Generating Physical Addresses Directly for Saving Instruction TLB Energy

I. Kadayif, A. Sivasubramaniam, M. Kandemir, G. Kandiraju, and G. Chen *

Dept. of Comp. Sci. & Eng., The Pennsylvania State University, University Park, PA, 16802, USA
{kadayif,anand,kandemir,kandiraj,gchen}@cse.psu.edu

Abstract

Power consumption and power density for the Translation Lookaside Buffer (TLB) are important considerations not only in its design, but can have a consequence on cache design as well. This paper embarks on a new philosophy for reducing the number of accesses to the instruction TLB (iTLB) for power and performance optimizations. The overall idea is to keep a translation currently being used in a register and avoid going to the iTLB as far as possible — until there is a page change. We propose four different approaches for achieving this, and experimentally demonstrate that one of these schemes that uses a combination of compiler and hardware enhancements can reduce iTLB dynamic power by over 85% in most cases.

These mechanisms can work with different instruction-cache (iL1) lookup mechanisms and achieve significant iTLB power savings without compromising on performance. Their importance grows with higher iL1 miss rates and larger page sizes. They can work very well with large iTLB structures, that can possibly consume more power and take longer to lookup, without the iTLB getting into the common case. Further, we also experimentally demonstrate that they can provide performance savings for virtually-indexed, virtually-tagged iL1 caches, and can even make physically-indexed, physically-tagged iL1 caches a possible choice for implementation.

1. Introduction

Power optimization has become as important a criterion as performance across a spectrum of computing devices. While the need for conserving battery energy on some devices is well understood, power dissipation has a crucial consequence on chip design — fabrication, packaging, and cooling.

Reducing the power dissipation requires an indepth examination of each system component [5, 27]. Power is consumed whenever computing elements are accessed/activated (called dynamic power), and even when the elements are idle (called leakage power). While the latter becomes important with billions of transistors clocked

at high frequencies expected to be packed on a single chip, today's main concern is still the dynamic power, which is proportional to the number of times that the device is exercised. This is particularly the case for small components that are frequently exercised such as the Translation Lookaside Buffer (TLB) which is the focus of this study.

Several research projects have looked at reducing dynamic power consumption by reducing the device activity or to reduce the cost of an access itself (e.g., [14, 12, 28] for caches, [7] for DRAMs, [29, 11, 23] for datapath components). However, there is one specific component, namely the TLB, which has not drawn very much attention from the architectural/software angle for power optimization. In fact, this component is much more frequently accessed than DRAMs and many other components. An instruction fetch and data reference go through address translation via the TLB which is a cache of recent virtual-to-physical address translations. Even though this unit is typically kept small (to keep access times low), its associativity is usually high to keep miss rates low.

These high-associative storage structures are an important candidate for dynamic power optimization [10, 18]. This is particularly very important for the instruction TLB (iTLB) which is accessed on every instruction memory reference. The address translation logic consumes as much as 17% of on-chip power on the Intel StrongARM [18] and 15% on the Hitachi SH-3 [20]. This is more or less evenly split between the instruction and data parts. Further, power density, which is important for thermal management [3, 4], is another consideration when designing a high associative structure within a small space as is the case for a TLB.

There are several strategies for iTLB power optimization. First, one can attempt to optimize power at the circuit level as has been conducted in Juan et al [18]. Their proposal includes modifications to the basic cells and to the structure of TLBs to give 15% improvement. The second approach is to reduce the power consumption per access at the architectural level by restructuring the TLBs, e.g., using a smaller structure, reducing associativity, or working with multi-level TLBs (the smaller one has lower power and can help as long as we have high hit rates within this level). Choi et al [10] propose a two-way banked filter TLB and a two-way banked main TLB. One could even do TLB reorganizations dynamically [2]. While these approaches can reduce the power consumption per access, they do not reduce the number of accesses themselves. Instead, in the third ap-

*This work was supported in part by NSF Grants 0103583, 0093082, 0130143, and 9701475.

proach, which is the strategy used in this paper, we attempt hardware and software based strategies which can reduce the number of times that we need to access the TLB. We show that this can provide as much as 85% savings which is much higher than what the previous studies have presented. Further, this approach can be used in conjunction with the others (which lower the per access cost itself) to produce even higher savings.

We identify different ways of reducing the number of times a TLB is accessed:

- *Delaying TLB lookup:* If we can make the caches (at least the L1 cache) virtually-indexed and virtually-tagged (denoted as VI-VT in this paper), then we need to access the TLB only on an L1 miss (assuming that L2 is physically addressed). While this may cause an extra cycle latency for L1 misses, it can considerably reduce power requirements. In fact, this is the approach that is used on the Intel StrongARM processor [15]. One could even try to extend the VI-VT lookup for the L2 cache as well. However, in this case, a hardware implementation can become cumbersome if the L2 is off-chip, and [17] suggests software-based TLB maintenance.
- *Implementing the TLB in software:* As mentioned in the previous solution, delaying the TLB lookup beyond L2 accesses can lessen the importance of the TLB latency and dynamic power, potentially allowing an implementation in software. This also helps save real-estate on-chip, in addition to power as mentioned in [17]. If cache misses become high (for some commercial workloads, even L2 misses can be quite important as reported in [1]), then the performance penalties can mitigate the benefits of this approach.
- *Generating physical addresses directly:* If the software/hardware can directly provide the physical address of the page being referenced, then we do not need the TLB for that instruction/reference. While this may appear to be a radical shift from the current view (why have a virtual address at all if this is the case?), we believe (and demonstrate in this paper) that there are several circumstances when one can correctly generate physical addresses directly, at least for the instruction stream which is the target of our optimizations. This approach can even be used in conjunction with the other two solutions without any loss of generality, and constitutes the underlying philosophy of the mechanisms proposed in this paper.

There has been some amount of prior work done a long time ago in generating addresses without going to the dTLB for data references [21, 22, 9], but our focus here is on the instruction stream. In the context of instruction streams, we are only aware of a similar philosophy as this work in the context of the VAX architecture which uses a register to keep translations of the current instruction page, to alleviate TLB lookup latencies [26]. This is similar to one of the strategies that is evaluated in this work (called HoA, as will

be detailed within the paper), with the focus now on power consumption. Our results will show that while this may do well for performance, it does not give the best power savings.

To our knowledge, this is the first paper to explore the ability of a program to directly generate physical addresses for instructions towards iTLB power savings. Such an ability can be used in a system that has a virtually-indexed, physically-tagged (VI-PT) iL1 cache, to lower iTLB power considerably. It can even lower iTLB power in a system with a virtually-indexed, virtually-tagged (VI-VT) iL1 cache by reducing lookups upon cache misses. Further, it can save cycles expended in iTLB lookups upon an iL1 miss for a VI-VT iL1 cache where the iTLB is in the critical path. Finally, if we are able to successfully provide translations in most cases, then we may want to even reconsider incorporating physically-indexed, physically-tagged (PI-PT) caches which are largely ignored today because of translation getting in the critical path.

It should be noted that the mechanisms investigated in this paper generate physical addresses directly only when we are absolutely sure (i.e., they are not speculative). Specifically, they target optimizing references to the page that has just recently been referenced. Since there is considerable spatial locality in instruction streams, we believe that one can get substantial savings even with such a simple strategy. One could ask how this philosophy is different from having a very small iTLB (degenerating even to a single entry iTLB). The differences are in that the consequence of a really small iTLB can lead to performance problems (higher miss rate), and there is still a comparison involved in matching tags (consuming power). In contrast, our approach can still work with a reasonably sized iTLB, and can generate the addresses directly in several situations without requiring comparisons, and without incurring any performance penalties. We also show in this paper how this approach can be better than a multi-level iTLB from both power and performance angles.

2. Background: Cache and TLB Lookup

The iTLB needs to be consulted upon a virtual page address to generate the physical page number before eventually going to the DRAM. However, there are caches (L1 and L2) before going to the DRAM, and how these caches are looked up can have an impact on iTLB performance and power. It should be noted that cache lookup requires an indexing part to determine the set under consideration, and a subsequent tag comparison for the blocks within the set. Either of these can be done with a virtual address or a physical address, leading to four possible combinations: virtually-indexed, virtually-tagged (VI-VT), virtually-indexed, physically-tagged (VI-PT), physically-indexed, physically-tagged (PI-PT), and physically-indexed, virtually-tagged (PI-VT). The last option (PI-VT) is not really in much use (MIPS R6000 uses it) and is not under consideration here. In this paper, we focus on the other three options for the L1 instruction cache (iL1) and assume that L2 is always PI-PT.

A brief summary of how these mechanisms work for the different iL1 addressing schemes [8] is given below:

- *PI-PT iL1*: The physical address needs to be obtained before the cache can even be indexed, making the iTLB fall in the critical path.¹ This is also a reason why this configuration is not very popular today. In terms of power as well, the iTLB is consulted on every instruction fetch regardless of whether it is in the iL1 or not. The advantage of this scheme is that there are no aliasing problems across different virtual address spaces.
- *VI-PT iL1*: One way to remove the iTLB from critical path is to index the sets of iL1 using the virtual address, and iTLB is concurrently looked up to obtain the physical address (which is expected to take less time than the iL1 indexing). Consequently, the tag from the physical address is used for the comparison with the corresponding tag bits of the set. As a result, iTLB is not in the critical path anymore, but the downside is that it is still accessed on every instruction fetch incurring energy costs. Further, write-backs require a translation, and this can be handled by storing the physical indexes/addresses with each block. Many current microprocessors use this configuration (e.g., AMD K6, MIPS R10K, PowerPC).
- *VI-VT iL1*: With this configuration, iL1 is both indexed and tagged with virtual addresses, implying that iTLB is not required at all until an iL1 miss. One could either lookup iTLB at this time (which may add an extra cycle latency to the iL1 miss path if L2 is PI-PT, but is very good in terms of power), or in parallel with iL1 access (in which case the iTLB lookups are not any different than in a VI-PT iL1). In this study, we use the former strategy as it is more power efficient, and may not suffer significant performance penalties if the iL1 locality is sufficiently good. The StrongARM is an example of this kind of iL1 indexing [15]. This strategy has aliasing problems, and the solution is to typically add a few most significant bits to differentiate between address spaces.

3. Our Approaches

As was mentioned earlier, our overall philosophy is to perform the translation for a page once, and subsequently keep reusing it directly without going to the iTLB, as long as it does not change. Two ways of achieving this is by:

¹There are several situations where by choosing an appropriate iL1 configuration — such that the cache indexing can work with just the offset within a page, and does not need frame number — one could implement a PI-PT iL1 without making the iTLB fall in the critical path. Some commercial processors (e.g., Sun UltraSPARC II) exploit such hacks. However, this restricts iL1 configurations and becomes very similar to a VI-PT iTLB lookup, which is evaluated in this paper. Consequently, in our PI-PT model, we do not put any restrictions on iL1, and the iTLB needs to be looked up before iL1 indexing.

- Storing the translations of several previously visited pages either in hardware (in which case it is no different than the iTLB itself except maybe a smaller version of it) or in software (in which case we incur high performance overheads).
- Storing just a single translation — namely the current page — and keep using it as long as we do not leave that page. When we do leave the page, we lookup the iTLB for the target page. This is the strategy that is adhered to in all the mechanisms proposed in this work.

3.1. Hardware Support

Whenever our mechanism cannot supply a translation, there needs to be a way of triggering an iTLB lookup based on the virtual address. The result of this lookup moves the corresponding iTLB entry (both the physical frame number and the protection bits) into a register called the **Current Frame Register (CFR)**, whose format is of the form

< Virtual Page Number, Physical Frame Number,
Protection/Other Bits >

The trigger mechanism itself (that is done in hardware or software) will be discussed in detail for each of our approaches in the subsequent discussion. Once we have the current physical frame number in the CFR, we can perform the next instruction fetch as follows depending on the cache addressing mechanism (described earlier in Section 2):

- *PI-PT iL1*: The page offset is obtained from the low order bits of the PC, and the physical frame number is obtained from the CFR. This constitutes the physical address, and the iL1 is looked up with this address. This is pictorially shown in Figure 1(a).
- *VI-PT iL1*: The virtual address is generated by appending the virtual page number in the CFR with the page offset bits of the PC. The index part of this address is used to address iL1. The physical address is generated by appending the physical frame number part of the CFR with the page offset bits of the PC, and the tag part of this result is used to compare the tags in the set that was indexed in iL1. This is shown in Figure 1(b).
- *VI-VT iL1*: We use the PC virtual address entirely to lookup iL1. If we obtain the data from there, then we are done. Only on a miss, we access the CFR to get the physical frame number concatenated with the page offset bits of the PC to lookup L2. This lookup mechanism is shown in Figure 1(c).

Our approach can work in conjunction with each of these cache addressing mechanisms to provide power savings. In fact, it can even provide performance savings in the case of PI-PT and VI-VT caches since the iTLB access can get in the critical path (it is always the case for PI-PT and happens on an iL1 miss for VI-VT). One could even hypothesize that we may want to re-think incorporating PI-PT, which is largely ignored today, as long as our approach can provide translations in most cases.

Figure 1. iL1 lookup with the presence of CFR assuming the translation is present there (a) PI-PT, (b) VI-PT, (c) VI-VT.

3.2. OS Support

The OS needs to ensure that the current page (the one whose translation is being used currently) is not evicted (i.e., its physical address does not change). This is not expected to be a problem since this page will be a very low candidate for LRU anyway (and we do this for at most 1 page per application process). If so desired, one could ask the OS to invalidate the CFR if this page has to be really evicted/re-mapped (just as the entry would be invalidated in the iTLB). Note that CFR is not explicitly available to the application program (either for reading or writing), and it is used directly by the hardware. However, in supervisory mode, the OS will be allowed to read/write the CFR (so that this page is not evicted) and maybe reset/invalidate it. Consequently, the program cannot change permissions to a page (which are also in the CFR) without going via the OS. Upon a context switch, the CFR can be treated as yet another register whose context is saved and restored.

3.3. Strategies

3.3.1 Hardware-only Approach (HoA)

This is an approach which does not require any software support. The hardware directly examines virtual addresses generated by the PC, and compares them with the virtual page number part of the CFR. If they match, then the target instruction is in the same page (requiring no translation) and the iL1 lookup is performed as described above. If they do not match, then we force an iTLB lookup. This lookup, in the case of a VI-PT iL1 is done in parallel with the iL1 indexing (incurring an energy cost in the iTLB). In a VI-VT iL1, on the other hand, even if the page numbers do not match, the iTLB is not looked up until an iL1 miss. The hardware that is needed is a comparator that compares the virtual page number produced by the PC and that in the CFR. As mentioned earlier, the VAX uses a similar strategy — holding the current instruction page translation in a register called the IPA — to alleviate TLB lookup latencies [26]. In this evaluation, we are looking at a more modern

processor with out-of-order execution and complex control flow structures, and our focus here is on power consumption.

The advantage of this approach is that we perform iTLB lookups exactly when needed (very accurate). The downside is the overhead of the comparison (energy cost) on every instruction fetch. We believe that the performance overhead can be hidden from the critical path by performing this operation as soon as the PC is updated (and before the subsequent instruction fetch cycle).

3.3.2 Software-only Conservative Approach (SoCA)

At the other end of the spectrum, we consider a scheme where all the triggering of the iTLB lookup is done explicitly by the software (i.e., the compiler). The reader should note that there are two ways by which a program execution can move from one instruction page to another: (a) explicit branch instructions whose target is in a different page (we call this the BRANCH case), and (b) two successive instructions which are on page boundaries (we refer to this as the BOUNDARY case), i.e., one is the last instruction of a page, and the next is the first instruction of the next page (we assume that instructions are aligned so that a single instruction does not cross page boundaries). Further, we assume that an iTLB lookup is done by the hardware for every target of a branch regardless of whether it crosses a page boundary or not, and all other instructions directly use the CFR. This automatically handles the BRANCH case. To handle the BOUNDARY case, the compiler explicitly inserts a BRANCH instruction at the end of each instruction page, with the target being the very next instruction (the first one on the next page).

The advantage of SoCA is that it does not even require the extra logic incurred by the previous mechanism, and there is no extra energy cost in the normal instruction fetch path. The downside is the extra instructions (both cycles and energy) incurred in the BOUNDARY cases (this overhead is negligible). The other problem is that we are being very conservative (which our results will show) in assuming that every branch target is in a different page and this is

Figure 2. Integration with branch prediction logic. BTB denotes the branch target buffer. BA and BTA correspond to branch address and branch target address, respectively. CFR is the current frame register. VPN, PFN, and PB correspond to three portions of the CFR, namely, virtual page number, physical frame number, and protection bits, respectively.

what the next two schemes try to address.

3.3.3 Software-only Less Conservative Approach (SoLA)

In this approach, we take the mechanism explained in Section 3.3.2 and try to be less conservative in the BRANCH cases. Specifically, we want to eliminate iTLB lookups when a static analysis of the code by the compiler can reveal that the branch target is within the same page as the branch instruction itself (this typically occurs when branch targets are given as immediate operands or as PC relative operands). The necessary compiler support is to check whether the target of a statically-analyzable branch is on the same page of the branch itself.

An implementation of this requires that the hardware be able to distinguish between two types of branches: one is the branch identified by the compiler as being on the same page (which does not go through the iTLB) and the other being the normal branches where the targets go through the iTLB. The first types of branches are called In-Page branches. We use an extra bit in branch instructions to differentiate between in-page branches and the others. One can envision this bit being part of the address itself, indicating whether the branch target needs to be looked up in the iTLB or not.

This approach enjoys the benefits of the previous one, with the additional benefit of avoiding iTLB lookups when branch targets are statically analyzable and found to be within the same page as the corresponding branches themselves. However, we are still being quite conservative in that we force lookups even if the targets are within the same page but this cannot be determined at compile time.

3.3.4 Integrated Hardware-Software Approach (IA)

While the hardware-only mechanism is quite accurate in finding out when to go to the iTLB, the downside is the energy cost on every instruction execution. The software-only approach avoids this, but can turn out to be conservative and

goes to the iTLB more often than needed. In this section, we propose an integrated approach that can get the better of these two extremes.

We can use the compiler-based approach to track the BOUNDARY cases since we are anyway accurate in predicting page transitions at these points with the software schemes. However, we can adopt a hardware mechanism (not the one used in Section 3.3.1) for the BRANCH cases, so that we can use runtime information to determine if the target is really going out of a page (and whether it is taken at all). We implement this within the existing framework of branch predictors. For instance, an implementation of this check with the Branch Target Buffer [25] is shown in Figure 2, and is the one evaluated in our studies.

The BTB (that is used in several commercial offerings such as Pentium, PA 8000, and PowerPC 620 [25]) indexed by the address of the branch instruction, keeps the address of target instruction to be executed next together with additional state information. As soon as the PC of the branch instruction is generated, this table is looked up concurrently with the IF stage of the branch instruction itself. Consequently, the IF of the (likely) branch target is performed in the next cycle if we hit in the BTB. Our enhancements to this mechanism is to simply check if the virtual address (page number bits) coming out of the BTB matches the CFR virtual page number (see Figure 2). If it does, then the iTLB is not used for the target instruction fetch. Otherwise, the iTLB may need to be consulted (not always in a VI-VT cache).

While the evaluations in this paper have been performed with what has been explained here, it is possible to make it work with other types of branch prediction mechanisms as well. The general idea is to wait until a branch target address is available and then perform a comparison of virtual page number with that in the CFR. For example, if a target address based predictor is not used, and the branches are handled with a predecoding mechanism, then the CFR comparison can be employed at that time.

The situations when the iTLB is looked up are expressed in pseudo-code format in Figure 3. Essentially, we avoid iTLB lookups when the branch target is predicted correctly and the target is within the same page, and default back to a iTLB lookup otherwise. As can be seen in this figure, there are four points of return (A, B, C, and D) from this routine. In (A), there is no iTLB lookup at all. In (B), we incur an iTLB lookup regardless of whether the taken target falls in the same page or not (this is a little conservative, but with high accuracies of branch predictors this may not be a major problem). In (C), we incur an iTLB lookup, but this would definitely be needed since there is a page change. Finally, in (D), we incur an extra iTLB lookup than actually needed in cases where the predictor failed but the target was still on the same page. As a result, we are being a little conservative in the (B) and (D) cases, but these penalties will be bounded by the inaccuracy of the predictor. One could try optimizing this further in future work.

There is no performance penalty that is additionally incurred by this mechanism. None of these mechanisms affect iL1 and L2 hits or misses, and thus they do not affect the

```

If BranchPrediction == NOT TAKEN
... proceed with execution until we know whether
  prediction was correct ...
If Prediction == SUCCESS
  return (A)
else
  Lookup iTLB for Target Address
  Update CFR
  return (B)
else
  if BranchTargetPage != VirtualPageNumber in CFR
    Lookup iTLB for Target Address
    Update CFR
    ... proceed with execution until we know whether
      prediction was correct ...
  if Prediction == SUCCESS
    return (C)
  else
    Lookup iTLB for Target Address (not taken path)
    Update CFR
    return (D)

```

Figure 3. Pseudo-code of iTLB lookups during branch executions in IA

rest of the memory system energy consumption. Further, in a VI-PT cache, these mechanisms will not affect the execution cycles either. In a VI-VT cache, our mechanisms are expected to help (rather than hurt) performance by possibly reducing address translation overheads on an iL1 miss.

4. Performance Results

4.1. Experimental Setup

In this section, we present a detailed energy and performance evaluation of the optimization strategies proposed in this work. Unless stated otherwise, we use the processor architecture whose parameters are listed in Table 1 (called the *default configuration*). Energy values are obtained using CACTI [24] for 0.1 micron technology.

To test the effectiveness of our strategies, we used six benchmarks from Spec2000 benchmark suite [13] (given in Table 2), and simulated 250 million instructions after skipping the first 1 billion instructions. These six benchmarks stress the iTLB more than the others due to the relatively worse instruction locality (their iL1 miss rates are higher). The second and third columns in Table 2 give the execution cycles and iTLB energy consumptions of our default configuration when iL1 is VI-PT. The fourth and fifth columns give the same information for the VI-VT iL1. The sixth column presents iL1 miss rates. The seventh column gives the number of branch instructions executed and their percentage with respect to the total number of instructions executed. The last two columns give the number of page crossings during execution. This number is divided into two portions: BRANCH case (i.e., the page crossings as a result of a branch instruction) and BOUNDARY case (i.e., the page crossings due to sequential execution on the page boundary). We clearly see that the overwhelming majority of dynamic page crossings are due to branches.

Parameter	Value
Processor Core	
RUU Size	64 instructions
LSQ Size	32 instructions
Fetch Queue Size	8 instructions
Fetch Width	4 instructions/cycle
Decode Width	4 instructions/cycle
Issue Width	4 instructions/cycle (out-of-order)
Commit Width	4 instructions/cycle (in-order)
Functional Units	4 Integer ALUs 1 Integer multiply/divide 4 FP ALUs 1 FP multiply/divide
Memory Hierarchy	
iL1	8KB, 1-way, 32 byte blocks, 1 cycle latency
dL1	8KB, 2-way, 32 byte blocks, 1 cycle latency
L2	1MB unified, 2-way, 128 byte blocks, 10 cycle latency
iTLB	32 entries, full-associative, 50 cycle miss penalty
dTLB	128 entries, full-associative, 50 cycle miss penalty
Page Size	4KB
DRAM	128MB (divided into 32MB banks), 100 cycle latency
Branch Logic	
Predictor	Bimodal with 4 states
BTB	1024 entry, 2-way
Mispred. penalty	7 cycles

Table 1. Default configuration parameters

All our experiments have been conducted using SimpleScalar [6], with the sim-outorder cycle-level model. The execution without any of our optimization mechanisms is referred to as the *base execution* in the rest of this paper, and the iTLB energy numbers (in columns three and five of Table 2) and execution cycles (in columns two and four of Table 2) are obtained with this model for the default configuration. SoCA and SoLA require an examination of the assembly code by the compiler to determine the page boundaries and branches. We also compare all our schemes with an OPT execution model, which gives the lowest iTLB energy without any further code transformations. In this model, iTLB energy is consumed only when there is an actual page change.

4.2. Results

We first give in Figures 4 and 5 the iTLB energy consumptions and overall execution cycles of our four strategies (HoA, SoCA, SoLA, and IA) normalized with respect to corresponding values of those for the base case. These schemes are compared with the OPT results. Examining the energy consumption graphs (Figure 4) we see that all our four schemes provide substantial reduction in iTLB energy for both VI-PT and VI-VT. On the average (over all 6 applications), the iTLB energy consumption is reduced to just 5.69%, 12.24%, 5.01%, and 3.82%, for VI-PT and 15.23%, 36.83%, 16.39%, and 14.04% for VI-VT, with HoA, SoCA, SoLA, and IA, respectively. We see that IA comes very close to the OPT energy consumption (3.20% for VI-PT and 12.74% for VI-VT on the average). While the savings in both iL1 addressing strategies are quite good, they are better for VI-PT. This can be explained based on the fact that in a VI-VT iL1, the address translation is done only on a iL1 miss. There is a higher probability (though not always as will be explained later) of the translation missing

Benchmark	VI-PT		VI-VT		iL1 Miss Rate	Number (Percentage) of Dynamic Branches	Page Crossings	
	Cycles	iTLB Energy	Cycles	iTLB Energy			BOUNDARY	BRANCH
177.mesa	188.1	109.1	196.1	3.345	0.002	23.6 (8.9%)	99016 (1.77%)	5503671 (98.23%)
186.crafty	331.7	124.1	350.5	8.385	0.014	36.4 (12.6%)	86925 (1.09%)	7969935 (98.91%)
191.fma3d	169.3	112.7	176.6	3.040	0.011	50.8 (18.6%)	13513 (0.11%)	12168347 (99.89%)
252.eon	263.1	134.5	274.7	5.221	0.010	38.7 (12.3%)	312314 (1.99%)	15344827 (98.01%)
254.gap	161.3	112.2	165.6	2.005	0.006	19.9 (7.3%)	722028 (11.31%)	5662714 (88.69%)
255.vortex	293.9	108.4	310.5	6.345	0.027	43.2 (16.6%)	577674 (5.75%)	9473056 (94.25%)

Table 2. Benchmarks and their characteristics using the default configuration. The seventh column shows the percentage of branch instructions of the total instructions executed. The actual page crossings for the BOUNDARY and BRANCH cases are shown, and their relative percentage of contribution to the crossings. The numbers in columns two, four, and seven are in millions. All energy values are in millijoules.

in our CFR as well (because of the worse locality when this occurs). Still, we should point out that we get over 85% iTLB energy reduction on the average for VI-VT with our IA scheme.

We next examine each of our four strategies in closer detail. With HoA, the energy consumption presented in these graphs is because of two factors: the iTLB lookup when the page comparison of the CFR indicates a page crossing, and the energy consumption of the comparison itself which is incurred on every instruction fetch (regardless of whether there is a page crossing or not). The latter factor accounts for the difference between HoA and OPT, and this does turn out to be reasonably significant.

As noted earlier, the last two columns of Table 2 give the actual page crossings incurred during the execution of these applications, broken down into the BOUNDARY and BRANCH cases. We can see that the BRANCH cases typically overwhelm the BOUNDARY cases. Table 3 gives the page crossings that are forced by the three schemes — SoCA, SoLA and IA — to look up the iTLB (sometimes conservatively). Note that the BRANCH case crossings are higher than the corresponding values in Table 2, and the BOUNDARY case crossings are the same (as these strategies differ from the optimal in only how the branches are treated). SoCA turns out to be much worse than OPT and the other three because of its conservative assumption that each branch crosses a page boundary. One can observe that the absolute numbers under the BRANCH column for SoCA in Table 3 is higher than the corresponding columns for the other schemes, and this is also the more dominating situation compared to the BOUNDARY case as Table 2 suggests.

SoLA, on the other hand, can optimize situations when there is no page crossing if the branch target is available at compile time. Consequently, this reflects on the lower iTLB lookups required by this scheme for the BRANCH cases. Table 4 shows the number of static occurrences of the branches whose target is available at compile time (termed ‘Analyzable’ in the table), and this table also shows how many times such branches occur in the dynamic execution. On the average, we find that these dynamic instances amount to 84.8% of the total, and of these 70.4% are within the same page (not requiring a lookup). This does turn out to be a significant fraction of the total branches

Figure 4. Normalized iTLB energy consumptions. Top: VI-PT, Bottom: VI-VT. These energy values are normalized with respect to that of the base case for each iL1 lookup mechanism as given in Table 2.

(nearly 60%), leading to the substantial reduction in energy for SoLA compared to SoCA.

Moving on to IA, we note that it is very close to OPT in most cases. As explained earlier, the only points where IA may need extra iTLB lookups over OPT is when the branch prediction is not accurate. Table 5 gives the percentage of dynamic branches that were predicted accurately by the branch prediction mechanism. As can be seen from this table, these misprediction rates are less than 15% explaining why IA comes close to OPT. In fact, if we can use a more accurate predictor, IA would come even closer to OPT.

Having covered the energy results, we present the execution time results with these schemes for the VI-VT cache in Figure 5. It is to be noted that there is no significant differ-

Benchmark	SoCA		SoLA		IA	
	BOUNDARY	BRANCH	BOUNDARY	BRANCH	BOUNDARY	BRANCH
177.mesa	99016 (0.41%)	23895619 (99.59%)	99016 (0.99%)	9893195 (99.01%)	99016 (1.48%)	6590313 (98.52%)
186.crafty	86925 (0.23%)	37174532 (99.77%)	86925 (0.66%)	13000618 (99.34%)	86925 (0.85%)	10133921 (99.15%)
191.fma3d	13513 (0.03%)	51083905 (99.97%)	13513 (0.07%)	19451932 (99.93%)	13513 (0.10%)	14043552 (99.90%)
252.eon	312314 (0.77%)	40386387 (99.23%)	312314 (1.52%)	20268715 (98.48%)	312314 (1.59%)	19277621 (98.41%)
254.gap	722028 (3.40%)	20531371 (96.60%)	722028 (6.83%)	9852715 (93.17%)	722028 (9.24%)	7092915 (90.76%)
255.vortex	577674 (1.31%)	43422782 (98.69%)	577674 (3.57%)	15595749 (96.43%)	577674 (5.28%)	10360962 (94.72%)

Table 3. Dynamic number of TLB lookups for SoCA, SoLA, and IA (VI-PT). The numbers in parentheses indicate the contributions of the BOUNDARY and BRANCH cases. Note that the number of page crossings due to the BRANCH cases is higher than the corresponding values in the last column of Table 2.

Benchmark	Static Statistics				Dynamic Statistics			
	Total	Analyzable	Page Crossings	In-Page	Total	Analyzable	Crossings	In-Page
177.mesa	563	472 (83.8%)	117 (24.8%)	355 (75.2%)	23645387	19175565 (81.1%)	5173141 (27.0%)	14002424 (73.0%)
186.crafty	2161	1985 (91.8%)	515 (25.9%)	1470 (74.1%)	36364110	31864428 (87.6%)	7690514 (24.1%)	24173914 (75.9%)
191.fma3d	532	477 (89.7%)	142 (29.8%)	335 (70.2%)	50803392	44644775 (87.9%)	13012802 (29.1%)	31631973 (70.9%)
252.eon	706	548 (77.6%)	204 (37.2%)	344 (62.8%)	38654600	28783921 (74.5%)	8666249 (30.2%)	20117672 (69.8%)
254.gap	883	785 (88.9%)	146 (18.6%)	639 (81.4%)	19989582	18034984 (90.2%)	7356328 (40.8%)	10678656 (59.2%)
255.vortex	2781	2548 (91.6%)	1078 (42.3%)	1470 (57.7%)	43248486	37933459 (87.7%)	10106426 (26.6%)	27827033 (73.4%)

Table 4. Static and dynamic branch statistics. Static statistics are obtained from the source codes. The analyzable column gives the number of branch instructions in the code whose target (when in-page or not) can be detected at compile time. It also shows the contributions of these branches to the total number of branches. The next two columns show how many (and what percentage) of the analyzable branches cross the page boundary or not. The next four columns give similar statistics for dynamic execution.

ence in execution cycles with these schemes (compared to the base case) for a VI-PT cache, since all the iTLB lookups are done in parallel with the iL1 cache. The overhead of the extra instructions for the BOUNDARY cases is very low. The schemes allow a translation to be already available in many situations even after one misses the VI-VT iL1. In such cases, we do not incur the extra latency for a iTLB lookup before we need to go to L2 which is physically addressed (both index and tag) in our evaluations. As can be observed from Figure 5, IA provides between 2-5% savings in execution cycles, with a saving of 3.55% on the average. These savings are a direct correspondence to how accurately IA is able to predict whether a iTLB lookup is really needed. Even though these applications are the ones with relatively high iL1 miss rates of the Spec2000 suite, it has been reported that commercial workloads (such as databases), have much higher iL1 miss rates [1]. In such situations, our approach can provide substantial cycle savings as well, in addition to energy savings for VI-VT caches as shown in [19].

4.3. Sensitivity to iTLB Configuration

4.3.1 Monolithic (Single-Level) iTLB Configurations

Tables 6 and 7 give the energy consumption and execution cycles for the base case as well as the OPT and IA executions for four different iTLB configurations (1 entry, 8 entry FA, 16 entry 2-way, 32 entry FA) with VI-PT and VI-VT

iL1. Note that the iTLB in our default configuration was 32 entry FA (its results are reproduced here for ease of comparison). While 8 through 32 may appear as reasonable sizes for an iTLB, the choice of also using a 1 entry iTLB was made to see if the instruction locality was good enough to itself provide good performance at a much smaller power consumption. Incidentally, the results for multi-level iTLB structures are given in Section 4.3.2.

The iTLB energy for a given execution is given by $n_a * E_a + n_m * E_m$, where n_a and n_m are the number of iTLB accesses and iTLB misses, respectively; and E_a and E_m are the energy cost per access and per miss, respectively. For any particular scheme (whether it is IA, OPT or the base case), n_a remains the same when we change the iTLB configuration. While n_m for a given scheme typically increases when we go for a smaller (or less associative) iTLB, the change is the same for all schemes. Hence, when the number of iTLB misses decreases, the importance of IA (or OPT) is felt even more (reflecting on the smaller percentage of energy consumption given in brackets in Table 6 for better iTLBs).

We find good benefits in terms of energy for all the configurations considered with IA for VI-PT and VI-VT (though the absolute energy for the latter itself is much lower than the former in the base case). While larger (and high-associative) iTLBs are good for reducing misses and providing good performance, their drawback is the high power consumption. The results presented above show that we can use a scheme such as IA in conjunction with a larger

Figure 5. Normalized execution cycles for VI-VT. These values are normalized with respect to the execution cycles of the VI-VT base case as given in Table 2. We did not observe any significant differences in execution cycles across the schemes, and compared to the base execution, for a VI-PT iL1.

177.mesa	186.crafty	191.fma3d	252.eon	254.gap	255.vortex
94.14%	91.16%	95.82%	85.23%	89.55%	97.38%

Table 5. Branch predictor accuracy.

iTLB, to get its performance benefits, while consuming as low power as a much smaller iTLB which does not employ any power optimizations.

More importantly, if we look at the absolute energy consumption values for VI-PT with IA, we can observe that they are in most cases comparable (and even smaller sometimes) to the absolute energy consumption of the base VI-VT. For example, with a 16 entry 2-way iTLB used in conjunction with VI-PT iL1 and IA has an energy consumption of 6.623 mJ for 255.vortex while the same iTLB for a base line VI-VT turns out to consume 9.047 mJ. These results show that the choice of the cache indexing does not need to be governed by the iTLB power consumption with our mechanism (the StrongARM has possibly chosen a VI-VT L1 addressing scheme for TLB power optimization). We achieve this result without compromising on the performance benefits of VI-PT (recall that VI-VT incurs an extra latency on an iL1 miss) — compare the cycles for the same two iTLB configurations where our approach with VI-PT does around 3% better in terms of cycles compared to the base VI-VT execution. On the average, for the VI-VT cache, the average savings due to IA in execution time amount to 18.1%, 11.0%, 5.4%, 3.55% respectively for 1-entry, 8-entry, 16-entry, and 32-entry iTLBs. Sometimes, even when we miss in iL1, we may be able to find the translation in the CFR with IA, avoiding an iTLB lookup (incurring a performance and energy penalty) before going to L2. This is particularly because of the larger spatial locality coverage provided by the CFR (which works at a page granularity) compared to the cache block granularity of iL1. For instance, a reference to a block within a page that is missing in both iL1 and the CFR will cause a miss in IA with

VI-VT as well. However, an immediate cache miss for another block within the same page will hit in the CFR, thus avoiding a iTLB lookup for IA.

4.3.2 Multi-Level iTLB Configurations

A multi-level TLB is not only a way of optimizing TLB performance, but can also be an effective way of reducing power consumption. By satisfying many lookups in a much smaller first-level TLB, we can reduce the dynamic power consumption of the larger second-level TLB (assuming they are looked up sequentially). However, this can typically increase the complexity of implementation (and area), and push latencies higher. In fact, on the Itanium, the first-level TLB can be looked up in one cycle, but the larger second-level TLB lookup takes as long as 10 cycles [16].

To compare how effective our scheme can be in relation to a multi-level iTLB structure, we have conducted numerous experiments with different configurations — (i) 1-entry level-1 and 32-entry, FA level-2, (ii) 32-entry, FA level-1 and 96-entry, FA level-2 (as in the dTLB of IA-64) — both of which have been evaluated with serial (i.e., the second level is looked up only on a first level miss) and parallel (i.e., both levels are looked up in parallel — this may have some performance benefits in terms of overlapping the lookup latency for the second level). We are not presenting the results for the parallel lookup here, because their energy consumption values are much worse. Here, we compare a monolithic 32-entry, FA iTLB using IA with configuration (i), and a monolithic 128-entry, FA iTLB using IA with configuration (ii). The normalized dynamic energy consumption and performance cycles are given in Figure 6. To give the multi-level iTLB structure the benefit of doubt, we have optimistically assumed a single (extra) cycle lookup for the second level when the first level misses.

When we look at the 32-entry experiments, the base execution with a two-level structure consumes 55.3% more energy than a monolithic 32-entry iTLB using IA. This is because in the IA scheme, the energy consumption in the common case (i.e., when the address is present in CFR) is just the register access/lookup. On the other hand, even with a 1-entry level-1 iTLB, there needs to be a comparison to check whether the translation exists. As a result, the energy differences between these two executions are a consequence of the extra comparison that is involved with a 1-entry, level-1 (it is to be noted that whenever we miss in the level-1 base case, we are also not going to be finding the translation in the CFR for the IA in the monolithic configuration). On the other hand, the performance of the monolithic iTLB with IA does turn out to be a better alternative (between 2-10%). This is because we do not incur any extra latencies looking up the second level if the first level (CFR) misses. When we have a 1-entry, level-1 iTLB, the performance penalties can become a concern and the additional second level lookup latency may be incurred often. To offset this, one could increase the number of entries in the first level as in configuration (ii) to ensure the working set is captured by the first level. However, the results presented in Figure 6 show that while performance is opti-

Figure 6. iTLB energy consumptions of two-level iTLB configurations (Top) and their execution cycles (Bottom). The energy consumptions and execution cycles of 1-entry level-1 and 32-entry, FA level-2 configuration and 32-entry, FA level-1 and 96-entry, FA level-2 configuration are normalized, respectively, with respect to the corresponding values of a 32-entry monolithic iTLB with IA and of a 128-entry monolithic iTLB with IA.

mized, the energy consumption deteriorates significantly.

In summary, our IA approach can provide more energy savings than a multi-level iTLB which uses a 1-entry first level, while not suffering from any performance deficiencies (which a multi-level structure can). Its benefits are more significant when the entries in the first-level iTLB become larger.

4.4. Sensitivity to iL1 Configuration and Page Size

We have experimented with different iL1 configurations for VI-VT iL1 (remember, VI-PT iTLB power is relatively insensitive to iL1 configuration), and the detailed results can be found in [19]. The benefits of IA are more significant at smaller or less associative iL1 configurations, since these incur more misses (which can get high in some commercial workloads [1]) and the iTLB can get in the critical path. On the other hand, IA with VI-PT can provide even lower energy consumption than the default VI-VT, while not suffering from this deficiency. Augmenting VI-VT with IA, is another way of reducing this overhead. As was explained in Section 4.3.1, the CFR may be able to satisfy some of the requests even after an iL1 miss because of its page level coverage.

A larger page size provides better coverage of the CFR,

thus improving the iTLB energy savings with our approaches, and detailed results can be found in [19].

4.5. PI-PT iL1 Lookup

This form of iL1 addressing is not really in fashion because of the additional latency in the critical path of iTLB lookup before iL1 is accessed (as mentioned earlier there are some ways of getting around this if the iL1 indexing can be done with just the page offset bits, in which case it is no different from VI-PT and it restricts iL1 configurations). However, our approach can also be used in conjunction with a PI-PT iL1, as long as we can provide translations most of the time. To investigate this issue, we have conducted experiments with a PI-PT iL1 cache, and the energy and performance results are presented in Table 8. This table compares (i) base PI-PT, (ii) PI-PT with IA, (iii) base VI-PT, and (iv) base VI-VT. All experiments have been performed with our default configuration parameters.

As can be expected, the base PI-PT does much worse than (iii) or (iv) in terms of execution cycles, while consuming as much energy as (iii). However, we find that incorporating IA into PI-PT substantially lowers the execution cycles, bringing its performance within 5.7% of the base VI-PT on the average, while doing significantly better than it in terms of energy. IA with PI-PT comes even more closer to the base VI-VT in terms of cycles (even beating it in three of our six applications). At the same time, it expends less energy than the base VI-VT in three applications. These results suggest that PI-PT (which is largely ignored today) may not be a bad idea at all for iL1 addressing when used in conjunction with our optimizations.

5. Summary of Results and Concluding Remarks

This paper has proposed hardware and software mechanisms for dynamic power optimizations within the iTLB. These mechanisms are intended to reduce the number of times that the iTLB is accessed, and can also work very well in conjunction with other circuit/architectural techniques for furthering the power savings.

Of the different techniques that were proposed and evaluated, the IA strategy which uses compiler analysis to track page boundary crossings, and a simple piece of hardware in conjunction with a branch predictor for branches out of a page, can effectively cut energy consumption by over 85%. It works well on both VI-PT and VI-VT iL1 caches. At the same time, these mechanisms are different from keeping a two-level iTLB (with the first level being 1 entry). In such a structure, a comparison is still needed to find out whether the translation exists, while three of our mechanisms (IA, SoCA, and SoLA) are already sure of this, leading to less energy consumption.

Some of the detailed observations and contributions of this work are in the following:

- Our optimization mechanisms achieve significant iTLB power savings without compromising on perfor-

		VI-PT Energy			VI-VT Energy			VI-VT Cycles		
TLB Size	Benchmark	Base	OPT	IA	Base	OPT	IA	Base	OPT	IA
1	177.mesa	6.585	0.245 (3.72%)	0.2719 (4.13%)	0.241	0.043 (17.99%)	0.046 (19.23%)	284.5	230.7 (81.10%)	232.8 (81.82%)
	186.crafty	7.144	0.337 (4.73%)	0.391 (5.47%)	0.574	0.089 (15.50%)	0.096 (16.89%)	510.7	415.4 (81.33%)	421.2 (82.47%)
	191.fma3d	6.804	0.556 (8.18%)	0.599 (8.81%)	0.216	0.040 (18.91%)	0.046 (21.37%)	252.1	207.2 (82.17%)	211.2 (83.76%)
	252.eon	7.353	0.642 (8.73%)	0.734 (9.97%)	0.532	0.082 (15.54%)	0.088 (16.55%)	436.5	340.6 (78.02%)	344.0 (78.80%)
	254.gap	6.560	0.269 (4.10%)	0.296 (4.51%)	0.148	0.030 (20.32%)	0.033 (22.39%)	224.9	188.9 (83.95%)	191.8 (85.22%)
	255.vortex	6.722	0.456 (6.79%)	0.478 (7.11%)	0.468	0.112 (23.91%)	0.117 (25.15%)	499.7	386.2 (77.27%)	389.8 (78.01%)
8,FA	177.mesa	99.256	2.454 (2.47%)	2.843 (2.86%)	3.440	0.472 (13.73%)	0.504 (14.65%)	250.5	206.8 (82.53%)	206.9 (82.56%)
	186.crafty	111.991	3.242 (2.89%)	4.016 (3.59%)	7.976	1.091 (13.67%)	1.172 (14.69%)	399.5	378.5 (94.73%)	378.6 (94.78%)
	191.fma3d	102.182	4.475 (4.37%)	5.132 (5.02%)	3.316	0.469 (14.16%)	0.520 (15.68%)	252.1	187.4 (74.33%)	187.8 (74.51%)
	252.eon	121.118	6.145 (5.07%)	7.544 (6.23%)	5.187	1.024 (19.75%)	1.088 (20.98%)	331.7	309.0 (93.16%)	309.8 (93.38%)
	254.gap	101.760	2.466 (2.42%)	2.977 (2.93%)	1.978	0.350 (17.72%)	0.384 (19.43%)	183.7	174.4 (94.95%)	175.5 (95.56%)
	255.vortex	98.919	4.392 (4.44%)	4.708 (4.76%)	6.280	1.463 (23.30%)	1.532 (24.39%)	378.9	352.4 (92.99%)	353.1 (93.20%)
16,2w	177.mesa	146.525	3.070 (2.09%)	3.664 (2.50%)	4.596	0.811 (17.66%)	0.878 (19.11%)	237.3	216.8 (91.39%)	218.8 (92.20%)
	186.crafty	166.305	4.266 (2.56%)	5.405 (3.25%)	11.273	1.097 (9.74%)	1.217 (10.79%)	353.1	336.1 (95.18%)	336.3 (95.24%)
	191.fma3d	151.138	6.537 (4.32%)	7.510 (4.96%)	4.204	0.624 (14.85%)	0.699 (16.63%)	188.5	180.6 (95.83%)	180.8 (95.93%)
	252.eon	179.744	8.814 (4.93%)	10.88 (6.05%)	7.389	1.292 (17.48%)	1.374 (18.59%)	308.2	289.8 (94.03%)	290.1 (94.15%)
	254.gap	150.565	3.512 (2.33%)	4.264 (2.83%)	2.818	0.448 (15.89%)	0.488 (17.31%)	175.4	169.2 (96.49%)	169.4 (96.57%)
	255.vortex	145.956	6.156 (4.21%)	6.623 (4.53%)	9.047	1.938 (21.43%)	2.044 (22.59%)	356.7	332.2 (93.12%)	333.7 (93.54%)
32,FA	177.mesa	109.075	2.199 (2.01%)	2.625 (2.41%)	3.345	0.370 (11.06%)	0.401 (11.99%)	196.1	189.4 (96.60%)	189.5 (96.64%)
	186.crafty	124.110	3.162 (2.54%)	4.011 (3.23%)	8.385	0.795 (9.48%)	0.884 (10.55%)	350.5	333.5 (95.16%)	333.8 (95.23%)
	191.fma3d	112.685	4.781 (4.24%)	5.517 (4.89%)	3.040	0.380 (12.52%)	0.440 (14.48%)	176.6	170.2 (96.39%)	170.4 (96.48%)
	252.eon	134.544	6.145 (4.56%)	7.689 (5.71%)	5.221	0.742 (14.21%)	0.808 (15.48%)	274.7	264.7 (96.36%)	264.9 (96.43%)
	254.gap	112.205	2.506 (2.23%)	3.067 (2.73%)	2.005	0.261 (13.06%)	0.291 (14.52%)	165.6	161.8 (97.73%)	161.9 (97.77%)
	255.vortex	108.424	3.944 (3.63%)	4.293 (3.96%)	6.345	1.151 (18.14%)	1.217 (19.18%)	310.5	298.7 (96.20%)	299.0 (96.28%)

Table 6. Energy consumptions (in millijoules) with different iTLB configuration (VI-PT and VI-VT) and execution cycles (in millions) with different iTLB configurations (VI,VT) - 1 entry, 8 entry FA, 32 entry 2-way, 32 entry FA. The numbers within the parantheses under the OPT and IA columns show their energy and cycles as a percentage of the base case.

mance. Their importance grows with higher iL1 miss rates (as in database applications) and larger page sizes (which is a trend these days). They can work very well with large iTLB structures (that can possibly consume more power and take longer to lookup), without them getting into the common case.

- These solutions are also very effective in removing the iTLB from the critical path of a VI-VT lookup mechanism, and can thus turn out to cut execution cycles as well in such cases.
- While a VI-VT mechanism can automatically provide good iTLB power savings over VI-PT, their drawback is in possible performance degradation with higher iL1 miss rates. At the same time, there are some drawbacks with VI-VT (even if we are to avoid cache aliasing with extra bits), since write-backs need to work with physical addresses — consequently, some VI-VT mechanisms keep both physical and virtual tags with each cache line to handle write-backs [15, 17]. Our mechanisms, on the other hand, can take VI-PT and provide as good power savings as VI-VT (if not better) without incurring any performance degradation. Further, they can take VI-VT and improve its performance to approach that of VI-PT while furthering its power savings. Our contributions thus make it possible to remove the iTLB power consumption from being an issue for iL1 design (indexing/lookup strategy).
- We have even ventured further to examine the ramifications with a PI-PT iL1 which is largely ignored

Benchmark	1-entry	8-entry, FA	16-entry, 2-way	32-entry, FA
177.mesa	437.6	244.5	198.0	188.1
186.crafty	650.7	372.8	333.9	331.7
191.fma3d	748.8	185.5	178.9	169.3
252.eon	897.4	331.6	310.5	263.1
254.gap	426.2	181.9	172.4	161.3
255.vortex	717.0	372.5	345.8	293.9

Table 7. Execution cycles (in millions) with different iTLB configurations for IA (VI-PT).

today (unless with very specific iL1 configurations). We have shown that our mechanisms can reduce the performance penalty with this kind of iL1 addressing considerably to make it competitive with a VI-PT iL1. Further, VI-PT and VI-VT iL1 caches require translations (storing physical addresses within each cache block) for write-backs increasing the iL1 complexity and power dissipation. On the other hand, PI-PT does not require this, and with our scheme we can provide the performance and power consumption of these fancier cache indexing mechanisms without the drawbacks.

- This work can be viewed as taking another step in the direction of removing the TLB altogether that was investigated in [17]. We are now less dependent on the actual iTLB structure in terms of its lookup latency. From the hardware point of view, this strategy can save on-chip area, in addition to optimizing power con-

Benchmark	PI-PT (Base)		PI-PT (IA)		VI-PT (Base)		VI-VT (Base)	
	E	C	E	C	E	C	E	C
177.mesa	104.01	250.6	2.48	195.5	109.07	188.1	3.34	196.1
186.crafty	115.24	410.4	3.70	343.7	124.11	331.7	8.38	350.5
191.fma3d	104.47	241.6	5.23	189.8	112.68	169.3	3.04	176.6
252.eon	115.03	330.4	6.77	282.9	134.54	263.1	5.22	274.7
254.gap	104.11	214.7	2.83	167.6	112.20	161.3	2.00	165.6
255.vortex	106.00	360.9	4.24	308.6	108.42	293.9	6.34	310.5

Table 8. iTLB energy (E, in millijoules) and cycle (C, in millions) comparison.

sumption and power density.

Finally, it is to be emphasized that the dynamic energy savings with our mechanisms are more a consequence of the reduced number of iTLB accesses, and the percentage improvements are likely to hold with technology or circuit level improvements.

Having identified the potential of this different philosophy in generating physical addresses for the instruction stream, we are currently examining similar approaches for data references. We are also looking to perform code layout transformations, and data/code restructuring to benefit from the reuse of the translation within the CFR.

References

- [1] A. Ailamaki, D. J. DeWitt, M. D. Hill, and D. A. Wood. DBMSs on a modern processor: where does time go?. In Proc. *25th International Conference on Very Large Data Bases*, Edinburgh, UK, September 1999.
- [2] R. Balasubramanian, D.H. Albonese, A. Buyuktosunoglu, and S. Dwarkadas. Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures. In Proc. *33rd International Symposium on Microarchitecture*, pp. 245–257, December 2000.
- [3] P. Bose et al. Early-stage definition of LPX: a low power issue-execute processor prototype. In Proc. *2nd Workshop on Power-Aware Computer Systems* (in conjunction with HPCA’02), Cambridge, MA, February 2, 2002.
- [4] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. In Proc. *International Symposium on High-Performance Computer Architecture*, January, 2001.
- [5] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In Proc. *27th International Symposium on Computer Architecture*, Vancouver, British Columbia, June 2000.
- [6] D. Burger, T. Austin, and S. Bennett. Evaluating future microprocessors: the SimpleScalar tool set. *Technical Report CS-TR-96-103*, Computer Science Department, University of Wisconsin, Madison, July 1996.
- [7] F. Catthoor, S. Wuytack, E. D. Greef, F. Balasa, L. Nachtergaele, and A. Vandeappelle. *Custom Memory Management Methodology – Exploration of Memory Organization for Embedded Multimedia System Design*. Kluwer Academic Publishers, 1998.
- [8] M. Cekanov, and M. Dubois. Virtual-address caches. Part 1: problems and solutions in uniprocessors. *IEEE Micro*, 17(5):64–71, September–October, 1997.
- [9] T-C. Chiueh and R. H. Katz. Eliminating the Address Translation Bottleneck for Physical Address Cache. In Proc. *ASPLOS*, pages 137–148, 1992.
- [10] J-H. Choi, J-H. Lee, S-W. Jeong, S-D. Kim, and C. Weems. A low-power TLB structure for embedded systems. *IEEE Computer Architecture Letters*, Volume 1, January 2002.
- [11] D. Folegnani and A. Gonzalez. Energy-effective issue logic. In Proc. *28th International Symposium on Computer Architecture*, Goteborg, Sweden, June 30 – July 4, 2001.
- [12] K. Ghose and M. B. Kamble. Reducing power in superscalar processor caches using subbanking, multiple line buffers, and bit-line segmentation. In Proc. *1999 International Symposium Low Power Electronics and Design*, 1999, pages 70–75.
- [13] J. L. Henning. SPEC2000: Measuring CPU performance in the new millennium. *IEEE Computer Magazine*, July 2000, pp. 28–35.
- [14] K. Inoue, T. Ishihara, and K. Murakami. Way-predicting set-associative cache for high performance and low energy consumption. In Proc. *International Symposium on Low Power Electronics and Design*, pages 273–275, 1999.
- [15] Intel StrongARM Processor. http://www.intel.com/design/pca/applicationsprocessors/1110_brf.htm
- [16] Itanium Manual. <http://developer.intel.com/design/itanium/manuals.htm>.
- [17] B. Jacob and T. Mudge. Uniprocessor virtual memory without TLBs. *IEEE Transactions on Computers*, vol. 50, no. 5, pp. 482–499, May 2001.
- [18] T. Juan, T. Lang, and J. J. Navarro. Reducing TLB power requirements. In Proc. *International Symposium on Low Power Electronics and Design*, 1997.
- [19] I. Kadayif, A. Sivasubramaniam, M. Kandemir, G. Kandiraju, and G. Chen. Generating physical addresses directly for saving instruction TLB energy. *Technical Report CSE-TR-02-012*, Department of Computer Science and Engineering, Pennsylvania State university, June 2002.
- [20] S. Kim. Low power MMU design for embedded processors. <http://supercom.yonsei.ac.kr/temp/sam.ppt>
- [21] J. Knight, and P. Rosenfeld. Segmented virtual to real translation assist. *IBM Technical Disclosure Bulletin*, 27(2):1077–1078, July 1984.
- [22] R. Maddock, B. Marks, J. Minshull and M. Pinnell. Hardware address relocation for variable length segments. *IBM Technical Disclosure Bulletin*, 23(11):5186–5187, April 1981.
- [23] D. Parikh, K. Skadron, Y. Zhang, M. Barcella, and M. Stan. Power issues related to branch prediction. In Proc. *Eighth International Symposium on High-Performance Computer Architecture*, pp. 233–244, February 2002.
- [24] G. Reinman and N. P. Jouppi. CACTI 2.0: an integrated cache timing and power model. Compaq, WRL, *Research Report 2000/7*, February 2000.
- [25] D. Sima, T. Fountain, and P. Kacsuk. *Advanced Computer Architecture: A Design Space Approach*, Addison-Wesley, 1997.
- [26] W. D. Strecker. VAX-11/780: A virtual address extension to the DEC PDP-11 family. In Proc. *AFIPS NCC*, vol. 47, pp. 967–980, 1978.
- [27] N. Vijaykrishnan, M. Kandemir, M. J. Irwin, H. Y. Kim, and W. Ye. Energy-driven integrated hardware-software optimizations using SimplePower. In Proc. *International Symposium on Computer Architecture*, June 2000.
- [28] J. Yang, Y. Zhang, and R. Gupta. Frequent value compression in data caches. In Proc. *33rd International Symposium on Microarchitecture*, pages 258–265, Monterey, CA, December 2000.
- [29] V. Zyuban and P. Kogge. Split register file architectures for inherently lower power microprocessors. In Proc. *Power-Driven Microarchitecture Workshop* (in conjunction with ISCA’98), pages 32–37, 1998.