Designing a Practical Data Filter Cache to Improve Both Energy Efficiency and Performance

ALEN BARDIZBANYAN, Chalmers University of Technology
MAGNUS SJÄLANDER and DAVID WHALLEY, Florida State University
PER LARSSON-EDEFORS, Chalmers University of Technology

Conventional Data Filter Cache (DFC) designs improve processor energy efficiency, but degrade performance. Furthermore, the single-cycle line transfer suggested in prior studies adversely affects Level-1 Data Cache (L1 DC) area and energy efficiency. We propose a practical DFC that is accessed early in the pipeline and transfers a line over multiple cycles. Our DFC design improves performance and eliminates a substantial fraction of L1 DC accesses for loads, L1 DC tag checks on stores, and data translation lookaside buffer accesses for both loads and stores. Our evaluation shows that the proposed DFC can reduce the data access energy by 42.5% and improve execution time by 4.2%.

Categories and Subject Descriptors: B.3.2 [**Design Styles**]: Cache Memories

General Terms: Energy efficiency, Performance Improvement, Data Cache Design

Additional Key Words and Phrases: Speculation, filter cache

ACM Reference Format:
Bardizbanyan, A., Själander, M., Whalley, D., and Larsson-Edefors, P. 2013. Designing a practical data filter cache to improve both energy efficiency and performance. ACM Trans. Architec. Code Optim. 10, 4, Article 54 (December 2013), 25 pages.
DOI: http://dx.doi.org/10.1145/2555289.2555310

1. INTRODUCTION

The performance demanded for computing continues to escalate as computer systems become more pervasive. As a result, the electricity cost and the environmental impact for computing infrastructure are increasing at an alarming rate. Energy efficiency is of critical importance today for both embedded and general-purpose computing [Hennessy and Patterson 2011]. Clearly it is vital that mobile devices efficiently run their increasingly complex applications as these devices are depending on power supplies with limited capacity. For general-purpose processors, clock rates and per-core performance are now constrained by thermal limitations. Energy-efficient solutions may allow more of the processor power budget to be spent on performance improvements [Huang et al. 2011]. Thus, the design solutions we pursue must be able to reconcile high energy efficiency with high performance.

This work is supported by the National Science Foundation, under grant CNS-0964413 and grant CCR-0915926, and by the Swedish Research Council, under grant 2009-4566.
Author's addresses: A. Bardizbanyan (corresponding author) and Per Larsson-Edefors, Computer Science and Engineering Dept., Chalmers University of Technology, 412 96 Gothenburg, Sweden; email: alenb@chalmers.se; M. Själander and D. Whalley, Computer Science Dept., Florida State University, Tallahassee, FL 32306-4530, USA.

DOI: http://dx.doi.org/10.1145/2555289.2555310

Recent studies show that servicing data accesses through a Level-1 Data Cache (L1 DC) and a Data Translation Lookaside Buffer (DTLB) account for up to 25% of the total power of an embedded processor [Dally et al. 2008; Hameed et al. 2010]. One technique for reducing processor energy dissipation is the use of filter caches that are tiny caches that are accessed before the L1 caches [Kin et al. 1997, 2000]. Since a filter cache is a smaller structure than the L1 DC, there is less capacitance to switch on an access. Thus, accessing a Data Filter Cache (DFC) instead of the L1 DC reduces energy usage. However, filter caches are generally considered impractical by performance-driven processor manufacturers as they typically incur an execution time penalty for each filter cache miss. Furthermore, prior filter cache designs that propose to transfer an entire cache line between the L1 DC and the DFC in a single cycle cause an increase in both the area of the L1 DC and the energy to access it.

While several different techniques associated with the L1 DC have been proposed to either reduce the data access energy dissipation *or* improve processor performance [Inoue et al. 1999; Zhang et al. 2005; Powell et al. 2001; Nicolaescu et al. 2006; Austin et al. 1995], we propose a practical DFC design that makes the processor both more energy efficient *and* faster. This article makes the following contributions: First, we describe practical techniques for accessing a small DFC early in an in-order pipeline that both avoid the DFC miss penalty and can improve performance by eliminating load hazards on DFC hits. Second, we recognize that filling a DFC line in a single cycle has a negative impact on L1 DC area and power, and we describe an approach for transferring a cache line from the L1 DC to the DFC over multiple cycles without incurring any execution time overhead. Third, we demonstrate that our DFC design not only eliminates many L1 DC and DTLB accesses when data is accessed from the DFC, but that it also eliminates a substantial fraction of the L1 DC tag checks and DTLB accesses when storing data to the L1 DC. Finally, we provide a more detailed DFC implementation study as compared to prior filter cache research and show that using a standard-cell DFC can be a practical approach to realizing an efficient DFC.

The remainder of this article is organized as follows: First, we review a typical design of an L1 DC. Second, we describe our proposed DFC design and also show that DTLB accesses can be eliminated on DFC load hits, discuss an approach for accessing the DFC earlier in the pipeline, describe a technique for efficiently transferring an entire line between the L1 DC and the DFC over multiple cycles, and present techniques for making the majority of stores to the L1 DC more efficient. Third, we outline our evaluation framework and present the results of our DFC design. Finally, we review related work on improving data access efficiency and summarize our future work and conclusions for the article.

2. BACKGROUND ON L1 DC DESIGN

Memory operations are commonly accomplished by first performing an address calculation that consists of adding an offset to a base address (displacement addressing mode). The calculated address is then used to access the L1 DC. The address calculation is often computed by an Address Generation Unit (AGU) that is placed before the L1 DC in the pipeline. This approach is illustrated for a segment of the pipeline in Figure 1, where the address calculation is performed in the address generation (ADDR-GEN) stage before the L1 DC access (L1 DC-ACCESS) stage.

The calculated address can be divided into three parts. The line offset is used to identify the word/half-word/byte to be accessed within a cache line, the line index is used to index into the tag and data arrays, and the tag is used to verify if the specified memory item resides in the cache.

An L1 DC access starts with reading out the tag(s) (and data for loads in performance-oriented cache designs). The line index is therefore on the critical path, as it is used to

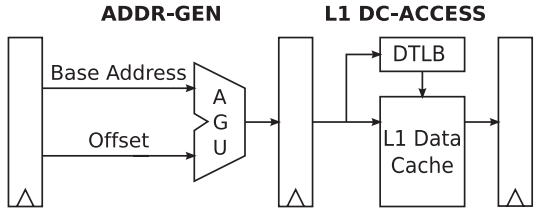


Fig. 1. Segment of a pipeline where address generation is performed in the stage before the L1 DC access.

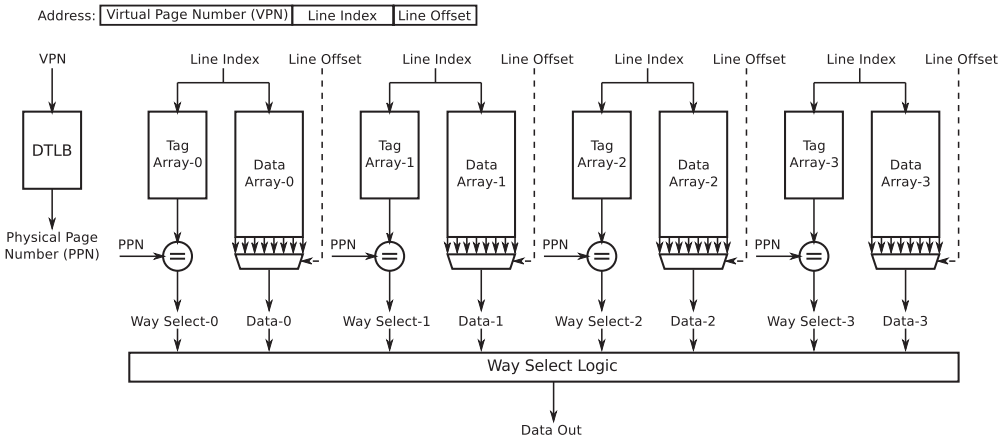


Fig. 2. Overview of a virtually-indexed physically-tagged four-way set-associative data cache.

index into the tag (and data) arrays. Furthermore, as most memories are synchronous, it is not possible to modify the address before the memories are accessed within a single pipeline stage. To avoid additional delays, the line index is commonly part of the page offset and therefore is not translated from the virtual address space.

The entire tag or at least part of the tag is commonly translated to the physical address space as virtually tagged caches cause synonyms and aliasing problems or are required to be flushed on a context switch [Cekleov and Dubois 1997]. The virtual-to-physical translation is performed by a Data Translation Lookaside Buffer (DTLB) that translates a Virtual Page Number (VPN) to a Physical Page Number (PPN). The DTLB access can be performed in parallel with the access to the tag array(s), which significantly shortens the critical path. The translated tag is then compared with the tag(s) read from the tag array(s) to determine if the specified memory item resides in the cache.

Figure 2 shows a four-way set-associative cache where the size of each way is equal to the page size, which results in the tag being the same size as the VPN. The cache is Virtually Indexed and Physically Tagged (VIPT) as discussed earlier. To limit the impact of load hazards, the tag and data arrays are accessed in parallel on load operations. The correct data is then selected by the way-select logic, based on the way select (hit) signal from the tag comparisons. In contrast, store operations are performed across multiple cycles, as the tag comparisons need to be completed before the store to the correct data array can be performed. Load operations are, therefore, performed in less time than store operations at the cost of accessing all the data arrays in parallel.

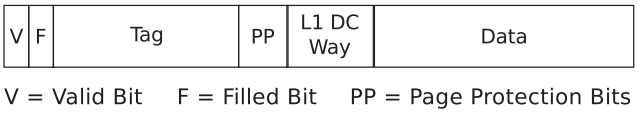


Fig. 3. Information associated with a DFC line.

3. A PRACTICAL DFC DESIGN

In this section, we describe our proposed design for an energy-efficient DFC that also provides a performance improvement. Some principles of the DFC in this work have been previously presented [Bardizbanyan et al. 2013b]; however, that preliminary study targets a very limited number of DFC configurations, does not address other issues related to the practical DFC implementation, and does not provide any energy evaluation.

Our DFC design utilizes virtual tags, employs a write-through policy, accesses the DFC early in the pipeline, supports efficient filling of DFC lines, and allows for more efficient stores to the L1 DC. Figure 3 shows the components of a DFC line in our design. The filled bit (F) indicates if the line has been filled. The Page Protection bits (PP) are copied from the DTLB upon allocating the DFC line. We also identify, for each DFC line, the L1 DC way in which the corresponding L1 DC line is stored.

3.1. Utilizing a Virtually Tagged DFC

Our proposed DFC is accessed using complete virtual addresses, which implies storing virtual tags in the DFC. There are two main advantages of using virtual tags for a DFC. First, the data access energy is reduced as DTLB lookups are not required for DFC accesses. Second, the DTLB is removed from the critical path when accessing the DFC, which is useful for accessing the DFC in parallel with the memory address generation, as described in Section 3.3.

Using virtual caches leads to some potential problems, all of which are cheaper to address in a small DFC, as opposed to a much larger L1 DC:

(1) Multiple different virtual addresses can map to the same physical address. These duplicate addresses, referred to as synonyms, can cause problems after stores as the processor needs to avoid having different values associated with the same location. This problem does not appear on Direct-Mapped (DM) caches because two synonym cache lines will always evict each other in a direct-mapped cache due to the lines having the same line index. Since we also will evaluate Fully Associative (FA) DFCs, we propose a method in order to handle the synonym problem for these DFCs, in which the problem cannot be inherently avoided. Our fully associative DFC design uniquely identifies for each DFC line the L1 DC line that contains the same memory block of data. The L1 DC way associated with each DFC line is explicitly retained, as shown in Figure 3. Note that the L1 DC index need not be explicitly stored, as it is equivalent to the Least Significant Bits (LSBs) of the DFC tag because the L1 DC is virtually indexed. Furthermore, all DFC lines also reside in the L1 DC. Thus, when a DFC line is replaced, the L1 DC index and way for the other DFC lines are compared to the replacement line's L1 DC index and way and a synonym DFC line is invalidated if it exists. Note that this check is only required on DFC line replacements and the small number of DFC lines limits the overhead of using such an approach.
(2) A single virtual address may map to multiple physical locations in different virtual address spaces, which are referred to as homonyms. This homonym problem can be resolved by simply invalidating all the cache lines in the DFC on context switches, which is possible because we employ a write-through policy for the DFC that always

keeps the L1 DC up to date with the latest data. Hence, the invalidation can happen in a single cycle. The overhead of invalidating the cache lines in the DFC is negligible because the DFC is small, context switches are infrequent, and it is highly likely that all cache lines in the DFC would be evicted anyway before switching back to the same application. To handle interrupts, the DFC can simply be disabled and service all accesses from the L1 DC. Though the DFC is disabled, it needs to be kept up to date with writes, which can be done using the same information for detecting synonyms as previously described. On a write, the index and way are checked for all the cache lines in the DFC, and if a match is found, then the data in the DFC is updated.

(3) The DTLB contains information regarding page protection. This protection information from the DTLB can be copied to the DFC when a DFC line is replaced. The overhead of storing this information is insignificant due to the small number of DFC lines.

(4) To support multiprocessor cache coherency, the DFC is strictly inclusive with respect to the L1 DC. When an L1 DC cache line is evicted due to the replacement on a cache miss or due to a cache coherency request, the index and way of the cache line are checked against those stored in the DFC. If a match is found, the cache line in the DFC is also evicted. No modifications are required of existing cache coherency protocols, which makes the DFC compatible with any multiprocessor configuration.

3.2. Employing a DFC Write-Through Policy

We propose a DFC that uses a write-through policy, which ensures that the L1 DC is always consistent with the DFC and has the latest up-to-date data. Using this policy simplifies the use of a DFC because if the DFC cannot be accessed, then the L1 DC can instead service the request. The use of a write-through policy requires that the L1 DC is accessed for every store instruction. However, stores are less frequent than loads, so if we can optimize the loads, the overall design may be improved. Furthermore, a DFC can be used to more efficiently perform the majority of the writes to the L1 DC. The tag check can be avoided by storing in the DFC the L1 DC way that the cache line resides in (see Section 3.5).

There are also disadvantages of using a DFC write-back policy. A DFC write-back policy would cause a DFC miss penalty when a dirty line is replaced because it would take several cycles to write back the dirty line before the desired line in the L1 DC can be accessed. As mentioned earlier, we propose to store the L1 DC way in the DFC to optimize store energy. A write-back policy would, therefore, only save energy for the case where there are more writes to the cache line before the line is evicted from the DFC than the number of writes required to write back the dirty line. Otherwise, the energy expenditure will increase. As described in the previous section, the proposed DFC is virtually tagged. If a write-back policy would be employed, then all dirty lines in the DFC would have to be written back on a context switch, which would incur high overheads. A DFC write-back policy would also complicate DFC line evictions due to L1 DC line evictions. An evicted dirty DFC line would have to be read from the DFC instead of simply invalidating it. Likewise, multiprocessor cache coherency would be more complicated.

A DFC write-through policy has less impact on performance, it is less complex to implement (which might improve timing), and it has the potential to achieve most (or even more) of the store energy benefits of a write-back policy. In contrast, a write-back policy would only reduce the store energy for the case where there are a large number of writes to a line before it is evicted. Thus, we have chosen to only evaluate the DFC with a write-through policy.

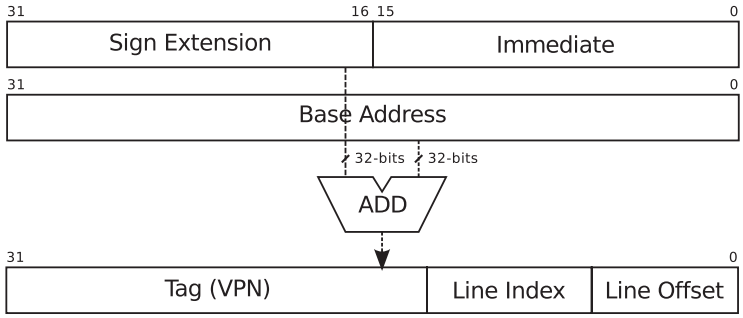


Fig. 4. Address calculation for MIPS-like instruction set.

3.3. Accessing the DFC Early in the Pipeline

If the DFC can be accessed earlier in the pipeline than the L1 DC, then a DFC can be used with no performance penalty. Assume that a DFC can be accessed in the address generation stage and that a write-through policy is utilized between the DFC and L1 DC. If there is a DFC miss, then the DFC miss penalty can be avoided because the L1 DC can be accessed directly in the pipeline stage after performing the address generation. If there is a DFC hit, then performance is potentially improved by avoiding load hazards normally associated with L1 DC hits. The performance penalty of a DFC can also be avoided if only the tags of the DFC are accessed early, since the miss will be detected earlier [Duong et al. 2012]. But in order to improve the execution time, the data also needs to be accessed early.

In order to do the tag comparison in the DFC, the line index and the tag values of the memory address are needed. For a conventional address calculation scheme (see Figure 4 for a MIPS-like address calculation), where an offset is added to a base address, it has been shown that for most of the address calculations the line index and the tag portion of the base address do not change since most of the address offsets are narrower than or the same size as the line offset [Austin et al. 1995]. Thus, since carries are not frequently generated from the line offset to the line index during the address calculation addition, the line index and the remaining Most Significant Bits (MSBs) often remain unaffected. This property of the address generation can be exploited by speculatively comparing tags earlier in the pipeline.

We propose to speculatively access the DFC in the address generation stage when the value of the offset does not exceed what can be represented in the line offset of the address. We perform speculative accesses only for load operations, since load misses are more likely to stall the pipeline due to data dependencies. If a processor configuration cannot hide the latencies related to stores, then speculation can also be used for the store operations.

Figure 5 shows how a filter cache can be speculatively accessed in the same pipeline stage that the memory address is generated. The figure shows conceptually how a successful speculative access can be verified by comparing the base tag and index with the computed tag and index of the memory address. In reality, it is enough to verify that the offset portion of the address is smaller than the line size and that during the address calculation there is no carry out from the line offset. This assures that the tag and index of the base address has not changed, as shown in Figure 6. Our timing evaluation (see Section 5.2) shows that it is possible to access the DFC in a single cycle.

The speculation is attempted for the bitwidth range of negative five bits to positive four bits. The analysis for selecting this range is described in prior work [Bardizbanyan et al. 2013a]. For the given bitwidth range, we found that a speculative DFC access was

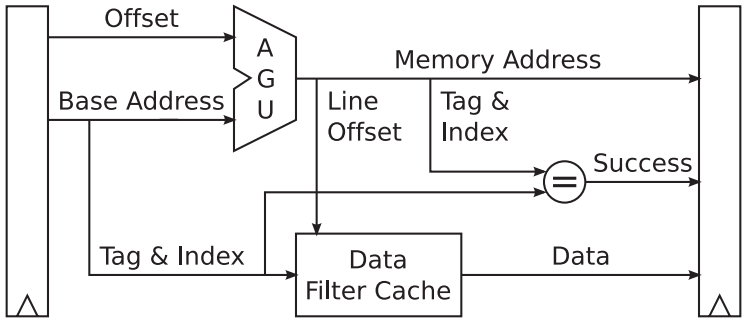


Fig. 5. Speculative data filter cache access.

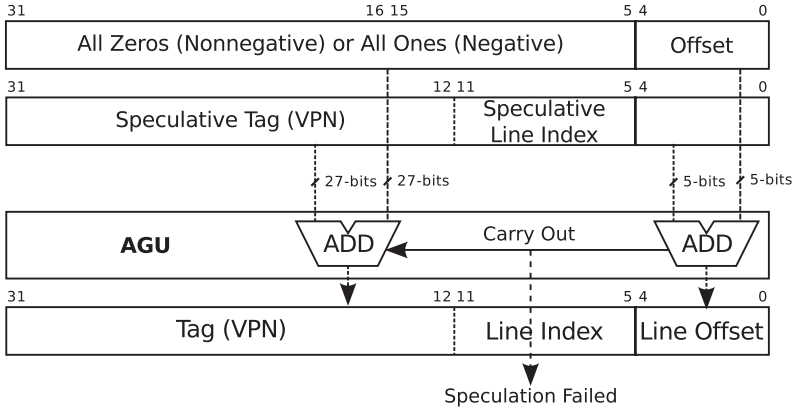


Fig. 6. Technique for detecting if the tag and index are not modified. The AGU is for illustrative purposes shown as two separate adders.

attempted for 73.8% of the load operations and that the speculation success rate (no carry out) for these accesses was 97% for our benchmark suite. Thus, 71.9% of all loads achieve a successful speculative access to the DFC. When a successful speculative DFC access does not occur, either because the offset was too large or because speculation was not successful (carry out), then there are two remaining alternatives: (1) The L1 DC is directly accessed in the following stage to avoid any additional delays. (2) The DFC is accessed in the following stage, which can provide energy benefits when there is a DFC hit and a one-cycle miss penalty when there is a DFC miss. Thus, these two alternatives present us with a tradeoff between improving energy efficiency and degrading performance. Note, however, that both alternatives give better performance than the conventional approach of always accessing the DFC in the L1 DC access stage.

3.4. Supporting Efficient Filling of DFC Lines

Since a DFC requires reading an entire line from the L1 DC when a DFC line is replaced, a relevant issue is how to read the entire line in an efficient manner. Prior filter cache studies have either explicitly stated or implied that a filter cache line can be filled in a single cycle [Kin et al. 1997, 2000; Tang et al. 2001; Hines et al. 2007, 2009; Duong et al. 2012]. Such an approach is not a viable implementation solution when integrating a filter cache with an L1 cache. First, reading an entire line in a single cycle would require a redesign of the L1 DC, which in turn could require a redesign of the whole memory hierarchy. Reading a larger bitwidth from the relatively small SRAM

Table I. Normalized Area and Energy Overheads for Wide Access 4kB SRAM Blocks

SRAM Block [rows × columns]	Area	Read Energy per Access	Total Line Fetch Energy
128 × 256b	1.91	4.15	1.43
256 × 128b	1.30	2.87	1.26
1024 × 32b	1.00	1.00	1.00

blocks that are used in L1 caches can have a significant area overhead. Since L1 caches occupy a significant portion of a processor core, increasing the area of these structures is very undesirable. In addition to the area problem, reading wider bitwidths increases the wire count and creates congestion, which deteriorates the layout efficiency. Second, L1 caches need to be fast structures to improve processor performance and a larger cache area due to wide data ports can have a negative impact on the L1 cache access time. Third, a fast set-associative L1 cache, in which a large number of bits are read in one cycle, will have a significant energy access overhead. The L1 DC should still be optimized with respect to the access of individual data items due to the relatively high DFC miss rate and the use of a DFC write-through policy.

Table I shows the area and energy overhead of using two 65nm SRAM blocks that provide wider access bitwidths than the reference case of a single 32b word. The energy dissipation is given for one line fetch operation (assuming a 32B line size) that is normalized to the SRAM block 1024x32b, in which a single word can be read in each cycle. Assuming a 16kB L1 DC that is four-way set associative, four tag accesses are required before the line can be fetched. Completing the line fill then takes one, two, and eight cycle(s) for the 128×256b, 256×128b, and 1024×32b SRAM memory, respectively.

The column count clearly has a stronger influence on overall area and energy than the row count has. This is due to the replication of double bitlines, bitline conditioning circuits, multiplexing, and so forth, which is more resource demanding than adding rows that entails expanding the address decoder and adding single wordlines. Thus, for the same SRAM block size, area and energy per access are lower for a tall and narrow memory, than for a short and wide one. Thus, an L1 DC in which a single 32b word can be read in a cycle is preferred for both area and energy efficiency. For example, the trial layout of an ARM Cortex-A5 processor has L1 DC SRAM blocks in which only a single word can be read in one cycle [Halfhill 2009]. Since a significant fraction of the data memory references still access the L1 DC even when a DFC is present, the DFC implementation needs to adapt to the L1 DC configuration, not the other way around.

One of the main focuses in this research is to design a data filter cache that does not require any modifications to the L1 data cache, which is designed for area and energy efficiency. A DFC based on standard cells will have at least one read and one write port because latches or flip-flops are used. During line fill operations, we use a critical-word-first fill strategy that starts from the word that causes the miss [Hennessy and Patterson 2011]. This strategy is appropriate, since the word that is missed in the DFC will be first accessed from the L1 DC and the fetched word will be forwarded to the pipeline and written to the DFC. During the line fill operation, the DFC read port is not occupied, hence load operations that hit in the filter cache can still be serviced. Bits are associated with the fill operation, indicating which words in the DFC line have been filled, in case this line is referenced by a load instruction before the fill operation has completed. Subsequent load operations that miss in the filter cache during the line fill operation are diverted to the L1 DC and no DFC allocation is permitted, since a DFC line fill operation is already being performed. In addition, when a load is diverted to the L1 DC, the line fill operation is stopped for that cycle in order to service the

current load operation. This approach allows the DFC to be filled without affecting the area and energy efficiency of the L1 DC and without causing any execution time penalty. Note that our DFC design uses a write-through policy, which ensures that the L1 DC always has up-to-date data and avoids the execution time penalty associated with writing back a dirty line over multiple cycles.

3.5. Making L1 DC Writes More Efficient

All store instructions cause writes to be attempted to both the DFC and the L1 DC, since a write-through policy is used in our design. The DFC is strictly inclusive, so all of the DFC lines are also resident in the L1 DC. For each DFC line, we also store information indicating the L1 DC way where the DFC line resides. Thus, on DFC store hits, there is no need to either perform an L1 DC tag check or to access the DTLB since the L1 DC is virtually indexed. Only the write operation to the data memory is performed on the L1 DC. A similar technique is used to save energy in level-2 caches, in which the L1 DC is write-through [Dai and Wang 2013]. However, using this technique for an L1 DC requires much more space since an L1 DC contains many more lines than a DFC. In addition, a store to the L1 DC is accomplished in one less cycle when the store is a DFC hit, since an L1 DC tag check is not performed. Performing an L1 DC store in one cycle can eliminate a structural hazard when an L1 DC load immediately follows the store. Thus, our design significantly reduces the cost of using a DFC write-through policy.

A write-allocate policy means that a line is allocated and fetched on a store miss. A no-write-allocate policy indicates that a line is not allocated in the cache on a store miss, and the store is done to the next level in the memory hierarchy. Typically, no-write-allocate is used with write-through as the value is going to be written to the next level of the memory hierarchy anyway and the miss rate can be reduced by not allocating a line that may never be read. When using a DFC, there is a tradeoff between using a no-write-allocate and a write-allocate policy. A no-write-allocate DFC policy reduces the line fetch operations from the L1 DC due to store misses not causing any allocation, but the L1 DC store energy increases due to each store miss requires a DTLB access and L1 DC tag checks to be performed. Using write-allocate with write-through will increase the ratio of DFC store hits, which can reduce the L1 DC store energy, as previously explained in this section. A detailed evaluation of both write-allocate and no-write-allocate policies is presented in the results section.

3.6. Supporting Multiprocessor Systems

The DFC is easily integrated into existing multiprocessor systems and does not require any modifications of existing cache coherency protocols. The DFC has a write-through policy that ensures that the data in the DFC and L1 DC are always consistent. The L1 DC is also strictly inclusive of the cache lines in the DFC. Whenever a cache line is evicted from the L1 DC due to a L1 DC miss or a cache coherency request, the way and index of the evicted line are compared to those stored in the DFC. If a match is found, that line is evicted from the DFC by clearing the valid bit (the V field in Figure 3).

In a fully associative DFC the least significant bits of the tag represent the index of the virtually tagged L1 DC and the conventional tag comparison logic can be used to detect matching indexes by simply ignoring the comparison of the most significant bits. The L1 DC way is stored with each DFC cache line when a new cache line is written to the DFC (the L1 DC Way field in Figure 3). A new line is only written to the DFC when a DFC miss has occurred. When a miss occurs, a conventional access to the L1 DC is performed. The hit signals from the L1 tag comparison indicate in which L1 DC way the data resides and this L1 DC way information is stored in the DFC. Thus, the only modification needed in the L1 DC is to be able to read the hit signals. As L1 DC cache lines are rarely evicted, this additional check in the DFC imposes an insignificant overhead.

Table II. 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

Table III. Processor Configuration

Load Latency	1 cycle
BPB, BTB	Bimodal, 128 entries
Branch Penalty	2 cycles
Integer & FP ALUs, MULDIV	1
Fetch, Decode, Issue Width	1
DFC	128B–512B (FA,DM) 32B line, 1 cycle hit
L1 DC & L1 IC	16kB, 4-way assoc, 32B line, 1 cycle hit
L2 Unified	64kB, 8-way assoc, 32B line, 12 cycle hit
DTLB & ITLB	16-entry fully assoc, 1 cycle hit
Memory Latency	120 cycles

4. EVALUATION FRAMEWORK

In this section, we present the tools, the benchmarks, and the methods used for the evaluation.

4.1. Benchmarks

We use 20 different benchmarks (see Table II) from six different categories in the MiBench benchmark suite [Guthaus et al. 2001]. All the benchmarks are compiled with VPO using the large dataset option [Benitez and Davidson 1988].

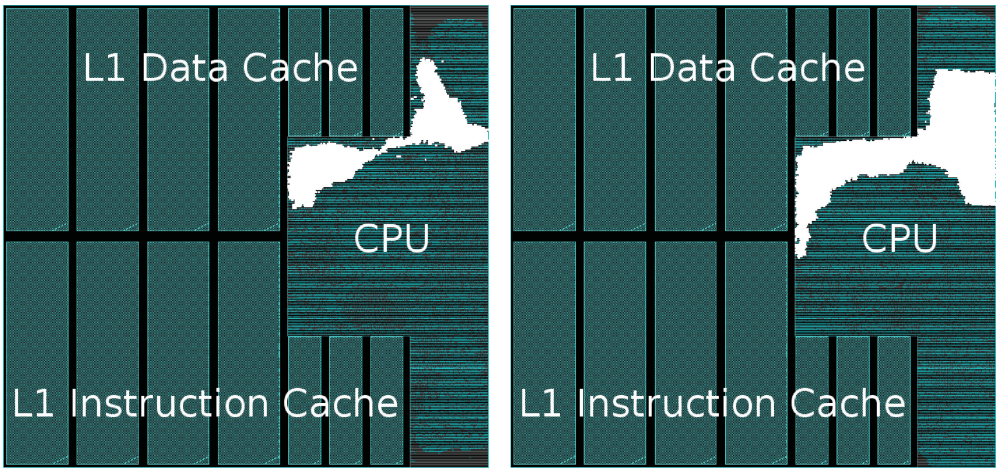
4.2. Simulator

We use the SimpleScalar simulator with the PISA instruction set to model a five-stage in-order processor [Austin et al. 2002]. The processor configuration is presented in Table III. In order to calculate the energy values, we backannotate energy values obtained from layout; for details see the next section.

4.3. Layout and Energy Values

Figure 7 shows the layout of two different five-stage in-order processors with 16kB four-way associative L1 instruction and data caches. Figure 7(a) shows a processor that uses a 128B DFC, while the DFC in Figure 7(b) is 256B. The SRAM blocks and the standard cells are laid out in a way similar to the trial layout of the Cortex-A5 processor [Halfhill 2009]. As described in Section 3.4, we use 32b wide SRAM blocks to reduce L1 DC area and energy. The DFCs are implemented using standard-cell flip-flops, which is the reason that they take up a relatively larger area per stored bit than the L1 caches that store the data in area-efficient SRAMs.

It is difficult to evaluate innovative implementations using approximative energy estimators, like Wattch and CACTI, so we use layout implementations to ensure that the energy values are accurate for the different processor components. All energy



(a) Processor layout with a 128B DFC. (b) Processor layout with a 256B DFC.

Fig. 7. Processor layout in which the DFC is highlighted in white to show its impact in terms of area.

values are derived from RC-based netlists that are extracted from placed and routed layouts such as the ones in Figure 7. All components except the DTLB are integrated and verified inside the processor layout. While the DTLB is implemented separately to facilitate overall verification of overall processor function, the DTLB is placed and routed as a component embedded in its proper context, for example, as a component driving tag comparators.

The processor designs are synthesized in Synopsys Design Compiler using a commercial 65nm Low-Power (LP) CMOS process technology with standard cells and SRAM blocks. The layout work is done in Cadence Encounter. The placed and routed layout implementations meet a 400MHz clock rate, assuming the worst-case process corner, a supply voltage of 1.1V, and 125°C. The layouts are functionally verified using small benchmarks from the EEMBC benchmark suite [Embedded Microprocessor Benchmark Consortium 2013]. Synopsys PrimeTime PX Power Analysis is used to obtain power and energy values from the RC-extracted netlists. These power and energy values are collected assuming the nominal process corner, 1.2V, and 25°C.

The L1 DC consists of four 1024x32b SRAM blocks for data and three 128×32b SRAM blocks for tags. Three 32b SRAM blocks are sufficient for storing the tags, since four tags, with 21 bits each, including valid bit, can be concatenated. The DTLB is a 16-entry fully associative structure built from standard cells. It is a very common configuration to have a first-level fully associative DTLB with few entries and a second-level DTLB with less associativity and more entries. Since the bulk of the accesses are captured by the fully associative structure, we only evaluate the first-level DTLB.

The total energy is calculated by taking, for all processor components, each component's energy per operation (obtained from layout) and multiplying this value by the total number of operations for this component (obtained from SimpleScalar). Before presenting the energy per operation, we first describe the events that take place in our DFC-enabled processor.

Table IV shows the various DFC events and the components of the DFC, DTLB, and L1 DC that are accessed for each of these events. DTLB and L1 DC misses happen much less frequently and are not depicted in this table. Furthermore, the inclusion of a DFC does not typically change the total number of DTLB or L1 DC misses. Therefore, DTLB and L1 DC misses are not accounted for in our evaluation.

Table IV. Components Accessed for Each DFC Event

DFC Event		DFC					L1 DC			
		Read Tags All Ways	Read Data All Ways	Write Tag	Write Data	DTLB	Read Tag All Ways	Read Data All Ways	Write Data	Read Data
Load	Spec. Failure	X	X							
	Hit	X	X							
	Miss (No Fill)	X	X			X	X	X		
	Miss (Fill)	X	X	X	8X	X	X	X		7X
Store	Spec. Failure	X								
	Hit	X			X				X	
	Miss (No Fill)	X				X	X		X	
	Miss (Fill)	X		X	8X	X	X		X	8X

DFC Speculation Failure implies that an access to the DFC was attempted but that the tag and/or index was modified during the address calculation. For loads, all tags and data for all ways within a set are accessed, while for stores, only the tags for all ways within a set are accessed.

DFC Hit implies that the speculative access to the DFC was successful and that the cache line resides in the DFC. For loads, all tags and data for all ways within a set are accessed, while for stores, only the tags for all ways within a set are accessed. As the DFC implements a write-through policy, the store also causes the data to be written to the L1 DC. The DFC stores the way of the associated L1 DC line, so no DTLB access or tag checks are required to write the data to the L1 DC.

DFC Miss (No fill) implies that the sought data does not reside in the DFC and that a no-write-allocate policy is implemented or that another cache line is currently being filled, which prevents a second cache line to be filled on a DFC load miss. For loads, a conventional access is attempted that accesses all tags and data for all ways within a set. The L1 DC is then accessed and for performance reasons all tags and data for all ways within a set of the L1 DC are accessed. This ensures that the sought data word is provided with as few stall cycles as possible. For stores, a conventional access is attempted that accesses all tags for all ways within a set. The data item is then written to the L1 DC, which requires a DTLB access, all tags for all ways within a set of the L1 DC to be read, and the data to be written to a single way of the L1 DC.

DFC Miss (Fill) implies that the sought data does not reside in the DFC and that a load miss or a write miss with a write-allocate policy implementation occurs. In addition to the events performed for a DFC miss with no fill, the cache line is read from the L1 DC and written to the DFC. For loads, the first data word is provided by the parallel access to the tags and data that is performed to reduce the amount of stall cycles. The remaining seven data words of the cache line are then read sequentially from a single way of the L1 DC. For stores, the conventional store of the data to the L1 DC is performed first and afterwards the eight words of the cache line are read sequentially from a single way of the L1 DC. For both loads and stores, the tag and eight data words of the cache line are written to the DFC.

Table V shows the energy required for accessing the components of the DFC for the various configurations. In this table, the tag and data read energy of the DFC are not separated. This leads to a pessimistic energy evaluation, since the total energy of data and tag is used for some events, for example, for store misses, instead of only tag energy.

Table VI shows the energy used for accessing the various components of the L1 DC and the DTLB. Similar to the DFC, loads are more expensive than stores due to the need to read all data on each load to minimize the access time. It should be

Table V. Energy for Different DFC Components

	FA			DM		
DFC Component	128B	256B	512B	128B	256B	512B
Read Data & Tag - All Ways	13.0pJ	29.5pJ	61.5pJ	10.5pJ	22.4pJ	48.0pJ
Write Tag	0.7pJ	1.5pJ	4.0pJ	0.7pJ	1.5pJ	4.0pJ
Write Data	3.4pJ	6.4pJ	18.0pJ	3.4pJ	6.4pJ	18.0pJ

Table VI. Energy for Different L1 DC Components and the DTLB

Component	Energy
Read Tags - All Ways	57.3pJ
Read Data - All Ways	112.7pJ
Write Data	33.9pJ
Read Data	28.2pJ
DTLB (16 entries, fully associative)	17.5pJ

noted that the energy dissipation of the separate components is shown. For example, to calculate the energy of a regular load operation from the L1 DC, the energy of reading tags from all ways should be added to the energy of reading data from all ways. To calculate the energy of a line fetch operation on a DFC load operation miss, the energy of making seven single data reads should be added to the energy of a regular load operation. Because on a line fetch operation the first word is fetched as a regular load, and the remaining words in the line are read from a single data SRAM as the tag match operation has already occurred and the way information is known. If a line fetch operation is initiated by a store miss in the DFC, then the energy is the cost of reading all the tags and eight single data reads. The reason is that the data in the first cycle is not critical hence it is possible to wait until after the tag match is made. The idle L1 DC energy is neglected, since the leakage power of the LP process technology is insignificant. Since miss events for the L1 DC are not very frequent, we assume the L1 DC miss energy is insignificant. In doing this simplification, we introduce an error of around 4% on the total L1 DC energy.

The store energy in a fast set-associative cache is substantially smaller compared to the load energy. The reason is that store operations only enable the data way in which there is a tag match. Load operations, on the other hand, enable all data ways and the correct word is selected using a final multiplexer driven by the tag hit signals, as depicted in Figure 2. This approach ensures the shortest delay. Due to data dependencies between load operations and consecutive operations (load hazards), load operations are typically performed as fast as possible.

5. RESULTS

In this section, we first evaluate the energy and performance benefits of the proposed DFC for the in-order five-stage pipeline that we implemented in hardware and that is described in Section 4.3. We then describe a more aggressive processor configuration with a two-cycle load latency and use this configuration to evaluate timing and performance improvements of the proposed technique.

5.1. Energy and Performance Evaluation

Figure 8 shows how successful a speculative access to the DFC is across the benchmark suite. As shown, the speculation, on average, is successful 71.9% of the time. Speculation is attempted but fails for 1.9% of the load accesses. For the remaining 26.2% of the load accesses, the offset portion of the address is greater than or equal to

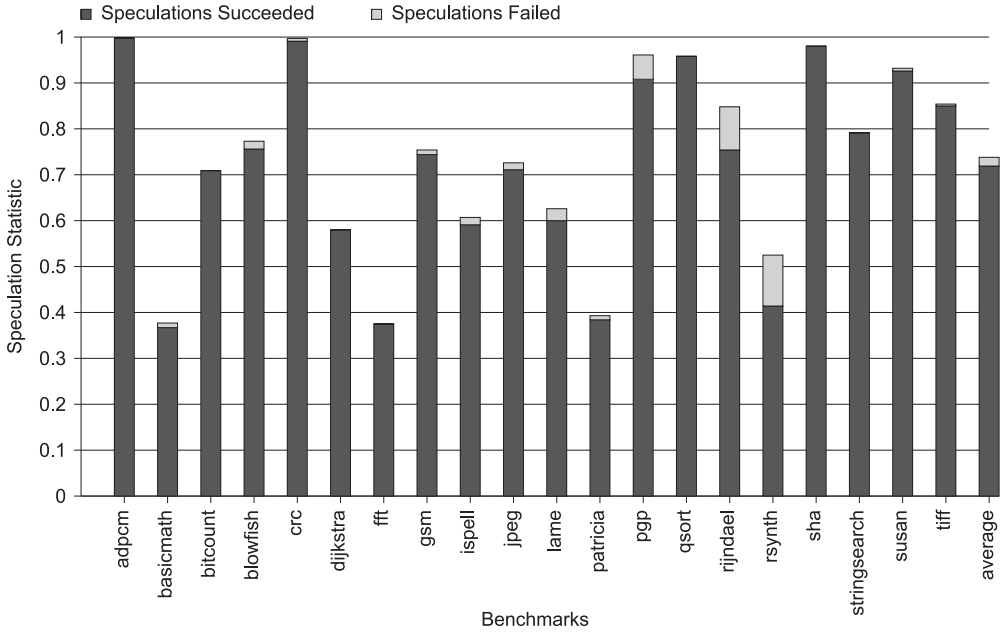


Fig. 8. Statistics for speculative access to the DFC.

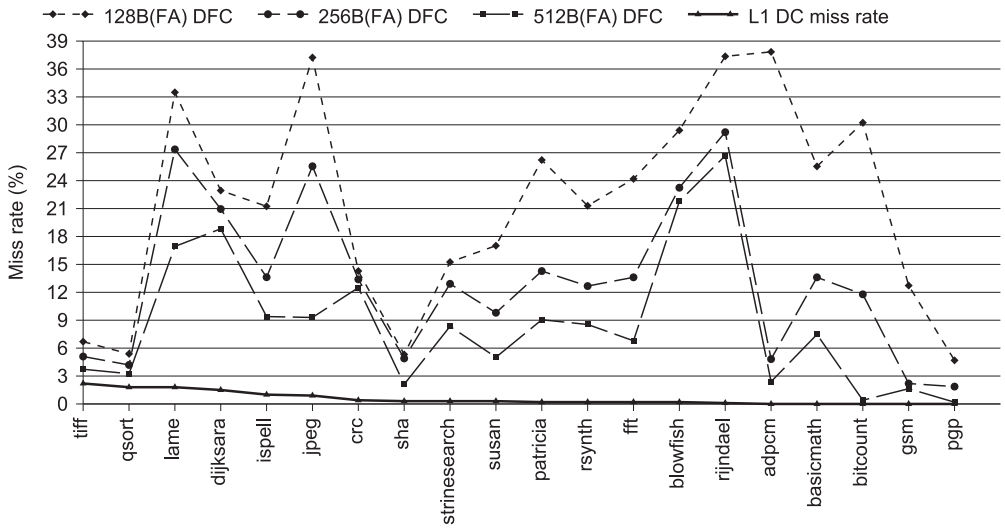


Fig. 9. L1 DC and fully associative DFC miss rates.

the line size, which causes speculation to not be attempted. These results show that speculatively accessing the DFC is worthwhile as almost three-fourths of the memory references for load operations can obtain the appropriate address, enabling the DFC to be accessed in the same cycle that the address is generated.

Figure 9 shows the L1 DC and fully associative DFC miss rates for the MiBench applications. The average miss rate for the L1 DC is 0.6%. The benchmarks are sorted by decreasing L1 DC miss rate. The figure clearly shows that there is no correlation

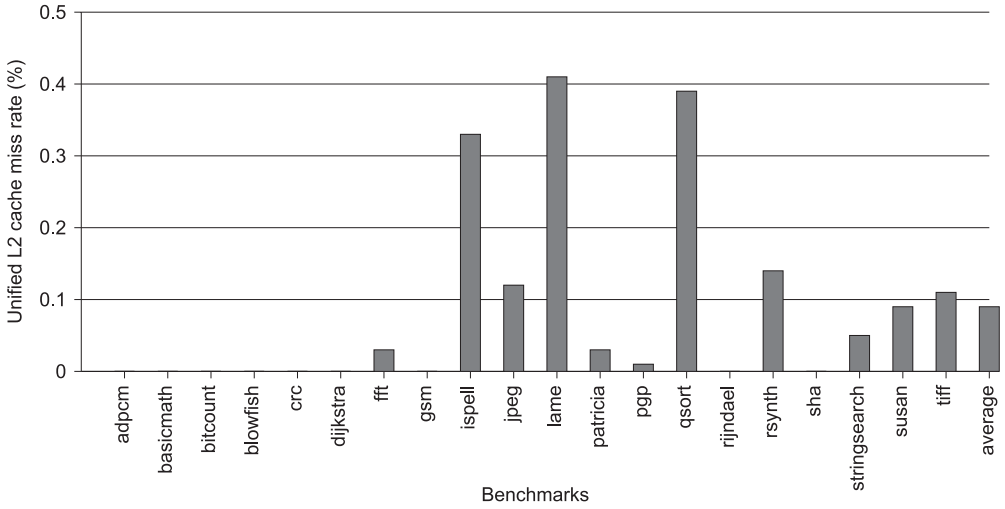
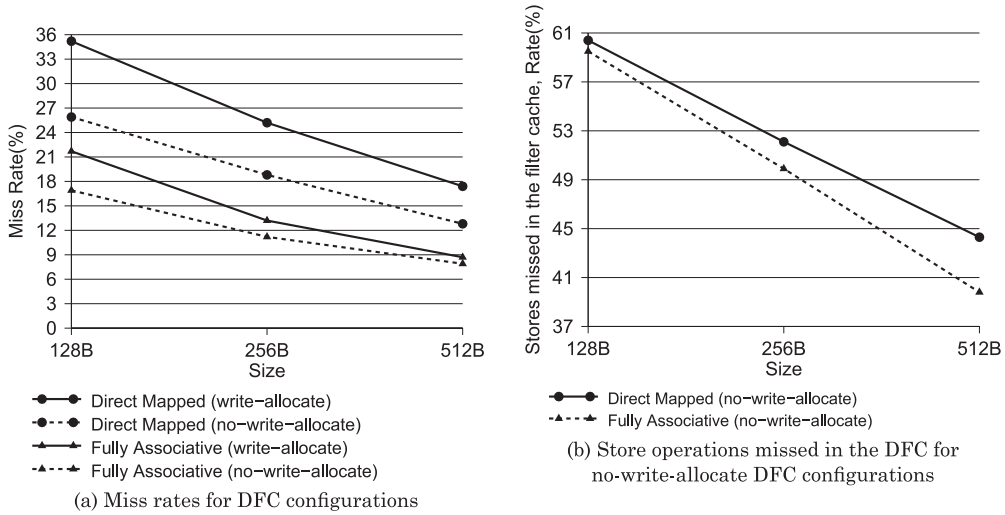


Fig. 10. Unified L2 cache miss rate (L2 misses / (L1 DC accesses + L1 IC accesses)).



(a) Miss rates for DFC configurations

(b) Store operations missed in the DFC for no-write-allocate DFC configurations

Fig. 11. Miss rates and store misses for no-write-allocate DFC configurations assuming an ideal one-cycle line fetch operation.

between the L1 DC miss rate and the DFC miss rates. The DFC miss rates are under the assumption that the line fetch operation takes one cycle. As described in Section 3.4, an L1 DC would not be designed for reading and writing an entire line in a single cycle unless the processor can benefit from it. But using the miss rate for one cycle line fetch operation gives good insights to the trends in DFC design.

Figure 10 shows the unified L2 cache miss rates for MiBench applications. This miss rate is calculated by the following formula:

$$\text{L2 cache miss rate} = (\text{L2 misses}/(\text{L1 DC accesses} + \text{L1 IC accesses})) \quad (1)$$

On average, the unified L2 cache miss rate is 0.1%. Figure 11 shows the average miss rates for different DFC sizes, organizations, and write policies under the assumption of one-cycle line-fetch operation. In addition, average store misses are shown for DFCs

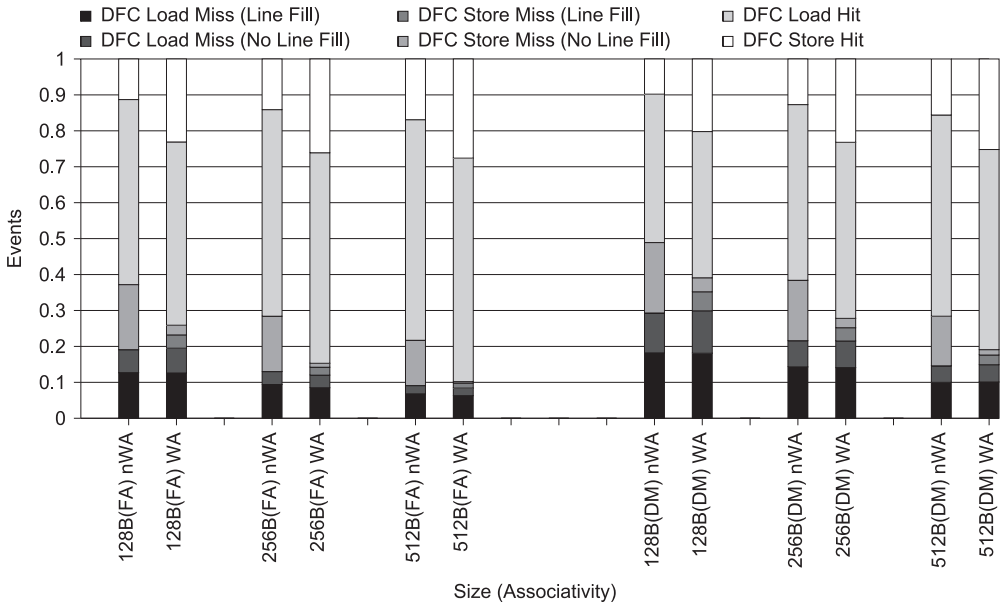


Fig. 12. Frequency of various DFC events. Two write policies are evaluated: no-write-allocate (nWA) and write-allocate (WA). The evaluated cache organizations are either fully associative (FA) or direct mapped (DM), of sizes 128B (4-entry), 256B (8-entry), or 512B (16-entry).

with the no-write-allocate policy. As shown in Figure 11(a), fully associative DFC configurations have better miss rates compared to direct-mapped DFC configurations. The no-write-allocate policy reduces the miss rate due to store operations not causing a line allocation, which reduces the contention in the DFC. The no-write-allocate policy is more effective on direct-mapped DFC configurations, since there are more conflicts in a direct-mapped DFC. But in the no-write-allocate policy there are many store operations that miss in the DFC due to the line not being allocated on a store miss. The ratio of stores that are missed in the DFC for the no-write-allocate policy is shown in Figure 11(b).

When the practical line fill approach is employed for the DFC, as described in Section 3.4, the miss rate cannot be used directly for evaluation because there are many other events happening. For example, some misses will not cause a line fetch because a miss happens while another line fetch operation is ongoing in which the miss operation will access the L1 DC instead of starting a line fetch operation. Hence, the results for a practical DFC implementation are given in terms of events, which cover all different activities that can occur in a this implementation. Figure 12 shows how frequent various DFC events are for different DFC sizes, organizations, and write policies. The main differences between the write policies are that the no-write-allocate policy has (1) no stores that cause line fills and (2) fewer store hits. As the size of the DFC increases, the ratio of hits (both loads and stores) increases. As expected, the total misses for a fully associative DFC is approximately the same as that of a direct-mapped DFC twice that size. As a result, given the same DFC size, a fully associative DFC can be more efficient than a direct-mapped one.

Figure 13 shows how frequently the various memory hierarchy components are accessed. As shown, the DFC tags are accessed on every DFC event. The number of DFC writes is large due to each word written during a line fill is separately counted. Note that most of the DFC writes are overlapping with instructions that do not perform any

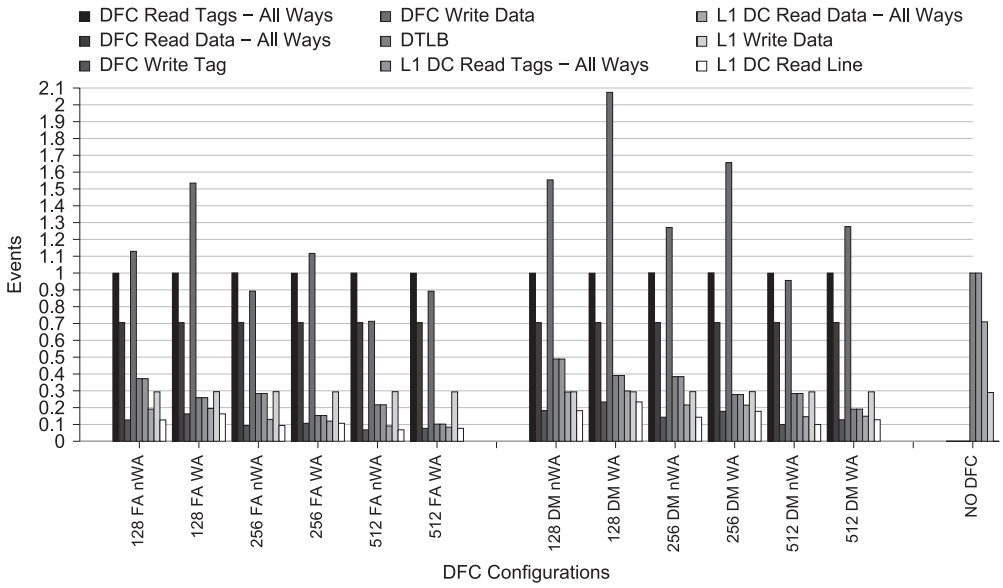


Fig. 13. Frequency of accesses to various components in the memory hierarchy.

data accesses. The number of DTLB accesses is higher for the no-write-allocate policy, since the number of DFC store misses is greater and each miss causes a DTLB access. Figure 12 shows that these missed stores are significant. Figure 13 also shows that the L1 DC line read operations are fewer for the no-write-allocate policy as compared to the write-allocate policy. This is expected because no store operations cause a line fetch for the no-write-allocate policy. This is even more apparent for direct-mapped caches because of their high miss rates. However, as the DFC size increases, the advantage of the no-write-allocate policy becomes smaller. The advantage of the no-write-allocate policy is smaller for the fully associative DFC, because the miss rate is relatively lower. It should be noted that the speculation failure events are not included in Figures 12 or 13.

Figure 14 shows the data access (DFC, L1 DC, and DTLB) energy usage for each configuration, where the baseline is an L1 DC and DTLB without a DFC. The breakdown is shown as the component list given in Section 4.3. A 256B fully associative DFC that uses the write-allocate policy can save 42.5% of the data access energy. This includes the overhead of speculation failures. The no-write-allocate policy saves more energy for direct-mapped caches, as expected. However, since the miss rate is relatively low for fully associative DFCs, as the DFC size increases, the overhead caused by the relatively higher number of store misses makes this approach less energy-efficient compared to the write-allocate policy. In addition, the DTLB energy dissipation is higher in the no-write-allocate policy due to the high number of store misses. Note that the best energy savings are for a DFC size of 256B for both direct-mapped and fully associative DFCs. It should also be noted that the area overhead is becoming significant for the 512B DFCs, since the implementations are based on standard cells. Furthermore, a fully associative, 512B DFC might be challenging to implement while meeting strict timing requirements and, therefore, might not be a realistic design choice. We have included the fully associative, 512B DFC in the results only to show the trends when the DFC size increases.

Figure 15 shows the same data access energy as Figure 14, but the breakdown is in terms of L1 DC, DFC, and DTLB energy. As the DFC size increases, the energy

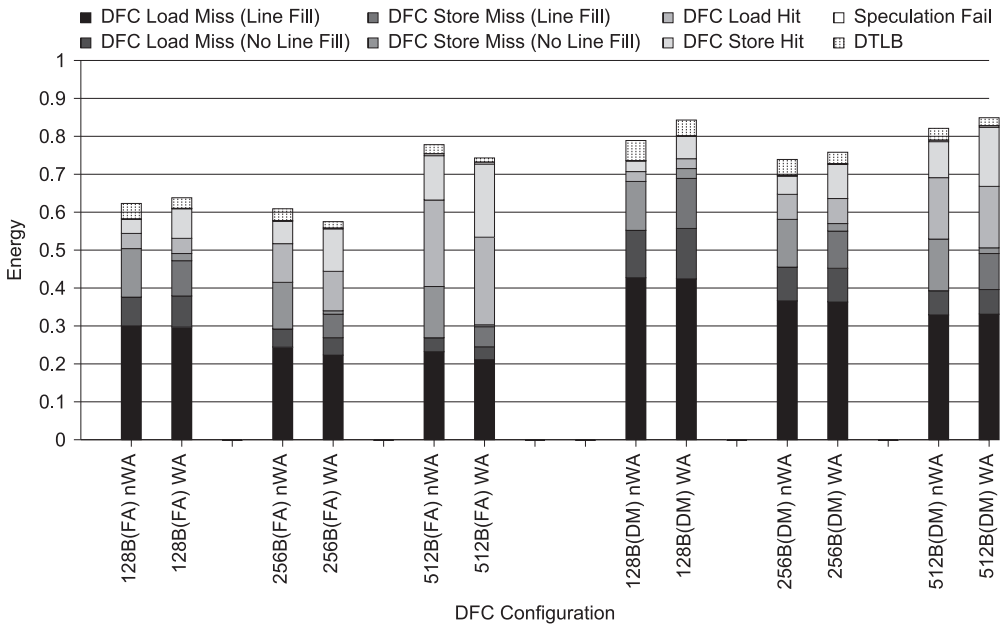


Fig. 14. Data access energy for different DFC configurations with event breakdown.

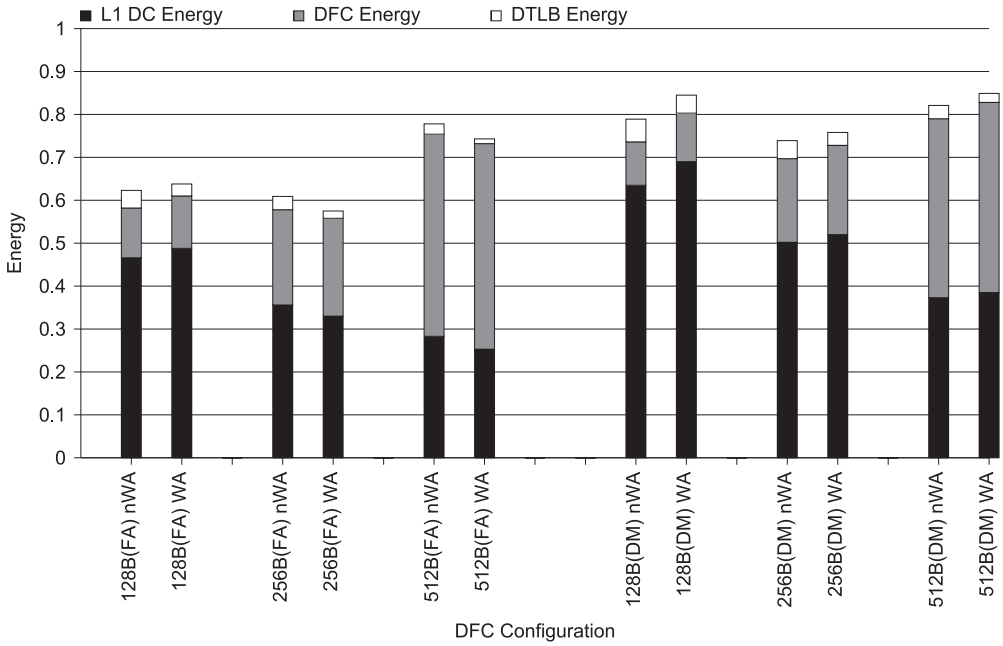


Fig. 15. Data access energy for different DFC configurations with unit breakdown.

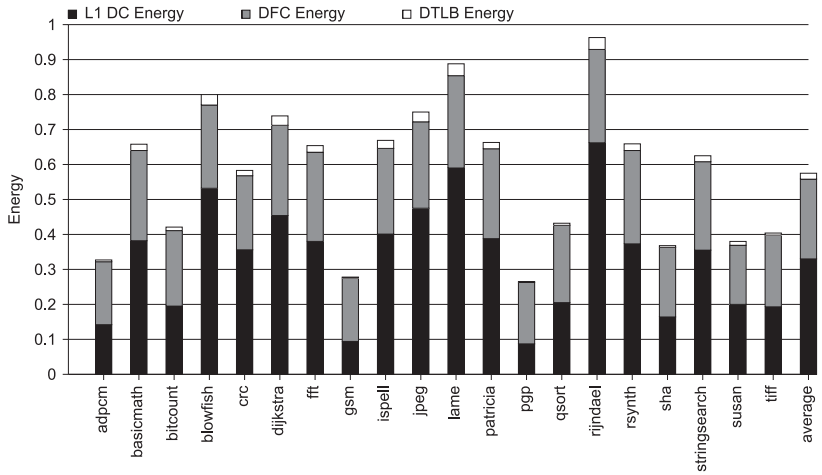


Fig. 16. Data access energy for 256B fully associative write-allocate DFC with unit breakdown.

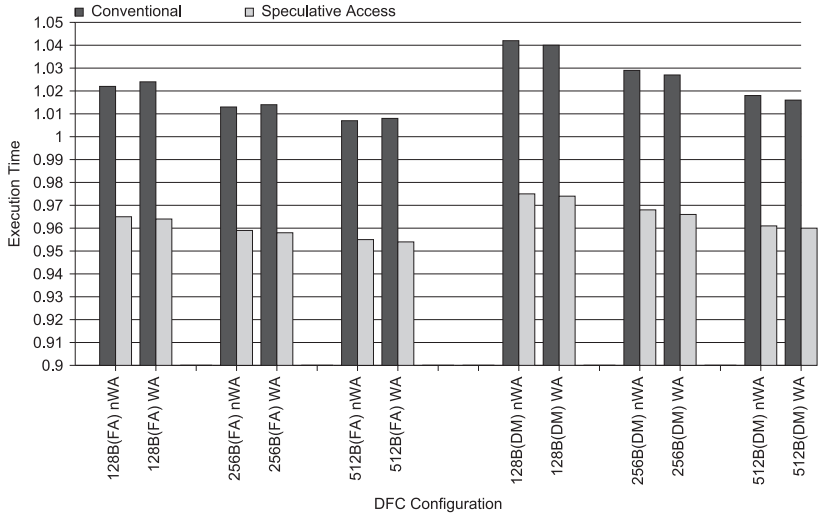


Fig. 17. Normalized execution time for different DFC configurations.

expenditure of the L1 DC and DTLB decreases due to a reduced DFC miss rate, but the energy dissipation of the DFC increases as the structure is becoming larger. Figure 16 shows the data access energy savings for each benchmark with the most energy efficient DFC configuration of 256B fully associative DFC with write-allocate policy. The energy savings are greater for benchmarks that have a high DFC hit rate. For instance, the energy for *pgp* is reduced by 73.5%. The energy dissipation is not increased for any of the benchmarks, even for those with high DFC miss rates.

Figure 17 shows processor performance results for the different DFC configurations, as compared to a processor with no DFC. *Conventional* implies a DFC implementation in which the DFC is accessed in a separate stage after the address generation stage. This causes a performance overhead due to frequent DFC misses. The *speculative access* execution time is improved due to DFC hits accessing data earlier in the pipeline. The performance benefit is thus affected by the hit rate for DFC loads. Increasing the size of

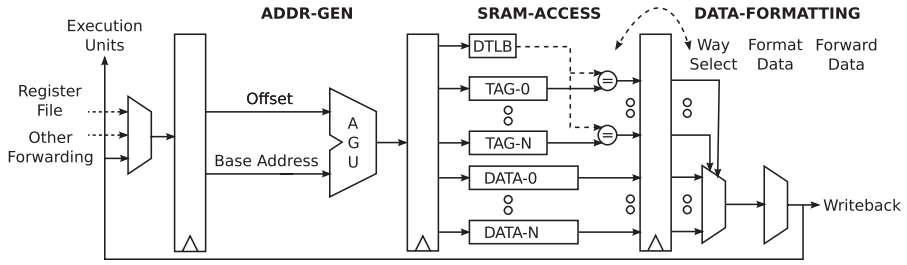


Fig. 18. Three-stage L1 DC access.

the DFC, using a fully associative DFC organization, and using a write-allocate policy all provide better performance as each option increases the ratio of DFC load hits. The 256B write-allocate DFC, which provides a 42.5% energy reduction, provides a 4.2% execution time improvement when the speculative access scheme is used.

5.2. Timing and Extended Performance Evaluation

The processor configuration used for the performance evaluation in the previous section is the same configuration as for the hardware implementation that is used for the energy evaluation. In this section, we evaluate timing and performance using a more aggressive processor configuration that has a load latency of two cycles instead of only a single cycle. A two-cycle load latency is common in contemporary state-of-the-art in-order processors[Halfhill 2009; Williamson n.d.; MIPS Technologies 2009].

Figure 18 illustrates a three-stage L1 DC pipeline that has a two-cycle load latency. In the address generation stage, the memory address is generated. In the second stage, the DTLB and the SRAMs for tag and data are accessed. The tag comparison can occur in this stage or in the third stage. Since the access times of the synchronous SRAMs can be considerably lower than their cycle times, it might be possible to do the tag comparison in this stage. During the third stage, the way selection is performed and the data are formatted if it is a half-word or byte-level operation. Finally, the data value is forwarded to the corresponding units in the pipeline.

Introducing a DFC into a three-stage L1 DC pipeline reduces the load latency by one cycle. The DFC is accessed during the address generation stage, the same way as shown in Figure 5. On a DFC hit, there is no need to access the SRAMs and DTLB of the L1 DC. The SRAM access stage can, therefore, be bypassed and the data from the DFC can be directly forwarded to the final stage of data formatting. Since the DFC is much smaller and is likely to have a faster access time than an L1 DC, some processor configurations may access both the DFC and do the data formatting in the address generation stage. However, the results we present assume that the load latency is reduced by only one instead of two cycles.

Timing Evaluation. We evaluate timing for the eight-entry fully associative DFC since (1) the critical path is much longer compared to a direct-mapped DFC and (2) the eight-entry fully associative DFC is the most energy-efficient DFC configuration. We assume a common L1 DC configuration of 16kB size and four-way set associativity. This data cache uses four multi-V_T 4kB SRAMs (1024wx32b-mux8), in which the cycle time is 1.2ns, while the access time is 0.85ns. These timing values are the final timing values for a placed and routed SRAM macro. This means that if the cycle time of the processor is limited by the data SRAMs it will be 1.2ns. The DFC is built using a multi-V_T design flow, whereas the SRAM cells and the flip-flops of the DFC are built using high-V_T transistors. The remaining logic is a mix of standard-V_T and low-V_T cells. The same

Table VII. Aggressive Processor Configuration

Load Latency	1–2 cycles
BPB, BTB	gshare L2:1024, 256 entries
Branch Penalty	7 cycles
Integer & FP ALUs, MULDIV	1
Fetch, Decode, Issue Width	1
DFC	128B-512B (FA, DM) 32B line, 1 cycle hit
L1 DC & L1 IC	16kB, 4-way assoc, 32B line, 2 cycle hit
L2 Unified	64kB, 8-way assoc, 32B line, 12 cycle hit
DTLB & ITLB	16-entry fully assoc, 1 cycle hit
Memory Latency	120 cycles

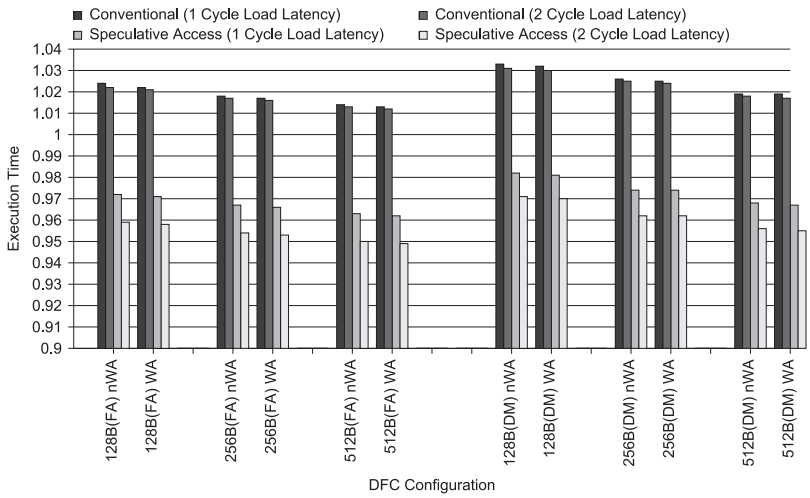


Fig. 19. Normalized execution time for different DFC configurations.

output load is used for the SRAMs in the L1 DC and the DFC. The timing path for accessing the eight-entry fully associative DFC is 0.78ns after synthesis. The timing will degrade after the DFC has been placed and routed. Assume the delay increases to 1ns after place and route, an increase of 28%, which is a reasonable assumption for an advanced process technology. This delay is still much lower than the critical path of 1.2ns for the L1 DC. Hence, it is possible to access the DFC in a single stage. The data that is output from the L1 DC SRAM needs to go through a way-select multiplexor, whereas the data item from the DFC is directly available. In addition, the maximum frequency will be defined by the cycle time of the SRAM and not the access time.

Performance Evaluation. For the performance evaluation, we not only increased the load latency but also increased the branch penalty, from two cycles to seven cycles, to represent a processor with a deeper pipeline. The more aggressive configuration is presented in Table VII.

Figure 19 shows the performance of the processor with an aggressive configuration (Table VII). The execution time is shown with both one-cycle and two-cycle load latency to show the impact of different load pipeline latencies. The execution time is normalized to a processor without DFC and a L1 DC with one-cycle and two-cycle load latencies,

respectively. In an aggressive processor pipeline, it is expected that the load pipeline will have a two-cycle latency. As the load latency is increased to two cycles, the execution time increases due to more data dependencies with load instructions. This has two consequences: First, the execution time overhead of the conventional DFC reduces slightly with a two-cycle load latency since the miss count of the DFC does not change, but the overall execution time increases. Second, the execution time benefit of our proposed DFC increases with a two-cycle load latency. The reason is that with a longer load latency there are more data dependencies that the proposed DFC can help resolve. A 256B fully associative DFC can improve the execution time by 4.6% on a processor with a two-cycle load latency. Hence, the DFC becomes more beneficial as the depth of the load pipeline increases.

6. RELATED WORK

A multiple line buffer approach has been proposed in order to reduce L1 cache energy usage [Ghose and Kamble 1999], but without affecting the cycle time, which was a drawback in the first line buffer study [Su and Despain 1995]. In the multiple line buffer approach, a parallel check is performed using the tag and line index to verify if the current access resides in the line buffers. When there is a hit in the line buffers, the discharge of the bitlines is avoided, hence energy is saved. The placement of the cache lines inside the line buffers is handled in a fully associative manner in order to keep the reuse of data as high as possible. One of the disadvantages of this approach is that it requires a customized SRAM implementation in order to disable the discharge signal when appropriate. In addition, the guarantee mechanism requires to have a line buffer for each way of the cache for every entry (line index). Given a line index, there is no significant locality between the lines that are residing on different ways. Hence, many of the line buffers become wasteful.

The original filter cache proposal places both an Instruction Filter Cache (IFC) and a DFC between the CPU and the L1 cache [Kin et al. 1997, 2000]. This proposed organization can potentially reduce energy usage at the expense of an execution time penalty of one cycle on each filter cache miss. It appears that their design also assumes a one cycle fill of an entire filter cache line, which can negatively impact both the area of the L1 cache and the power to access these caches. We believe this performance penalty and negative impact on L1 DC area and power has prevented filter caches from being adopted by processor manufacturers.

Duong et al. [2012] propose to use a fully associative DFC in order to reduce the energy dissipation of an L1 DC. Their DFC tag comparison is performed in the execute stage after the address generation, in order to prevent the DFC miss penalty. They do not mention using virtual tags, but instead just state that the DFC tag comparison is possible in the execute stage due to the small DFC size and the slow clock rate of an embedded processor. Unlike our DFC design, their approach provides no execution time benefit. It also appears their evaluation was performed assuming that entire DFC cache lines can be filled or written back in a single cycle. In addition, they assume that the L1 DC has separate read and write ports, which will significantly increase the area of the L1 DC SRAMs. Unless there is an important benefit due to the processor configuration, an L1 DC will typically have a single shared read/write port for area efficiency. This issue will have an important impact on their proposed writeback filter cache design.

Austin et al. [1995] 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. While this technique improves performance, it increases the L1 DC energy since the whole cache needs to be accessed again on speculation failures.

Nicolaescu et al. [2006] propose to use a 16-entry fully associative table to store the way information of the last accessed lines. 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 of a DTLB structure, incurs a significant area and power overhead, which will cancel out some of this technique's energy benefits. This technique can only reduce the energy dissipation of the L1 DC, while the DFC proposed in this work improves the execution time considerably in addition to reducing the overall data access energy.

There have also been some techniques proposed to avoid DTLB accesses. Block buffering has been used to detect recent TLB translations by performing comparisons with the virtual address as soon as it is generated [Lee et al. 2005; Chang and Lan 2007]. Opportunistic virtual caching is a technique to allow some blocks in the L1 caches to be cached with virtual addresses by changing the operating system to indicate which pages can use virtual caching [Basu et al. 2012]. Our DFC design is more effective at reducing data access energy usage, as our approach not only avoids most DTLB accesses but also avoids most L1 DC accesses as well.

7. FUTURE WORK

A lazy fill write-allocate policy could be employed to further improve the energy efficiency as compared to the traditional write-allocate policy. On a store miss, the line can be allocated by only writing the tag. This way a store miss does not cause a line fetch. If a load happens for a line which is only allocated and not filled, then the bytes of the line that have not yet been written are fetched from the L1 DC. This lazy fill technique may avoid fetching many words from the L1 DC into a DFC line when cache lines are partly or entirely written first.

For multithreaded processors that periodically switch between a set of threads, a DFC per thread can be employed, as each DFC has a relatively small area footprint. This not only avoids any issues regarding the use of virtually tagged DFCs, but it also reduces contention as each thread has its own private DFC. Multithreaded processors tend to require a highly associative L1 DC to avoid contention between the threads, which makes each L1 DC access more costly in terms of energy. The DFC eliminates a large portion of the L1 DC accesses. It might, therefore, be beneficial to increase the associativity to reduce contention even further, as long as timing requirements can be met. Employing multiple DFCs would require a cache coherency mechanism to be implemented to keep shared data coherent across the DFCs.

Out-of-order processors have a more complex load-store pipeline. Commonly the address generation is followed by a load buffer to support speculative load operations. The use of address speculation and a DFC to access data earlier has the potential to reduce both energy and execution time for out-of-order processors. However, further evaluations are required to determine how the DFC would interact with the conventional behavior of an out-of-order load-store pipeline.

8. CONCLUSIONS

We have described how to design and implement data filter caches (DFCs) that are practical in the sense that they not only improve processor energy efficiency, but they also improve processor performance and comply well with rational cell-based implementation flows. Our evaluations show that, for example, a 256B fully associative DFC can provide an overall 42.5% energy reduction for data accesses at the same time as the execution time is improved by 4.2% using speculative DFC accesses. Our energy evaluations have been based on constant clock rates. Thus, the reduction in execution time

can offer further energy saving opportunities at the processor level, but this depends on the processor configuration. For low-power systems, there are mainly opportunities to reduce energy in the clock network, while for high-performance systems, leakage energy reductions are possible.

REFERENCES

AUSTIN, T., LARSON, E., AND ERNST, D. 2002. SimpleScalar: An infrastructure for computer system modeling. *Computer 35,* 2, 59–67.

AUSTIN, T. M., PNEVMATIKATOS, D. N., AND SOHI, G. S. 1995. Streamlining data cache access with fast address calculation. In *Proceedings of the International Symposium on Computer Architecture*. ACM, New York, NY, 369–380.

BARDIZBANYAN, A., SJÄLANDER, M., WHALLEY, D., AND LARSSON-EDEFORS, P. 2013a. Speculative tag access for reduced energy dissipation in set-associative L1 data caches. In *Proceedings of the International Conference on Computer Design*. IEEE Computer Society, Washington, DC, 302–308.

BARDIZBANYAN, A., SJÄLANDER, M., WHALLEY, D., AND LARSSON-EDEFORS, P. 2013b. Towards a performance and energy-efficient data filter cache. In *Proceedings of the Workshop on Optimizations for DSPs and Embedded Systems*. ACM, New York, NY, 21–28.

BASU, A., HILL, M., AND SWIFT, M. 2012. Reducing memory reference energy with opportunistic virtual caching. In *Proceedings of the International Symposium on Computer Architecture*. ACM, New York, NY, 297–308.

BENITEZ, M. E. AND DAVIDSON, J. W. 1988. A portable global optimizer and linker. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, New York, NY, 329–338.

CEKLEOV, M. AND DUBOIS, M. 1997. Virtual-address caches. Part 1: Problems and solutions in uniprocessors. *IEEE Micro 17,* 5, 64–71.

CHANG, Y. AND LAN, M. 2007. Two new techniques integrated for energy-efficient TLB design. *IEEE Trans. Very Large Scale Integrated Systems 15,* 1, 13–23.

DAI, J. AND WANG, L. 2013. An energy-efficient L2 cache architecture using way tag information under write-through policy. *IEEE Trans. Very Large Scale Integration (VLSI) Systems 21,* 1, 102–112.

DALLY, W. J., BALFOUR, J., BLACK-SHAFFER, D., CHEN, J., HARTING, R. C., PARIKH, V., PARK, J., AND SHEFFIELD, D. 2008. Efficient embedded computing. *IEEE Computer 41,* 7, 27–32.

DUONG, N., KIM, T., ZHAO, D., AND VEIDENBAUM, A. 2012. Revisiting level-0 caches in embedded processors. In *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems*. ACM, New York, NY, 171–180.

EMBEDDED MICROPROCESSOR BENCHMARK CONSORTIUM. 2013. [Online]. Available: http://www.eembc.org.

GHOSE, K. AND KAMBLE, M. 1999. Reducing power in superscalar processor caches using subbanking, multiple line buffers and bit-line segmentation. In *Proceedings of the International Symposium on Low Power Design*. ACM, New York, NY, 70–75.

GUTHAUS, M. R., RINGENBERG, J. S., ERNST, D., AUSTIN, T. M., MUDGE, T., AND BROWN, R. B. 2001. Mibench: A free, commercially representative embedded benchmark suite. In *Proceedings of the International Workshop on Workload Characterization*. IEEE Computer Society, Washington, DC, 3–14.

HALFHILL, T. R. 2009. ARM's Midsize Multiprocessor. Technical Report. Microprocessor Report.

HAMEED, R., QADEER, W., WACHS, M., AZIZI, O., SOLOMATNIKOV, A., LEE, B. C., RICHARDSON, S., KOZYRAKIS, C., AND HOROWITZ, M. 2010. Understanding sources of inefficiency in general-purpose chips. In *Proceedings of the International Symposium on Computer Architecture*. ACM, New York, NY, 37–47.

HENNESSY, J. AND PATTERSON, D. 2011. *Computer Architecture: A Quantitative Approach* 5th Ed. Morgan Kaufmann, San Francisco.

HINES, S., GAVIN, P., PERESS, Y., WHALLEY, D., AND TYSON, G. 2009. Guaranteeing instruction fetch behavior with a lookahead instruction fetch engine (LIFE). In *Proceedings of the Conference on Languages, Compilers, and Tools for Embedded Systems*. ACM, New York, NY, 119–128.

HINES, S., WHALLEY, D., AND TYSON, G. 2007. Guaranteeing hits to improve the efficiency of a small instruction cache. In *Proceedings of the Internatinal Symposium on Microarchitecture*. ACM, New York, NY, 433–444.

HUANG, W., RAJAMANI, K., STAN, M. R., AND SKADRON, K. 2011. Scaling with design constraints: Predicting the future of big chips. *IEEE Micro 31,* 4, 16–29.

INOUE, K., ISHIHARA, T., AND MURAKAMI, K. 1999. Way-predicting set-associative cache for high performance and low energy consumption. In *Proceedings of the International Symposium on Low Power Electronics and Design*. ACM, New York, NY, 273–275.

KIN, J., GUPTA, M., AND MANGIONE-SMITH, W. H. 1997. The filter cache: An energy efficient memory structure. In *Proceedings of the International Symposium on Microarchitecture*. IEEE Computer Society, Washington, DC, 184–193.

KIN, J., GUPTA, M., AND MANGIONE-SMITH, W. H. 2000. Filtering memory references to increase energy efficiency. *IEEE Trans. Computers 49,* 1, 1–15.

LEE, J. H., WEEMS, C., AND KIM, S.-D. 2005. Selective block buffering TLB system for embedded processors. *IEE Proceedings on Computers and Digital Techniques 152,* 4, 507–516.

MIPS Technologies 2009. *MIPS® 1004K™ Coherent Processing System Datasheet*. MIPS Technologies.

NICOLAESCU, D., SALAMAT, B., VEIDENBAUM, A., AND VALERO, M. 2006. Fast speculative address generation and way caching for reducing L1 data cache energy. In *Proceedings of the International Conference on Computer Design*. IEEE Computer Society, Washington, DC, 101–107.

POWELL, M. D., AGARWAL, A., VIJAYKUMAR, T. N., FALSAFI, B., AND ROY, K. 2001. Reducing set-associative cache energy via way-prediction and selective direct-mapping. In *Proceedings of the 34th Annual ACM/IEEE International Symposium on Microarchitecture*. ACM, New York, NY, 54–65.

SU, C. AND DESPAIN, A. 1995. Cache design tradeoffs for power and performance optimization. In *Proceedings of the International Symposium on Low Power Design*. ACM, New York, NY, 63–68.

TANG, W., GUPTA, R., AND NICOLAU, A. 2001. Design of a predictive filter cache for energy savings in high performance processor architectures. In *Proceedings of the International Conference on Computer Design*. IEEE Computer Society, Washington, DC, 68–73.

WILLIAMSON, D. N.D. ARM Cortex A8: A High Performance Processor for Low Power Applications. ARM.

ZHANG, C., VAHID, F., YANG, J., AND NAJJAR, W. 2005. A way-halting cache for low-energy high-performance systems. *ACM Trans. Archit. Code Optim. 2,* 1, 34–54.

Received June 2013; revised September 2013; accepted November 2013