A Low-Power dTLB Design Based on Memory Region Encoding*

XIE Jinsong, LI Xianfeng, TONG Dong, CHEN Jie and CHENG Xu
(Microprocessor Research and Development Center, Peking University, Beijing 100871, China)

Abstract — Translation lookaside buffers (TLBs) consume significant power due to their highly associative structure. This is getting worse with the increasing width of Virtual page number (VPN) and TLB capacity in 64-bit computing. In this paper, we present a new data TLB (dTLB) design that reduces VPN width to a large extent, thereby saving considerable power during TLB lookups. This design is motivated by an observation: the VPN can represent a much larger set of memory regions than what can be cached by the TLB at any time. We exploit this redundancy by encoding some high-order VPN bits with a shorter memory region id before the VPN is sent to dTLB. The consistency of the encoding and the recycling of memory region ids are taken cared by a small amount of hardware with little timing/power overhead. Experimental results on SPEC CPU2000 show a 37% energy reduction of dTLB with negligible performance penalty.

Key words — Memory region, Low-power, Data TLB.

I. Introduction

Virtual memory has become a commonplace in modern computers. To speed up virtual-to-physical address translation, an on-chip memory known as Translation lookaside buffer (TLB) is used to cache recently accessed page table entries. Since it is on the critical path, and a TLB might incur an expensive memory table walk, it has small size (typically 32 to 64 entries) and high associativity for low access latency and high hit rate. In particular, the tag part containing Virtual page numbers (VPNs) is often implemented with Content addressable memories (CAMs) while the data part containing Physical page numbers (PPNs) is implemented with SRAMs.

The consequence of this design is high power consumption in the CAM structure, where each lookup triggers multiple VPN comparisons. With the fast growth of program working set size in recent years, processors are being designed with larger TLBs. Sun T2 has 64-entry instruction TLB (iTLB) and 128-entry dTLB[17], while AMD Opteron has a two-level iTLB/dTLB structure, in which a 40-entry L1 TLB is backed by a 512-entry L2 TLB[10]. On the other hand, the advent of 64-bit computing and the adoption of Address space identifier (ASID) significantly increase VPN width. These trends together will push the power consumption of TLBs to even higher levels. Wider VPN also has a negative impact on TLB access latency.

TLB power reduction has been investigated by many other researchers. Most existing techniques save TLB power by trying to reduce the number of TLB entries accessed each time, such as classical banked design[12,13], multi-level design[10,18], and their variants[14,11].

In contrast to these entry-reduction techniques, we address TLB power and latency issues by trying to reduce VPN width in this paper, which we call a width-reduction technique. We choose dTLB as our target because iTLB power reduction is easier to achieve given the more predictable pattern of instruction streams. By raising view granularity from memory page to memory region of mega-byte size, we observe that at any time, the active memory regions (each of which contains at least one page with its entry in TLB) are very few. In other words, the VPN can represent a much larger set of memory regions than what can be cached by TLB. We exploit this redundancy by encoding some high-order VPN bits to a shorter memory region id before the VPN is sent to TLB. The compacted VPN will then be compared with those in the CAM, which are also in compacted form. Thus power reduction is achieved with narrowed CAM bit arrays, and latency is reduced with smaller comparators.

We choose SPEC CPU2000 as benchmarks, and use SimpleScalar[11] along with CACTI 4.0[19] as tools for experimental evaluation. The results show a 37% dTLB energy reduction with negligible performance penalty.

The rest of the paper is organized as follows. Section II introduces the concept of memory region and the motivating observation. In Section III, the new dTLB design is described with details. Experimental results are given in Section IV. Section V discusses the related work. Conclusions and future work appear in Section VI.

II. Motivation

In a conventional TLB design, a virtual address is viewed in two parts: a page offset and a virtual page number. The page offset corresponds to the low-order bits, and the rest high-order bits are identified as VPN. The cache-like TLB has two

*Manuscript Received Aug. 2007; Accepted Jan. 2008. This work is supported by the National Natural Science Foundation of China (No.60703067).
parts: a tag part storing VPNs of recent translations, and a data part storing the corresponding PPNs. The tag part is typically implemented with CAMs because of its high associativity. Before proceeding to further discussions, we formally introduce a few terms used in this paper and a reference TLB design used for comparison.

A memory region is defined as an area of $2^K$ ($K \geq 0$) contiguous pages with identical $|\text{VPN}| - K$ high-order bits among the constituent pages ($|\text{VPN}|$ is VPN width); a cached page is one with its translation in TLB; and an active memory region is one that contains at least one cached page.

Table 1. Virtual memory and reference
number of memory regions to be represented, and set the width of memory region id accordingly.

By analyzing the data in Fig.1, we choose a memory region size of 2MB (21 bits), which is 512 pages of 4kB. With this size, we can encode 27 high-order bits out of the 36-bit VPN. We use 4 bits to encode the 27 high-order bits, which means 16 memory regions can be covered simultaneously. The encoded VPN comprises two parts: a 4-bit memory region id and a 9-bit page number. For convenience, the encoded VPN is called eVPN, and VPN is reserved for its original meaning. The 13-bit eVPN is only 36% of the 36-bit VPN in width.

Note in this paper, there are two important terms which are very similar: VPN encoding and memory region encoding. They have slight difference in meaning. VPN encoding refers to the whole process from the original VPN to eVPN, in which memory region encoding performs part of the work. Or from another angle, VPN encoding stands for a more abstract approach that aims to reduce VPN width, while memory region encoding is a concrete technique that achieves this goal. In some cases, these two terms can be used interchangeably.

2. New TLB and translation

The block diagram of the new TLB is shown in Fig.2. Only the tag part is drawn, as there is no change to the data part. In this design, a translation is accomplished in two steps: memory region encoding and eVPN lookup.

The part on the left-hand side performs memory region encoding. It is activated in the same cycle as address generation, therefore one clock cycle before the lookup operation in a normal TLB. In a typical RISC architecture, an Address generation unit (AGU) computes a virtual address with a base address register and an offset. The offset is usually 16 bits. According to Ref.[6] (Appendix B.4), over 90% offsets can be represented with less than 10 bits for CINT2000, and for CFP2000, the ratio is close to 60%. On the other hand, the VPN part to be encoded starts at bit 21, thus it needs many carry bits in a row to change the high-order VPN bits during AGU's address calculation. This is an event of very low probability. Therefore we can use the base address register, and send its high-order VPN bits to a structure called Region store for VPN encoding. The Region store is a CAM structure that saves current VPN encodings. If a match happens, it means the virtual address belongs to an existing memory region. In this case, the corresponding memory region id is obtained and latched for use in the next clock cycle. Otherwise it is a region miss, which means the incoming virtual address does not fall in any existing memory region. In this case, an encoding for a new memory region needs to be created; the details will be covered in next subsection when we discuss recycling. Note that the 4-bit memory region id can be directly obtained from the match lines of the CAM; thus, unlike most cache-like structures, it does not need an data part for storing memory region ids.

The part on the right-hand side performs eVPN lookup. It is activated after the complete virtual address has been generated from the AGU. This is also the clock cycle when a TLB lookup is carried out in a normal TLB design. The 4-bit region id latched last cycle is concatenated with the 9 low-order VPN bits to form an encoded VPN. Then the eVPN is sent to the TLB tag part, now called eVPN Store, for a lookup. Each entry in the eVPN Store is a 13-bit tag, in which the 4 high-order bits denote a memory region, and the 9 low-order bits denote a page number in that region. As can be seen, this 128-entry eVPN Store has a much narrower bit array compared to the original TLB tag part.

This design does not lengthen TLB data path, because memory region encoding is performed one cycle in advance. It does not introduce an extra clock cycle for TLB lookup since Region Store is obtained one cycle in advance with a partial address. But this is essentially a speculative technique. Although in most cases, the early memory region encoding using a partial virtual address is correct, it is possible that the offset added by AGU changes the high-order VPN bits. This is detected by monitoring bit 21, the first bit in the VPN portion to be encoded. If it changes after AGU’s operation, the early memory region encoding is incorrect, and an “encode fault” signal will be set, as shown in Fig.2. Encode fault will trigger a new memory region encoding in this cycle, and an eVPN lookup in the following, thus is has a single-cycle penalty.

It is possible to implement a non-speculative design. An AGU operation usually has a delay shorter than the clock cycle, leaving considerable slack for other operations. This has been exploited in Pentium 4[7], in which an ALU performs two operations in a clock cycle. In addition, since the Region store has only 16 entries, its bitline pre-charge and discharge take less time than a 128-entry CAM, making its lookup a much faster operation. Therefore, it is possible to perform AGU operation and memory region encoding sequentially in the same cycle, and perform eVPN lookup in the next cycle. In this design alternative, encode fault is completely avoided.

Memory region encoding also has a certain amount of overhead on power, mainly contributed by the Region store. Again, because of its small number of entries, its power consumption is much lower compared to that of the 128-entry eVPN Store, and even lower than the original TLB tag part.

3. Memory region id recycling

As the whole program working set is far beyond 16 memory regions that can be represented by the Region store, a memory region id cannot be globally bound to a specific memory region. In other words, a memory region id recycling mechanism is needed.

Selecting an entry for recycling is much like the cache/TLB replacement problem, and there are many choices available. We decide to use the LRU policy, not only because it pro-
duces fewer misses in usual, but also because we can afford it here. As a program is supposed to stay in a set of memory regions in a relatively long while before it accesses a new memory region (The low TLB miss rate proves this assumption), recycling happens infrequently. Also, the Region store has only a few entries. Thus it is justified to use a few additional book-keeping bits and a multi-cycle operation to find the least recently used entry.

Once a candidate is picked, the main action for recycling is to flush all relevant eVPN entries of the old region. For example, if a region 0010 is selected, eVPN entries with their first four tag bits being 0010 will be flushed out.

The multi-entry flushing can be done within a cycle with specific circuitry added the CAM cells. For example, in SUN OpenSPARC T1[16] processor, it has a TLB Megacell that supports invalidation of multiple TLB entries in a single cycle. The flushing operation can be hidden by the TLB missing, because a flushing is always accompanied by a TLB miss, thus it does not contribute additional performance penalty.

After flush operation, the rest work is simple just to place the high-order VPN bits of the incoming VPN into the recycled Region Store entry, and a new active memory region is encoded. This accomplishes the memory region id recycling. In this paper, we assume a single-cycle flushing implementation.

2. Column “Ref. TLB” gives the energy and access time of the reference TLB. Column “FF”*** means region flushing and encoding fault and includes average energy consumption of region flush and encoding fault. The ratios of region flushes and encoding faults are given in Table 3. Our experiment shows a 37% energy reduction and an 8.36% reduction on access latency.

To further understand where the savings come from, a TLB energy dissipation breakdown on the major components is presented in Table 4. Energy reduction of the new TLB design is contributed by eVPN store (data part inclusive) as well as Region store (no data part), which correspond to the eVPN column and Region column respectively. The first row tag comparison gives the energy consumption of the CAM structure in the respective data structure, and the rest rows are the various components in the data part. Since the Region store has no data part, the data of the corresponding rows are not applicable. Note this table only presents energy consumption of a single port, thus the overall energy consumption on the dual-port TLB should be doubled as shows in the last row in the Table 4.

<table>
<thead>
<tr>
<th>Table 4. TLB energy dissipation breakdown</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>Tag comparison</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
observe any energy savings; this is reasonable, as the data part of the TLB is largely left intact.

2. Performance

The performance is measured in terms of TLB miss rate and its equivalent. In the new TLB, in addition to TLB miss, there is a source of TLB related penalty: encode fault, which has one cycle penalty. For ease of comparison, it penalty is translated into the number of TLB misses. Suppose a TLB miss takes 120 clock cycles, each encode fault is equivalently 0.0083 TLB miss***.

TLB miss rates of CINT2000 and CFP2000 on the reference TLB design are presented in Table 5. Fig.3 and Fig.4 give the relative TLB miss rate of the new design over the reference design for CINT2000 and CFP2000 respectively (a few benchmarks with their miss rates below 0.01% in the reference design are omitted). We have the following findings. First, TLB miss rate is almost the same as the reference TLB across CINT2000/CFP2000 benchmarks.

![Fig. 3. Relative miss rate of the new TLB over the reference design for CINT2000](image1)

![Fig. 4. Equivalent TLB miss rate contributed by encode fault](image2)

Second, for three CFP2000 benchmarks (wupwise, applu, sixtrack), we observe significant amount of encode fault in terms of relative TLB miss rate. This indicates that for some floating-point programs, their offset operands are not always small in magnitude. Fortunately, the rise of their absolute TLB miss rate is very low (0.004%, 0.031% and 0.019% respectively). Thus the high relative TLB miss rate has little impact on actual performance.

V. Related Work

TLB power has received great attention in the past decade, and various techniques with different tradeoffs on performance, energy and implementation cost have been proposed.

Two common techniques are banked design[12,13] and multi-level design[10,19]. They save power by accessing fewer TLB entries each time. Banked TLB has higher miss rate because associativity is sacrificed, while multi-level TLB suffers a performance penalty when a lookup failed in the smaller L1 TLB takes additional cycles to access the L2 TLB. Since multi-level TLB is also very effective in reducing access latency, it has been adopted in some commercial processors such as AMD Opteron[10] and Unity-863 microprocessor[15]. A recent work by Chang and Lan[4] provides methods that reduce performance penalty of these techniques.

Ballapuram et al.[29] observe that multiple virtual addresses issued in the same cycle or in consecutive cycles may go to the same page. They exploit this finding by compacting synonymous addresses into a single one. This technique may not work well if address synonymy is presented in a form of interleaved streams. Its hardware complexity is another concern, e.g. it needs $C(N,2)$ high-fanout comparators for an $N$-wide machine.

Kandemir et al.[8] propose a combined software and hardware technique. It employs a set of Translation registers (TRs) to store frequently used translations. To use these TRs, extended load/store instructions are added to specify whether an address translation goes through a TLB lookup or is obtained from a specific TR. The compiler is responsible for making such decisions (which means no dynamic adaptation is made from runtime information). This technique is reported to be effective for array-based benchmarks.

Lee and Ballapuram[11] present a decoupled dTLB design based on program semantic regions. In particular, it has two small TLBs: sTLB and gTLB for stack and global data regions respectively, and a larger hTLB for heap data. It saves energy because most stack and global address translations go to the smaller TLBs. This technique requires OS participation. It has a longer data path than a single-bank design, which may affect the clock rate. This work shares some similarities with ours, in that both rely on the concept of memory region. But our definition of memory region is more general and flexible. More importantly, our objective is to reduce VPN width instead of accessing TLB entries.

Delaluz et al.[5] propose a power-saving technique by resizing TLB according to program needs during different execution intervals. It relies on a carefully designed training process to estimate the demand on TLB size at the beginning of each

*** Generally, 64-bit computer system uses a three-level or more hierarchical page table to map the address space. A TLB miss must access main memory (i.e. page table walking) three times at least if hardware mechanism handles table walking. We assume every page table walking takes 40 clock cycles, thus three times will take 120 clock cycles at least. If using software mechanism to handle TLB miss, this will take much more cycles to complete table walking.
interval. At the circuitry level, hardware is inserted into CAM cells to turn them on/off selectively.

In a recent work, Ballapuram et al.\(^3\) propose a dTLB design for reducing the number of VPN bits compared in TLB lookup. Their work is based on the observation that the entropy of VPNs is low. This observation shares some similarities with our observation on the redundancy of high-order VPN bits. However, their solutions are limited to stack and global data, while our work covers all types of data accesses, including heap data, which often exhibit much more irregular access patterns than stack and global data. Another difference is that our technique physically reduces the number of VPN bits stored in the TLB (the eVPN store), while their work still maintains the full VPN bits they only achieve a reduction on the number of compared bits for stack and global data accesses.

Petrov and Orailoglu\(^{14}\) pursue a software approach that saves energy by reducing VPN width. Despite the same objective as ours, there exist some important differences. First, it is only applicable to specific program parts called “hot spots”. Second, it simply disregards some high-order bits only if they are confirmed to be identical for all addresses in a hot spot; disregarding is very restrictive compared to encoding used in this our work, and it is likely to lose a lot opportunities. Third, it is a software approach that needs substantial modifications to compiler and OS, especially for handling dynamically allocated data and dynamically linked libraries; our technique poses no need on software modification. Last, the compiler responsible for the offline part has to make conservative decisions because it lacks accurate runtime information.

Superpaging\(^{18}\) is a mechanism to extend the TLB coverage. It uses a single entry for reserving a large number of consecutive virtual pages. A superpage is a physical object, which occupies a large chunk memory, and its drawback is that it may waste considerable memory when only a small part is actually in use, which is often the case. Because of this reason, superpages should be used with great care. In contrast, memory region in this paper is a virtual concept; it captures program behavior without paying the cost of extra physical resources (in fact, it reduces the size of the TLB tag part). In addition, memory region encoding is applicable to a much broader scope: the program needs not to exhibit a special data access pattern like what superpaging is based on.

In summary, most related works fall in the category of reducing accessed TLB entries. Our work is fundamentally different: it saves TLB energy by reducing VPN width. The related works that fall in the same category as ours are a software approach\(^{14}\) and entropy-based design\(^{3}\) which both suffer some restrictions absent here. Meanwhile, our work is complementary to those in the other category, and they can work in combination for further improvements.

**VI. Conclusions**

TLB power consumption has received much attention in the past decade. Most existing techniques save TLB power by trying to reduce the number of accessed entries. In this paper, we pursue a fundamentally different approach. It aims to reduce the width of TLB tag array, thereby saving considerable power. Our work is motivated by two factors: (1) current architectures are using wider VPNs to provide larger memory space; (2) the VPN can represent a much larger set of memory regions that can be cached by the TLB at any time. We propose a technique called memory region encoding to exploit this redundancy. The results show significant power reduction with negligible performance penalty. Furthermore, if the virtual address is increasing from 40-bit by now to 48-bit, our techniques will become more significant in reducing the dTLB power. Meanwhile, this work targets dTLB, but the proposed technique also works for ITLB, which is easier to handle.

In the future, we will try to combine this width-reduction method with existing entry-reduction techniques for further TLB power saving. Meanwhile, the memory region concept and its encoding technique can be applied to study other memory-related issues, as they provide a higher level of view on memory, which is in continuous growth to meet application needs.

**References**


