Entropy-Based Low Power Data TLB Design

Chinnakrishnan Ballapuram
School of ECE
Georgia Institute of Technology
Atlanta, GA 30332
chinnak@ece.gatech.edu

Kiran Puttaswamy
School of ECE
Georgia Institute of Technology
Atlanta, GA 30332
kiranp@ece.gatech.edu

Gabriel H. Loh
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
loh@cc.gatech.edu

Hsien-Hsin S. Lee
School of ECE
Georgia Institute of Technology
Atlanta, GA 30332
leehs@gatech.edu

ABSTRACT

The Translation Look-aside Buffer (TLB), a content addressable memory, consumes significant power due to the associative search mechanism it uses in the virtual to physical address translation. Based on our analysis of the TLB accesses, we make two observations. First, the entropy or information content of the stack virtual page numbers is low due to high spatial locality of stack memory references. Second, the entropy of the higher order bits of global memory references is low since the size of the global data is determined and fixed during compilation of a program. Based on these two characteristics, we propose two techniques: an entropy-based speculative stack address TLB and a deterministic global address TLB to achieve energy reducing. Our results show an average of 47% energy savings in the data TLB with less than 1% overall performance impact.

Categories and Subject Descriptors

B.3.2 [**Low-Power Architectures**]: Virtual memory, TLB, Associative memory

General Terms

Design, Experimentation, Performance

Keywords

Entropy, Low-power TLB, Spatial and temporal locality

1. INTRODUCTION

Power and thermal control is a first class priority in designing the next generation embedded and high performance microprocessors [13]. Among the microprocessor components, the ever-increasing on-chip memory and the memory sub-system have been a primary focus for architects to explore energy reduction opportunities. As most processors support virtual memory, the Translation Lookaside Buffer (TLB) is an indispensable component in the processor design. The TLB, a content addressable memory to expedite address translation, provides virtual-to-physical address translation for each virtual address accessed. The TLB can draw a considerable amount of power as it is typically organized as fully or highly associative structure and is accessed upon every instruction and data fetch. Ekman et al. in [4] reported the TLB power consumption to be 20-25% of the total cache power consumption, and Fan et al. [5] reported 13% of the total processor power consumption for a unified TLB. In modern embedded and high-performance processor design, it may be infeasible to increase the size of TLB due to power and thermal constraints. The TLB power consumption increases further as we move from 32-bit processors to 64-bit processors since the TLB will have to compare and match almost twice the number of bits.

The virtual page numbers (VPNs) typically exhibit a large degree of spatial and temporal locality. Due to the spatial and temporal locality of the data accesses, a given sequence of data accesses may all map to the same physical page as a page size is typically much larger than a normal cache line. Therefore, very little information is conveyed by consecutive virtual page number lookups in the TLB. Also, many applications do not utilize the entire address space, resulting in rather stable higher order bits or low entropy in the VPN. In this paper, we analyze the entropy of the VPNs exhibited in the memory reference stream and exploit their characteristics for energy reduction opportunities. Each memory access by a running program carries a certain amount of information. The entropy or information rate is a measure of unexpectedness and varies with the program behavior. In particular, we find that the stack references contain the least information content, and are thus highly predictable. As the number of the pages are fixed during compile time for the global static data, most of the higher order bits in the global data memory references do not change (zero entropy) during the program execution.

CASES'06, October 23–25, 2006, Seoul, Korea

Thus, a small number of VPN bits is sufficient for address translation. To exploit these properties, we propose a novel *Entropy based SPeculative - Translation Lookaside Buffer (ESP-TLB)* mechanism for stack references, and an *Entropy based DeTerministic - Translation Lookaside Buffer (EDT-TLB)* mechanism for global static references to reduce the overall data TLB energy. These mechanisms are complexity-effective and power-efficient, with minimal impact on the performance and low hardware overhead. The contributions of this paper are:

- We quantify the entropy content of the VPNs for data TLB lookups.
- We propose two novel entropy-based schemes: a speculative address translation for stack addresses, and a deterministic address translation for global static addresses that exploit the low entropy in the respective VPNs. We show that our techniques provide an average of 47% energy savings with less than 1% overall performance impact.

The rest of this paper is organized as follows: Section 2 provides an overview of entropy and its measurement. Section 3 motivates our work by characterizing the entropy content of the VPNs. Section 4 presents entropy based address translation mechanism. Section 5 presents our simulation results. Section 6 discusses related work. Section 7 concludes the paper.

2. OVERVIEW OF ENTROPY AND ITS MEASUREMENT

The entropy or information content is a measure of "uncertainty" or "unpredictability" of a random variable X [16]. In other words, simple repetitive patterns contain low entropy. If X is a random variable that represents the VPN during the program execution, then the average amount of information or the zeroth order Markov source entropy is given by

$$H_0 = -\sum_{i=1}^{N} p(x_i)\ log_2\ p(x_i)\ ...\ (1)$$

where $p(x_i)$ is the probability of occurrence of VPN x_i, $-log_2\ p(x_i)$ is called self information, and H_0 is called the average self information. The entropy has the dimension of bits per VPN reference.

For example in a virtual memory model, if the virtual address is 32-bits and the page size is 4KB (12-bit index), the VPN size will be 20 bits. It means the program can access any of the 2^{20} virtual pages during the dynamic program execution.

First, let's assume all the virtual addresses refer to the same page. The average information for this type of VPN accesses is zero by applying Eq.(1). Because there is only one x_i, and the probability of occurrence of this x_i is 1, and log 1 is equal to zero. This means that there was no useful information in this type of repetitive behavior.

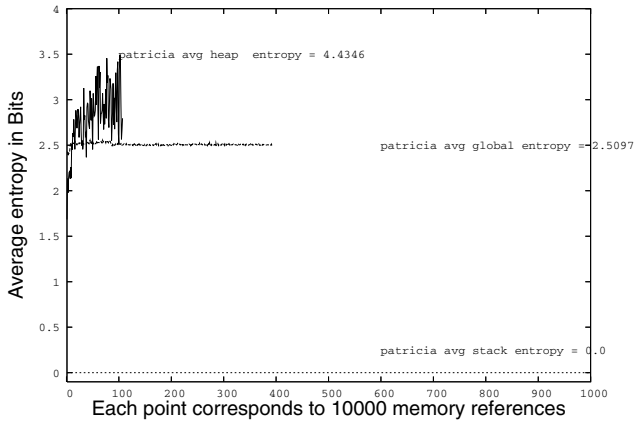
Now assume each of the virtual address refers to a unique page. Since there are 2^{20} references, the average information is 20 bits/VPN reference by applying Eq.(1). This is the maximum amount of information that can be obtained. Here x_i varies from 0 to 2^{20}-1.

The first order Markov source entropy is a measure of the unpredictability of the next VPN given the knowledge of the previous VPN. The weighted average first order entropy is given by

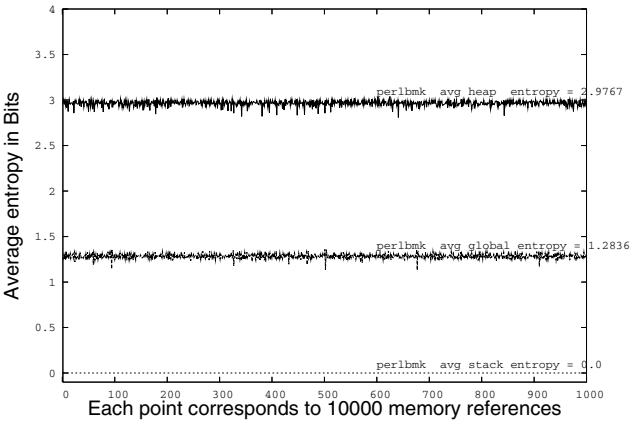
$$H_1 = -\sum_{i=1}^{N} p(x_i) \sum_{j=1}^{N} p(x_j/x_i)\ log_2\ p(x_j/x_i)\ ...\ (2)$$

In general, an nth order Markov source entropy is a measure of the unpredictability given the knowledge of the previous n-1 references. The entropy is zero when the VPNs generated by the processor is deterministic or can be predicted correctly, and it is maximum when the probabilities are equal for all the VPNs.

2.1 VPN entropy measurement



(a) Mibench patricia



(b) SPEC 2000 perlbmk

Figure 1: Entropy content of virtual page numbers

Figure 1 shows the measured zeroth order Markov source entropy of virtual page numbers for heap, global and stack regions of patricia from MiBench [6] and perlbmk from the SPEC benchmark using Eq.(1). Each point on the x-axis contains 10,000 VPNs, and the corresponding y-axis value is the average entropy for those 10,000 VPNs. The VPNs in the x-axis were plotted by their access order in time.

The entropy of the stack VPNs (the bottom dashed line) for these two benchmark programs is negligible. This means that there is very little or no change in the stack VPNs. The stack region shows low entropy as the memory accesses to this region happen in an orderly fashion. An example

of this behavior is function calls, where the stack frame increases and decreases linearly and the memory references to the stack are all close to each other (e.g. within a memory page) during the execution lifetime of that function. The size of the stack frame depends on the function's activation record.

The middle curves in both Figure 1(a) and Figure 1(b) show the entropy for the global VPNs. The global data size is fixed based on the global variables declared in the program. For example, 16KB of global data corresponds to four pages based on a 4KB page size. The 20-bit global VPN activity in the plot comes from the accesses within this global data size. The accesses to the global variables by a program is more random compared to the stack region accesses, showing that they are a bit more difficult to predict than the stack region. Note that, out of the 20-bit global VPN, only few least significant bits vary during the entire program's execution. The remaining higher order bits do not vary. For the above example, only two least significant bits (for four pages) of the VPN changed and the remaining higher order 18-bits did not change. Based on this fact, the entropy of the higher order global VPN bits is zero, meaning that it is deterministic throughout the program execution. As the sizes of various sections are clearly defined in the section headers as part of the executable file, the global data size is determined during the program compilation. Therefore, the number of pages occupied by the global data can be determined prior to a program's execution.

The top-most curves in both Figure 1(a) and Figure 1(b) show the entropy for the heap region. It exhibit the highest variation among all regions indicating that it is the most difficult of all to predict, as each VPN contains more information. Since accesses to dynamically allocated objects (through function calls such as malloc) cannot be easily tracked, the entropy of these VPNs is much higher than those of the stack and global regions. The global and heap entropy plot is shorter for the Mibench patricia benchmark because it has fewer memory references compared to perlbmk. Also, patricia benchmark has more global references compared to heap, since the global plot is longer than the heap plot.

2.2 Alternative VPN entropy measurement using GZIP

Another way to measure the entropy is to measure the compression ratio of the VPNs of the stack, global and heap memory accesses, using a standard compression program such as gzip program. The gzip utility uses Lempel-Ziv-Welch (LZW) algorithm [17], a universal algorithm for sequential data compression. A universal algorithm is a coding scheme where the coding process is interleaved with a learning process for varying source characteristics. One of the main advantages of this algorithm is that it does not need any knowledge of the probability for the source symbols *a priori*, in our case the VPNs, unlike Huffman encoding [8]. The LZW algorithm relies on the recurrence of strings in its input. Instead of assigning codewords to the symbols in advance, the algorithm assigns codewords to repeating source words, or patterns in the text. This algorithm is still the basis of most lossless modern data compression techniques today. Hence, the size of the compressed trace file compared to the original one gives a very good indication of the entropy contained in the VPNs.

3. MOTIVATION

Figure 2 shows the uncompressed VPN trace file size for each of the stack, global and heap regions in megabytes (MB). Each trace file is an ASCII text file that contains the 20-bit VPNs[1] of memory references that occurred during the dynamic program execution of the respective programs. The empty bars such as global references in blowfish indicate that the file size is very small. Also, Figure 2 shows that many benchmark programs have more stack references followed by global references, and then heap references.

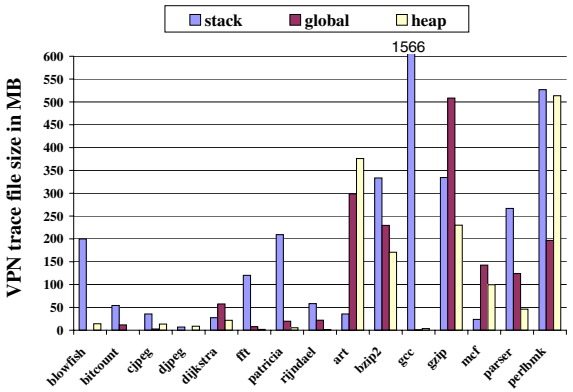


Figure 2: VPN file size before compression

Figure 3 shows the percentage ratio of the compressed VPN trace file size to the original file size for Mibench and SPEC2000 benchmark programs. The standard gzip utility that uses the LZW algorithm [17] is used to compress the trace file. For example, the compressed stack VPN file size for blowfish is only 0.3% of the original VPN file size. The stack VPN has the highest compression (lowest compression ratio) of all the three regions, indicating that the gzip algorithm was able to find repeating sequences much more often in the stack VPNs than in the global and heap VPNs.

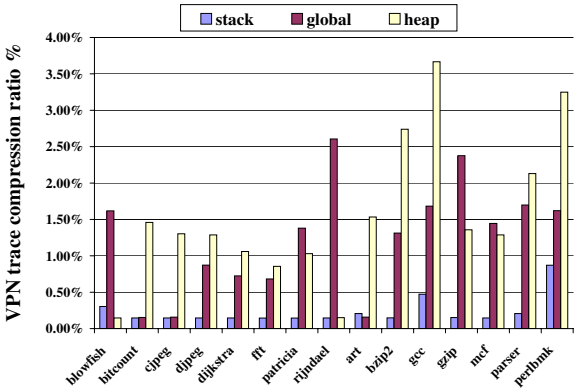


Figure 3: Percentage compression ratio of VPNs

From the above three figures, we deduce that the stack VPNs contain the lowest entropy in spite of the total number of memory references to the stack being much higher than the global and heap memory references for most of the benchmark programs. One outstanding example of this behavior is shown by gcc in Figure 2 and Figure 3.

[1]A 4KB page size is used in the simulations for a 32-bit processor, thus the VPN has 20 bits.

Though the stack accesses dominate the gcc program as shown in Figure 2, its compression ratio is only 0.5% as shown in Figure 3 indicating very low information content.

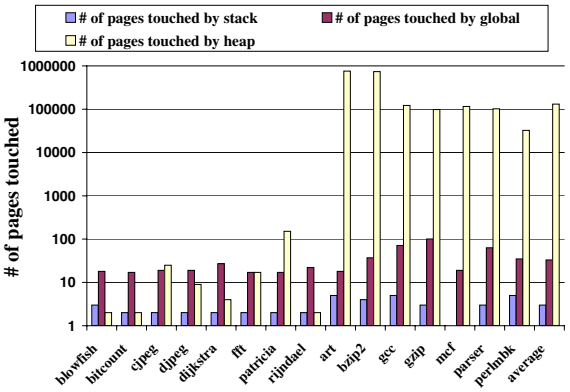


Figure 4: Total number of pages accessed

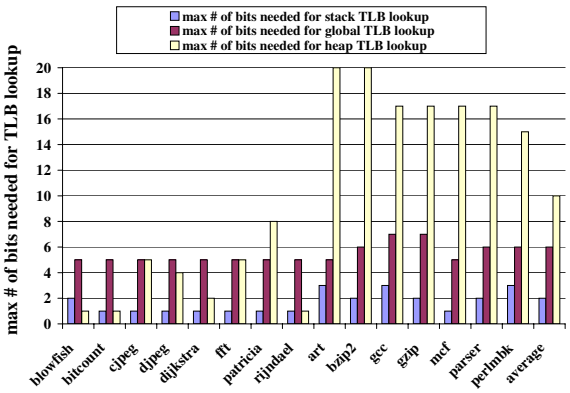


Figure 5: Maximum number of bits required

Figure 4 shows the number of pages covered by the program from its respective base during the program execution. Sometimes the program may dynamically allocate memory (through function calls such as malloc) in the heap region, but may not utilize all the pages. In such cases, though the virtual page is covered by the program, it may not have accessed any memory location in that non-utilized page. This is particularly observable for heap references since the memory locations are not sequentially accessed. For example, a program may allocate ten-thousand quad words (∼156 pages) using the malloc function expecting this amount of memory to be used during run-time. But all of the allocated pages may not be accessed by the program. For example, only some pages in the middle or top may be accessed based on some condition checks. In Figure 4 patricia covered 151 pages and most of the SPEC benchmark programs covered thousands of unique pages during the program execution. This random access behavior of heap references makes it difficult to predict as shown by the high entropy value for heap references in Figure 1.

Figure 5 shows the maximum number of bits needed for the entropy-based address translation based on the number of pages covered. As the number of accessed pages increases, more TLB bits are needed for entropy-based address translation. The small bars for the stack and global regions in Figure 5 suggest that lesser number of bits are sufficient for TLB tag match instead of the complete 20-bit VPN during address translation, as the number of pages covered by them is less than that of the heap region. For example, if the stack region covers eight pages from its base, it requires three bits for the tag match to translate the address correctly. For most of the SPEC benchmark programs, all 20-bits of VPN are required to translate the heap address correctly. An important characteristic of the global region is that its data size (the total number of pages shown) can be determined at compile-time. So, unlike the heap and stack regions, the global data size is fixed throughout the program execution and does not change.

In summary, we observe the following characteristics during program execution.

- The entropy is very low for stack VPNs and therefore can be predicted accurately.
- The global data size is determined at compile-time. The number of bits actually needed for the global TLB tag match is proportional to the number of global data pages.
- As a result, the number of bits needed for the stack and global address translation is much less than the complete VPN (20-bits each in our example).

We exploit these memory reference characteristics and propose a complexity-effective, entropy-based, low-power data TLB.

4. ENTROPY BASED DATA TLB

Based on the above discussion it is clear that a few lower order VPN bits are sufficient instead of the full 20-bit VPN tag match during the TLB lookup for the stack and global memory references. Figure 6 shows a semantic-aware memory (SAM) architecture [11] enhanced with Entropy-based SPeculative (ESP-TLB) and Entropy-based DeTerministic (EDT-TLB) mechanisms. The Entropy-based SAM (ESAM) architecture exploits the characteristics demonstrated in each semantic region by reorganizing the first-level TLB structure into two small structures — stack and global static micro-TLBs, while leaving the second level TLB for all the data accesses.

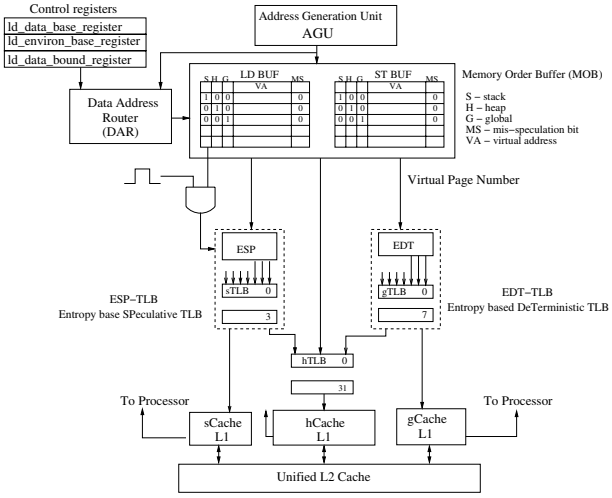


Figure 6: Entropy-based SAM microarchitecture (ESAM)

The virtual address (32-bits) from the address generation unit (AGU) is written to the the memory order buffer

(MOB), which consists of the load and store buffers. In the same cycle the virtual address from the AGU also enters the Data Address Router (DAR). The DAR writes 3 bits (100 for stack, 010 for heap, 001 for global) to the corresponding load/store buffer based on the ranges derived from the control registers (ld_data_base_register, ld_environ_base_register, ld_data_bound_register) initialized by the system loader[2]. The MOB uses these three bits (100-stack, 010-heap, 001-global) during the virtual address dispatch and clock gates the other two semantic TLBs that do not participate in the address translation as shown in Figure 6 to reduce dynamic power consumption. The addition of four bits (3-bits – one each for stack, heap and global, and 1-bit for mis-speculation) to the MOB is not a significant overhead as it already contains many bits such as page fault information, data size of the load/store, and load/store color ids. Since these four bits do not participate in associative search, the power consumed by these bits is very less. Though it is not a significant overhead, we account for the power consumed by these four bits in our power estimation. The ESAM speculatively translates the stack addresses using the ESP-TLB structure, and deterministically translates the global static address using the EDT-TLB structure. The functional and implementation details of these two structures are described in the following sections.

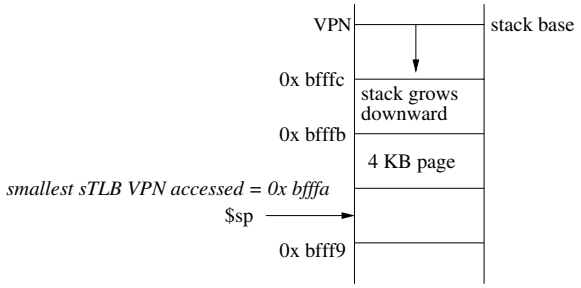
4.1 Entropy-Based SPeculative TLB

As we have discussed previously, the stack addresses contain low entropy and are highly predictable. To utilize this characteristic, stack addresses are speculatively translated. Figure 7 shows the block diagram of the ESP-TLB. ESP-TLB first speculatively precharges the same number of bits that was used in the previous VPN translation for the current VPN translation and evaluates its correctness. If it finds that it is incorrect, the logic block does two things: (1) corrects itself by increasing the number of precharge bits and (2) squashes the incorrect translation by invalidating the cache tag match, and finds the correct translation in the subsequent cycle. The main functionality of this logic block is to enable and precharge the minimum required number of bits for the stack TLB (sTLB) address translation in order to save energy. The *smallest sTLB VPN accessed* register denoted "p" is initialized to 0xffffffff.[3] During the program execution, this register will hold the smallest VPN that was accessed by the stack pointer ($sp).[4] The counter bits, mis-speculation bit (MS-BIT) in the MOB, and the TLB valid bits (V) for each entry are initialized to zero. The MS-BIT is set/reset based on the correctness of the speculatively translated VPN. Figure 7(a) shows the stack growth in the ARM memory model. As visually represented in Figure 7(a), when the stack grows, the VPN decreases.

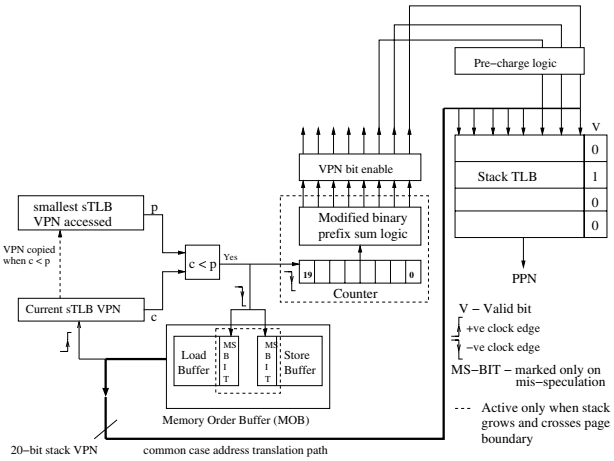
[2]Alternatively, a compiler can annotate loads and stores with semantic region information. The MOB can use this information during the virtual address dispatch to the semantic TLBs and eliminate the DAR logic by using compiler support.

[3]In our simulation model based on ARM, the stack region begins from higher memory address and grows downward.

[4]Following software convention, the smallest VPN should represent the memory page pointed by the current $sp. Any memory access beyond the current $sp (i.e. addresses lower than $sp) is considered a violation of the software convention.



(a) Stack growth in ARM memory model



(b) Entropy based SPeculative stack TLB

Figure 7: Entropy based speculative stack TLB

The memory order buffer (MOB) dispatches the virtual address that needs to be translated to semantic TLBs. But only one TLB will be enabled based on the three bits stored as part of the MOB. If the stack TLB (sTLB) is enabled, the sTLB is precharged only for the bits that are high at the output of the *VPN bit enable* register, a bit mask. The VPN bits sent from the MOB to the sTLB represent the common case for stack address translation. This common case path is shown as a bold line from the MOB to the sTLB in Figure 7(b).

At the same time the address reaches the sTLB, the *current sTLB VPN* denoted "c" that contains the current VPN is compared with the *smallest sTLB VPN accessed* during the high phase of a clock cycle. If the current stack reference did not cross the page boundary, the comparator output will be FALSE. If the comparator output is FALSE ($c \geq p$), the speculative translation was indeed correct and no further correction is needed. The counter value stays the same and is not decremented. Note that the comparison is not on the critical path of the stack address translation.

When the stack region grows (downward) as shown in Figure 7(a) and crosses the page boundary, the comparator output will be TRUE ($c < p$) indicating that the speculatively translated address was incorrect, and a mis-speculation

recovery is required as follows: (1) set the mis-speculation bit (MS-BIT) in the MOB for this virtual address entry, (2) update the *smallest sTLB VPN accessed* register with the *current sTLB VPN* value, (3) increment the counter, and (4) invalidate the output of the cache tag match hit/miss signal by qualifying it with the comparator output signal; all four events happen on the negative edge of the same clock cycle since there are no dependencies among them.

The counter keeps track of the number of pages the stack region has covered until now, starting from the stack base. We use a *modified binary prefix sum logic* to determine the minimum number of the sTLB precharge bits to be enabled based on the counter value. The *VPN bit enable* register stores the output of the *modified binary prefix sum logic* and is ready for correct address translation in the subsequent cycle in case of a mis-speculation. The comparator (that writes the MS-BIT) squashes the incorrectly translated address by invalidating the tag match for the cache data. The instruction scheduler cancels and reschedules the instructions dependent on this cache data. The number of mis-speculations for the sTLB is very low and is analyzed in detail in Section 5. The corresponding memory reference is re-issued by the MOB, and the MS-BIT is reset after the re-issue. The re-issued memory reference will now succeed through the common case path, as both the *current sTLB VPN* and *smallest sTLB VPN accessed* contain the same value. The instructions dependent on the memory reference get scheduled to execute on their respective execution units. The power consumption of all the components in Figure 7 and the performance issues on the mis-speculation path are further discussed in Section 5.

The MOB is extended by the addition of the MS-BIT to each entry as shown in Figure 7. During normal operation, the MOB captures events such as TLB page-faults and memory-access-violations, and reschedules the affected memory operations after a TLB page walk. This existing MOB mechanism can be extended to set and reset the MS-BIT, and no further logic is required.

The *modified binary prefix sum logic* is a combinational circuit that determines the minimum number of the sTLB precharge bits to be enabled based on the counter value. For example, if the program has covered twelve pages from the stack base, the counter value will be 12 (binary 1100). The 20-bit VPNs of these twelve pages can be uniquely identified using the four lower order bits [3:0] as the stack frame grows sequentially. When a new VPN looks up the sTLB for an address translation, these four bits are enough to resolve the twelve pages unambiguously for the VPN tag match. Therefore, the sTLB needs to be precharged with only four least significant bits enabled, which is [000....1111]. The circuit that accepts the binary value from the 20-bit counter as input (say, [000...01100]) and provides a 20-bit binary [000...1111] as output is implemented using a modified binary prefix sum logic. In general, given an n-bit binary input sequence,

$b_{n-1}, b_{n-2}, ...b_2, b_1, b_0$

the binary prefix sum logic provides a n-bit binary output sequence,

$p_{n-1}, p_{n-2}, ...p_2, p_1, p_0$

where $p_i = (b_{n-1} + b_{n-2} + ... + b_{i+1} + b_i) mod 2$ is called the i^{th} binary prefix sum [12].

We need to take care of an additional case when the counter value is a power of 2. For example, when the counter value is 4 (binary 100) indicating four stack pages are now covered by the program, the output should be binary 011, as two bits are enough to uniquely identify these four VPNs. But the output of the binary prefix sum logic will be 111. This precharges one extra bit that is not needed for address resolution, consuming more energy during every stack TLB lookup. Therefore, whenever the counter value is a power of 2, the MSB of the binary prefix sum logic output must be changed to zero instead of one. We modified the truth table to accommodate this case, and implemented the logic using CMOS gates. This *modified prefix sum logic* enables pre-charging and comparing only the few least significant bits of the stack TLB, thus saving energy by avoiding the complete 20-bit VPN precharge. The power consumption and the delay for this logic is detailed in Section 5.

The following things are to be noted during the stack address translation mechanism:

1. The stack address is translated speculatively through the common-case (highlighted) path shown in Figure 7.

2. Since the stack address translation is speculative, the comparator, the modified binary prefix sum logic, and MS-BIT update in the MOB do not affect the critical path of the TLB lookup.

3. The modified binary prefix sum logic becomes active and consumes power only during mis-speculation.

4.2 Entropy-Based DeTerministic TLB

The compiler, assembler, and linker tool chain creates a widely used ELF or COFF file format that is loadable, relocatable, and executable. As part of the process, the global variables in the program become part of the global static data in the executable. The size of the various sections such as .text, .init, .data, .rodata, and .bss sections are clearly defined as part of the executable file format. Therefore, the number of pages occupied by the global static data can be determined after the program compilation from the difference between the loader variables *ld_data_start* (the address where the global static data starts) and *ld_data_bound* (the address where the global static data ends). Once the global data size is known, the maximum number of pages occupied by the global data is obtained by dividing the global data size by 4096 (the page size). The minimum number of bits required to address these global pages, and the bit sequence sent to the precharge logic to enable the global TLB is calculated by the loader as shown below. By going through a few lines of code once, the loader can deterministically find the bits that need to be precharged and stores it in the *Deterministic fixed bits* register.

global data size = ld_data_bound - ld_data_base;
number of pages = global data size / 4096;
number of bits needed = log_2(number of pages);
Deterministic fixed bits = mod_binary_prefix_sum(number of bits needed);

The last line is a software equivalent of finding the modified binary prefix sum bit sequence that was described in Section 4.1.

The global entropy activity comes from the memory references between the defined boundaries of the global data size. As the activity is confined to these few pages occupied by the global data, only a few lower order bits change between memory references and most of the higher order bits of VPN remain unchanged. This provides an opportunity to enable and precharge only the few lower-order bits for global TLB (gTLB) lookup in order to save energy.

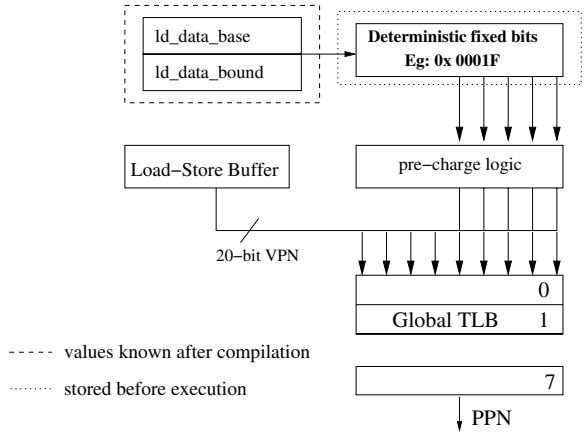


Figure 8: Entropy based deterministic global TLB

32-bit Processor Parameters	Values
Execution Engine	4-wide out-of-order
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	DM, 32KB, 32B line
L2 Cache	4-way 512KB, 32B line
Number of TLB ports used	1
20-bit comparator power	300μW
Modified prefix sum logic delay	160 ps
Modified prefix sum logic power	668 μW
Dynamic power for counter	956 μW

Table 1: Processor configuration

In Figure 8, the *Deterministic fixed bits* is derived from the loader variables and stored in the register. The gTLB is precharged and compared only for the bits that are high in this register to save energy. Since the precharged bits are deterministically found before program execution, there is no performance penalty involved.

5. 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 modified Simplescalar to model the ESP-TLB and EDT-TLB. We use 100nm technology in our simulations. We simulated the Mibench benchmark programs to completion. For the SPEC2000 benchmark suite, we first fast forwarded by one billion instructions and simulated the next 300 million instructions. Table 1 describes our machine configuration and power data.

To evaluate the energy savings using the ESP-TLB and EDT-TLB mechanisms, the power consumption of the 20-bit comparator, modified binary prefix sum logic, and the counter are taken into account in the Wattch simulation. We simulated these logic components using HSPICE. For our HSPICE simulations, we use BSIM transistor models [3]. The energy consumed by the comparators, and counter are added every cycle, and the modified binary prefix sum logic power on mis-speculation. We charge two extra cycles on each stack address translation mis-speculation. In the first cycle, during the positive edge, the VPNs are compared; during the negative edge, if the speculative stack address translation is incorrect the counter is incremented and the modified binary prefix sum logic changes its output. In the second cycle, the MOB reschedules the virtual address for sTLB lookup. We add an extra 10% [15] of the dynamic power to account for the leakage power when there is no TLB access activity. Sery et al. [15] report 15% of dynamic power as leakage power in the 65nm technology, and 10% in the 90nm technology.

Figure 9 presents the energy savings and overall performance after applying the ESP-TLB and EDT-TLB mechanisms. The baseline in this figure is a conventional 32-entry d-TLB. The total energy savings is 47% compared to the baseline d-TLB with an overall performance penalty of less than 1%. The penalty from the stack address translation mis-speculation is negligible as the number of mis-speculations are very few. In some cases there is a performance improvement as the ESAM architecture reduces the number of expensive TLB misses.

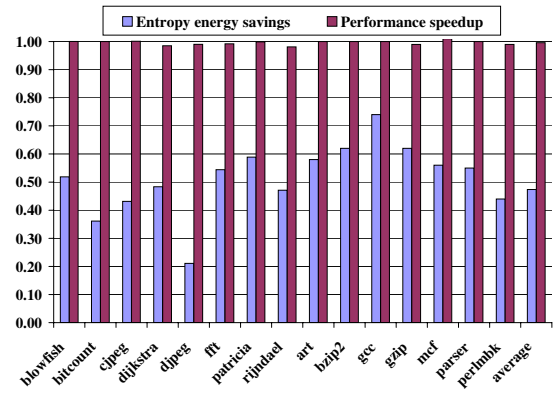


Figure 9: Energy savings and performance using ESP-TLB and EDT-TLB

Figure 10 shows the effectiveness of the ESP-TLB. The maximum number of bits precharged for the stack address translation is less than or equal to three bits, as the stack region has not crossed more than eight page boundaries. The maximum number of mis-speculations is three; when the counter increments from 0 to 1, 1 to 2, and finally from 2 to 3. The first bar in this figure shows the number of accesses when the sTLB lookup needed only one bit to be precharged, and the second and third bar when the sTLB lookup needed only two and three bits to be precharged, respectively. The experimental results confirm the graphs in Section 3; the stack addresses have very low entropy and can be predicted with high accuracy to save energy.

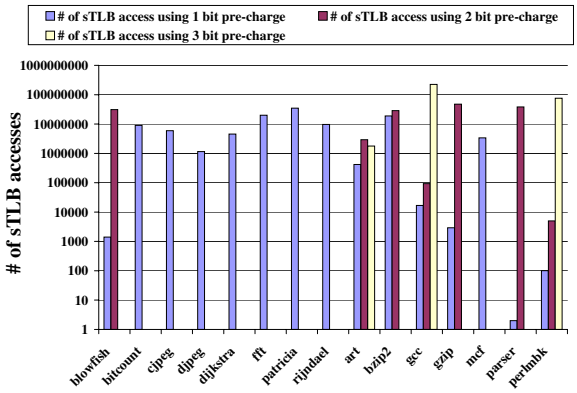


Figure 10: Effectiveness of the ESP-TLB

6. RELATED WORK

Hammerstrom et al. [7] proposed a method to estimate the information content of memory references and the address generation overhead. They proposed a few optimization techniques to reduce the address calculation overhead based on the measured entropy. Becker et al. [1] proposed a scheme to dynamically encode a typical 32-bit address into four to seven bits using Huffman coding. Their experiments show that 83% of address bits in the traces contain redundant information.

Kandemir et al. [10] proposed a compiler-based data layout restructuring combined with smaller translation registers, to reduce the d-TLB energy. Petrov et al. [14] proposed a software-controlled reduced virtual page number mechanism to reduce the TLB energy. Their method consists of two phases, an off-line and an on-line phase. In the off-line phase, a special algorithm code is inserted prior to entering and after exiting the hot-spot with reduced tag information. Additionally, the compiler also attaches special tables with information about the hot-spots. During the on-line phase, the OS provides information regarding the libraries. Our technique requires some hardware changes, but does not require any profile feedback support.

Juan et al. [9] proposed a circuit design to modify the CAM cell by adding another transistor in the discharge path. With the modified cell, the control line can be used to precharge the match line without resetting the bit lines. Now, the bit lines toggle due to changes in the virtual addresses of consecutive accesses. As the unified TLB is accessed by all the memory references from different semantic regions, the entropy is higher leading to their measured change of four bits per access, unlike our semantic-aware technique where the addresses are segregated based on different memory regions.

7. CONCLUSION

In this paper, we proposed two architectural techniques to reduce the data TLB energy. By exploiting the low entropy of stack and global data, energy savings are achieved via (1) a speculative but highly accurate stack address translation, and (2) a deterministic global static address translation. In both schemes, we enable and precharge only the minimal required number of least significant address bits to reduce the data TLB energy. We show that the TLB energy consumption is reduced by 47% with less than 1% overall performance overhead. As shown in our results, the performance degradation due to mis-speculation is negligible for the stack TLB, and there is no performance penalty for the global TLB translation as it is deterministic. Our techniques incur very little hardware overhead. As the TLB power continues to rise and as we migrate from 32-bit to 64-bit architectures supporting large page sizes, our techniques will become more significant in reducing the data TLB energy.

8. ACKNOWLEDGMENT

This research was supported in part by National Science Foundation Grants CCF-0326396 and CNS-0325536.

9. REFERENCES

[1] Jeffrey C. Becker, Arvin Park, and Matthew Farrens. An Analysis of the Information Content of Address Reference Streams. In *Proceedings of IEEE/ACM 24th International Symposium on Microarchitecture*, 1991.

[2] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: a Framework for Architectural-level Power Analysis and Optimizations. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, 2000.

[3] Y. Cao, T. Sato, D. Sylvester, M. Orshansky, and C. Hu. New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Design. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, 2000.

[4] M. Ekman, F. Dahlgren, and P. Stenstrom. TLB and snoop energy-reduction using virtual caches in low-power chip-multiprocessors. In *Proceedings of the 2002 International Symposium on Low Power Electronics and Design (ISLPED-02)*, 2002.

[5] Dongrui Fan, Zhimin Tang, Hailin Huang, and Guang R. Gao. An energy efficient tlb design methodology. In *Proceedings of the 2005 International Symposium on Low Power Electronics and Design (ISLPED-05)*, pages 351–356, 2005.

[6] Mathew R. Guthaus, Jeff S. Ringenberg, D. Ernst, Todd M. Austin, Trevor Mudge, and Richard B. Brown. MiBench: A Free, Commercially Representative Embedded Benchmark Suite. In *the 4th Workshop on Workload Characterization*, 2001.

[7] D. Hammerstrom and E. S. Davidson. Information Content of CPU Memory Referencing Behavior. In *Proceedings of the 4th Annual International Symposium on Computer Architecture*, 1977.

[8] D. A. Huffman. A method for the construction of minimum redundancy codes. In *Proceedings of the IRE*, 1952.

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

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

[11] Hsien-Hsin S. Lee and Chinnakrishnan S. Ballapuram. Energy Efficient D-TLB and Data Cache using Semantic-aware Multilateral Partitioning. In *Proceedings of the 2003 International Symposium on Low Power Electronics and Design (ISLPED-03)*, 2003.

[12] R. Lin, K. Nakano, S. Olariu, and A. Zomaya. An Efficient Parallel Prefix Sums Architecture with Domino Logic. In *IEEE Transactions on Parallel and Distributed Systems*, 2003.

[13] Trevor Mudge. Power: A First Class Design Constraint. *IEEE Computer*, 34(4):52–57, 2001.

[14] P. Petrov and A. Orailoglu. Virtual Page Tag Reduction for Low-power TLBs. In *Proceedings of the 2003 IEEE International Conference on Computer Design*, 2003.

[15] G. Sery, S. Borkar, and V. De. Life is CMOS: Why Chase Life After? In *Proceedings of the 39th Conference on Design Automation*, 2002.

[16] Claude E. Shannon. Prediction and Entropy of Printed English. *Bell Systems Technical Journal*, 30:50–64, 1951.

[17] Jacob Ziv and Abraham Lempel. A Universal Algorithm for Sequential Data Compression. In *IEEE Transactions on Information Theory*, volume IT-23, No. 3, 1977.