Synonymous Address Compaction for Energy Reduction in Data TLB

Chinnakrishnan S. Ballapuram
chinnak@ece.gatech.edu

Hsien-Hsin S. Lee
leehs@ece.gatech.edu

Milos Prvulovic[†]
milos@cc.gatech.edu

School of Electrical and Computer Engineering
College of Computing[†]
Georgia Institute of Technology, Atlanta, GA 30332

ABSTRACT

Modern processors can issue and execute multiple instructions per cycle, often performing multiple memory operations simultaneously. To reduce stalls due to resource conflicts, most processors employ multi-ported L1 caches and TLBs to enable concurrent memory accesses. In this paper, we observe that data TLB lookups within a cycle and across consecutive cycles are often *synonymous* — they go to the same page. To exploit this finding, we propose two new mechanisms — *intra-cycle compaction* and *inter-cycle compaction* of address translation requests in order to save energy in the data TLB. Our results show that average energy savings of 27% using intra-cycle, 42% using inter-cycle in a conventional d-TLB, and 56% using inter-cycle compaction in semantic-aware d-TLBs can be achieved. When these 2 compaction techniques are combined together and applied to both the i-TLB and semantic-aware d-TLBs, an average energy savings of 76% (up to 87%) is obtained.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*Associative memories, Cache memories, Virtual memory.*

General Terms

Design, Experimentation, Performance.

Keywords

Low-power TLB, Spatial and temporal locality, Multi-porting.

1. INTRODUCTION

Multi-issue superscalar processors have become the de facto standard not only for high performance computing but also in embedded computing platforms. These sophisticated processors issue and execute multiple instructions per cycle and rely on accurate branch predictors, multiple address generation units (AGU), and multi-ported translation lookaside buffers (TLBs) and caches to keep the processor supplied with instructions and data. With virtual memory support, address translation must also be done for instruction and data fetches. Due to the different needs of virtual memory management and cache coherency maintenance, most caches are either physically indexed and physically tagged (PIPT), or virtually indexed and physically tagged (VIPT). In both cases, an address translation using the TLB is needed for each access. Multiple instructions issued in each cycle require multi-ported instruction TLBs and data TLBs to avoid stalls due to resource conflicts. Additionally, TLBs are typically organized as a fully-associative cache to eliminate memory intensive page walks due to conflict misses. As a result, TLBs are often implemented as content-addressable memory (CAM), where all CAM cells are probed and compared to find a match each time a TLB access is initiated. Measured data [5, 6] from commercial processors such as Intel's StrongARM and Hitachi's SH-3 indicates that as much as 17% on-chip power is consumed in the TLBs with an escalating trend.

In this paper, we analyze the access pattern of memory operations performed within a cycle and in successive cycles, and exploit the characteristics of the addresses for energy reduction opportunities In particular, we find that concurrent and consecutive memory operations demonstrate very high locality and are often *synonymous* — accessing the same memory page. As a result, a single TLB lookup often suffices to find the correct translation for multiple accesses in the same cycle, thus eliminating redundant look-ups that could draw additional power. Similarly, the most recently accessed data TLB entry can be latched and reused in subsequent TLB look-ups. We propose two new hardware-based mechanisms that exploit this behavior to reduce the number of TLB lookups. These mechanisms are complexity-effective and power-efficient, with minimal impact on the hardware budget. In addition to reducing power, these mechanisms can also be used, when chip area is a concern, to reduce the number of TLB ports.

The rest of this paper is organized as follows: Section 2 motivates our work by characterizing data memory references, Section 3 presents intra-cycle and inter-cycle address compaction mechanisms, Section 4 presents our simulation results, Section 5 discusses related work, and Section 6 presents our conclusions.

2. MOTIVATION

Using MiBench [3] and SPEC CPU2000 benchmarks, Figure 1 shows that more than 40% of dynamic instruc-

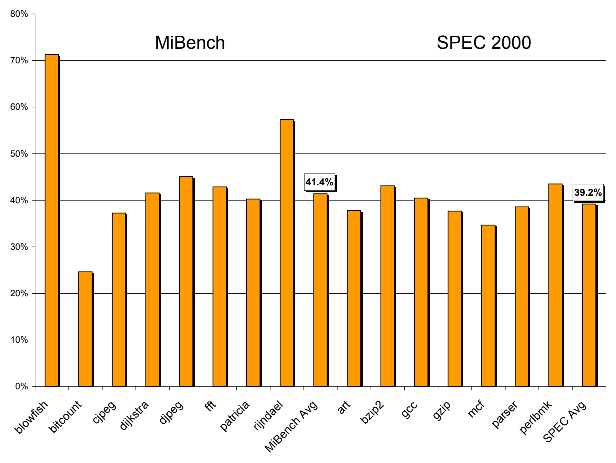


Figure 1: Dynamic memory references as a fraction of all dynamic instructions

tions executed in a program are memory references,[1] in other words, there is a memory instruction for almost every other instruction issued. Therefore, in a superscalar processor, multiple memory operations will be performed concurrently in each cycle, using multiple AGUs, larger memory order buffers for address disambiguation, and multiported TLBs and caches. To study the behavior of memory references in a given cycle (intra-cycle) and in consecutive cycles (inter-cycle), we examine the distribution of dynamic memory accesses.[2]

2.1 Intra-cycle behavior of memory references

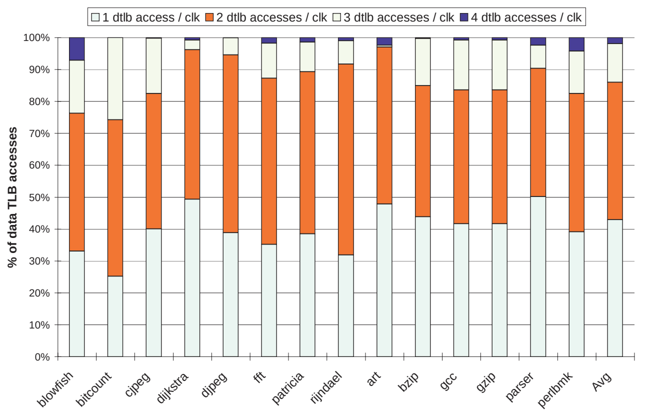


Figure 2: Breakdown of d-TLB accesses

Figure 2 shows the breakdown of data TLB accesses according to the number of concurrent references per cycle. On average for 58% of accesses, the processor issues more than one data TLB lookup in a cycle, that requires a multiported TLB to avoid stalls. An interesting property of these simultaneous memory accesses is that they often access the same page (thus using the same virtual-to-physical translation in the TLB). We call these *intra-cycle synonymous* accesses.

Figure 3 shows the breakdown of memory accesses according to the number of d-TLB access synonyms. In the figure, *syn(0)* means no access synonym, i.e., all memory references in the intra-cycle are unique. An access is *syn(1)*, when one other access in the intra-cycle is its synonym, i.e., the same page is used by two memory references in the same cycle. More generally, *syn(N)* is when there are N other memory references in the intra-cycle accessing the same page. Within each *syn(N)* group, accesses are further broken down according to the number of memory references per cycle, yielding a total of 10 categories. As the simulated machine is 4-wide, a maximum of 4 memory accesses can be issued in each cycle. The bottom four segments in each bar represent non-synonymous, i.e., *syn(0)*, accesses in 1, 2, 3, or 4 simultaneous memory accesses per cycle. The next 3 bar segments represent *syn(1)* accesses in 2, 3, or 4 memory references per cycle. Similarly, the next 2 bar segments show the portion of *syn(2)* accesses and, finally, the top bar segment represents *syn(3)* accesses that can occur only when there are 4 memory operations issued in the same cycle.

As shown in Figure 3, 30% of references have synonyms,

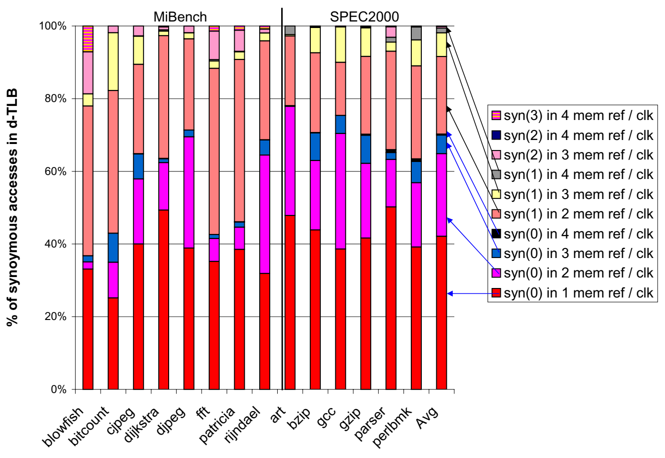


Figure 3: Breakdown of synonymous intra-cycle accesses in d-TLB

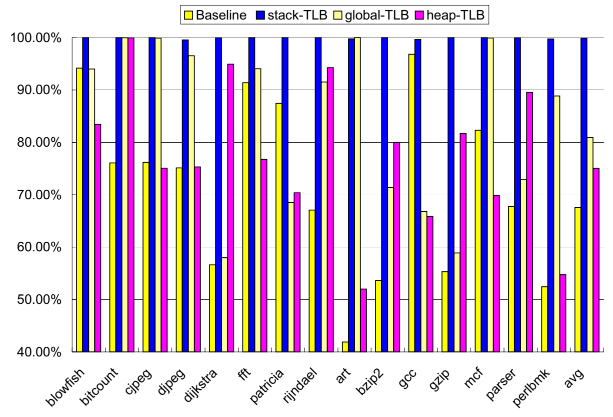


Figure 4: Inter-cycle reuse of d-TLB translations.

[1]We ran all the benchmark programs to completion except for `art` which stopped at 500 billion instructions. For SPEC2000, reference inputs were used. We randomly selected the programs from each benchmark suite without prejudice for figure's readability. Our full SPEC2000INT and FP benchmark runs show 40% and 41% of dynamic instructions are memory references.

[2]A 4-wide machine with 4KB pages is simulated.

indicating that there is access redundancy that can be eliminated by using a single d-TLB lookup to satisfy these synonymous accesses. With this technique, we can remove d-TLB lookups for $\frac{1}{2}$ of all the *syn(1) accesses*, $\frac{2}{3}$ of all the *syn(2) accesses*, and $\frac{3}{4}$ of all the *syn(3) accesses*.

2.2 Inter-cycle behavior of memory references

Another congruent memory reference behavior that occurs during program execution is when two consecutive memory references go to the same page. We call these type of memory references as *inter-cycle synonymous accesses*. Inter-cycle synonymous accesses can be exploited by a simple mechanism that detects the reuse of immediately preceding address translations. Our mechanism keeps the most recently accessed TLB translation and reuses it if the next access is synonymous. As shown in the leftmost bar (baseline) of Figure 4, using a fully-associative TLB 76% of accesses could reuse the last address translation. This rate of reuse can be further increased if the data TLB is horizontally segregated into discrete differentially-sized TLBs based on semantic regions — stack, global, and heap, as proposed in [8]. As shown in Figure 4, the stack-TLB shows almost perfect inter-cycle reuse of nearly 99%, followed by 82% for the global-TLB and 80% for the heap-TLB. In addition to improving the probability of reuse, these semantic-aware d-TLBs also allow reuse detection to be applied selectively, that will be shown in experimental results section.

3. VPN COMPACTION MECHANISMS

Intra-cycle, and inter-cycle synonymous memory references provide an opportunity to reduce the energy consumed by TLBs. In this section, we describe two orthogonal virtual address compaction techniques that make use of these properties. Figure 2 and Figure 3 show that 58% of memory references have companions in the same cycle, and even three or four memory references per cycle are not uncommon, 14% on average. Furthermore, access locality can be very high and synonymous accesses may look up the same memory page or even the same cache line in the same cycle or in consecutive cycles.

3.1 Overview of VPN Compaction

cycle i	0xdeadbeee	0xdeadbeef	0xdeadbef0	0xfffffff
cycle (i+1)	0xdeadbef2	0xdeadbef3	0x12345678	—

Table 1: Virtual address access sequence

cycle i	0xdeadb	0xdeadb	0xdeadb	0xfffff
cycle (i+1)	0xdeadb	0xdeadb	0x12345	—

Table 2: VPN translation lookup in d-TLB

Table 1 shows an example of two back-to-back execution cycles for a 4-issue machine with 4KB pages and a 4-ported d-TLB. The corresponding virtual page numbers (VPN) looked up in d-TLB are shown in Table 2. This example will be used in the following sections to illustrate our compaction mechanisms.

3.1.1 Intra-cycle compaction

Combining same-cycle synonymous lookups into one can be regarded as *compaction* of virtual page numbers called *intra-cycle compaction*. This compaction is shown in Table 3, where in the first cycle (cycle i) three lookups (for addresses 0xdeadbeee, 0xdeadbeef, and 0xdeadbef0 in Table 1) can be compacted into one and, similarly, in cycle (i+1) two VPN lookups can be compacted into one. As shown, after intra-cycle compaction only two TLB accesses are needed in each cycle, that saves power and reduces the number of required d-TLB ports. To perform intra-cycle compaction, dedicated logic to be discussed in Section 3.2.1 is designed to detect access synonyms and eliminate redundant d-TLB accesses.

cycle i	0xdeadb	—	—	0xfffff
cycle (i+1)	0xdeadb	—	0x12345	—

Table 3: VPNs after intra-cycle compaction

3.1.2 Inter-cycle compaction

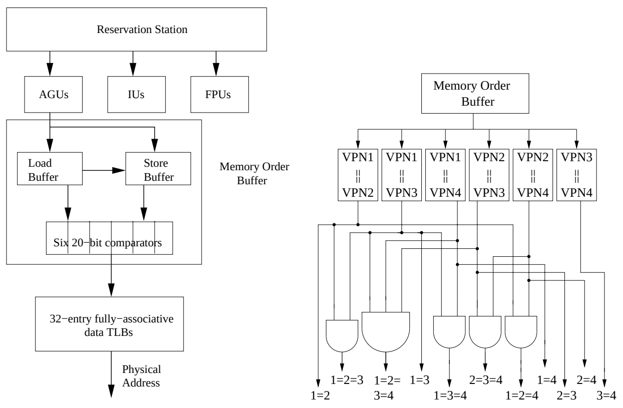
The reuse of address translations in consecutive cycles can also be regarded as *compaction* of virtual page numbers, or *inter-cycle compaction*. Using the access example in Table 2, this technique latches the d-TLB translation used in cycle i for addresses 0xdeadbeee and 0xdeadbeef, and reuses it for the addresses 0xdeadbef2, and 0xdeadbef3 in cycle (i+1). Table 4 shows address translations needed after inter-cycle compaction. Effectively, two d-TLB lookups are saved in cycle (i+1). We note that the reused translation remains latched and could be reused again as long as the same memory page is accessed consecutively. For example, Figure 4 shows stack address translation often fits in this category. The extra prediction logic for intra-cycle compaction also needs careful trade-off evaluation to ensure that extra energy consumed does not exceed the energy saved by compaction.

cycle i	0xdeadb	0xdeadb	0xdeadb	0xfffff
cycle (i+1)	—	—	0x12345	—

Table 4: VPNs after inter-cycle compaction

3.2 Implementation of VPN Compaction

3.2.1 Intra-Cycle Compaction



(a) Modified Microarchitecture (b) Comparator Logic

Figure 5: Architectural enhancements for intra-cycle compaction

Based on prior discussion, we propose an intra-cycle compaction mechanism to eliminate the same-cycle synonymous d-TLB lookups. For an N-wide machine, we add C(N,2) comparators at the end of the memory order buffer after address disambiguation but before addresses are used for actual memory accesses. For instance, a 4-issue machine needs C(4,2)=6 comparators. The width of each comparator has the same width of the VPN. The comparators are used in each cycle to eliminate VPN redundancy so only unique VPNs are sent to the fully-associative d-TLB for address translation. Figure 5(a) illustrates the proposed microarchitecture enhancement assuming a 4-issue machine with 4KB pages and 32-bit address space (thus six 20-bit comparators). The comparator logic is detailed in Figure 5(b) that asserts one of the output signals to indicate the degree of access synonym for eliminating redundant lookups.

In addition to saving energy, this technique also offers potential benefits in reducing d-TLB lookup latency and the port requirement. For instance, for a 4-issue machine, the d-TLB does not need to be designed for the worst-case, i.e. 4-ported, since the occurrence of $syn(3)$ is rare for 4 memory references in a cycle, though $syn(2)$ is not uncommon as shown in Figure 3. This means that to avoid performance loss at least three ports are needed for the d-TLB. Using our proposed intra-cycle compaction, as some of $syn(2)$ can be compacted to 1 or 2 memory references when the opportunity arises, a d-TLB with two ports would suffice. A d-TLB with fewer ports is a smaller structure with a shorter lookup latency. This could be used to reduce the number of cycles needed for a d-TLB lookup when operating frequency is high, or it can be used to increase the number of entries in the d-TLB without increasing the lookup latency.

3.2.2 Inter-cycle Compaction

The inter-cycle compaction mechanism can be implemented by simply latching the most recently used (MRU) TLB entry and its virtual page number, and later reading the latch to detect reuse and obtain the translation. Figure 6 shows a semantic-aware memory (SAM) architecture [8] extended with this mechanism. The SAM architecture uses a Data Address Router to decouple a single memory stream into stack, global, and the rest (primarily heap data) substreams. In this case, the semantic-aware TLB splits a conventional TLB into a 2-entry stack-TLB, a 4-entry global-TLB, and a 32-entry heap-TLB for exploiting the semantic-region affinity. In the same figure, inter-cycle compaction is enabled for each individual semantic-aware TLB. For a multi-ported TLB, the number of reuse latches can be equivalent to the number of ports. Typically, TLBs are already designed with latches to test the read and write circuitry. During address translation, the VPN of the virtual address is compared against these latches, that hold input and the result of the previous translation lookup. If the VPN matches (hit), the latched physical address is used. If the VPN does not match (miss), a lookup to the corresponding fully associative semantic-aware TLB will be performed. The performance impact of misses in the reuse detection circuitry will be discussed in the next section, along with methods of reducing the penalty impact.

4. EXPERIMENTAL RESULTS

Our performance evaluation infrastructure is based on Simplescalar for the ARM ISA. We integrated Wattch [2] into the Simplescalar ARM model for energy simulation and made the necessary changes to enable our studies for semantic-aware memory architecture, and compaction of synonymous virtual addresses. The MiBench simulations were run to the end. The SPEC2000 simulations were first fast forwarded by 1 billion instructions, and simulated the next 3 billion instructions. The total memory references for these benchmarks vary from as low as 25% for bitcount to as high as 72% for blowfish in MiBench, and from 34% for mcf to 43% for bzip2 in SPEC benchmarks. Table 5 describes our machine model.

To evaluate the energy savings using intra-cycle compaction mechanism, the power consumption of the six 20-bit comparators used in eliminating redundant TLB lookups are taken into account in the Wattch simulation. We designed the comparators in Verilog and synthesized them using Synopsys Design Compiler targeting at 0.35μm technology. Each 20-bit comparator consumes 300μW, and takes about 550ps according to our synthesized results. Similarly, each MRU latch comparison for the inter-cycle compaction mechanism consumes 140μW. The energy consumed by the comparators, and latches are added each cycle even if only one d-TLB access is issued. We charge one extra cycle for the six 20-bit comparators, and one extra cycle for the MRU latch comparison. We also added an extra 10% of the dynamic power to account for the leakage power when there is no TLB access activity. The rest of the processor modules also use the same 0.35μm technology scaling for power measurements using Wattch.

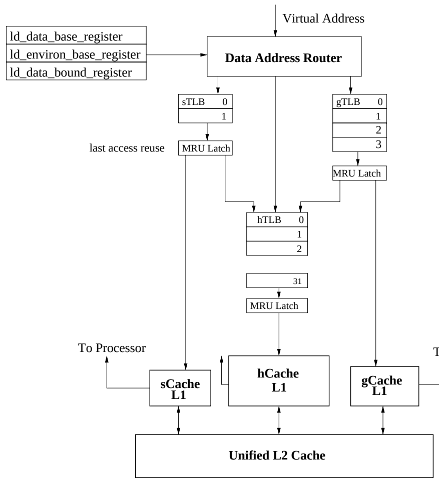


Figure 6: Semantic-aware memory architecture with inter-cycle compaction

32-bit Processor Parameters	Values
Execution Engine	out-of-order
Fetch/Decode Width	4 / 4
Issue/Commit Width	4 / 4
Number of data TLB entries	32
Page size	4 KB
L1/L2 cache hit latency	1 / 6 cycle
Memory latency	150 cycles
TLB hit/miss latency	1 / 30 cycles
L1 Cache baseline	Directed-mapped, 32KB, 32B line
L2 Cache	4-way 512KB, 32B line
Number of TLB ports used	2
Each 20-bit comparator power	300μW
Each MRU latch power in TLB	140μW

Table 5: Processor model parameters

4.1 Energy Reduction of Intra-cycle and Inter-cycle Compaction

Figure 7 presents the energy savings by applying intra-cycle and inter-cycle compaction mechanism separately to the d-TLB. The baseline in this figure is a conventional 32-entry d-TLB. The rightmost bar shows the total energy savings attained by applying intra-cycle compaction mechanism to baseline d-TLB. In average, nearly 27% of d-TLB energy savings is achieved with 9% penalty as shown by the last bar in Figure 8.

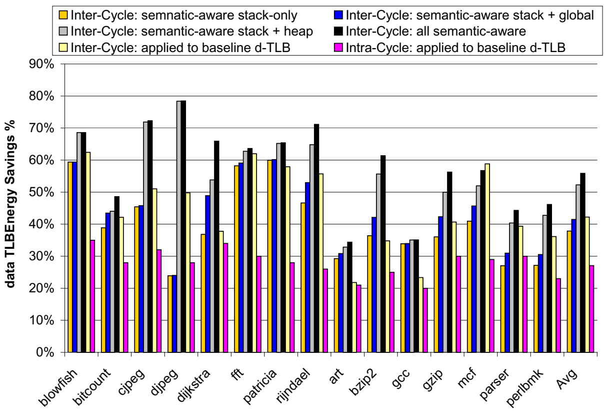


Figure 7: Energy savings using intra-cycle or inter-cycle compaction mechanism

Also shown in Figure 7 are the combinations of inter-cycle compaction mechanism applied to selective semantic-aware d-TLBs to trade-off the miss penalty. The first four bars show the energy savings achieved by employing stack-only, stack+global, stack+heap, and stack+global+heap (i.e., all semantic-aware) using inter-cycle compaction. In

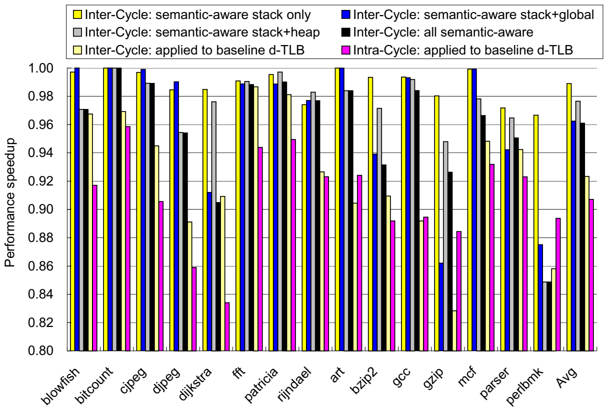


Figure 8: Performance impact due to intra-cycle or inter-cycle compaction mechanism

general, the energy savings reaches a maximum of 56% when inter-cycle compaction is applied to all semantic-aware d-TLBs while encountering less than 4% performance penalty. (the 4th bar from the left in Figure 8). Using inter-cycle compaction mechanism in a conventional d-TLB results in 42% energy reduction, with 8% performance slowdown (fifth bar in Figure 7 and and Figure 8.) The inter-cycle compaction scheme can be advantageous over the intra-cycle compaction when the reuse address translation hit rate is higher. Thus, most of the benchmarks show better performance using inter-cycle compaction over the intra-cycle compaction scheme.

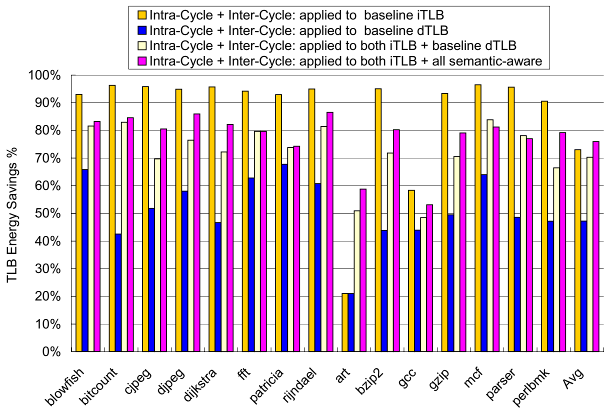


Figure 9: Overall i-TLB and d-TLB energy savings using both intra- and inter-cycle compaction mechanism

4.2 Synergistic Synonymous Address Compaction

The intra-cycle and inter-cycle compaction can be combined together to further reduce the overall energy consumption. Hence, we applied the intra and inter-cycle compaction mechanisms to instruction TLB (i-TLB) as well for it also exhibits high synonymous access behavior. Note that the instructions change memory pages only when (1) subsequent instructions cross the page boundary, or (2) a cross-page branch is taken. First, by applying both the compaction mechanisms to i-TLB and d-TLB separately, an average of 85% of the i-TLB energy and 52% of the conventional d-TLB energy are reduced as shown by the first two bars in Figure 9. A more encouraging observation is that several benchmark programs even demonstrated more than 90% energy savings in the i-TLB. Next, we combined both compaction schemes and applied to both i-TLB and d-TLB together to evaluate the overall TLB energy savings. As shown by the last two bars in Figure 9, 70% and 76% energy reduction are attained. It shows that applying both compaction schemes to i-TLB and semantic-aware d-TLB saves most of the energy spent in the TLBs.

4.3 Performance Impact for Combined Compaction

In the worst-case design, the two compactions require two extra pipeline stages if the TLB lookup cycle budget cannot accommodate them. We evaluated the performance impact due to the increased latency in Figure 10. In the worst case, 14% (3rd bar from left) performance is lost, when the latency penalties for intra- and inter-cycle compactions were charged to both the i-TLB and the conventional d-TLB. The loss is reduced down to 11% (4th bar from left) when the conventional d-TLB was replaced with semantic-aware d-TLBs as the reuse address translations has higher hit rate. If the compaction mechanisms are applied to only i-TLB, and d-TLB separately the performance degradation is 5%, and 13% (1st, and 2nd bar) as shown in Figure 10. The performance overhead can be eliminated from the critical path for the i-TLB, if the design can perform the comparison against the MRU latch as soon as the PC is updated before the subsequent fetch cycle.

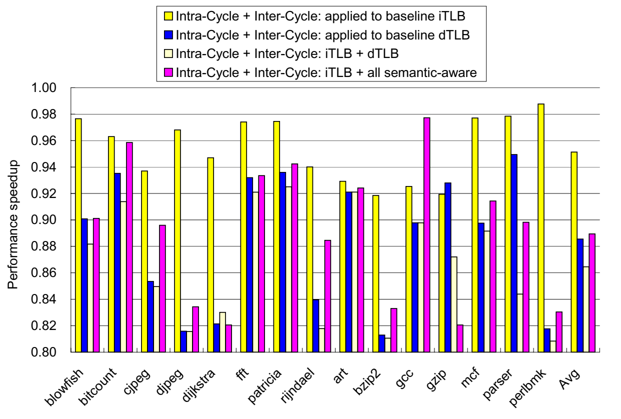


Figure 10: Overall i-TLB and d-TLB performance impact using intra- and inter-cycle compaction mechanism

5. RELATED WORK

A number of TLB power reduction schemes have been proposed. Kadayif et al. [6] added a register called Current Frame Register (CFR) to the instruction address translation. Instead of looking up the i-TLB, the processor fetches the translated address from the CFR unless there is a memory page change. Our intra-cycle compaction differs from this, as we also exploit redundancy among synonymous accesses in the same cycle, while the scheme in [6] only eliminates lookups in subsequent cycles, after the CFR has been filled. A compiler-based data layout restructuring was proposed in [7], combining with smaller translation registers, to reduce d-TLB energy. The scheme is similar to our inter-cycle compaction, but requires re-compilation. Way-prediction [4] was proposed to reduce cache energy by speculating one predicted way instead of looking up all ways in a set-associative cache. Other approaches for TLB power reduction include selective filter-bank TLB [9] and semantic-aware memory [8]. Both are complementary to our techniques.

Austin and Sohi [1] proposed re-translation and piggybacking ports to increase the address translation bandwidth. To achieve higher bandwidth, a 4-ported 8-entry pre-translation cache was added in the ID stage. The output of the pre-translation cache is attached to the TLB entry in the ID stage, making the TLB translation available at the start of the execution speculatively. The first difference is that, piggybacking the translation happens for the next cycle, unlike our intra-cycle compaction technique. The second difference is that, this high bandwidth is achieved at the expense of higher power, and inter cluster changes to the baseline architecture. The mechanisms that we proposed consumes less power with minimum hardware changes required in the memory cluster.

6. CONCLUSIONS AND FUTURE WORK

In this paper, we proposed intra-cycle and inter-cycle synonymous address compaction techniques that exploit address translation redundancy to reduce TLB energy. First, it is found that the concurrent and consecutive memory accesses are often *synonymous* — going to the same memory page. The same-cycle synonymous accesses can be eliminated using intra-cycle compaction mechanism that uses comparators to eliminate the redundant TLB lookups. Similarly, the inter-cycle synonymous accesses are eliminated by reusing the most recently used address translation stored in latches. These address comparators and the MRU latches add little power overhead and access time, allowing significant energy savings.

We also quantified the frequency of synonymous accesses in d-TLBs. More than 30% of intra-cycle, and nearly 76% of inter-cycle d-TLB accesses are synonyms. In semantic-aware TLBs, the ratio is even higher: 99% for stack, 82% for global, and 80% for heap. To exploit these intra-cycle and inter-cycle synonyms, we proposed two energy-efficient microarchitecture mechanisms — (1) intra-cycle compaction to eliminate the same-cycle synonymous accesses by using simple comparators, (2) inter-cycle compaction that reuses the address translation with an MRU latch. These techniques result in an average energy savings of 27% and 42% using intra- and inter-cycle compaction in conventional d-TLBs with roughly 9% and 8% perfromance penalty. And, when inter-cycle compaction techniques is applied to semantic-aware architecture the energy savings is 56% with 4% performance penalty. We also evaluated our compaction techniques by combining i-TLB and d-TLB together. The overall TLB energy savings is 70% for an i-TLB with a conventional d-TLB and 76% for an i-TLB with semantic-aware d-TLBs.

For area-constrained processors, our scheme can enable more memory operations per cycle without adding extra access ports for the TLB. Our simple hardware-only technique for saving energy is just one way of exploiting synonymous access. Our future work will explore other interesting mechanisms. These include compiler scheduling to group accesses known to be synonymous in the same cycle to take advantage of our hardware mechanism. Similar kind of compaction techniques can be used for instruction and data caches to exploit the synonymous accesses that go to the same cache line.

7. REFERENCES

[1] T. M. Austin and G. S. Sohi. High-bandwidth Address Translation for Multiple-issue Processors. In *Proceedings of International Symposium on Computer Architecture*, 1996.

[2] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: a Framework for Architectural-level Power Analysis and Optimizations. In *Proceedings of International Symposium on Computer Architecture*, 2000.

[3] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *the 4th Workshop on Workload Characterization*, 2001.

[4] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting Set-associative Cache for High Performance and Low Energy Consumption. In *Proceedings of International Symposium on Low Power Electronics and Design*, 1999.

[5] T. Juan, T. Lang, and J. J. Navarro. Reducing TLB Power Requirements. In *Proceedings of International Symposium on Low Power Electronics and Design*, 1997.

[6] I. Kadayif, A. Sivasubramaniam, M. Kandemir, G. Kandiraju, and G. Chen. Generating physical addresses directly for saving instruction TLB energy. In *Proceedings of International Symposium on Microarchitecture*, 2002.

[7] M. Kandemir, I. Kadayif, and G. Chen. Compiler-Directed Code Restructuring for Reducing Data TLB energy. In *Proceedings of International Conference on Hardware/Software Codesign and System Synthesis*, 2004.

[8] H.-H. S. Lee and C. S. Ballapuram. Energy Efficient D-TLB and Data Cache using Semantic-aware Multilateral Partitioning. In *Proceedings of International Symposium on Low Power Electronics and Design*, 2003.

[9] J.-H. Lee, G.-H. Park, S.-B. Park, and S.-D. Kim. A Selective Filter-bank TLB System. In *Proceedings of International Symposium on Low Power Electronics and Design*, 2003.