Energy-Efficient Address Translation for Virtual Memory Support in Low-Power and Real-Time Embedded Processors

Xiangrong Zhou
University of Maryland
College Park, USA
xrzhou@glue.umd.edu

Peter Petrov
University of Maryland
College Park, USA
ppetrov@ece.umd.edu

ABSTRACT

In this paper we present an application-driven address translation scheme for low-power and real-time embedded processors with virtual memory support. The power inefficiency and nondeterministic execution times of address-translation mechanisms have been major barriers in adopting and utilizing the benefits of virtual memory in embedded processors with low-power and real-time constraints. To address this problem, we propose a novel, Customizable Translation Table (CTT) *organization, where application knowledge regarding the virtual memory footprint is used in order to eliminate conflicts in the hardware translation buffer and, thus, achieve tag-free address translation lookups. The set of virtual pages is partitioned into groups, such that for each group only a few of the least significant bits are used as an index to obtain the physical page number. We outline an efficient compile-time algorithm for identifying these groups and allocate their translation entries optimally into the CTT. The proposed methodology relies on the combined efforts of compiler, operating system, and hardware architecture to achieve a significant power reduction. The experiments that we have performed on a set of embedded applications show power reductions in the range of 55% to 80% compared to a general-purpose Translation Lookaside Buffer (TLB).*

Categories and Subject Descriptors

B.3 [**Hardware**]: Memory structures; C.1 [**Computer Systems Organization**]: Processor Architectures; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems

General Terms

Algorithms, Design, Experimentation, Performance

1. INTRODUCTION

Since its conception, virtual memory [1] has been shown to be an elegant and efficient solution for abstracting away from the application the complexity of *memory allocation*, and *code/data relocation and sharing*, while efficiently providing *memory protection* between user applications and system software; all these being completely transparent to the application and controlled by the operating system (OS). Such features would tremendously benefit many embedded systems, if virtual memory is to be supported for them. General-purpose virtual memory, however, requires unacceptably high amounts of power and introduces execution time nondeterminism, thus rendering itself unusable for a large number of embedded systems with stringent power constraints and real-time requirements.

The program accesses a virtual address space separated into pages, typically 4K in size, and at each such access a translation is needed to map the virtual address into a physical one. The translation is performed at a page granularity in order to control the complexity of the translating mechanism. The *Translation Lookaside Buffer (TLB)* is a hardware cache responsible for capturing the most recently used *Page Table Entries (PTE)* for dynamically translating virtual addresses generated by the processor to physical addresses. The mapping between virtual and physical addresses is typically maintained by the OS and established by the OS loader, dynamic linker and memory manager. TLB misses typically result in trapping into the OS where the missed PTE is retrieved from *page tables* maintained by the kernel. Consequently, the TLB is usually implemented as a highly associative cache structure, which consumes a significant amount of power. It has been shown that the TLB power consumption constitutes 20-25% of the total cache power consumption [2].

The presence of a data cache does not eliminate the need for address translation since physical tags need to be used when accessing the cache. If, instead, virtual addresses are directly used to access the cache, this introduces the *cache synonym* problem [3], a situation where a shared data is relocated to distinct cache locations for each process, thus introducing consistency hazards and cache capacity underutilization. In order to avoid increasing cache access time and to facilitate the solution of the synonym problem, caches are typically indexed with the virtual address and tagged with the physical address. In such virtually-indexed and physically-tagged caches, the *cache index* is obtained from the virtual address while the tag is obtained from the *Physical Page Number (PPN)*, which is translated from the *Virtual page Number (VPN)*. Consequently, an address translation is needed each time a memory location is referred to by the processor.

The need for energy costly address translation on each memory reference, as well as the introduced execution time uncertainty caused by the cache-like TLB lookups, have been the two major factors preventing the introduction of virtual memory and its concomitant benefits to low-power and real-time processors.

The novel address translation approach that we outline in this paper, attacks these two problems. Through the utilization of application-specific information regarding the virtual memory footprint of the application, the set of VPNs accessed by the program is *partitioned into groups* so that by using only a small number of least significant VPN bits as an index into a special translation buffer, *a conflict-*

CODES+ISSS'05, Sept. 19–21, 2005, Jersey City, New Jersey, USA.

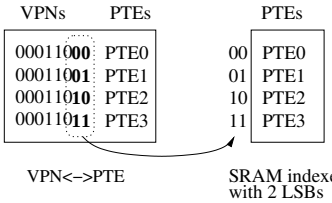

Figure 1: Set of consecutive VPNs

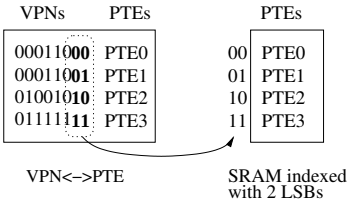

Figure 2: Indexing SRAM table with LSBs

free, and thus tag-less lookup can be achieved. This not only results in low-power address translation, but also to highly predictable execution times as the conflict free CTT access guarantees a translation with no OS intervention or additional page table lookups.

2. RELATED WORK

A low-power TLB organization for chip-multiprocessors has been proposed in [2]. By incorporating a special *Page Sharing Table* to the TLB and using virtual caches, the authors reduce the amount of TLB activities, at the same time eliminating a large number of snoop accesses. A similar work in the direction of employing virtual caches with specialized TLB support is presented in [4]. The authors propose replacing the TLB with the more scalable and power efficient *Synonym Lookaside Buffer*, as it stores only the current synonym instances. In [5], the authors evaluate the power consumption of a number of TLB organizations and propose a new cell implementation for low-power set-associative TLBs. A low-power and high-performance TLB architecture has been proposed in [6]. A TLB organization is proposed that dynamically supports up to two pages per entry with a banked fully-associative structure. In [7], the TLB accesses are redirected to a register file which holds a few recent TLB entries.

All the aforementioned techniques target the architectural organization of general-purpose virtual memory support. In contrast, our work takes a step further into the domain of embedded processors where application knowledge can be exploited and the architecture dynamically customized. A noteworthy difference of our approach is that it *does not trade-off performance for power* and is also orthogonal to many of the proposed approaches. Since the proposed technique eliminates accesses to the TLB for most of the memory accesses by replacing them with deterministic and tag-less indexing, the number of TLB misses is drastically reduced. Consequently, our approach slightly *improves performance* and significantly *improves predictability of execution times* - a characteristic of great importance to any real-time embedded system.

3. FUNCTIONAL OVERVIEW

General-purpose processor architectures are typically designed with the assumption that a large variety of programs will be executed and that there is no program information made available to the microarchitecture prior of its execution. It is also assumed that the program to be executed could come in a binary only form. Embedded processors and systems, however, have the distinctive advantage of complete application knowledge, as the embedded software is usually developed concurrently with the hardware design or is available in a source code format. The low-power address translation methodology that we outline in this paper, is fundamentally a technique that with the help of the compiler and the operating systems exploits dynamically such an application-specific knowledge.

The set of virtual pages accessed by the application is available when compiling and linking the program. The general-purpose TLB architecture features tag arrays and comparators, which purpose is to ensure that conflicts within the data arrays are identified and handled properly. This organization assumes no knowledge regarding the set of VPNs accessed by the application. If, however, information regarding the virtual pages is used in the address translation process in such a way so that all conflicts are eliminated, then no tag arrays and operations are needed while obtaining the PPN with direct indexing.

If the possibility of TLB conflicts can be avoided through a judicious analysis of the VPN set, then a direct indexing for finding the PPN, free of any tag operations, can be achieved. Figure 1 shows an example where the compiler/linker has identified that only four consecutive data virtual pages are accessed throughout execution. It can easily be seen that among all the VPN bits, the two *Least Significant Bits (LSBs)* are enough to differentiate the four VPNs. And if we use these two LSBs as an index into a translation table, all the VPNs will be mapped into the table in a conflict-free manner. A more complex example is illustrated in Figure 2, in which four distinctive virtual pages are accessed. Although they are not consecutive, their two LSBs are still enough to uniquely distinguish them. Consequently, only these two bits of the VPN can be used to form an index into a 4-entry memory block which holds the physical page addresses of these virtual pages, as shown in the figure. By avoiding the VPN tag lookup and using only these two bits as an index there would be no performance implications, while the overall reduction on the TLB power consumption is to be quite significant. All the power associated to the VPN tag arrays, the corresponding sense amplifiers, and the comparator cells is eliminated.

The fundamental idea of the proposed approach is to identify such a conflict-free indexing scheme in order to avoid the power consuming VPN tag operations. Given a set of *n* data VPNs, we need to find an answer to the question what is the minimal number *m* of VPN LSBs that could differentiate these VPNs and thus be used as an index. Even more importantly, how efficiently will the introduced translation table be utilized after storing the translation entries of each VPN in such a 2^m sized memory. The above examples show an ideal situation. However, in some cases the utilization of the memory could be quite low if no additional measures are taken. Such an example is depicted in Figure 3. For these eight VPNs, six LSBs are needed to differentiate them and use them as an index. Therefore, this set of VPNs occupies a memory array with 2^6 entries, while only eight of them will be actually used. This extreme case shows that a low memory utilization is possible with a large waste of memory and its associated power, if the set of VPNs accessed by the application is targeted as a whole. However, it can be seen that VPNs 0,1,4,5 can be differentiated by two bits, while the VPNs 2,3,6,7 can be differentiated by two bits as well. Consequently, if the initial set of eight VPNs is divided into two partitions, $0, 1, 4, 5$ and $2, 3, 6, 7$, then the two LSBs can still be used to form an index into two non-overlapping segments within a translation table as long as information regarding which partition is being used is known prior to access the table. Additionally, the two partitions need to be allocated into two different four-entry sections of the translation table, so that VPNs across different partitions are guaranteed not to overlap.

The approach that we propose in this paper involves the iden-

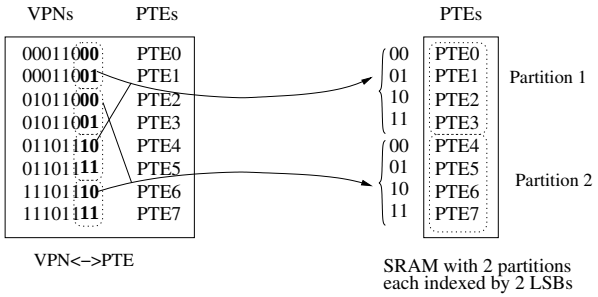

Figure 3: Two VPN partitions, each indexed by 2 LSBs

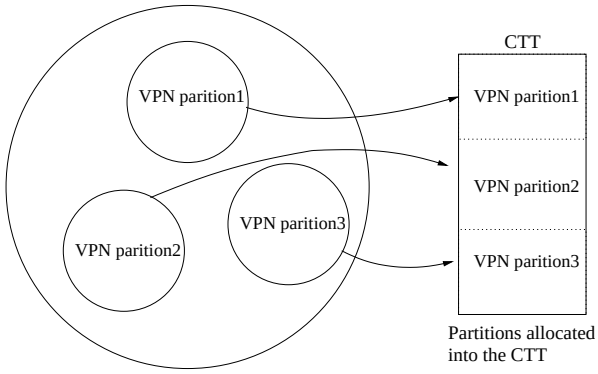

Figure 4: Mapping VPN partitions to CTT segments

tification of partitions of VPNs which result in optimal indexing scheme maximizing the utilization inside each partition while reducing the overall number of partitions. Minimizing the number of partitions, while maintaining high utilization of the translation table is important to control the cost of hardware needed to identify partitions and to compute their translation table index.

As partitions are identified on a per load/store instruction basis, it is very important to ascertain that each load/store instruction that we target with this approach accesses VPNs that belong to a single partition. This additional restriction guarantees that there is no ambiguity when accessing the translation table..

After identifying the partitions, each partition is mapped to its own part of the introduced *Customizable Translation Table (CTT)*, as illustrated in Figure 4. The CTT is implemented as a small SRAM array containing the translation entries for all VPNs. As each partition of VPNs is mapped to a distinct part of the CTT, a special indexing logic is need to form the final index. As we align the partitions inside the CTT on address boundaries proportional to their size, a very simple logic is needed to compute the CTT index; the CTT segment offset needs to be simply concatenated to the few VPN LSBs selected as a partition index.

Application programs typically spend most of their execution times in tight loops or function calls, which are generally referred to as "hot-spots". Inside the hot-spot, the program would typically access only a few arrays or codec stacks which occupy a very few VPN pages. By targeting the application hot-spots, practically all the benefits from the proposed technique can be achieved with only a low-cost hardware support needed to capture the information regarding the VPN partitions. Consequently, the proposed scheme is applied only on the application hot-spots, while for the rest of the infrequently accessed VPNs, a default D-TLB is still used for address translation. Upon entering or exiting a hot-spot, the compiler inserts a special setup code which stores certain information into special registers and tables implemented as a part of the specialized hardware support and, thus, informs the hardware that a hot-spot has just been entered.

As a first step in the proposed approach, the application is profiled and its hot-spots identified. When compiling the program, information regarding all virtual pages accessed by the application within each "hot-spot" is extracted. Furthermore, the frequency of access can also be computed at that step.

For a given set of VPNs, there is a minimal number *m* of LSBs that differentiate all the VPNs. We refer to such a set of VPNs as an *m-bit* partition, while *m* is referred to as a *dimension* of the partition. As we saw in the previous section, *m* depends on the total number *n* of VPNs in the set, as well as on their particular values. Evidently, $\lceil \log_2 n \rceil$ is a lower bound for *m*, as this is the minimal number of bits needed to distinguish a set of *n* elements. If the partition dimension *m* is equal to $\lceil \log_2 n \rceil$, where *n* is the number of VPNs, such partition is referred to as *m-bit complete* or just *complete* partition. Consequently, a complete partition is a partition with minimal number of index bits and high utilization of the index space. Such a partition requires a minimal segment of translation table to map its VPNs.

4. ALGORITHM OVERVIEW

In order to achieve efficient hardware support, we need to identify the minimal number of VPN partitions with the highest utilization of translation table resources. The proposed approach partitions the set of VPNs into a minimal number of complete partitions, thus achieving very high utilization of the translation table resources. Consequently, an algorithm is required to find the minimal number of complete VPN partitions for a given set of VPNs. As discussed later in this paper, the case of dynamic data allocation can be efficiently dealt with, as such events typically happen outside the application hot-spots and a special partition can be reserved for the involved VPNs.

An apparent first step in the proposed algorithm is to separate the groups of consecutive VPNs. Such groups of VPNs correspond to application arrays and buffers; it can be easily observed that in many applications they constitute an overwhelmingly large part of the application hot-spots. Furthermore, such groups of VPNs have the very desirable property to constitute *complete* VPN partitions. This can be easily observed from the fact that a set of *n* consecutive numbers can always be differentiated through the $\lceil \log_2 n \rceil$ LSBs.

Separating the set of VPNs into partitions of consecutive VPNs is only a first step in achieving the desired final result. Very often, an *m-bit complete* partition does not utilize the index range and can, thus, be merged with some smaller VPN partition without increasing the number of least significant bits *m* to form an index. Figure 5 illustrates such a case. In this example, the algorithm starts with three partitions. All of them are complete, as they contain consecutive VPNs; the largest partition is a *4-bit* partition, the next one is a *3-bit*, while the smallest one is a 1-bit partition. It can be easily seen that the 4-bit partition can "absorb" the 3-bit partition, without increasing its initial dimension of 4. The resulting partition consists of 14 VPNs and is still a complete partition of dimension 4. In a subsequent step, the 1-bit partition can be merged with the new 4-bit partition, resulting to a single complete partition of dimension 4, consisting of 16 VPNs.

This step in the algorithm fundamentally tries to "pack" the initial set of VPN partitions. At this step, the proposed algorithm needs to find the optimal strategy for merging the initial VPN partitions in such a way so that minimal number of complete partitions remain at the end. An important requirement that needs to be imposed here is that at no step of the merging process should the dimension of a partition be increased. That is, when merging two partitions, the resulting partition must have the dimension of

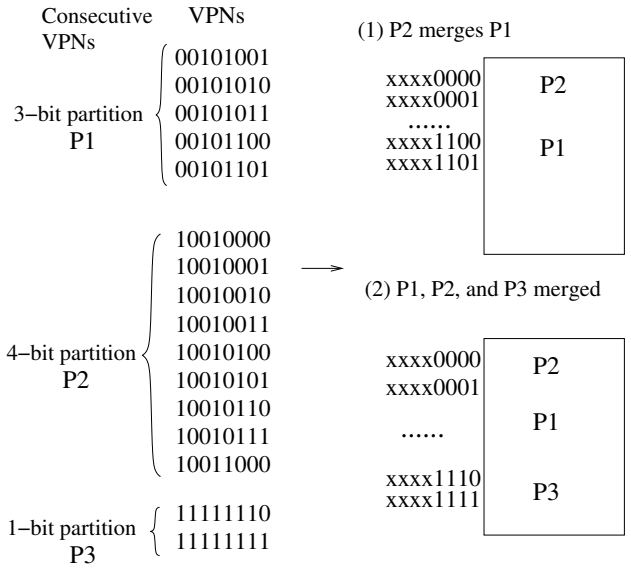

Figure 5: VPN partition merging

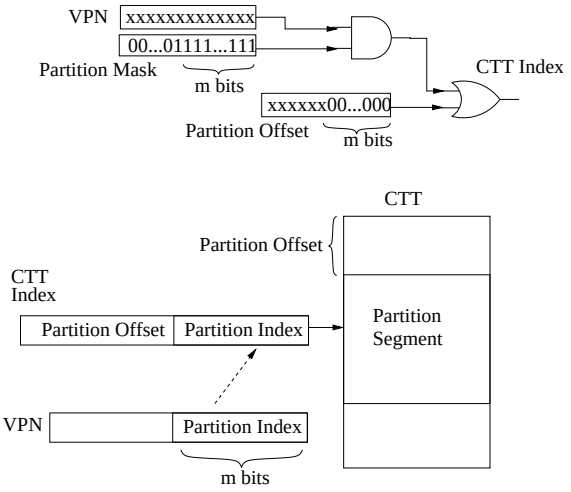

Figure 6: Access PTE in the SRAM

the larger partition from the initial pair of partitions. In order to merge two partitions, the LSBs of the smaller partition when extended to the dimension of the larger one must not conflict with the LSBs of the larger partition; such conflicts can exist between some of the partitions which would prevent their merging. Therefore, the algorithm to identify the optimal scheme of merging is a multidimensional combinatorial optimization problem, very similar to the well known Bin-Packing problem. We have developed an algorithm which in its essence is a heuristic, somewhat similar to the First-Fit Decreasing [8] heuristic used for Bin-Packing.

Fundamentally, the aforementioned merging step of the algorithm tries to utilize the empty index space that is left in some partitions. As can be seen from the example in Figure 5, partition P2 has 9 VPNs. Since it is a 4-bit partition, the index space of the 4 LSBs is 16, which results in utilization of 9/16 of the available index space and, thus, translation table resources. After the merging steps, though, it can be seen that the utilization of the final partition is 100% as it is a 4-bit partition with 16 VPNs. This final VPN partition can be mapped into a translation table with 16 entries indexed by the 4 LSBs of the VPNs. Consequently, the driving force behind the proposed algorithm is the goal of using the empty space in the initial partitions by fitting there smaller partitions. Consequently, the merging phase of the algorithm starts with the partition having the largest dimension and then tries to merge as many smaller partitions as possible; the step is repeated until no more merging is possible. If there are more than one partition of the same dimension, we use their empty index space as a tie-breaker by picking the one with larger empty space. This step is repeated until all the partitions not merged yet are tried.

At the end of this algorithm, a small set of VPN partitions is produced with very high utilization of the index space. The high utilization implies that very few entries in the CTT segments allocated for these partitions will remain unused. An important requirement in the above algorithm is that in the process of merging, the dimension of the large partition is left unchanged. This might leave a few very small partitions each having several VPNs. However, SRAM arrays are typically implemented as a set of smaller banks. Therefore, it will be beneficial to combine the small partitions into a larger partition with a size equal to the memory bank size. This step is performed by starting from the smallest partitions (size 1 or 2) and trying to merge them with the next larger one by possibly increasing the partition dimension while keeping it under the memory bank size.

As we mentioned in the previous section, it is very important that each load/store instruction generates virtual addresses that belong to the same VPN partition. It is typical that a load/store instruction generates addresses within the same VPN or across adjacent VPNs. Because of the nature of our partitioning algorithm, all such load/store instructions will generate addresses within the same VPN partition. This property ensures that there is a one-to-one mapping between a memory reference instruction and a VPN partition. Consequently, it is at the level of load/store instructions where we identify the VPN segment which is to be accessed. As shown in Section 7, the proposed algorithm results in a very few VPN partitions within each application hot-spot (fewer than 8), thus, enabling various hardware schemes for mapping load/store instructions to VPN partitions. One such possibility is to use 2 or 3 extra bits from the instruction encoding (in a way identical to the scheme used in [7]) to mark which VPN partition should be accessed for the particular load/store instruction. The memory reference instructions outside hot-spots or the very few load/store instruction which access virtual addresses from different VPN partitions are handled by a default D-TLB.

5. HARDWARE SUPPORT

The proposed approach requires a specialized hardware support which purpose is to map load/store instructions to their corresponding VPN partitions, to compute the appropriate CTT index, and to access the CTT in order to obtain the physical page number. The VPN partitions are identified by using two or three extra bits from the instruction encoding of the memory reference instructions. An alternative approach is to use a small table accessed by the load/store PC in cases where the total number of such instructions inside the application hot-spot is very limited. The partition identifying bits are used to access one of the four or eight registers, which in turn contain information regarding the VPN partition (the partition dimension *m* and the partition offset within the CTT). A default value is used for the memory references that need to be translated through the default D-TLB.

Each VPN partition must be mapped to an exclusive segment within the CTT. As multiple VPN partitions are mapped, an efficient index computation logic is needed. If VPN segments are allocated in arbitrary positions within the CTT, accessing such a segment would require the hardware to compute the CTT index by

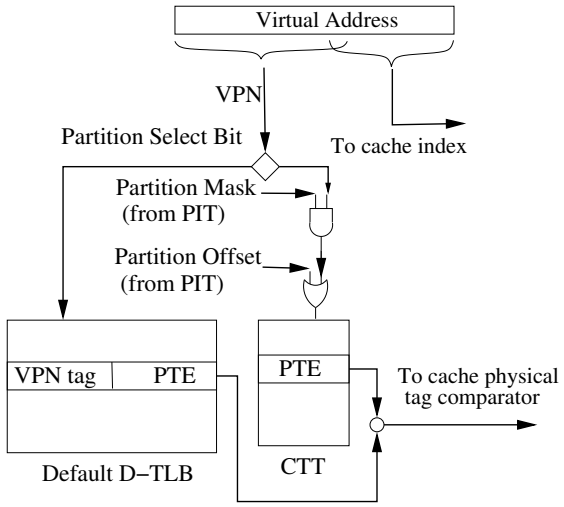

Figure 7: Hardware architecture

adding the segment offset to the LSB index of that partition. While such an addressing scheme is not difficult to implement, it would introduce a significant delay on the critical path of address translation, possibly increasing the L1 cache access time. We propose, instead, an alternative hardware scheme, which requires only a concatenation of the partition LSB index and the CTT segment offset. This could be achieved by allocating the VPN segment at an address boundary multiple to the partition size. Each VPN partition is defined with a pair of numbers. The first one is the dimension *m* of the partition, which indicates how many LSBs are used as an index inside the CTT segment. The second number is the offset within the CTT where this VPN partition is allocated. Since this offset is aligned on address multiple to the partition dimension, when forming the final CTT index, the partition offset and the *m* LSBs of the VPN are concatenated. This scheme and the required hardware logic are illustrated in Figure 6. To facilitate the hardware implementation, instead of storing directly the dimension *m* for each partition, a partition mask is used, containing *m* 1s on the least significant bits, extended with zeroes towards the most significant bits. By using such a representation, the logic needed to compute the final CTT index comprises of one *OR* and one *AND* gate per bit position. The pairs of *Partition Offset* and *Partition Mask* defining each partition are stored in a small set of registers or a small table, referred to as the *Partition Identification Table (PIT)*. As demonstrated in Section 7, eight such entries will be typically enough to target all the VPN partitions for each hot-spot.

Figure 7 shows the entire address translation architecture. It is noteworthy that the lookup into the PIT is performed early in the pipeline when the load/store instruction is decoded. Therefore, the PIT lookup is outside the critical path of cache lookup or data memory access. Only the circuitry for computing the CTT index, which consists of two logic gates per bit, is needed when the virtual address is generated by the processor.

6. SYSTEM ISSUES

As the proposed approach can be used in multitasking environments or in a program with multiple hot-spots, special care needs to be taken to efficiently utilize the CTT space. If all the VPN partitions of the application can be accommodated within the CTT, they are loaded during system setup. However, when this is not possible due to large number of VPN partitions across all the hot-spots, the PIT and CTT have to be initialized with the appropriate data just prior to entering the hot-spot, thus sharing CTT space with other hot-spots and processes. This scheme can be easily applied when the total number of CTT entries is relatively small and amounts to tens or hundreds of entries. The performance overhead of such a setup code prior to entering the hot-spot is practically zero as executing the subsequent application hot-spot would take millions of cycles. Another alternative could be implemented as well. If multiple hot-spots are cycled through frequently, eliminating the overhead of the setup code must be needed. For such cases, an on-demand CTT loading scheme can be effected. To achieve this, each CTT entry needs to be associated with a hot-spot identifier. When CTT entry is accessed, the hot-spot id stored in it is compared against the id of the current hot-spot. A match would indicate that the CTT entry contains a valid translation information. Otherwise, the default D-TLB is looked up to perform the default, general-purpose address translation. Such a default D-TLB translation could be invoked only once per CTT entry while executing a hot-spot. The translation entry obtained through the default translation mechanism is provided to the CTT for later utilization by the proposed approach. If multiple hot-spots have their VPN segments only partially overlapped in the CTT, some entries would be preserved for the next execution of the same hot-spot. The same methodology can be easily extended to multitasking environments for sharing the CTT space among multiple processes.

Dynamic memory allocation, a software technique rarely used in embedded application, presents an issue that needs special consideration. Even if an application requires dynamic memory allocation, such allocation is typically performed outside the hot-spots and only references to these locations are allowed inside the hot-spots. Although the virtual address for such memory references are not available at compile time, the data heaps are normally assigned by the OS and reside within consecutive virtual pages. Therefore, a separate VPN partition can be reserved in the CTT, and all the references to dynamically allocated memory are directed to this partition. As the physical memory is allocated by the OS at run time, the OS also fills the appropriate CTT entry. Inside the hot-spot, references to dynamically allocated data are therefore treated as any other memory references and mapped to their own CTT partition.

7. EXPERIMENTAL RESULTS

In evaluating the proposed technique, we have performed a quantitative analysis and comparison of baseline D-TLB architecture and the proposed application-driven address translation organization. The baseline D-TLB contains 64 entries, with either 4-way or 8-way set associativity. The virtual page size is fixed to 4K. The baseline D-TLB access energy for a 0.18μ process technology is obtained by using the CACTI tool [9].

For the proposed technique, a CTT of 512 entries is used, where each CTT entry consists of four bytes. A banking memory architecture is assumed with each bank having 32 CTT entries. Consequently, partitions are merged to maximum dimension of 9 and minimum of 5. The maximum number of partitions per hot-spot is set to 8, thus resulting to a total of 8 PIT entries, each consisting of 9-bit *partition offset* and 9-bit *partition mask*. From the results reported in Figure 9, it can be seen that the number of VPN partitions per hot-spot is always below eight even for the most complex benchmarks. The width of the *AND* and *OR* gates for computing the CTT index is also set to 9 bits. Additionally, a default D-TLB with 64 entries is assumed for VPNs outside the hot-spots. The CTT access energy is obtained by using CACTI; this is achieved by subtracting the tag-related energy from the total energy of a direct mapped cache. The access energy of the PIT register file is estimated by using the data presented in [10]. The energy for 0.2μ and 2V Vdd process technology parameters has been scaled down

	adp	g721	gsm	epic	jpeg	mpeg	mp3
h-s	1	1	1	1	2	3	5
freq(%)	100	100	100	100	11,72	81,1,11	25,13,24,16,18
E(4sa)	0.21	14.1	20.7	2.29	23.6	131.2	94.9
Miss(4sa)	8	12	14	2295	2513	1118	46719
E(8sa)	0.36	24.8	36.2	4.02	41.4	230.1	166.4
Miss(8sa)	8	12	14	2323	2510	1126	53730

Figure 8: Baseline D-TLB characteristics

	adp	g721	gsm	epic	jpeg	mpeg	mp3
Init Part	2	2	3	4	4,7	5,3,5	8,11,7,7,12
Part	1	1	1	3	1,1	3,2,4	3,2,2,1,2
Index	{2}	{2}	{2}	{3,6,7}	{9} {9}	{1,6,7} {1,7} {7,7,5,1}	{3,5,4} {5,5} {5,5} {5} {5,5}
E(4sa)	0.08	5.37	7.86	0.87	1.08	55.6	39.0
Red(%)	62.0	62.0	62.0	62.0	54.3	57.9	58.9
E(8sa)	0.08	5.37	7.86	0.87	13.0	62.6	42.6
Red(%)	78.3	78.3	78.3	78.3	74.0	75.8	76.5

Figure 9: Energy reductions of the proposed technique

to 0.18μ, 1.7V Vdd process technology by applying the same estimation methodology as the one utilized in CACTI. The power consumption of the few logic gates needed in the computation of the final CTT index is accounted for as well, even though their contribution is orders of magnitude less than the power consumption of the CTT table.

Our experimental study has been performed on a set of widely used embedded applications from the Mediabench [11] set of benchmarks. The simulation is performed with the SimpleScalar toolset [12]. Through benchmark simulation and analysis, the hot-spots and their virtual memory layout are identified. The final power consumption is computed by summing up the energy for all the VPN to PPN translations including the energy needed for the hardware.

Figure 8 shows the baseline characteristics. The first row in the table contains the benchmark name. The first 6 applications are from the Mediabench set of benchmarks, while the seventh application is a widely used open source *mp3* encoder. The subsequent row shows the number of hot-spots identified for each benchmark with the execution frequency for each hot-spot in percentage presented in the next row. The last two pairs of rows correspond to 4-way set associative and 8-way associative TLB organizations respectively. The first row of each pair shows the energy dissipation in mJ, while the second row shows the number of D-TLB misses in all the hot-spots of each benchmark.

Figure 9 shows the energy dissipation for the proposed methodology with a default general-purpose D-TLB of 64 entries. The first row presents the number of initial partitions for each hot-spot at the beginning of the merging phase of our algorithm. The next row shows the number of VPN partitions after applying the merging algorithm. The improvements in terms of minimizing the total number of complete VPN partitions per hot-spot is evident from this data. The next row in the table presents the dimension m of each VPN partition. This number indicates the CTT volume needed to handle the VPN partition. The last two pairs of rows report the energy achieved by the proposed approach and the percentage improvement compared to the baseline TLB architecture. The first pair of rows present the data for the proposed approach with a 4-way set-associative default D-TLB accessed outside the hot-spots, while the second pair of rows corresponds to an 8-way set-associative default D-TLB. The first row for each pair shows the energy consumption in mJ for the proposed technology, while the second row shows the energy reduction in percentage against the baseline case. It can be observed that the energy reductions are in the range of 54% to 79%. It is evident that the proposed technique consistently achieves high energy reductions even for complex benchmarks, such as *mpeg* and *mp3*, with multiple hot-spots and large VPN partitions.

8. CONCLUSIONS

In this paper, we have presented a methodology for energy-efficient and time-deterministic address translation for virtual memory support in embedded processors. Compiler and hardware support are used to extract and utilize application knowledge regarding the virtual page accesses of the program in order to achieve conflict-free and, thus, tag-less translation table lookup. By allocating each VPN partition into a separate segment within a tag-free translation table, not only are significant power reductions achieved, but also the execution time of the program is made more easy to asses in advance. The proposed hardware support captures in a set of registers and a small translation table the information regarding the VPN partitions and address translations. As this is performed in a software-controlled way, the proposed hardware architecture is highly programmable and the proposed approach is applied across multiple programs or even across important program fragments. The application-customizable address translation mechanism that we have proposed in this paper, enables the adoptions of virtual memory support with its significant benefits in embedded systems, where energy efficiency, multitasking, and real-time response are requirements of utmost importance.

9. REFERENCES

[1] B. Jacob and T. Mudge, "Virtual memory: issues of implementation", *IEEE Computer*, vol. 31, n. 6, pp. 33–43, June 1998.

[2] M. Ekman, F. Dahlgren and P. Stenstrom, "TLB and snoop energy-reduction using virtual caches in low-power chip-microprocessors", in *ISLPED*, pp. 243–246, August 2002.

[3] M. Cekleov and M. Dubois, "Virtual-address caches. Part 1: problems and solutions in uniprocessors", *IEEE Micro*, vol. 17, n. 5, pp. 64–71, September 1997.

[4] X. Qiu and M. Dubois, "Towards virtually-addressed memory hierarchies", in *HPCA*, pp. 51–62, January 2001.

[5] T. Juan, T. Lang and J. J. Navarro, "Reducing TLB power requirements", in *ISLPED*, pp. 196–201, August 1997.

[6] J. H. Lee, J. S. Lee, S. Jeong and S. Kim, "A banked-promotion TLB for high performance and low power", in *ICCD*, pp. 118–123, September 2001.

[7] M. Kandemir, I. Kadayif and G. Chen, "Compiler-Directed Code Restructuring for Reducing Data TLB Energy", in *CODES+ISSS*, pp. 98–103, September 2004.

[8] S. Baase and A.V. Gelder, *Computer Algorithms*, Addison-Wesley, Boston, MA, 2000.

[9] G. Reinman and N. Jouppi, "An Integrated Cache Timing and Power Model", Technical report, Western Research Lab, 1999.

[10] V. Stojanovic and V.G. Oklobdzija, "Comparative analysis of master-slave latches and flip-flops for high-performance and low-power systems", *IEEE Journal of Solid-State Circuits*, vol. 34, n. 4, pp. 536 – 548, April 1999.

[11] C. Lee, M. Potkonjak and W. H. Mangione-Smith, "MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems", in *30th MICRO*, pp. 330–335, December 1997.

[12] T. Austin, E. Larson and D. Ernst, "SimpleScalar: An infrastructure for computer system modeling", *IEEE Computer*, vol. 35, n. 2, pp. 59–67, February 2002.