buffer (DTLB) is accessed in both the address generation and SRAM access stage, which gives rise to a more complex floorplan, where the input signal to the DTLB has to be routed from multiple locations on the chip. To make matters worse, the input from the address generation stage is produced by forwarding logic, which can lead to additional delay and constrain the timing of the DTLB even further. With more constrained timing, the DTLB could be on the critical path or require stronger, more power-dissipating gates.

Our proposed SHA approach accesses the halt tags in the address generation stage instead of the SRAM access stage. Thus, SHA determines which L1 DC tag and data arrays need to be accessed by the beginning of the SRAM access stage, which enables a conventional SRAM implementation to be used for the L1 DC. SHA should not impact the critical path since the halt-tag arrays can be accessed in parallel with the address generation. Since we propose to access the halt-tag arrays before the DTLB is accessed, the DTLB is only accessed during the SRAM access stage. Thus, the SHA halt-tag arrays contain bits from the virtual address while the complete physical tags are stored in the L1 DC tag arrays. To avoid aliasing and synonyms we enforce the same restrictions on the OS as used in PTC and WHC and, thus, the LSBs of the virtual and physical tags are the same. While the SHA approach does not decrease the size of the L1 DC tag storage, it does simplify the design.

III. Speculative Halt-Tag Access (SHA)

The speculative halt-tag access (SHA) technique is based on the observation that there is a high probability that only the cache-line offset of an address will change during address calculation if the displacement is small [8]. When the magnitude of the displacement is small we speculatively access the halt tags in the address generation stage using the index of the base address. The halt tags are then compared with the low-order bits of the tag of the base address. The speculation succeeds when the index and tag fields of the base address are the same as the index and tag fields of the effective address, see Fig. 2. If the speculation succeeds, then only those ways that have a halt-tag match (i.e., the halt tag is equal to the low-order bits of the tag) can possibly contain the sought memory location. During the SRAM access stage, the tag and data arrays of only those ways that had a halt-tag hit are enabled, see Fig. 1.

The halt-tag comparison is performed with the stored halt tag and the base address, see Fig. 1, to eliminate the need to calculate the effective address before performing the comparison. The validity of the halt-tag hit signals is assured by detecting, in the address generation unit (AGU), if a carry is propagated from the line offset to the line index. The speculative halt-tag access is successful if no carry is detected for a non-negative displacement. Conversely, the speculation is successful if a carry out is detected for negative displacements. Since it is readily available in conventional parallel adders, extracting this information has negligible impact on performance. The sign and bit-width of the displacement can

be detected in parallel with the sign extension of the immediate field, which is commonly done to obtain a 32-bit offset. The detection can be done by performing an AND (negative) and OR (positive) on the higher order bits (all bits except for the maximum bit-width of the displacement for which speculation is to be attempted).

For large displacements, the line index and tag are likely to change during the address calculation. Accessing the halt tags would then cause an overhead as the speculation is likely to fail. For large displacements the cache is therefore accessed conventionally without a halt-tag access in the address generation stage, with all ways being accessed in the SRAM access stage. In the event of a speculative access failure, the cache is conventionally accessed the next cycle without any performance penalty. Like in the STA technique, energy usage is only reduced when there is a successful speculative halt-tag access.

Accessing the halt tags speculatively in the address generation stage has the advantage that the access to the halt-tag array and comparison of the halt tags can be done in parallel with the address calculation. The resulting hit signals can then be used to set the enable signal of the tag and data arrays of the ways that have a halt-tag match. As the hit signal is available the cycle before the tag and data arrays are accessed, conventional SRAMs can be employed. The halt-tag array is of equal or smaller size than a single tag array, see Sec. V. With the proposed technique the access to the halt-tag array, comparison of the halt tags, and routing of the signal to enable port of the SRAMs can be done during a whole cycle. In contrast, with previous halt-tag proposals, the access, comparison, and routing are attempted in the same cycle as the SRAMs are to be read, which severely constrains the timing and requires custom SRAMs.

The stored halt tags are virtual to avoid the need to speculatively access the DTLB in the address generation stage. To ensure correct behavior, all halt tags are invalidated on a context switch. Because we guarantee that the LSBs of the virtual and physical tag are identical and because the full physical tag is accessed and compared for the ways with a halt-tag hit, there are no issues with aliasing or synonyms.

Our approach incurs very little additional complexity to a conventional CPU core design.

1) The size of the displacement has to be checked in a stage, such as instruction decode, before the address generation stage.
2) The halt-tag array has to be added to the address generation stage and enabled only if the displacement was determined to be small.
3) The resulting halt-tag hit signals have to be routed to the enable inputs of the tag and data arrays, and disable them if the displacement was determined to be small and there is no halt-tag match.

No other modifications are required. In particular the only modification to the L1 DC is the halt-tag hit signals that need to be included for driving the enable signals of the SRAMs.

IV. Evaluation Method

To accurately evaluate the data access energy, we implement 16kB 4-way data and instruction caches in the context of a 5-stage in-order processor. The RTL implementation of the pipeline is synthesized using the Synopsys Design Compiler [9] based on a commercial 65-nm 1.2-V CMOS low-power process technology, with standard cells and mixed-V_T SRAM macros. The physical implementation is performed using Cadence Encounter [10]. The final placed and routed processor implementation meets a timing constraint of 2.5 ns (400 MHz), assuming the worst-case process corner, a supply voltage of 1.1 V, and 125 °C. To validate pipeline and cache functionality, the netlist of this physical implementation is verified using the EEMBC benchmark suite [11].

Power analysis is performed using Synopsys PrimeTime PX [12] on the physical implementation and its RC-extracted netlist. Based on the nominal process corner, 1.2 V, and 25 °C, power values are retrieved for different components at a clock rate of 400 MHz. For each component, the energy per cycle is obtained as the average power dissipation multiplied by the clock period. Since this evaluation is based on a low-power process technology, the leakage energy is negligible; 0.75 pJ in static energy is dissipated per 2.5-ns cycle.

In order to perform energy evaluations we use SimpleScalar [13] and a selection of MiBench benchmarks [14]. The simulator is modified to simulate the 5-stage in-order processor described above. A bimodal branch predictor with 128 entries is used. The access time for the memory hierarchy is 1, 12 and 120 cycles for L1 caches, L2 caches and main memory respectively. In total we use 20 MiBench benchmarks distributed across six categories; Automotive, Consumer, Network, Office, Security and Telecom. The benchmarks are compiled with GCC using the large data set option. The final energy values are calculated by combining the energy per cycle results with activities tracked by SimpleScalar while running MiBench until completion.

The L1 DC consists of four 1024x32b-mux8 4kB SRAM macros for the data arrays. For our 32B cache line size and 4kB page size, the tag bit-width is 20 bits. Considering the additional valid bit, four tag arrays together represent 84 bits, which means that the four 128x32b-mux4 512B SRAM macros used for the tags are not fully utilized. The halt tags are assumed to be no larger than 8 bits. Four halt tags can then be stored in a single 128x32b-mux4 512B SRAM macro.

We use the same energy evaluation methodology as in our earlier STA work [6]. We consider the energy dissipated internal to the SRAM blocks as well as energy dissipated by logic transitions on any of the SRAM block's peripheral address, data in, and mask pins. As for the output, energy is dissipated when driving the output load. The energy values for accessing the different structures of the L1 DC are shown in Table I.

The energy estimations include peripheral circuits in addition to the tag and data arrays. All estimations of these peripheral units are done pessimistically, i.e., all combinatorial

TABLE I
L1 DC COMPONENT ENERGY

Component	Energy (pJ)
Read Halt	19.1
Write Halt	17.7
Read Tag	19.1
Write Tag	17.6
Read Data	26.5
Write Data	27.2
DTLB	17.5
Peripheral	18.8
Arbiter	2.0

nodes are assumed to switch on each load and store operation. The peripheral units include the replacement unit (LRU), the cache controller, and the remaining multiplexers. Additionally, an arbiter is used to grant access to either the instruction or data cache to access lower levels in the memory hierarchy. We use a 16-entry fully associative DTLB, once again with the same pessimistic assumption for the combinatorial nodes.

We estimate the total L1 DC energy by identifying the set of use cases in which the cache can be accessed and then for each case we determine the accessed cache structures and their energy dissipation. The relative frequency of the different use cases can then be used to calculate the relative energy usage.

Table II shows the accessed components for the identified L1 DC use cases. The **BL** and **BS** use cases represent loads and stores, respectively, of a baseline L1 DC. As depicted in the table, the baseline case only accesses three tag arrays. This optimization can be performed if all tags are read in parallel, as all four tags fit in a total of three 32-bit memories. The **STA0** and **STA1** use cases represent an L1 DC cache where the tags are speculatively accessed. **STA0** represents the case when the access is successful and only a single data array is accessed, while **STA1** represents a failed access and all tag arrays need to be accessed a second time. The STA technique only works for loads and is used in combination with the **BS** use case.

The remaining use cases in Table II represent our proposed approach. **SHA0** (load) and **SHA1** (store) represent the case where the halt arrays are not accessed (the displacement is too large). As depicted in the table, four tag arrays are accessed in our approach. This is necessary to be able to disable individual tag accesses as decided by the hits in the halt-tag array. **SHA2:**X (highlighted in gray) represents a successful speculative load where X represents the number of matching halt tags. **SHA3** represents a failed speculative load, which in terms of accessed structures and energy dissipation is equal to a successful access where all four halt tags match, both of which are rare events as shown in Sec. V. **SHA4:**X (highlighted in gray) represents a successful speculative store where X represents the number of matching halt tags. **SHA5** represents a failed speculative store.

The use cases in Table II assume that the access hits in the cache. In case of a cache miss, additional energy is dissipated as determined by Table III. The energy of a cache miss depends on if the cache line is dirty and needs to be written back (WB). The baseline/STA cases (**B-NoWB** and **B-WB**) do not require a halt tag to be written, which is required for the SHA cases (**H-NoWB** and **H-WB**).

TABLE II
COMPONENTS ACCESSED FOR EACH CASE

Case	Read Halt	Read Tag	Read Data	Write Data	DTLB	Peripheral	Energy (pJ)
BL	-	3	4	0	1	1	182.1
BS	-	3	0	1	1	1	103.3
STA0	-	3	1	0	1	1	102.6
STA1	-	6	4	0	1	1	239.4
SHA0	0	4	4	0	1	1	201.2
SHA1	0	4	0	1	1	1	122.4
SHA2:0	1	0	0	0	1	1	37.9
SHA2:1	1	1	1	0	1	1	83.5
SHA2:2	1	2	2	0	1	1	129.1
SHA2:3	1	3	3	0	1	1	174.7
SHA2:4	1	4	4	0	1	1	220.3
SHA3	1	4	4	0	1	1	220.3
SHA4:0	1	0	0	0	1	1	37.9
SHA4:1	1	1	0	1	1	1	84.2
SHA4:2	1	2	0	1	1	1	103.3
SHA4:3	1	3	0	1	1	1	122.4
SHA4:4	1	4	0	1	1	1	141.5
SHA5	1	4	0	1	1	1	141.5

TABLE III
COMPONENTS ACCESSED ON MISS EVENTS

Case	Write Halt	Write Tag	Read Data	Write Data	Peripheral	Arbiter	Energy (pJ)
B-NoWB	-	1	0	8	8	8	251.2
B-WB	-	1	8	8	16	16	479.1
H-NoWB	1	1	0	8	8	8	268.9
H-WB	1	1	8	8	16	16	496.8

V. RESULTS

The speculative and halt components of the SHA technique are orthogonal in the sense that the speculation success ratio does not affect the choice of halt-tag width. With that in mind, we start by evaluating halt-tag widths. In Fig. 3 the simulation results using bit-widths between two and eight are shown. Our evaluation is limited by only having access to a 32-bit wide SRAM macro for the halt array. Energy benefits due to smaller halt tags are therefore not shown in the figure. From the results we can see that a larger bit-width consistently produces better results. The rationale for this trend is that a wider halt tag enables an increasing number of tag and data array halts as a larger halt tag is less likely to match. As can be seen, the benefit of a halt tag larger than six bits is diminishing. With an SRAM macro adapted to the halt tag size, the energy cost of accessing the halt array would be reduced and the best case would likely be a five or six bit wide halt tag. However, since we are unable to adapt the size, we choose to proceed with eight bits to fully utilize the available SRAM macro.

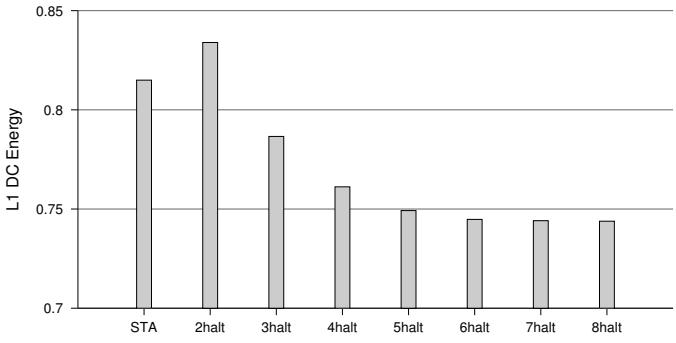

Fig. 3. Energy for different halt-tag widths.

Just like in the original STA technique, more energy is saved if the speculation ratio is kept high, as this enables more ways to be halted. Speculation failures for the STA technique are very costly as the tags have to be accessed a second time. A major concern for STA is therefore to minimize the number of

failed speculations. An SHA failure only incurs an overhead of the halt-tag array access, which is one third[1] of the energy overhead of STA. An SHA failure is therefore significantly less costly than an STA failure. Hence, we need to identify the new offset range for which speculations should be attempted under SHA. The L1 DC used in this study has a line size of 32B with a line offset of five bits. While it is possible to speculate on offsets exceeding the line offset using an OR-based approach [6], [8] this would necessitate additional logic on the critical path. The OR-based index and tag would have to be compared with the effective address computed by the AGU, which puts this comparison in series with the AGU. Since one of the primary goals of the SHA approach is a practical implementation, we decided to dismiss the OR-based method reassured by simulation results showing very limited energy gains. In the original STA paper, a positive offset limit of four bits was found to be best while increasing it to five bits increased the speculation failures too much. However, in Table IV we can see that for the SHA approach, using the full line offset is the best design point. For negative offsets, the original STA implementation already used the full line offset and for reasons already discussed we decided against increasing this.

TABLE IV
ENERGY FOR 4-BIT AND 5-BIT SPECULATION LIMITS (NORMALIZED TO L1 DC ACCESS ENERGY)

Offset	Miss energy	Store energy	Load energy
4 bits	0.01332	0.18909	0.55426
5 bits	0.01332	0.18828	0.54250

An analysis of the frequency of SHA events is shown in Fig. 4 where we can see that the majority of all L1 DC accesses are SHA2&4:X accesses (successful speculative accesses) shown in gray. Most prevalent of these are SHA2:1 and SHA4:1 (load and store respectively) where three tag and data arrays or three tag arrays are halted, respectively. Approximately 38% are SHA0 and SHA1 accesses, shown in black, that fall outside of our speculation window, i.e., the displacement is too large. 5% are SHA3 and SHA5 accesses, shown in white, which are speculation failures that in addition to a normal cache access also waste energy in the halt-tag array. Note, however, that no performance penalty is incurred on speculation failures. Around 0.5% of all memory accesses are SHA2:0 and SHA4:0 accesses, which correspond to the majority of the cache misses in the benchmarks; the rest fall into either SHA2:1 or SHA4:1. Even less prevalent are SHA2:2-4 and SHA4:2-4 accesses, which constitute less than 0.15% of all accesses. In Fig. 5 we can see that if we neglect the accesses exceeding our speculation offsets, the SHA approach achieves a halt ratio (ratio of accessed L1 DC ways) of almost 0.25 for all benchmarks, which corresponds to the case where only one way is activated.

The SHA approach results in considerable energy savings as shown in Fig. 6. Across all benchmarks we have an energy

[1]Adapting the SRAM macro width to the halt-tag width would further lower this overhead.

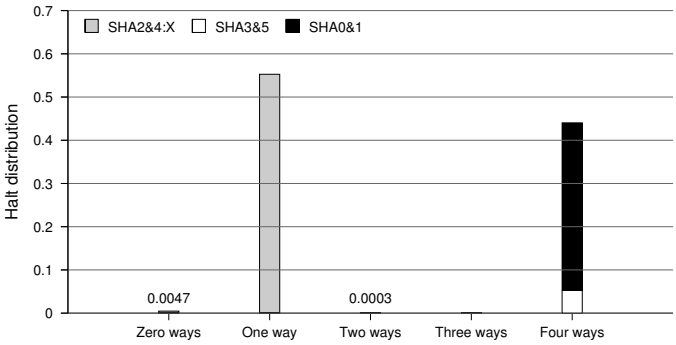

Fig. 4. Distribution of halt accesses where X denotes Zero-Four ways on the x-axis.

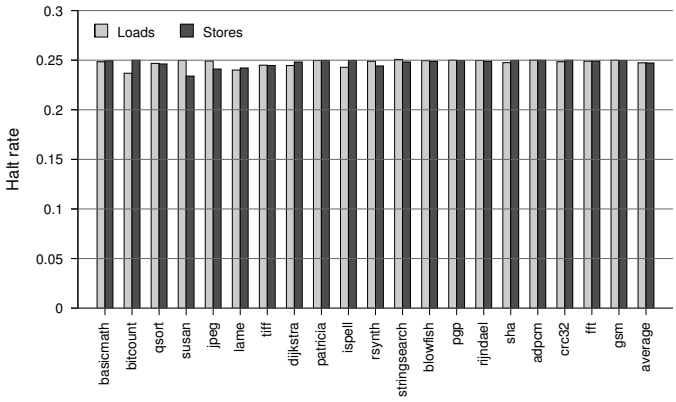

Fig. 5. Halt rates for a halt bit-width of seven bits.

saving of at least 9% with some benchmarks showing energy savings of almost 50%. The average energy saved is 25.6% compared to the baseline and 7.1% compared to the STA technique[2]. In Fig. 7 we compare the SHA approach to the baseline cache and the STA technique. The majority of the savings comes from loads where SHA saves 24.5% energy normalized to the baseline cache, which is 6% better than STA. Energy savings for stores are more modest at 1.2% compared to both the baseline and STA. The SHA approach requires the halt tag to be updated on a miss, which results in higher miss energy than the baseline and STA. However, misses are infrequent in MiBench so this only has a minor impact on the results. In relative numbers, the SHA approach saves 31% of the load energy and 6% store energy compared to the baseline with a slight energy penalty of 5% on misses.

In order to quantify how the speculative nature of SHA affected the energy result we decided to replicate the WHC technique [4]. In contrast to the original WHC evaluation, energy numbers from placed and routed implementations are used instead of CACTI [15]. Several simplifying assumptions had to be used in our replica implementation: 1) We do not have access to the same custom SRAM memories so we replace these with standard SRAM macros in our energy analysis. 2) The original paper uses a cache size of 8kB but we

[2]In contrast to the original STA paper, we are using GCC as compiler. This impacts the speculation statistics and, thus, makes our STA energy reduction slightly smaller than in the original paper.

scale our implementation up to 16kB. 3) Because of limited SRAM sizes we are unable to customize the tag array size causing our WHC replica to suffer the same energy overhead as our SHA approach. 4) A fully associative 128-entry memory was implemented in 65-nm standard cell logic to estimate the halt tags used in the WHC implementation. 5) We do not consider any timing constraints that this design might entail. Simulations put the halt cache at approximately 36% energy saved compared to our baseline cache, only 11% above our SHA approach. As stated previously, we believe the SHA approach has a number of advantages over WHC that makes it much more practical to implement.

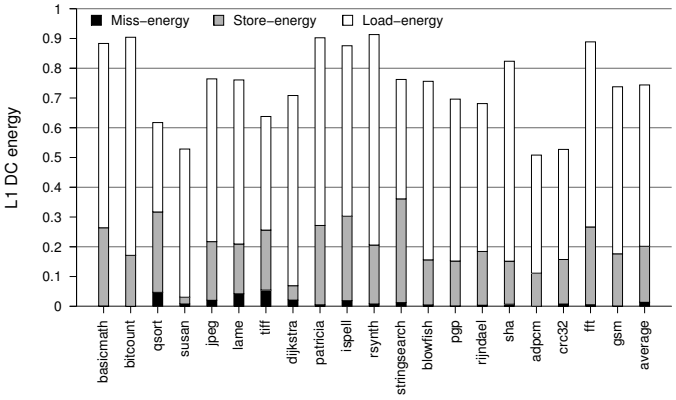

Fig. 6. Total SHA miss, store and load energy.

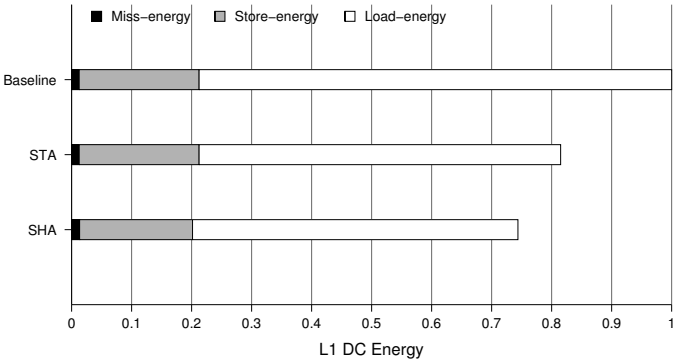

Fig. 7. Average energy distribution for baseline, STA and SHA caches.

As mentioned in Sec. IV, our implementation of SHA suffers from an energy overhead in the tag arrays, which only utilize 75% of the capacity of the SRAM memories. This has a negative impact on our results and in order to quantify this effect, we decided to scale the energy values on the tag arrays to reflect the utilization. Furthermore, we observed that beyond six halt bits only moderate energy gains were achieved. If the halt tag size is reduced to six, the bit-width drops to 75%. By using CACTI to study memory size scaling trends, we estimate this bit-width reduction to scale down the read and write energy by 19%. Simulations put the scaled SHA approach at 33% energy saved on average, which is an additional improvement of 7.4%.

VI. Conclusion

We have proposed a practical way-halting approach that can reduce the energy dissipated in set-associative caches implemented with standard synchronous SRAM memory. Our approach enables the processor to speculatively access a halt tag in the address generation stage instead of the SRAM access stage to eliminate accesses to cache ways that cannot possibly be a hit without causing any execution time penalties. The approach gives an average energy saving of 25.6% compared to an optimized baseline cache. We estimate that significantly larger energy savings are achievable with more narrow (yet standard) halt and tag SRAM macros, which are better adapted to the bit-widths in question.

Acknowledgment

This research was supported in part by the Swedish Research Council grant 2009-4566 and the US National Science Foundation grants CNS-0964413, DUE-1241525, and CCF-1533846.

References

[1] W. J. Dally, J. Balfour, D. Black-Shaffer, J. Chen, R. C. Harting, V. Parikh, J. Park, and D. Sheffield, "Efficient embedded computing," *IEEE Computer*, vol. 41, no. 7, pp. 27–32, Jul. 2008.

[2] K. Inoue, T. Ishihara, and K. Murakami, "Way-predicting set-associative cache for high performance and low energy consumption," in *Proc. Int. Symp. on Low Power Electronics and Design*, Aug. 1999, pp. 273–275.

[3] A. Bardizbanyan, M. Själander, D. Whalley, and P. Larsson-Edefors, "Reducing set-associative L1 data cache energy by early load data dependence detection (ELD^3)," in *Proc. Conf. on Design, Automation and Test in Europe*, Mar. 2014.

[4] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," *ACM Trans. on Architecture and Code Optimization*, vol. 2, no. 1, pp. 34–54, Mar. 2005.

[5] R. Min, Z. Xu, Y. Hu, and W.-b. Jone, "Partial tag comparison: A new technology for power-efficient set-associative cache designs," in *Proc. Int. Conf. on VLSI Design*, Jan. 2004, pp. 183–188.

[6] A. Bardizbanyan, M. Själander, D. Whalley, and P. Larsson-Edefors, "Speculative tag access for reduced energy dissipation in set-associative L1 data caches," in *Proc. IEEE Int. Conf. Computer Design*, Oct. 2013, pp. 302–308.

[7] A. Shafiee, N. Shahidi, and A. Baniasadi, "Using partial tag comparison in low-power snoop-based chip multiprocessors," in *Proc. Annual Int. Symp. Computer Architecture*, 2012, pp. 211–221.

[8] T. Austin, D. Pnevmatikatos, and G. Sohi, "Streamlining data cache access with fast address calculation," in *Proc. Annual Int. Symp. Computer Architecture*, Jun. 1995, pp. 369–380.

[9] *Design Compiler®, v. 2010.03*, Synopsys, Inc., Mar. 2010.

[10] *Encounter® Digital Implementation (EDI), v. 10.1.2*, Cadence Design Systems, Inc., Jul. 2011.

[11] Embedded Microprocessor Benchmark Consortium. [Online]. Available: http://www.eembc.org

[12] *PrimeTime® PX, v. 2011.06*, Synopsys, Inc., Jun. 2011.

[13] T. Austin, E. Larson, and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling," *Computer*, vol. 35, no. 2, pp. 59–67, Feb. 2002.

[14] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "MiBench: A free, commercially representative embedded benchmark suite," in *Proc. Int. Workshop on Workload Characterization*, Dec. 2001, pp. 3–14.

[15] S. Wilton and N. Jouppi, "CACTI: An enhanced cache access and cycle time model," *IEEE J. Solid State Circuits*, vol. 31, no. 5, pp. 677–688, May 1996.