2009 18th International Conference on Parallel Architectures and Compilation Techniques

Zero-Value Caches: Cancelling Loads that Return Zero

Mafijul Md. Islam and Per Stenstrom
Department of Computer Science and Engineering
Chalmers University of Technology, SE-412 96, Goteborg, Sweden
Email: {mafijul.islam, pers}@chalmers.se
Members of the HiPEAC Network of Excellence

***Abstract*— The speed gap between processor and memory continues to limit performance. To address this problem, we explore the potential of eliminating Zero Loads — loads accessing memory locations that contain the value "zero" — to improve performance and energy dissipation. Our study shows that such loads comprise as many as 18% of the total number of dynamic loads. We show that a significant fraction of zero loads ends up on the critical memory-access path in out-of-order cores. We propose a non-speculative microarchitectural technique — Zero-Value Cache (ZVC) — to capitalize on zero loads and explore critical design options of such caches. We show that with modest investment, we can obtain speedups up to 78% and reduce the overall energy dissipation by up to 39%. Most importantly, zero-value caches never cause performance loss.**

Keywords- Zero-Value Cache, Load Scheduling, Load Criticality, Frequent Value Locality, Zero Load

I. INTRODUCTION

On-chip cache hierarchies are widely used in modern processors to address the gap between processor speed and DRAM access time. While caches are successful in reducing the accesses to memory, even a small fraction of the remaining accesses may cause substantial performance loss. Moreover, despite advances in cache management, load latency is still a limiting factor in achieving high performance in dynamically scheduled processors. This clearly calls for novel techniques to improve efficiency of load scheduling and cache hierarchies.

Typically, a load accesses the first level data cache (DL1) and searches the load/store queue (LSQ) in parallel to find a matching store. If the load does not obtain its value from either source, it accesses the next level of cache hierarchy and experiences longer latency that limits performance. Performance and energy losses increase with the number of visited cache/memory levels. Ideally, if a load request could be satisfied without injecting it into the cache hierarchy, it would not result in performance or energy loss. To this end, we propose and evaluate a non-speculative microarchitectural technique to satisfy load requests as early as possible to reap performance gains and energy savings.

Numerous approaches for load scheduling and execution to enhance performance, power, and energy efficiencies have been proposed. Load-value prediction schemes [4, 5, 19] aim to reduce average memory access time and rely heavily on speculation to predict load values with high accuracy. However, these techniques require detecting misspeculation and consequently, re-executions lead to performance and power loss. On the other hand, store-to-load forwarding techniques [22, 25, 26] can cancel load accesses early if there is an outstanding store to the same address. In contrast, we propose a non-speculative technique that is capable of cancelling load accesses that are not cancelled by store-to-load forwarding techniques.

Our approach leverages *frequent value locality,* as observed by Yang and Gupta [28], to track loads that access zero-valued memory locations. In particular, we establish that a significant fraction of dynamic loads read the value "zero" from the memory. We refer to such loads as *Zero Loads* (ZLDs). Remarkably, on average, 18% of the total number of executed loads are ZLDs for applications of the SPEC CPU2000 benchmark suite. Such loads can be executed without accessing the cache hierarchy, and this opens a number of opportunities. We contribute a new architectural concept referred to as *Zero-Value Cache* (ZVC). Unlike a conventional cache, a ZVC keeps track of zero-valued locations compactly and responds quickly to such load requests. We find that this simple structure can improve performance by up to 78% with a low additional complexity.

While Frequent Value Cache (FVC) [29] can be thought of as a general mechanism of capturing the value zero, it trades energy efficiency for lower performance. The FVC always increases load latency of non-frequent values. In contrast, our proposed ZVC takes a different view on how to leverage frequent value locality by targeting only the value zero as well as ensuring that performance will never suffer and overall energy dissipation will diminish. Several earlier studies have leveraged the prevalence of the value zero for other purposes: Alameldeen and Wood [1], Villa *et al.* [31] and Dusser *et al.* [7] for cache compression, and Ekman and Stenstrom [8] for main memory compression. But no prior study has exploited this phenomenon to cancel load requests before they reach the memory hierarchy as done in this study.

The remainder of the paper is organized as follows: We introduce and characterize zero loads in Section II to motivate the architectural innovation presented in Section III. We describe the experimental methodology in Section IV before we present and analyze the results in Section V. We discuss related work in Section VI and provide concluding remarks in Section VII.

1089-795X/09 $25.00 © 2009 IEEE
DOI 10.1109/PACT.2009.29

II. CHARACTERIZATION OF ZERO LOADS

We first introduce *Zero Loads* (ZLDs) and quantify such loads as fraction of the total number of executed loads. We then discuss the criticality and L2 miss rate of ZLDs to anticipate their impact on performance. This serves as motivation for studying opportunities to exploit ZLDs in terms of architectural innovations.

A. Frequency of Zero Loads

We quantify the occurrences of dynamic zero loads for a set of applications of the SPEC CPU2000 benchmark suite using the experimental methodology presented in Section IV. The results are shown in Fig. 1. The Y-axis represents the percentage of all dynamic loads that are ZLDs. We see that the frequency of ZLDs lies between 5% and 55% of the total number of executed loads, with an arithmetic mean of 18%. It is noteworthy that ZLDs are indeed common in every application.

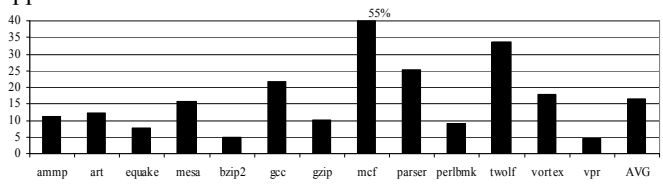

Figure 1. Frequency of zero loads.

This finding provides several optimization opportunities. Most importantly, the latency of a ZLD can be reduced to provide performance and power/energy benefits if the request can be satisfied without issuing it to the cache hierarchy. The gain will be particularly high if a ZLD is not resident in data caches and would have to access main memory. A large fraction of ZLDs actually exhibits this behavior.

B. Criticality of Zero Loads

Traditionally, cache hierarchies exploit locality of references to increase the fraction of memory accesses that can be satisfied by the cache. This, in turn, reduces the average load latency and improves performance. However, Srinivasan *et al.* [24] show that latencies of all load operations do not equally penalize overall performance in a dynamically scheduled, out-of-order processor. Loads that must complete early to avoid performance degradation are *Critical Loads* (CLDs). In this study, we classify a load as *Zero Critical Load* (ZCLD) if a critical load returns zero. Srinivasan *et al.* [24] classify a load as critical if it satisfies any of the following criteria: 1) The load feeds into a mis-predicted branch, 2) The load feeds into another load that incurs an L1 cache miss, or 3) The number of independent instructions issued in an N-cycle window following the load is below a threshold.

We establish the relative occurrences of ZCLDs based on the stated criteria using a window of four cycles and a threshold of four instructions. We also quantify the fraction of L2 misses caused by ZLDs, assuming the cache hierarchy parameters that we present in Section IV. The Y-axis of Fig. 2 shows the frequency of ZCLDs as percentage of the total number of executed ZLDs. The X-axis shows the application name, and the first, second and third rows below show the frequency of ZCLDs as percentage of all dynamic loads, L2 miss per 1000 memory references and the percentage of L2 misses caused by ZLDs, respectively.

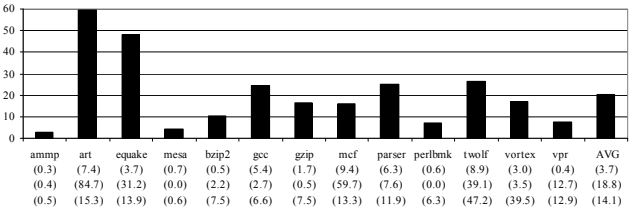

Figure 2. Frequency of ZCLD and L2 miss rate.

The figure shows that about 20% of all dynamic ZLDs are critical and that such loads comprise about 4% of all executed loads. Furthermore, the figure shows that about 14% of all L2 misses are caused by ZLDs. More detailed analysis reveals that (a) ZCLDs constitute a moderately large fraction of all dynamic loads in many applications (art, equake, gcc, mcf, parser, twolf, and vortex), (b) ZLDs contribute significantly to L2 miss rate (at least 10% of total number of L2 misses) in many applications (art, equake, mcf, parser, twolf, vortex, and vpr), and (c) the impact of ZLDs on L2 miss rate and the frequency of ZCLDs are relatively lower in several applications (ammp, mesa, bzip2, and perlbmk).

The results make ZLDs a natural target for optimization. However, it is important to note that the notion of criticality adopted in this study merely indicates that a critical load degrades performance, but does not quantify performance loss caused by such loads. In the next section, we present a novel, non-speculative architectural scheme to exploit ZLDs.

III. ZERO-VALUE CACHES

A *Zero-Value Cache* (ZVC) is a small, cache-like structure dedicated to a subset of memory locations guaranteed to contain the value zero. We say that a load request is a *data hit* in the ZVC if there is an entry for that zero-valued location. Conversely, it is a *data miss* if the entry contains a non-zero value and an *entry miss* if there is no entry for that location. We consider two placements for the ZVC: placed between the CPU and the L1 data cache (DL1) or alongside the DL1. We first present the microarchitecture of the baseline processor model in Section III-A. In the subsequent sections, we discuss the organizational issues of the ZVC and elaborate on its design tradeoffs.

A. Baseline Microarchitectural Model

While ZVC can be integrated in most conventional microarchitectural models, we assume for simplicity a straightforward model with six pipeline stages (fetch, decode, issue, execute, write back, and commit) similar to that of Sun Microsystems' Niagara [16]. Unlike Niagara, however, we assume an out-of-order core. Both instruction decoding and register renaming happen at the decode pipeline stage as in the MIPS R10000 superscalar processor [30]. The entries are allocated in the reorder buffer (ROB),

and the instructions proceed to the issue stage. Instructions execute as soon as dependences are resolved and the required functional units are available. The execution outcome is then broadcast to wake up other dependent instructions at the write back stage. Finally, the resources used by the instructions are reclaimed, and caches and memory are updated at the commit stage.

B. Organization of the ZVC

Each entry of the ZVC stores a tag corresponding to the address of a *B*-Byte block, a bit-vector of that entire block, a Cache Indicator Bit (CIB), and a valid bit, as shown in Fig. 3. Each bit in the bit vector designates whether the corresponding byte or word of a block is zero ('1' represents zero whereas '0' represents unknown/non-zero value). A new design parameter is the granularity to represent zero-valued locations. We evaluate two options: one bit per byte (BB), and one bit per four-byte word (BW).

Tag	Bit-vector	Cache Indicator Bit (CIB)	Valid Bit

Size in bits (Block size in byte):
Tag = 32 - $\log_2$(Number of Sets) + $\log_2$(Block Size),
Bit-vector = (ZVC Block Size) / W, W = 1 if BB or W = 4 if BW,
CIB = (ZVC Block Size) / (L2 Block Size), and
Valid bit = (ZVC Block Size) / (L2 Block Size).

Figure 3. A ZVC entry and its size.

To exploit spatial locality offered by larger blocks for better cache utilization, we assume the proposed ZVC is not inclusive with respect to the conventional cache hierarchy. However, we assume inclusiveness across cache levels in the baseline cache hierarchy. To this end, the CIB indicates whether a particular block of the ZVC is present in the last level cache or not. If not, a load request bypasses the traditional cache hierarchy to reduce miss penalty and power consumption as proposed by Memik *et al.* [20]. Each ZVC entry may have *N* CIBs if the block size of the ZVC is N times larger than the L2 cache block size. Finally, a valid bit serves the same purpose as in a conventional cache and similar to the CIB, each ZVC entry may have *N* valid bits.

C. ZVC placed between the CPU and the DL1

The proposed ZVC can be placed between the CPU and the DL1 to treat it as another level in the cache hierarchy. The advantage is that the load latency is completely eliminated if there is a data hit in the ZVC, since the load can be satisfied immediately. However, this design is constrained by the clock frequency of the pipeline and loads that miss in the ZVC may experience longer latency.

Let us now explain the different possible scenarios on a read access in the presence of the ZVC. At the issue stage of the pipeline, a load request is issued to the ZVC in parallel with carrying out the address translation. First, if there is a data hit in the ZVC, the load is not issued to the cache hierarchy and the LSQ lookup is not done. Second, if either an entry miss or a data miss occurs, the request is issued to the LSQ and the DL1 as it happens in the baseline processor model. However, on a data miss, if the load is not served by the LSQ and the DL1, the status of the corresponding CIB in the ZVC dictates how the request will travel through the rest of the cache/memory hierarchy. If the CIB is '0', i.e., the block is not present in the last level cache (L2 in the baseline model), the main memory is accessed directly without searching the next level caches, the block is installed into the caches and the corresponding CIB is set to '1'. This reduces the miss penalty. Conversely, if the CIB is '1', the load traverses the cache/memory hierarchy as in the baseline processor model. Fig. 4(a) shows the integration of the proposed ZVC into the baseline processor model. In the figure, the additional logic and connections are shown using dotted rectangles and lines, respectively.

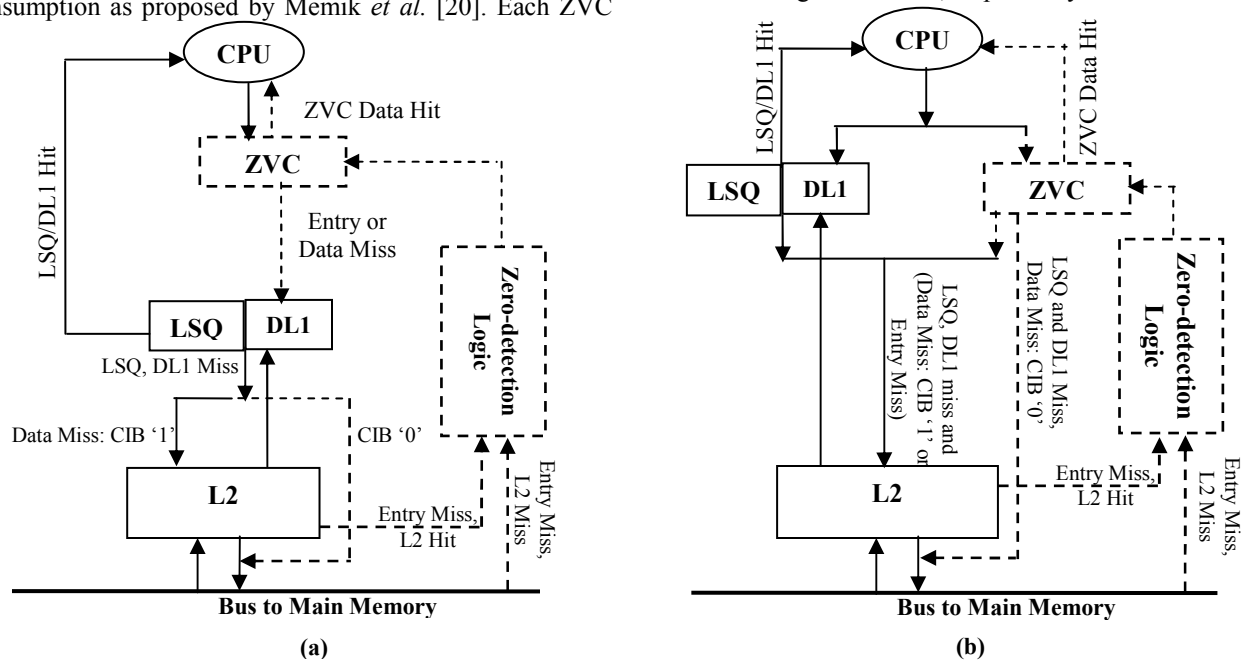

Figure 4. The integration of the ZVC into the baseline processor model.: (a) ZVC placed betweem the CPU and the DL1, (b) ZVC alongside the DL1.

The allocation of a ZVC entry and the modifications of the corresponding bits are done at the issue stage of the pipeline model. First, if there is an entry miss and a miss in the L2, the block is fetched from the main memory at the ZVC block size granularity. The entire block passes through the zero-detection logic to create the bit vector following the specified granularity (BB or BW) and place it in the ZVC. However, only the relevant fraction (L2 block size) of the fetched block is placed in the L2. At this point, the CIBs and valid bits of the newly allocated ZVC entry are updated. This may increase the memory traffic on read misses if the ZVC block size is larger than the L2 block size. Second, if there is an entry miss but a hit in the L2, the block is fetched from the L2 and placed in the ZVC. The bit vector formation and the modifications of the CIBs as well as valid bits are done similarly as in the first case. In both cases, the ZVC stores only non-speculative data. Finally, as soon as an in-flight store completes, i.e., the effective address and the value are known, a new entry is allocated if it is not in the ZVC. Only the bytes for which the value zero is written are marked by '1' in the bit vector. The valid bits are set but the CIBs are not set. This approach allows the ZVC to hold speculative data and support store-to-load forwarding if the load finds a matching store in the ZVC with the corresponding bit(s) of the bit vector set.

Since the proposed scheme is non-speculative, the entries of the ZVC have to be validated to maintain correctness during program execution. During the recovery from the misspeculation, the ZVC is validated against each squashed store. This requires clearing the corresponding bit vector positions of a cached ZVC entry to '0'. At the commit stage of the pipeline, the ZVC lookup is performed for each ready-to-commit store to detect zero silent stores – a special case of silent stores [17] that update zero-valued memory locations with the same value. Thus the proposed ZVC detects zero silent stores basically for free and this may reduce the overall write traffic.

The virtually-indexed ZVC allows the TLB lookup to proceed in parallel. In the proposed placement, this will partially hide the additional latency of accessing the ZVC and the DL1 sequentially. As the ZVC size is quite small, the cost of dealing with virtual aliases will be low.

D. ZVC alongside the Level-1 Cache

The ZVC may alternatively be placed next to the first level data cache (DL1) to perform lookups simultaneously. Fig. 4(b) shows the integration of the proposed ZVC into the baseline processor model with the associated additional logic marked using dotted rectangles and lines. In this case, the LSQ, the ZVC and the DL1 are searched in parallel on a load request. If the request is satisfied by any, the L2 cache is not accessed. But on a ZVC data miss in addition to the LSQ and the DL1 miss, the CIB of the ZVC entry determines whether to access the L2 cache. If the CIB is '1', the L2 cache is accessed as in the baseline processor model. Otherwise, the main memory is accessed directly to bring in the required block to the caches. The allocation of a new ZVC entry, modifications of the bit vector, CIBs and valid bits, and the maintenance of non-speculative nature are performed in the same way as done in the ZVC placed between the CPU and the DL1.

The placement of the ZVC alongside the DL1, unlike the other approach, offers more flexibility in the design as it only needs to be as fast as the L1 cache and does not incur any performance penalty on data/entry miss. However, the performance and energy benefits are now limited to zero loads that access the L2 cache in absence of the ZVC.

E. Threshold-based Insertion/Replacement

We have thus far assumed that a particular block is placed in the ZVC without determining the relative frequency of zero-valued bytes/words in the block. As a result, in the worst case, the ZVC may entirely fill up with blocks that primarily consist of non-zero values. To address this issue, we propose and evaluate a scheme that inserts a block into the ZVC only if the number of zero-valued bytes/words is above a preset threshold. Though the threshold-based approach requires extra hardware to count the number of zero bytes/words, it may increase ZVC utilization. It is notable that this approach is not applicable if the ZVC is updated via an in-flight store. We assume the LRU replacement policy in all cases.

F. Cache Coherence Issues

Since inclusion between the ZVC and conventional cache hierarchy is not maintained, all cache coherence transactions will travel all the way to the ZVC through the cache hierarchy. This may pose severe performance loss in multi-cores. However, we show in Section V that even a small ZVC is effective in cancelling a large number of loads destined to zero-valued locations. Therefore, it is reasonable to duplicate the ZVC at the last level private cache. We assume that the content of the duplicated ZVC is consistent with the ZVC placed between the CPU and the DL1 or next to the DL1. The replicated ZVC filters out all cache coherence transactions not involving blocks contained in the ZVC. However, proper actions must be taken, including invalidating ZVC entries for the blocks involved in a cache coherence transaction. Note that we do not evaluate the impact of the ZVC on multi-cores.

IV. Experimental Methodology

We evaluate the performance potential of the proposed mechanism via a modified version of *sim-outorder* from the SimpleScalar toolset [3]. The baseline processor model supports dynamic scheduling and speculative execution. We describe the pipeline model in Section III-A. We modify a memory implementation [10] to model SDRAM-based memory subsystem and integrate it into the SimpleScalar toolset. We model the corresponding controller and latencies, including contention according to JEDEC Standard No. 79-3C [13]. We present the parameters of the baseline processor model and memory subsystem in Table I. In addition, we perform sensitivity analysis of the proposed method using a larger L2 cache.

TABLE I. THE BASELINE PROCESSOR PARAMETERS.

Parameter	Value
Decode, Issue and Commit Width	4 instructions/cycle, out-of-order issue and execution
Register Update Unit (RUU), LSQ	(128, 64) instructions
L1 Instruction Cache	32KB, block size 32 bytes, 4-way set-associative, 256 sets, LRU replacement policy, 2 cycles latency
L1 Data Cache	32KB, block size 32 bytes, 4-way set-associative, 256 sets, LRU replacement policy, 2 cycles latency
Unified L2 Cache	512KB, block size 64 bytes, 8-way set-associative, 1024 sets, LRU replacement policy, 10 cycles latency
Branch Predictor	Combined: 2K-entry Bimodal + gshare with 2K entries in 2nd level; RAS: 8-entry; BTB: 4-way, 512 sets
Memory subsystem	Number of banks: 8, DIMM width: 64 bits, Number of SDRAM chips: 8, Capacity: 2 Gbits, Memory clock speed: 200 MHz, Bus clock speed: 800 MHz, Address map: interleaved, Control signals: pipelined, Latencies: 9-9-9-24 in the format "tCAS-tRCD-tRP-tRAS"

We use nine applications (bzip2, gcc, gzip, mcf, parser, perlbmk, vortex, vpr, and twolf) from the SPEC CINT2000 and four applications (ammp, art, equake, and mesa) from the SPEC CFP2000 of the SPEC CPU2000 benchmark suite. We compile each with optimization level –O3 and *gcc* version 2.7.2 for a MIPS-compatible processor. We omit two remaining C applications (gap and crafty) due to compilation problems. We use the reference inputs for all the selected applications. We use SimPoint3.2 to quickly simulate parts of a program's execution (100M instructions) to represent the entire execution [23]. Table II shows each selected application, the simulation point and the input set used in this study.

TABLE II. APPLICATIONS AND INPUT SETS.

Application	SimPoint	Input
ammp	846	ammp-ref
ert	257	art-470
equake	985	equake-ref
mesa	3909	mesa-ref
bzip2	384	bzip2-program
gzip	567	gzip-source
gcc	116	gcc-166
mcf	321	mcf-ref
parser	24	parser-ref
perlbmk	64	perlbmk-diffmail
twolf	1402	twolf-ref
vortex	696	vortex-lendian1
vpr	368	vpr-route

We incorporate the proposed ZVC designs into our processor model and evaluate their impact on performance, power, energy, chip-area and memory traffic. We evaluate many configurations of the ZVC, as shown in Table III. In the table, the ZVC size includes CIB and valid bit. In each case, the ZVC is 4-way set-associative with the LRU replacement policy.

TABLE III. CONFIGURATIONS OF THE ZVC.

Bit-vector	Block Size (in byte)	Threshold	ZVC Size (in byte)
BB, BW	64, 128, 256	0%, 50%, 100%	576, 2112, 4224

We use CACTI 5.3 [27] to estimate power consumption, access latency, and chip-area overhead of the ZVC, assuming a 2-GHz processor clock speed and 45nm technology. We present the estimated access time of the various configurations of the ZVC in Table IV. Note that all designs are accommodated within a single cycle for a 2-GHz processor core. We use Wattch [2] to estimate dynamic power and energy dissipation. We assume 1-cycle latency for each of zero-detection logic and the counter used in the threshold-based approach.

TABLE IV. ACCESS TIME (IN NS) OF THE ZVC.

Block Size	Bit-vector	Number of Sets	ZVC Size (in byte)	Access Time
64	BB	64	2112	0.481577
128	BB	32	2112	0.476364
256	BB	16	2112	0.476364
256	BB	32	4224	0.476364
256	BW	16	576	0.447053

V. RESULTS AND ANALYSIS

In this section, we present and discuss the results that we obtain through systematic exploration of the ZVC design space. Our default ZVC is four-way set-associative and assumes LRU replacement policy.

A. Impact of Placement on Performance

We evaluate the impact of ZVC placement and lookup on speedup using three different configurations and present the results in Fig. 5. There are three bars for each application in the figure and from left to right, they represent the speedup obtained using a 2KB ZVC with 128-byte blocks placed between the CPU and the first level cache (L0, 2KB), a 2KB ZVC with 128-byte blocks (L1, 2KB) and a 4KB ZVC with 256-byte blocks (L1, 4KB) alongside the L1. The estimated access latency is one cycle in the first case whereas it is two cycles in the other cases. Each ZVC model forms bit vector using byte granularity (BB).

We see from Fig. 5 that four applications (art, gcc, mcf, and twolf) achieve significant speedup (between 12% and 80%), three applications (equake, parser, and vortex) obtain moderate speedup (between 3% and 7%), and most importantly, no application experiences performance degradation. We observe that the ZVC placed alongside the L1, in general, performs slightly better than the ZVC placed

between the CPU and the L1 in all applications except vortex.

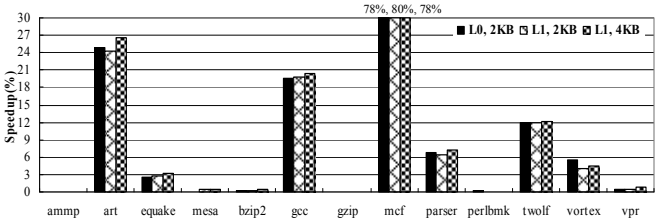

Figure 5. Impact of ZVC placement on speedup.

We observe that the speedup obtained is in line with the expectation if we recall the results on criticality and L2 miss rate of ZLDs presented in Fig. 2 in Section II-B. For example, in mcf, critical ZLDs constitute 9.4% of all dynamic loads and ZLDs contribute 13.3% to the overall L2 miss rate, and those are only 0.3% and 0.5%, respectively in ammp. We look into the data hit rate of the critical zero loads (ZCLD) of the 4KB ZVC placed alongside the L1 to understand the observed trend and present the results in Fig. 6. In the figure, the solid line represents the speedup in percentage and the dotted line represents the data hit rate of the ZCLDs as percentage of the total number of executed loads. In the X-axis, the row below the application name shows the data hit rate of the ZCLDs as percentage of all dynamic ZCLDs. We see from Fig. 6 that the higher data hit rate of ZCLDs leads to higher gains in performance and the proposed ZVC tracks majority of the executed ZCLDs in most applications (mesa, gcc, gzip, mcf, parser, perlbmk, twolf, vortex, and vpr).

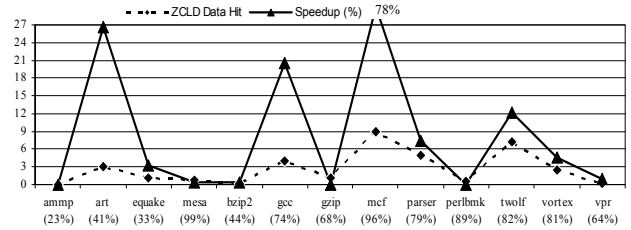

Figure 6. Data hit rate of the ZCLD.

The results discussed in this section reveal that the ZVC placed next to the L1 to perform lookups simultaneously, in general, performs better. More importantly, this design ensures that non-zero loads – about 82% of all dynamic loads – do not suffer from any performance penalty.

B. *Impact of ZVC Block Size on Performance*

We assume that the ZVC is searched in parallel with the L1 and evaluate the impact of its block size on speedup. We use BB to form bit vectors. Fig. 7 shows the results. In the figure, the three bars from left to right correspond to the obtained speedup in percentage for block sizes of 64, 128, and 256 bytes, respectively for a 2KB ZVC.

It is evident from Fig. 7 that the speedup increases in several applications (art, equake, gcc, parser, and vortex) as we increase the block size. However, we do not observe any noticeable impact on speedup in most of the applications (ammp, mesa, bzip2, gzip, perlbmk, twolf, and vpr) and observe a slight decrease in speedup in mcf. In general, the ZVC with 256-byte block size performs the best.

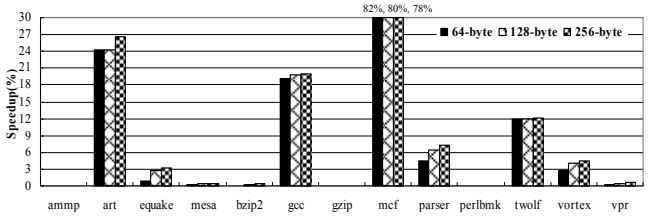

Figure 7. Impact of ZVC block size on speedup.

C. *Impact of Bit-vector Granularity*

We assess the impact of the granularity to represent zero-valued locations in the bit vector on speedup by considering byte (BB – 1 bit/byte) and word (BW – 1 bit/word) granularities. We assume that the ZVC is placed next to the L1 and uses 256-byte blocks. In Fig. 8, the left and the right bars correspond to the speedup obtained using BB and BW, respectively. The results in the figure reveal that speedup is almost entirely independent of the granularity.

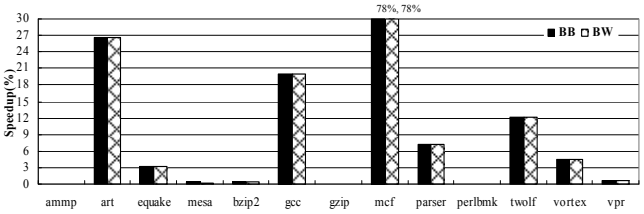

Figure 8. Speedup and bit-vector granularity.

To understand this trend, we quantify the relative occurrences of different granularities (byte, half-word, word) of ZLDs and present the results in Fig. 9. In the figure, the three bars from bottom to top correspond to the relative occurrences of byte (1-byte), half-word (2-byte) and word (>=4-byte) ZLDs, respectively as percentage of all dynamic ZLDs. We find that byte-ZLDs are dominant – at least 60% of all dynamic ZLDs – only in three applications (bzip2, gzip, and twolf). The results suggest that we use BW because it reduces the ZVC size by a factor of four with respect to BB and provides almost same performance benefit as BB.

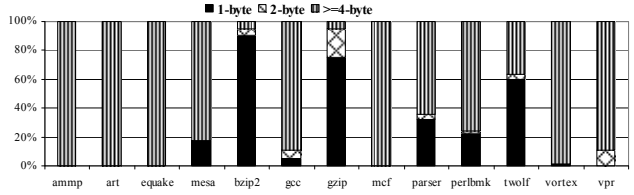

Figure 9. Granularity of dynamic zero loads.

D. *Impact of the Zero-Count Threshold*

In this section, we assume that a particular block is placed in the ZVC only if it satisfies a certain threshold of zero-count and evaluate the impact on speedup. For example, when the threshold is N, a block is placed in the ZVC if the

block contains N% zero words. The ZVC with 256-byte blocks is searched in parallel with the L1 and uses BW to form bit vectors. We apply three different thresholds and present the results in Fig. 10. In the figure, there are three bars for each application and from left to right, they correspond to the achieved speedup by applying no threshold (0%), thresholds of 50%, and 100%, respectively. Note that a threshold of 0% means that we do not count the number of zero-valued words in a particular block before placing it in the ZVC. We see from Fig. 10 that speedup decreases in most applications as we increase the threshold from 0% to 100%.

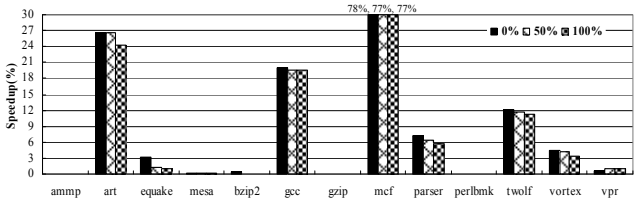

Figure 10. Impact of threshold on speedup.

E. Energy, Power and Area of the ZVC

We estimate the efficiency of our proposed ZVC in terms of energy, power and chip-area. We evaluate a ZVC that sits alongside the L1, applies no zero-count threshold, is 4-way set-associative with 16 sets, and uses 256-byte blocks. We assume that BW is used to create bit vectors. Thus, the size of the ZVC is 576 bytes including CIB and valid bit. We use CACTI 5.3 [27] to obtain the results and present those in Table V. We see from the table that the ZVC increases leakage power only by 0.18% and chip-area only by 0.15% with respect to data caches.

TABLE V. THE OVERHEAD OF THE ZVC.

Parameter	DL1	L2	ZVC	Overhead
Dynamic read energy (nJ)	0.0163	0.0903	0.0024	2.25%
Dynamic write energy (nJ)	0.0134	0.0444	0.0026	4.5%
Dynamic read power (nW)	46.872	74.048	16.209	13.4%
Leakage per bank (mW)	66.27	855.14	1.7037	0.18%
Area (mm^2)	0.288	3.621	0.0057	0.15%

We also use Wattch [2] to estimate the impact of the ZVC on the overall dynamic power consumption and energy dissipation. The proposed ZVC reduces the power consumption in two ways: (a) the L2 cache is not accessed on a read request if it is a data hit in the ZVC but misses in the L1, and (b) data caches are not accessed on a write request if the ZVC detects the store as a zero silent store. However, the ZVC itself consumes power. In addition, the ZVC accesses the L2 cache to maintain CIB, and fetch the required block on entry miss and L2 hit. This may increase the overall power consumption. We estimate that the dynamic power consumption of the baseline processor model increases by 3%. The ZVC may reduce the overall energy dissipation in the applications that achieve speedup and increase it in the other applications. We estimate the total dynamic energy dissipation and show the results in Fig. 11. We see from the figure the total energy usage of the processor is reduced in many applications (art, equake, gcc, mcf, parser, twolf, and vortex) and increased slightly (about 1%) in some applications (ammp, bzip2, gzip, perlbmk, and vpr).

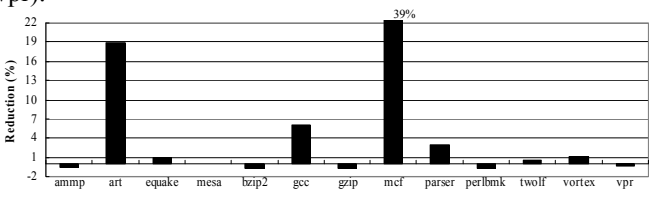

Figure 11. Impact of the ZVC on dynamic energy.

F. Impact of the ZVC on Memory Traffic

Memory bandwidth is a bottleneck in current and future processors [9]. This motivates us to evaluate the impact of the proposed ZVC on memory traffic. If the ZVC blocks are larger than the L2 blocks, we expect that the read traffic may increase as an entry miss in the ZVC along with L2 miss fetches additional data from the memory. In contrast, the write traffic may decrease as the ZVC is capable of detecting zero silent stores. We observe that such stores, on average, constitute about 11% of the total committed stores. It is important to note that only a critical silent store — a specific dynamic silent store that, if not squashed, will cause a cache line to be marked as dirty and, hence, require a writeback [17] — suppresses write traffic.

We assume the ZVC is placed next to the L1, is 4-way set-associative with 16 sets and 256-byte blocks, uses BW to form bit vectors and applies no threshold on zero-count. The L2 block size is 64 bytes. We present the results in Table VI. Note that a particular number following the '-' sign implies increase in traffic. We see from Table VI that memory read traffic increases up to 33% and all applications except gcc experience increase in the read traffic. We also observe that the memory write traffic decreases between 1% and 66% in seven applications (art, mesa, gcc, gzip, mcf, parser, and vortex) and remains unaffected in the remaining six applications. We find that the proposed ZVC detects about 62% of the total number of committed zero silent stores, i.e., about 9% of all committed stores.

TABLE VI. IMPACT OF THE ZVC ON MEMORY TRAFFIC (READ AND WRITE).

Application	ammp	art	equake	mesa	bzip2	gcc	gzip	mcf	parser	perlbmk	twolf	vortex	vpr
Read Traffic	-29	-17	-6	-14	-20	14	-11	-33	-4	-22	-25	-26	-31
Write Traffic	0	29	0	1	0	66	5	57	12	0	0	17	0

G. Sensitivity Analysis

Processor vendors such as IBM and Intel are incorporating large on-chip L2 cache [11, 18]. This motivates us to appraise our proposed ZVC in relation to the size of the L2 (last level) cache. We use a 4MB L2 cache that is 8-way set-associative with 128-byte blocks and 10-cycle access latency. The 576-byte ZVC placed next to the L1 is 4-way set-associative and uses 256-byte blocks with BW to create bit vectors. Fig. 12 shows the results. We see from Fig. 12 that the speedup is reduced if we use a 4MB L2 cache. In particular, the performance advantage disappears completely in art. We analyze the miss rates of both L2 caches to understand the observed differences in speedup and present the results in Table VII. We see from Table VII that the miss rate is reduced significantly in the 4-MByte L2 cache and in fact, it goes down to almost zero in art. This justifies the observed drop in speedup.

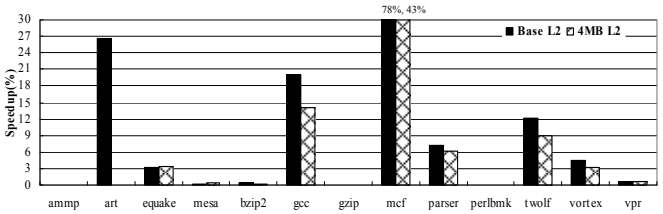

Figure 12. Sensitivity analysis of the ZVC.

TABLE VII. THE MISS RATE (MISS PER MEMORY REFERENCE) OF 512KB L2 (BASE) AND 4MB L2.

Application	ammp	art	equake	mesa	bzip2	gcc	gzip	mcf	parser	perlbmk	twolf	vortex	vpr
Base L2	0.08	0.56	0.22	0.00	0.03	0.04	0.01	0.45	0.06	0.00	0.08	0.01	0.10
4MB L2	0.02	0.00	0.14	0.00	0.00	0.03	0.01	0.28	0.01	0.00	0.01	0.00	0.02

VI. RELATED WORK

Previous research focuses on efficient techniques to improve memory performance. Lipasti *et al.* [19] recognize that load instructions exhibit *value locality* and conclude that there is potential for prediction. Last-value predictors, stride predictors, context predictors, and hybrid predictors have been proposed to predict load values [4, 5, 19]. Calder and Reinman perform a comparative study of load speculation techniques such as value prediction, address prediction, dependence prediction, and memory renaming [5]. Roth [21] proposes a filtering mechanism to reduce the re-execution overhead of a given speculative technique. Ceze *et al.* [6] and Kirman *et al.* [15] propose checkpointing to speculatively retire long-latency load accesses and unclog the ROB. On the other hand, store-to-load forwarding techniques [22, 25, 26] address non-scalability and power inefficiency of conventional LSQ designs. Unlike load speculation, checkpoint assisted load retirement, and store-to-load forwarding schemes, our proposed ZVC is non-speculative, simpler in terms of hardware resources, and efficient from performance and energy perspectives.

Kin and Mangione-Smith [14] introduce the *filter cache*, a small cache placed in between CPU and L1, to achieve power reduction. This, however, increases the execution time of programs in contrast to our proposed ZVC. The *Frequent Value Cache* (FVC) presented by Yang and Gupta [29] encodes frequent values in a compressed format. While frequent values are accessed directly, accessing a non-frequent value results in performance loss. In contrast, the ZVC proposed in this study is sufficiently small to be accessed in parallel with the L1 or even before the L1. The non-zero loads do not suffer from any speedup penalty. This is important because about 82% of all dynamic loads return non-zero values.

Several recent techniques exploit the potential of zero-valued memory locations [7, 12]. In our previous work [12], we observe that zero loads are common in certain integer applications, and study the upper limit of potential benefits of exploiting such loads. Concurrently, Dusser *et al.* [7] propose *Zero-Content Augmented* (ZCA) cache to avoid storing a block in conventional caches if the entire block contains zero values. While their proposal can improve cache utilization, it cuts down the load latency only if the entire block contains zero values. On the other hand, our proposed ZVC can be placed close to the pipeline allowing loads to be cancelled immediately.

VII. CONCLUSION

In this paper, we introduce the notion of Zero Loads to improve load scheduling and cache hierarchy efficiency to achieve performance and energy benefits. We observe that Zero Loads, on average, comprise about 18% of the total number of executed loads and about one fifth of them appear on the critical memory-access path of out-of-order cores. We propose and evaluate a novel, non-speculative micro-architectural technique — Zero-Value Caches — to exploit such loads. We find that our proposed scheme with additional storage of only about 576 bytes improves performance by up to 78% and reduces the overall energy dissipation of the processor core by up to 39%. Most importantly, the concept of zero-value cache requires only modest resources and never causes performance loss.

ACKNOWLEDGMENT

This research is sponsored by the SARC project under the EU funded FET program. Authors are members of the HiPEAC Network of Excellence. Authors are indebted to Sally A McKee for her comments on earlier drafts of the manuscript. Authors would like to also thank the colleagues in the research group for their input.

REFERENCES

[1] A. R. Alameldeen and D. A. Wood, "Adaptive Cache Compression for High-Performance Processors," *In Proc. ISCA-31*, pp. 212-223, 2004.

[2] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *In Proc. ISCA-27*, pp. 83-94, 2000.
[3] D. Burger and T. Austin, "*The SimpleScalar Tool Set Version 2.0*," Technical Report TR-CS-97-1342, University of Wisconsin-Madison, 1997.
[4] M. Burtscher and B. G. Zorn, "Hybrid Load-Value Predictors," *IEEE Transactions on Computers*, 51(7): 759 – 774, 2002.
[5] B. Calder and G. Reinman, "A Comparative Survey of Load Speculation Architectures," *Journal of Instruction-Level Parallelism*, 2000(1): 1-39.
[6] L. Ceze, K. Strauss, J. Tuck, J. Torrellas, and J. Renau, "CAVA: Using Checkpoint-assisted Value Prediction to Hide L2 Misses," *ACM Trans. Archit. Code Optim.* 3(2): 182-208, 2006.
[7] J. Dusser, T. Piquet, and A. Seznec, "*Zero-Content Augmented Caches*," Technical Report RR-6705, INRIA, October, 2008.
[8] M. Ekman and P. Stenstrom, "A Robust Main-Memory Compression Scheme," *In Proc. ISCA-32*, pp. 74-85, 2005.
[9] D. Burger, J. R. Goodman, and A. Kagi, "Memory Bandwidth Limitations of Future Microprocessors," *In Proc. ISCA-23*, pp. 78-89, 1996.
[10] M. Gries and A. Romer, "*Performance Evaluation of Recent DRAM Architectures for Embedded Systems*," TIK Report Nr. 82, Swiss Federal Institute of Technology (ETH), 1999.
[11] Intel Corporation, "Introducing the 45nm Next-Generation Intel® Core™ Microarchitecture," *Technology@Intel Magazine*, 4(10), 2007.
[12] M. M. Islam and P. Stenstrom, "Zero Loads: Canceling Load Requests by Tracking Zero Values," *In Proc. MEDEA 2008 Workshop*, October, 2008.
[13] JEDEC, "*JEDEC Standard No. 79-3C*," November, 2008.
[14] M. G. J. Kin and W. H. Mangione-Smith, "The Filter Cache: An Energy Efficient Memory Structure," *In Proc. MICRO-30*, pp. 184-193, 1997.
[15] N. Kirman, M. Kirman, M. Chaudhuri, and J. F. Martinez, "Checkpointed Early Load Retirement," *In Proc. HPCA-11*, pp. 16-27, 2005.
[16] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded SPARC Processor," *IEEE Micro*: 21-29, 2005.
[17] K. M. Lepak, G. B. Bell, and M. H. Lipasti, "Silent Stores and Store Value Locality," *IEEE Transactions on Computers*, 50(11): 1174-1190, 2001.
[18] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D.Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden, "IBM POWER6 microarchitecture," *IBM Journal of Research and Development*, 51(6): 639 - 662, 2007.
[19] M. H. Lipasti, C. B Wilkerson, and J. P. Shen, "Value Locality and Load Value Prediction," *In Proc. ASPLOS*, pp. 138-147, 1996.
[20] G. Memik, G. Reinman, and W. H. Mangione-Smith, "Just Say No: Benefits of Early Cache Miss Determination," *In Proc. HPCA-9*, pp. 307-316, 2003.
[21] A. Roth, "Store Vulnerability Window (SVW): A Filter and Potential Replacement for Load Re-Execution," *Journal of Instruction Level Parallelism*, vol. 8, September 2006.
[22] T. Sha, M. M. K. Martin, and A. Roth, "NoSQ: Store-Load Communication without a Store Queue," *IEEE Micro*, vol. 27, no. 1, pp. 106-113, Jan./Feb. 2007.
[23] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically Characterizing Large Scale Program Behavior," *In Proc. ASPLOS*, pp. 45-57, 2002.
[24] S. T. Srinivasan, R. D. Ju, A. R. Lebeck, and C. Wilkerson, "Locality vs. Criticality," *In Proc. ISCA-28*, pp. 132-143, 2001.
[25] S. S. Stone, K. M. Woley, and M. I Frank, "Address-Indexed Memory Disambiguation and Store-to-Load Forwarding," *In Proc. MICRO-38*, pp. 171–182, 2005.
[26] S. Subramaniam and G. H. Loh, "Fire-and-Forget: Load/Store Scheduling with No Store Queue at All," *In Proc. MICRO-39*, pp. 273–284, 2006.
[27] S. Thoziyoor, N. Muralimanohar, J. H. Ahn, and N. P. Jouppi, "*CACTI 5.1*," Technical Report HPL-2008-20, 2008.
[28] J. Yang and R. Gupta, "Frequent Value Locality and its Applications," *ACM Transactions on Embedded Computing Systems*, 1(1):79–105, Nov 2002.
[29] J. Yang and R. Gupta, "Energy Efficient Frequent Value Data Cache Design," *In Proc. MICRO-35*, pp. 197-207, 2002.
[30] K. C. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, 16(2): 28-40, 1996.
[31] L. Villa, M. Zhang, and K. Asanovic, "Dynamic Zero Compression for Cache Energy Reduction," *In Proc. of the MICRO-33*, pp. 214–220, 2000.