Speculative Tag Access for Reduced Energy Dissipation in Set-Associative L1 Data Caches

Alen Bardizbanyan[†], Magnus Själander[‡], David Whalley[‡], and Per Larsson-Edefors[†]
[†]Chalmers University of Technology, Gothenburg, Sweden
[‡]Florida State University, Tallahassee, USA
alenb@chalmers.se, msjaelander@fsu.edu, whalley@cs.fsu.edu, perla@chalmers.se

***Abstract*—Due to performance reasons, all ways in set-associative level-one (L1) data caches are accessed in parallel for load operations even though the requested data can only reside in one of the ways. Thus, a significant amount of energy is wasted when loads are performed. We propose a speculation technique that performs the tag comparison in parallel with the address calculation, leading to the access of only one way during the following cycle on successful speculations. The technique incurs no execution time penalty, has an insignificant area overhead, and does not require any customized SRAM implementation. Assuming a 16kB 4-way set-associative L1 data cache implemented in a 65-nm process technology, our evaluation based on 20 different MiBench benchmarks shows that the proposed technique on average leads to a 24% data cache energy reduction.**

I. Introduction

Energy efficiency is a very important metric for all types of processors. As the executed applications are becoming more demanding, energy-saving techniques that reduce performance is however not a viable option. The energy dissipation of the level-one (L1) data cache (DC) is significant. For example, the L1 DC energy was shown to be equal to that of the register file and all pipeline registers of an embedded processor even though the L1 DC is much less frequently accessed [1].

Store operations to data caches are usually sequential, since a tag hit signal needs to be generated before modifying any data. This scheme suits store operations, since these rarely create any data dependencies in the pipeline. But load operations, which often have data dependencies with the following instruction, must be carried out in a fast way for performance reasons [2].

Due to the poor locality of the data accesses, associativity is important for data caches in order to achieve a low miss rate [3]. In a set-associative cache, the fast load operation is accomplished by accessing all the tag and data arrays in parallel to ensure synchronous operation with a short critical path. However, this access scheme is energy inefficient because the data needed for the load operation can only reside in one of the ways. Hence, the unnecessary data read operations from the remaining ways waste energy.

We propose a technique that exploits the address generation stage in a processor to make a speculative access to the tag arrays. If the speculation is successful, only a single data array is enabled for load operations. This way the energy overhead of accessing the other data arrays is eliminated. The technique does not incur any execution time penalty and it does not require any customized SRAM implementation. We evaluate the energy efficiency of the technique in the context of a processor with a 16kB 4-way set associative L1 DC, using a physical implementation in a 65-nm process technology.

This research was supported in part by the Swedish Research Council grant 2009-4566 and the US National Science Foundation grants CNS-0964413 and CNS-0915926.

II. Background

In this section we briefly describe the functionality of a conventional set-associative data cache and of the address calculation for load and store operations.

A. Data Cache and Data Translation Lookaside Buffer

Fig. 1 shows a conventional 4-way set-associative data cache (DC), which is virtually indexed and physically tagged (VIPT). The figure also shows a data translation lookaside buffer (DTLB), which is used to convert the virtual page number (VPN) to the physical page number (PPN). VIPT caches are preferred because virtually-tagged caches cause problems related to the use of multiple virtual memory address spaces, such as synonyms and aliasing. In addition, the virtual index makes it possible to access the DTLB and the L1 DC in parallel, which shortens the critical path of the cache access significantly [4], [2].

The L1 DC load access is accomplished using the memory address that is usually calculated before the memory access stage. The memory address, as shown in Fig. 1, has three main parts: line offset, line index, and tag. The tag part contains the VPN, which is used to make a fully-associative search in the DTLB, which in turn outputs the translated PPN on a hit. The line index is used to read the tags from the tag arrays and, together with the line offset, it is used to read the data words from the data arrays. The tags, which contain PPNs, are compared with the PPN coming from the DTLB output and if any of them matches, then a hit signal is generated. This signal drives the final way-select multiplexer that forwards the correct data.

Store operations require the tag comparison to be performed before the data can be written to the correct data array. Since SRAM blocks are usually synchronous, store operations take two cycles: During the first cycle, if there is a tag match,

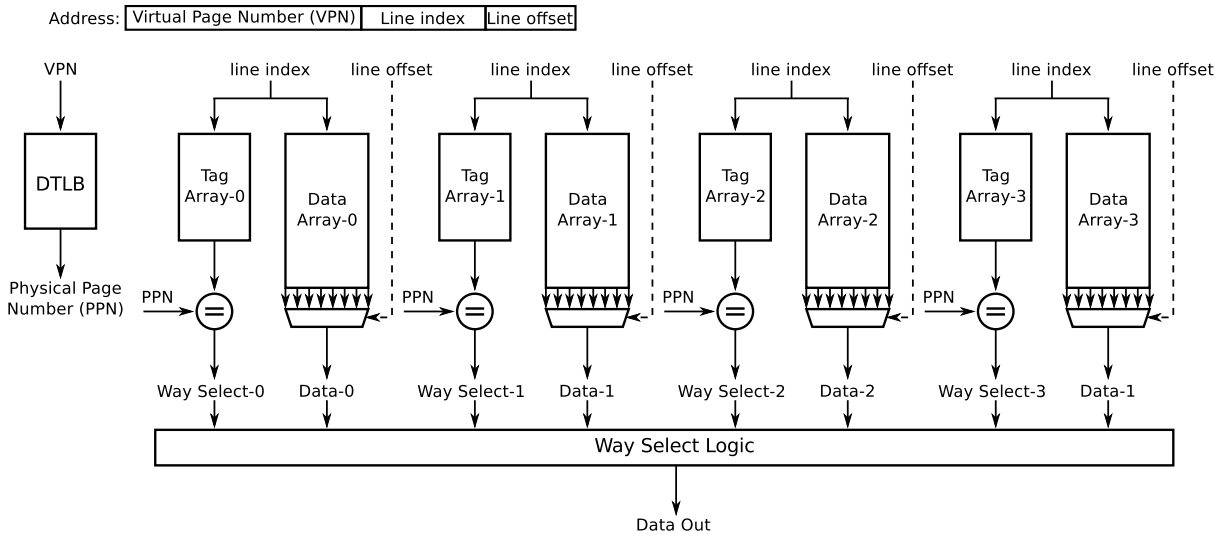

Fig. 1. Overview of a virtually-indexed physically-tagged 4-way set-associative data cache.

a hit signal is generated. During the second cycle, the write operation to the selected data array is accomplished.

In a set-associative cache, in which the tag and data are accessed in parallel for load operations, the energy dissipation of load operations is higher as compared to the energy of store operations. This is due to 1) only one data way being activated and written during store operations, and 2) store operations being much less frequent than load operations. For the 20 MiBench applications used in Sec. IV, on average only 29% of the memory operations are store operations. Consequently, optimizing the handling of load operations will give higher overall L1 DC energy savings.

B. Address Generation and Data Cache Accesses

The memory address for load and store operations is commonly calculated in an address-generator unit (AGU) by adding an offset to a base address. The AGU is usually located one stage before the L1 DC access stage [5]. Fig. 2 shows a segment of a pipeline, in which the AGU is located in the address generation (ADDR-GEN) stage before the L1 DC access stage.

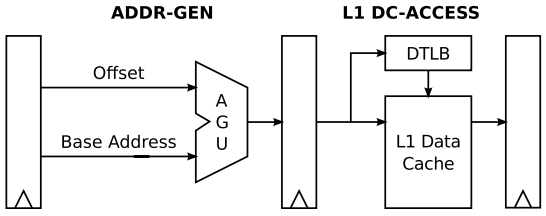

Fig. 2. Segment of a pipeline in which address generation and L1 DC access are in consecutive stages.

Fig. 3 shows the address calculation in a MIPS-like instruction set. A 16b offset is sign extended and added to a 32b base address value. This addition generates the virtual address, which includes tag, line index, and line offset. The tag portion is translated by the DTLB to a physical address in the L1 DC access stage for tag comparison (see Fig. 2).

The bitwidth of the tag, the line index, and the line offset depends on the cache line size and the page size. Throughout this paper, we will use the common configuration of a 32B cache line size and a 4kB page size. In this configuration, the bitwidth of the tag, the line index, and the line offset is 20 bits, 7 bits, and 5 bits, respectively.

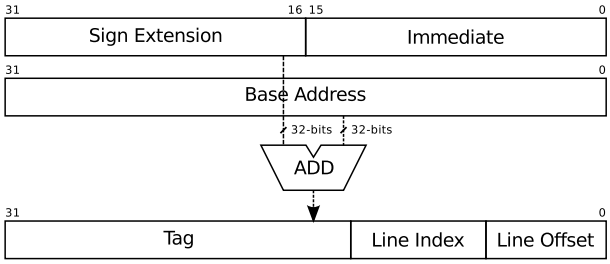

Fig. 3. Data memory address calculation.

III. Speculative Tag Access for Load Operations

In this section we give an overview of the speculative tag access technique in terms of pipeline functionality and implementation. We provide an implementation-oriented discussion on the impact of offset bitwidth on the success of speculations and how speculation successes and failures are detected.

A. Speculative Tag Access in the Pipeline

In order to perform the tag comparison in the L1 DC, the *line index* and the *tag* portions of the memory address are needed. These are used to index the tag memories and access the DTLB. For the conventional address calculation scheme described in Sec. II-B, it has been shown that for most of the address calculations the line index and the tag portions of the base address do not change [6]. The reason for this is that most of the address offsets are narrower than or of the same

size as the line offset. Thus, since carries are not frequently propagated from the line offset to the line index during the address calculation addition, the line index and the tag often remain unaffected. This property of the address generation can be exploited by speculatively comparing tags earlier in the pipeline. This is possible by using 1) the line index of the base address to speculatively access the tag memories and 2) the tag of the base address to speculatively access the DTLB. An overview of the speculation scheme is shown in Fig. 4. The address generation and tag access take place in parallel.

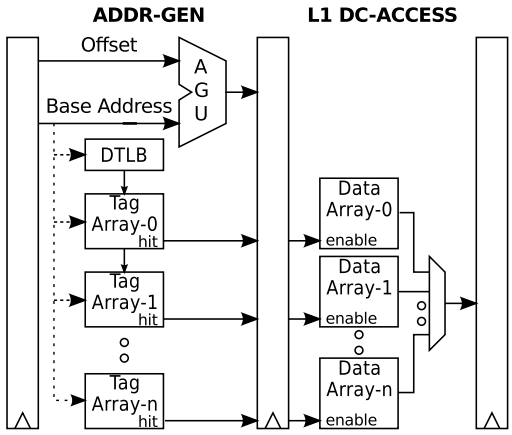

Fig. 4. Speculative access of tag arrays and DTLB in the pipeline.

If the speculation is successful, only one way of the L1 DC data arrays needs to be activated on a hit. In this manner, the energy dissipation from the other data ways will be eliminated. On a speculation failure, the L1 DC will be conventionally accessed on the following cycle. It should be noted that on a speculation failure there is an additional energy overhead in accessing the tag memories a second time. However, most of the speculation failures for tag accesses do not result in a speculation failure for the DTLB. This is because carry propagation in the most significant bits (MSBs) is rare. As a result, when the DTLB speculation is successful while the tag access is not, the DTLB need not be accessed again, and the output can be used on the following cycle. Hence, the energy overhead caused by the DTLB due to speculation failures will be very low.

Since the speculation is attempted earlier in the pipeline, speculation failures do not cause any execution time penalty. In the event of a speculation failure, a subsequent load operation cannot be speculated because the L1 DC will be busy. Our evaluations show that this situation occurs rarely and causes only 0.25% of the successful speculations not to be achieved. An additional small benefit of this speculation scheme is related to cache misses. If a speculation is successful on a load operation that causes a miss in the L1 DC, the miss will be detected earlier in the pipeline. Hence, no data array will be enabled and the access to the next level in the memory hierarchy can be initiated earlier, reducing the execution time penalty related to cache misses by one cycle.

B. Implementation

It is straightforward to speculatively calculate the line index and the tag, if the offset is narrower than or of the same size as the line index. The corresponding parts of the addresses are added as usual. However, it has been shown that if the address offset is wider than the line offset, then it may still be possible to calculate the line index in a very fast way by OR'ing the corresponding parts of the address offset and the base address [6]. If there is no carry propagation in the OR'ed portion, the result of the addition will be correctly generated by using the OR operation. This introduces an OR-gate delay in the critical path. Our energy evaluations (Sec. V-A) show that if positive offsets wider than four bits are included in the speculation, the energy dissipation increases due to a deteriorating ratio of successful and failing speculations. As a result, the OR-gate based fast address calculation scheme is not beneficial for reducing energy. It should be noted that for negative offsets, an additional adder is needed to perform the addition using two's complement of the line offset in order to detect the carry propagation [6].

Our evaluations on the MiBench benchmarks (see Sec. IV) show that only 3.6% of the load operations use a negative offset. Negative offsets up to five bits cover almost half of these cases. Consequently, the speculation scheme is only applied if the load operation has a negative offset, which is not wider than five bits, *or* has a positive offset, which is not wider than four bits. The sign and bitwidth of the address offset can be easily detected by applying AND or OR on the MSB of the immediate in parallel with the sign extension of the immediate that is commonly performed to achieve 32 bit wide offsets.

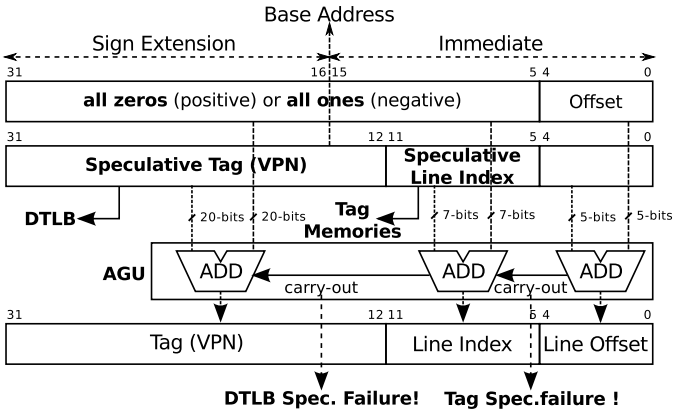

Fig. 5. Speculative address calculation and speculation failure detection.

The line index and tag portion of the base address can be directly supplied to the tag memories and the DTLB, without any additional delays. Speculation failures can be easily detected using only one carry-out from the MSB of the line offset (for tag memory speculation failures) and one carry-out from the MSB of the line index (for DTLB speculation failures). Fig. 5 shows the speculative address generation and the speculation failure detection. The three add operations are not separate, but the internal carry signals have been exposed

in the figure only to illustrate which signals are used for speculation failure detection.

Some processors can use register-based offsets for load operations in addition to immediate-based offsets. A register-based offset can be either read from the register file or it can arrive from the forwarding network. Hence, it might not be possible to detect the offset bitwidths for register-based offsets without affecting the critical path. Due to the lack of early offset bitwidth detection for register-based offsets, speculation should either be attempted for all load operations with register-based offsets or it should not be attempted at all. In this work a processor with a MIPS-like instruction set is evaluated, i.e., immediate-based offsets for load instructions are used.

IV. Evaluation Framework

In this section, we give an overview of the tools and the methodology used to evaluate the energy savings. We also give details about the energy dissipation in a 16kB 4-way set-associative cache implemented in a 65-nm process technology.

A. Energy Estimation Methodology and Tools

In order to perform an energy evaluation, we use 20 different benchmarks (see Table I) across six categories in the MiBench benchmark suite [7]. All benchmarks are compiled with the VPO compiler [8], using the large dataset option.

TABLE I
MiBench benchmarks

Category	Applications
Automotive	Basicmath, Bitcount, Qsort, Susan
Consumer	JPEG, Lame, TIFF
Network	Dijkstra, Patricia
Office	Ispell, Rsynth, Stringsearch
Security	Blowfish, Rijndael, SHA, PGP
Telecomm	ADPCM, CRC32, FFT, GSM

We implemented an RTL description of a 5-stage in-order processor including 16kB 4-way set-associative instruction and data caches [9] with 32B line size. Although a simple pipeline is used for the evaluations, our technique can be applied to more complex pipelines. The RTL implementation is synthesized in Synopsys Design Compiler [10] using a commercial 65-nm low-power process technology with standard cells and SRAM blocks. A multi-V_T design approach is used both in the SRAM blocks and in the logic cells to ensure reasonable performance with low leakage power [11]. In addition, clock gating is used to reduce switching energy when the circuits are idle. The layout work is done in Cadence Encounter [12]. The SRAM blocks and the processor core are placed in a similar manner (see Fig. 6) to a trial layout of a commercial processor [13]. The placed and routed implementation meets the timing constraint for a 400-MHz clock rate, assuming the worst-case process corner, a supply voltage of 1.1 V, and 125°C. The layout is verified using small benchmarks from the EEMBC benchmark suite [14]. We then use the power analysis of Synopsys PrimeTime PX [15] to extract energy

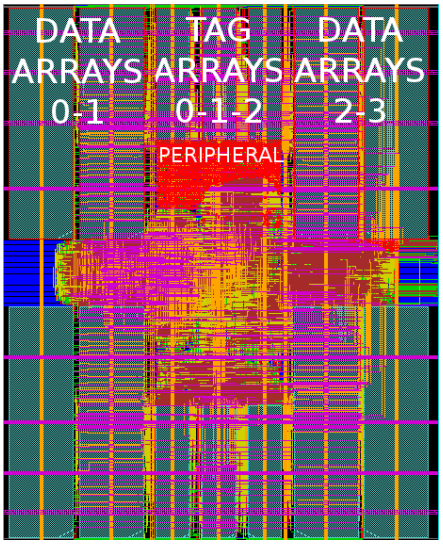

Fig. 6. Layout of a 5-stage in-order processor with 16kB 4-way set-associative instruction and data caches.

values of the L1 DC operations and the static energy from the RC-extracted netlist. The energy values, which are obtained using the nominal process corner, a supply voltage of 1.2 V, and 25°C, are then used in the SimpleScalar simulator [16], which is modified to simulate a 5-stage in-order processor with the same cache parameters as the implemented ones. The access time in terms of cycles assumed for each memory in the hierarchy is 1, 12, 120 for the L1 cache, L2 cache, and main memory, respectively. The simulator calculates the activities (loads, stores, cycles etc.) for the MiBench benchmark suite until completion. The final energy values are calculated from these activities.

B. L1 DC Energy Values

The L1 DC consists of four 1024x32b-m8 4kB SRAM blocks for the data arrays and three 128x32b-m4 512B SRAM blocks for the tag arrays. The reason for using three SRAM blocks for the tag memories is that each tag array consists of 20 bits and one additional bit for valid. As a result, four tag arrays together represent 84 bits, which can be accommodated in three 32b SRAM blocks. The remaining logic is built with standard cells. Table II shows the energy dissipated for reading and writing a single data (Table IIa) and tag (Table IIb) memory. Read and write internal energy is the fixed energy that is dissipated during read and write operations, respectively. Each SRAM block has four main peripheral pin types: Address (A), Output (Q), Data in (D), and Mask (M). Logic transitions that occur on any of the A, D, and M pins cause some internal energy to be dissipated in the SRAM blocks. As far as Q, energy is dissipated when driving an output load. The energy due to switching on the A pins is higher compared to the other peripheral pins. Hence, we simulated the switching using SimpleScalar on the 20 MiBench applications. The A pins on average switch for 34% of all load operations. For the other pins, we assume all of them switch on an operation.

TABLE II
SRAM BLOCK READ AND WRITE ENERGY

1024x32b-m8	Energy (pJ)
Read internal	23.2
A[11:0]	2.05
Q[31:0]	1.1
Clock	0.15
Read Total	26.5
Write internal	22
A[11:0]	2.05
D[31:0]	2.0
M[31:0]	1.0
Clock	0.15
Write Total	27.2

(a) Energy for a single data memory

128x32b-m4	Energy (pJ)
Read internal	17.3
A[6:0]	0.92
Q[31:0]	0.75
Clock	0.13
Read Total	19.1
Write internal	14
A[6:0]	0.92
D[31:0]	2.0
M[31:0]	0.6
Clock	0.13
Write Total	17.6

(b) Energy for a single tag memory

The energy estimation has been done in such a way that the overall energy savings due to the speculation technique are somewhat pessimistic. While the speculation technique mainly saves data array energy, there is also energy dissipated in other parts during a load operation, e.g., in the peripheral circuits. For the peripheral circuits, we assumed each combinatorial node switches during a read or write operation. As a result, the ratio of the data array energy compared to read energy becomes less. The peripheral circuits consist of the least recently used (LRU) replacement unit, the cache controller, and the remaining multiplexers. An arbiter is used during misses to control whether the instruction or the data cache can access the next level in the memory hierarchy. For the DTLB, we used a 16-entry fully-associative translation structure; again we pessimistically calculated the energy, assuming that all combinatorial nodes switch. These energy values are shown in Table III. Using the four equations below we obtain the final energy values, see Table IV.

$$E_{read} = 4\times E_{data\_read} + 3\times E_{tag\_read} + E_{peripheral} \quad (1)$$

$$E_{write} = 1\times E_{data\_write} + 3\times E_{tag\_read} + E_{peripheral} \quad (2)$$

$$E_{miss\_no\_wb} = E_{tag\_write} + 8\times E_{data\_write} + 8\times E_{arbiter} \quad (3)$$

$$E_{miss\_wb} = E_{miss\_no\_wb} + 8\times E_{data\_read} + 8\times E_{arbiter} \quad (4)$$

TABLE III
L1 DC PERIPHERAL AND DTLB ENERGY

Unit	Energy (pJ)
LRU	12.1
Other peripherals	6.7
Arbiter	2.0
DTLB	17.5

TABLE IV
L1 DC ENERGY VALUES FOR MAIN OPERATIONS

Operations	Energy (pJ)
Read	182.1
Write	103.3
Idle (per 2.5-ns cycle) (leakage+clock network)	0.75
Miss without writeback (wb)	251.2
Miss with writeback (wb)	479.1

V. RESULTS

Since the proposed technique is speculative, it is essential to keep the speculation success ratio high, otherwise the energy savings can be very limited. On a speculation success, we eliminate the read energy in three data arrays, that is, 79.5 pJ. On a speculation failure for the tag, there is an extra energy cost of 57.3 pJ; this is the energy required to access all the tag arrays. In addition, a speculation failure for the DTLB costs an extra 17.5 pJ.

A. Address Offset Bitwidth and Energy Savings

Given the energy values for speculation outcomes, it is possible to analyze the energy savings for different address offset bitwidths. We would expect that for larger address offset bitwidths there is a higher chance of carry-out propagation, hence the energy savings will be limited or even nonexistent. Note that in order to speculate using address offsets that are wider than the line offset, which is five bits for our cache design, the OR-based fast address calculation is needed as explained in Sec. III. We found that the OR-based speculation usually has a very low success ratio improvement over just using the speculative line index.

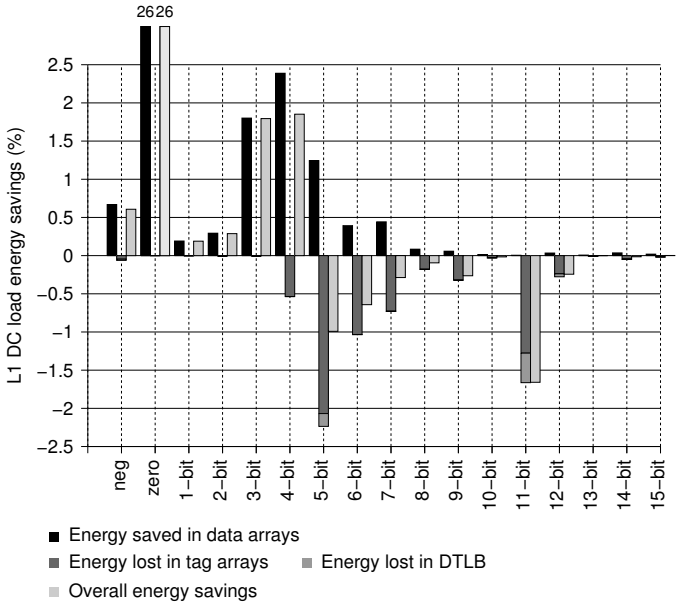

Fig. 7. The impact of address offset bitwidth on L1 DC load energy.

Fig. 7 shows, for different address offset bitwidths, how much L1 DC load energy is saved on speculation successes and how much energy is lost on failures. The figure also shows the overall L1 DC load energy reduction. The energy values are normalized to the total energy dissipated by load operations in the L1 DC, since the speculation scheme is only applied for the load operations. *neg* denotes the negative offsets up to five bits, and *zero* denotes the offsets that are zero. For the positive offsets that are wider than four bits, it is not possible to achieve any overall energy savings, so speculation should not be attempted under these circumstances. This is expected as a 32B cache line size results in a 5b line offset and address

offsets wider than four bits will therefore have a high chance of causing carries to propagate to the line index. The greatest energy reduction is associated with the zero offsets (26%) because they cannot cause speculation failures and because zero-offset load operations are common. For the selected offset range (negative 5b to positive 4b offsets), the total load energy is reduced by 30.7% as shown in Fig. 8.

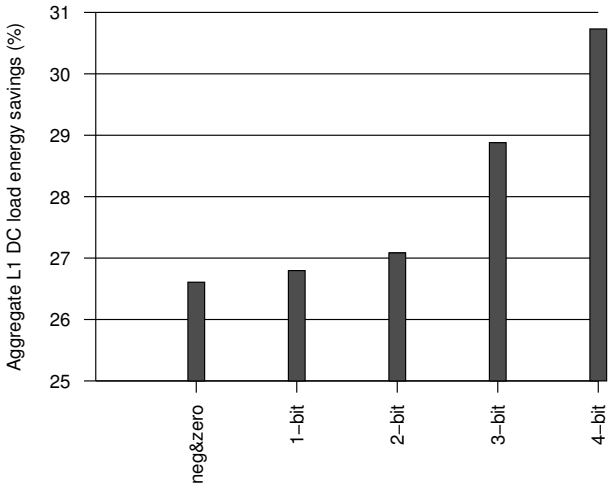

Fig. 8. Aggregate energy savings for different ranges of address offset bitwidths used for speculation. For example, "neg&zero" represents all negative offsets and the zero offset, while "1-bit" represents the aggregate of "neg&zero" and the positive 1b offset.

Fig. 9 shows the proportion of load operations that correspond to speculation success and failure, for the selected offset range of negative five bits to positive four bits. On average, speculation is used for 73.8% of the total number of load operations, since 26.2% of the loads have an offset outside the selected range. The speculation has a 97% success ratio, which translates into having 71.9% of all load operations accessing the tags early. Only 1.9% of the load operations cause speculation failures in the tag memories, and 0.2% cause speculation failures in the DTLB. Due to the carry propagation in the address calculation, a DTLB access failure always occurs together with a tag access speculation failure.

B. Overall L1 DC Energy Savings

Fig. 10 shows the overall L1 DC energy savings, which depend on the ratio of load energy to the total L1 DC energy, i.e., the sum of idle, load, store, and miss energy. It should be noted that the relation between idle, store, and miss energy in the L1 DC that uses our speculation technique is the same as in a conventional L1 DC. The presented load energy accounts for the total energy reduction resulting from our speculation technique, including the energy saved in data arrays and the energy lost in tag arrays and DTLB. On average, the L1 DC energy is reduced by 24%.

Detecting cache misses earlier, thanks to speculation successes on L1 DC misses, improves execution time by 0.1%. This is expected considering that only one cycle of a 12-cycle miss penalty is saved. Thus, this performance improvement has a negligible impact on the idle energy. Nevertheless, this is an optimization that the speculative tag access approach enables.

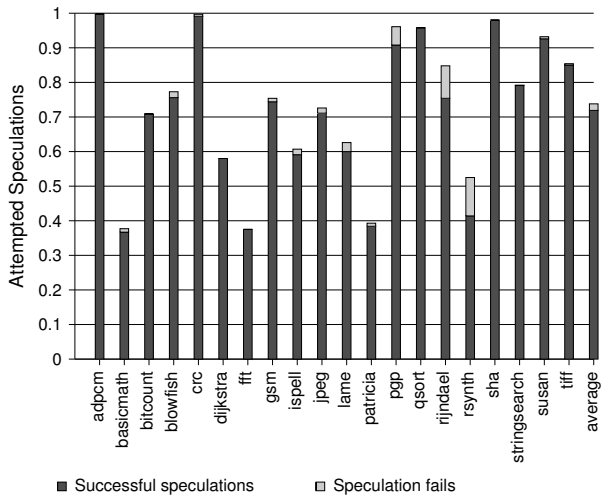

Fig. 9. Speculation successes and failures as compared to the total number of load operations.

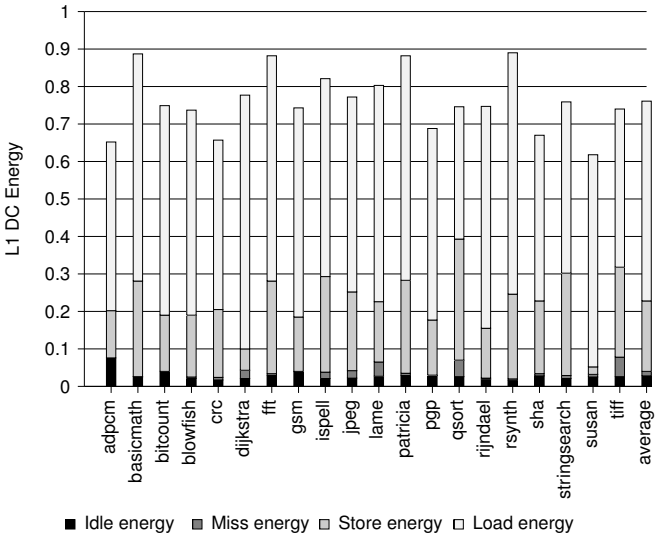

Fig. 10. Total energy of an L1 DC using the new speculation technique. The energy shown is normalized to a conventional L1 DC.

VI. Related Work

Inoue *et al.* propose a way-prediction technique to access only one tag and data array of a set-associative cache [17]. Using a table, which is addressed with the line index, this technique accesses the last way that was a hit for a given line index. This technique also depends on the possibility of calculating the line index earlier in the pipeline. Since the cache access happens in the L1 DC access stage, there is a one-cycle execution time penalty for each prediction failure. Even though this technique works relatively well with instruction caches, it increases the data cache access time by 12.5%. Our technique does not incur any execution time penalty.

Nicolaescu *et al.* propose to use a 16-entry fully-associative table to store way information of the last accessed lines [18]. The table is accessed in the stage before the data cache access, with a fast address calculation scheme based on Alpha 21264. If there is a match in the table, only one data cache way is accessed. The addition of the 16-entry fully-associative circuit, which has a complexity similar to a DTLB, incurs a significant area and power overhead. This additional hardware will cancel out some of the energy benefits. Our technique is less invasive as it only requires a few additional logic gates.

Fang *et al.* propose to exploit software semantics to eliminate unnecessary accesses to data ways [19]. The MSB of the virtual address is stored together with the physical address for each cache line. This bit is compared with the MSB of the virtual address in parallel with the address translation during a load operation. If there is a mismatch, the discharge of the SRAM cells and the sense-amplifier operations of the corresponding data arrays are disabled. The disadvantages are that the technique requires a customized SRAM implementation and that the table read and MSB bit comparison must be done in parallel with the precharge and decode of the SRAM arrays, making the technique dependent on technology and implementation style. In addition, handling of the virtual memory system gets complex. Our technique does not require any customized SRAM implementation and does not affect the handling of the virtual memory system.

Zhang *et al.* propose storing the least significant four bits of each tag for all the ways in the cache, in a so-called halt tag [20]. During a cache access, for each way a fully-associative search is done in the halt tag arrays. If there is no match, the discharge of the bitlines and the sense-amplifier operation are halted. This technique also requires a customized SRAM implementation. Furthermore, the critical path of the halt signal goes through a relatively big fully-associative memory that should complete its operation before the line index decoding, making the technique impractical.

Austin *et al.* propose to use fast address calculation to access the entire L1 DC earlier in the pipeline, to reduce execution time by resolving the data dependencies earlier [6]. They use OR-gates to speculatively compute the tag and line index. Data arrays are accessed assuming that the column multiplexing of the SRAM blocks are not on the critical path. Hence, the addition operations for the line offset can be done in parallel with the set index decoding and applied to the column multiplexers. While this technique improves performance, it increases the L1 DC energy since the whole cache needs to be accessed again on speculation failures. Furthermore, it requires a customized SRAM implementation to apply column multiplexing asynchronously. The column multiplexing factor is determined by speed and area requirements, hence it might not be as wide as the line offset, which will reduce the proportion of successful speculations.

VII. Conclusion

We have proposed a low-overhead speculation technique that significantly reduces the energy dissipated in set-associative L1 data caches. Our speculative tag access scheme enables a processor to serially access a set-associative L1 DC for improved energy efficiency, without causing any execution time penalty. The technique can be applied to any type of L1 DC; either custom designed or synchronous SRAM block based. The technique on average reduces the L1 DC energy by 24%, at almost no additional hardware cost.

The benefit of the speculative tag access proportionally increases with increasing cache line size, since the relative size of the tag arrays compared to the data arrays will become smaller. In addition, the line offset will become wider so there will be less carry-out propagation to the line index portion, which leads to a higher proportion of successful speculations.

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] C. McNairy and D. Soltis, "Itanium 2 processor microarchitecture," *IEEE Micro*, vol. 23, no. 2, pp. 44–55, 2003.

[3] A. Milenkovic, M. Milenkovic, and N. Barnes, "A performance evaluation of memory hierarchy in embedded systems," in *Proc. 35th Southeastern Symp. on System Theory*, Mar. 2003, pp. 427–431.

[4] M. Cekleov and M. Dubois, "Virtual-address caches. Part 1: problems and solutions in uniprocessors," *IEEE Micro*, vol. 17, no. 5, pp. 64–71, 1997.

[5] *Enabling Mobile Innovation with the Cortex™-A7 Processor*, TechCon, White Paper, ARM, Oct. 2011.

[6] T. Austin, D. Pnevmatikatos, and G. Sohi, "Streamlining data cache access with fast address calculation," in *Proc. 22nd Annual Int. Symp. on Computer Architecture*, Jun. 1995, pp. 369–380.

[7] 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.

[8] M. E. Benitez and J. W. Davidson, "A portable global optimizer and linker," in *ACM SIGPLAN Conf. on Programming Language Design and Implementation*, Jun. 1988, pp. 329–338.

[9] V. Saljooghi, A. Bardizbanyan, M. Själander, and P. Larsson-Edefors, "Configurable RTL model for level-1 caches," in *Proc. IEEE NORCHIP Conf.*, Nov. 2012.

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

[11] L. Wei, Z. Chen, M. Johnson, K. Roy, and V. De, "Design and optimization of low voltage high performance dual threshold CMOS circuits," in *Proc. Design Automation Conf.*, Jun. 1998, pp. 489–494.

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

[13] T. R. Halfhill, "ARM's midsize multiprocessor," *Microprocessor*, Oct. 2009.

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

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

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

[17] 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.

[18] D. Nicolaescu, B. Salamat, A. Veidenbaum, and M. Valero, "Fast speculative address generation and way caching for reducing L1 data cache energy," in *Proc. Int. Conf. on Computer Design*, Oct. 2006, pp. 101–107.

[19] Z. Fang, L. Zhao, X. Jiang, S.-l. Lu, R. Iyer, T. Li, and S. E. Lee, "Reducing L1 caches power by exploiting software semantics," in *Proc. ACM/IEEE Int. Symp. on Low Power Electronics and Design*, Jul. 2012, pp. 391–396.

[20] C. Zhang, F. Vahid, J. Yang, and W. Najjar, "A way-halting cache for low-energy high-performance systems," *ACM Trans. Archit. Code Optim.*, vol. 2, no. 1, Mar. 2005.