

# Heterogeneously Tagged Caches for Low-Power Embedded Systems with Virtual Memory Support

XIANGRONG ZHOU and PETER PETROV  
University of Maryland, College Park

---

An energy-efficient data cache organization for embedded processors with virtual memory is proposed. Application knowledge regarding memory references is used to eliminate most tag translations. A novel tagging scheme is introduced, where both virtual and physical tags coexist. Physical tags and special handling of superset index bits are only used for references to shared regions in order to avoid cache inconsistency. By eliminating the need for most address translations on cache access, a significant power reduction is achieved. We outline an efficient hardware architecture, where the application information is captured in a reprogrammable way and the cache is minimally modified.

Categories and Subject Descriptors: C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors)

General Terms: Design, Experimentation

Additional Key Words and Phrases: Embedded systems

## ACM Reference Format:

Zhou, X. and Petrov, P. 2008. Heterogeneously tagged caches for low-power embedded systems with virtual memory support. *ACM Trans. Des. Autom. Electron. Syst.* 13, 4, Article 32 (April 2008), 24 pages, DOI = 10.1145/1344418.1344428 <http://doi.acm.org/10.1145/1344418.1344428>

---

## 1. INTRODUCTION

Increasingly many new embedded systems are emerging for which miniaturization and communication are of great importance. Such applications can include nodes for industrial/environmental process control, systems for data acquisition and object tracking, wearable computing applications with data manipulation communication and security capabilities, etc. Such systems typically rely on autonomous power sources, thus imposing stringent requirements for energy efficiency.

---

Authors' address: X. Zhou, P. Petrov (corresponding author), Department of Electrical and Computer Engineering, University of Maryland, College Park, MD 20742; email: [ppetrov@ece.umd.edu](mailto:ppetrov@ece.umd.edu). Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or direct commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or [permissions@acm.org](mailto:permissions@acm.org). © 2008 ACM 1084-4309/2008/04-ART32 \$5.00 DOI 10.1145/1344418.1344428 <http://doi.acm.org/10.1145/1344418.1344428>

ACM Transactions on Design Automation of Electronic Systems, Vol. 13, No. 2, Article 32, Pub. date: April 2008.

The memory system is one of the most crucial components of any modern computer system. Typically, its design characteristics greatly determine the overall performance and power consumption of the system. Since its conception, virtual memory [Jacob and Mudge 1998] has been shown a very elegant and efficient solution for abstracting away from the application the complexity of *memory allocation, code/data relocation, memory sharing, and protection*. All these features are completely transparent to the application and controlled by the memory manager of the operating system (OS). Consequently, supporting this concept is a must for many complex embedded systems, where the application comprises multiple tasks that coexist and share the available physical memory and hardware resources. The program accesses a virtual address space separated into pages, and at each such access a translation is needed to map the virtual address into a physical one. The translation is performed at page granularity in order to control the complexity of the translation mechanism. However, traditional virtual memory support is very power consuming, which has prevented its adoption in energy-constrained embedded systems. In this article, we outline a methodology which utilizes application-specific knowledge regarding the embedded application to achieve energy-efficient virtual memory support for high-end embedded processors with caches.

Many contemporary embedded processors, such as the Intel XScale [Intel Corp. 2007] and the ARM9 [ARM Ltd. 1995], are offering a *memory management unit (MMU)*, which is responsible for providing the hardware support for translating addresses generated by the processor to addresses in the available physical memory. The MMU provides for address translation through the *translation lookaside buffer (TLB)*, a cache-like structure responsible for the dynamic translation from virtual to physical addresses on each memory access. The mapping between virtual and physical addresses is typically maintained by the OS memory manager. The fundamental purpose of the TLB is to serve as a translation cache, so that the overhead of having the OS perform the multilevel page table lookup is avoided. TLB misses typically result in trapping into the OS where the missed translation 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. As lookups need to be performed every time the memory is accessed, it introduces significant power overhead. It has been shown that TLB power consumption constitutes 20–25% of the total cache power consumption [Ekman et al. 2002]. It has also been shown that for the StrongARM and Hitachi SH-3 processors [Juan et al. 1997; Kadayif et al. 2002] around 17% of the total on-chip power is contributed by the TLB. In this article we propose a novel cache architecture which eliminates the need for TLB lookup for the majority of cache accesses, and thus significantly reduces power.

In the presence of virtual memory, caches can be accessed in several different ways. Since there are both virtual and physical addresses present in the system, either can be used to access the cache. Furthermore, the cache access operation can be split into two components: indexing and tagging. Consequently, four types of cache access mechanisms can be constructed depending on the type of address used for indexing and tagging. If virtual addresses only are used to

access the cache, the resulting cache is referred to as *virtually indexed and virtually tagged*. The benefit of this cache architecture is that there is no need for address translation when accessing the cache, which results in fast access time and, even more importantly for embedded processors, low power consumption [Qiu and Dubois 2001; Kim et al. 1995; Woo et al. 2006]. Virtually indexed and tagged caches, however, exhibit severe drawbacks, namely cache *aliasing* and *synonyms* [Cekleov and Dubois 1997; Woo et al. 2006]. Cache aliasing is a situation where the same virtual address from different tasks is mapped to different physical addresses. Such a situation necessitates flushing the cache on context switch, which can lead to significant performance degradation. The aliasing problem is avoided if the tags are extended to store a *process identifier (PID)*, a unique key associated with each process. Additionally, since no TLB lookup is performed, each cache line must be extended to capture the access control bits for the cache line address range. At the same time, however, the cache synonym problem has been traditionally difficult to overcome. Cache synonyms occur when different virtual addresses, usually from different virtual address spaces, are mapped to the same physical address. This situation occurs naturally when two processes share data. If virtual addresses are used to access the cache, the different virtual synonyms of the shared physical location will end up in different cache blocks. This, in turn, can easily lead to cache coherence problems if one of the processes writes into the synonym location, leaving the other cached copies of the shared data stale. Various solutions for avoiding cache synonyms have been offered for general-purpose processors [Kim et al. 1995; Qiu and Dubois 2001], all of which introduce nontrivial hardware structures with significant power overhead, thus making their adoption in embedded processors infeasible. Because synonyms occur when sharing writeable data, virtually indexed and tagged caches have been mostly used as I-caches where such sharing usually does not exist.

*Physically-tagged and physically-indexed* caches are the exact opposite. In this cache architecture, only physical addresses are used for indexing and tagging. Naturally, aliasing and synonyms are no longer an issue. However, address translation needs to be performed for each cache access and before forming the cache index, which results in delayed cache access time and significant power overhead.

*Physically-tagged and virtually-indexed* caches are the most typical D-cache architecture for processors with virtual memory support. By performing address translation only for the tags, cache indexing is overlapped with tag translation, thus effectively hiding the address translation latency. By imposing certain restrictions on the OS memory manager and by introducing additional hardware support, the physically-tagged and virtually-indexed cache eliminates the cache synonym problems with no performance implications. However, the power consumption of such a cache architecture is quite high, as address translation is performed each time the cache is accessed in order to obtain the physical tag.

In this article we propose a novel cache architecture which *selectively uses either physical or virtual tags*, with the objective of minimizing the number of address translations, that is, physical tags. In this way the power

benefits of virtual caches are combined with the synonym elimination benefits of physically-tagged caches. Application-specific information regarding the type of memory references is used to determine whether to use a virtual or a physical tag. Tag translation is performed only for memory references which can potentially refer to shared memory and, as such, result in cache synonyms. The majority of cache accesses which refer to private data in the process's virtual address space, that is, pages only mapped to that address space, are handled with virtual tags, thus necessitating no address translation on cache access. Not only are the majority of tag-related address translations eliminated, but also for the shared data memory references we introduce a new translation scheme. The new scheme utilizes knowledge of physical page locations in order to replace power-expensive TLB lookups with simple arithmetic operations. This novel *cache-tagging* approach coupled with *specialized address translations* for the shared physical pages results in very energy-efficient cache operations.

## 2. RELATED WORK

The memory subsystem has been known to be one of the major bottlenecks in terms of power and performance, not only for general-purpose computing systems but also, and even more so, for the typically resource-constrained embedded systems [Panda et al. 2001; Benini et al. 2003]. As virtual memory and caches coexist in many modern high-performance processors, the importance of caches and their associated address translations in terms of performance and power has been recognized in the microarchitecture industry and research communities. Consequently, techniques for power reduction and performance improvement for caches and TLBs have been the focus of many research efforts. Cache-conscious memory layouts have been explored for both code [Calder et al. 1998; Benini et al. 2004] and data [Chilimbi et al. 1999; Kulkarni et al. 2005]. In Juan et al. [1997], 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 TLB organization for chip multiprocessors has been proposed in Ekman et al. [2002]. By incorporating a special *page sharing table* to the TLB and using virtual addresses, the authors reduce the amount of TLB activities while 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 Qiu and Dubois [2001]. The authors have proposed to replace the TLB with the more scalable and power-efficient *synonym lookaside buffer*, as it stores only the current synonym instances. In Kim et al. [1995], the authors have proposed the *U-cache* architecture, which maintains reverse translation information of the cache blocks that belong to unaligned virtual pages only in order to handle the synonyms efficiently. A low-power physically-tagged cache has been proposed in Petrov et al. [2005]. A minimal set of tag bits is dynamically identified per hot-spot and used to access the cache instead of complete tags. In Kadayif et al. [2002] the authors have proposed an instruction address translation architecture which places the most recent I-TLB translation in a register that is subsequently reused until the instruction memory page is changed. For periods of time when instructions are

fetched from the same memory page, the translation register is used to obtain the physical address instead of the I-TLB, thus achieving faster and less power-consuming instruction address translation. A low-power TLB organization that dynamically supports up to two pages per entry with a banked, fully associative structure is proposed in Lee et al. [2001]. In Kandemir et al. [2004], the TLB accesses are redirected to a register file which holds a few recent TLB entries. The authors have proposed a special code reconstruction phase within the compiler in order to minimize the overhead of dynamic register entry updates. Compiler techniques have been presented in Kadayif et al. [2004]; these methods maximize the reuse of software-controlled translation registers. Methodologies which emulate the address translation process in software have been recently proposed in Middha et al. [2005] and Simpson et al. [2005]. In these approaches the compiler introduces code for runtime checks in order to enable applications to share physical memory for their stacks and to prevent any out-of-boundary memory accesses.

All previous techniques target the TLB organization for general-purpose processors, or aim at minimizing the power consumption of either physically- or virtually-tagged caches. Our work takes a step further in the domain of embedded processors where application knowledge is exploited and the architecture dynamically customized. In this way accesses to the TLB are eliminated for all of the nonsynonym VPN accesses, while at the same time a power-optimized address translation scheme is used for the synonym VPN accesses. As an end result we achieve significant energy reductions with practically no impact on performance.

### 3. HETEROGENEOUS CACHE TAGGING—A FUNCTIONAL OVERVIEW

The proposed cache-tagging policy takes into account application information regarding the page status of the memory location being accessed. A virtual page, known to be mapped to a physical page which (in turn) is mapped to at least one other virtual page from another address space (belonging to another process), is considered to be shared. Such mappings are established and controlled by the OS on request from the application, or when there is a need to share code or data between different address spaces. Physical tags are used with the purpose of resolving the serious problems which such shared memory pages can cause when stored in the data cache. The proposed heterogeneous cache tagging in essence is a hybrid approach which combines the low-power benefits of virtual tags with the cache-synonym-avoidance properties of physical tags.

Synonyms are multiple virtual addresses which are mapped to the same physical address. Synonyms can naturally appear when a shared data in physical memory is being mapped to different virtual addresses in more than one task, in order to facilitate data communication between processes. With synonyms, only a single copy of the shared data is maintained in physical memory; this is very beneficial for embedded systems, which typically have limited memory resources. Not only is memory usage minimized, but the performance overhead of copying the shared data between the processes' address spaces is eliminated.

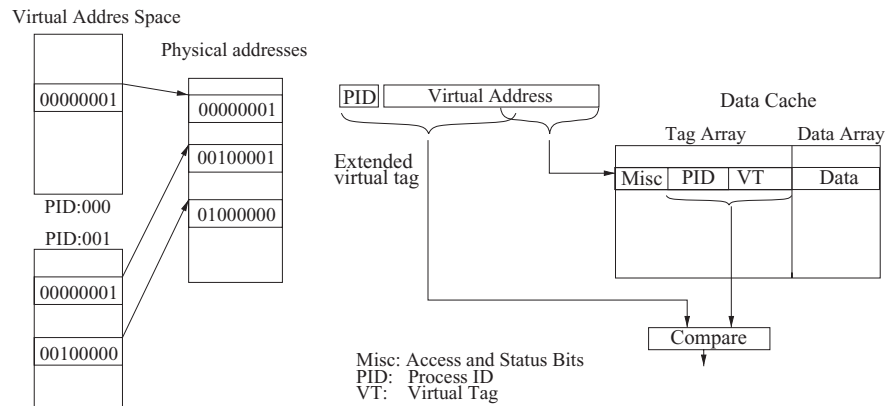


Fig. 1. Using PID to eliminate aliasing with virtually-tagged caches.

In general-purpose processors, the application usually comes in binary-only format. Consequently, no information regarding data sharing is available to the microarchitecture, which assumes that a large variety of programs will be executed. Thus every memory access is assumed a potential synonym address. 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 work, is fundamentally a technique that, with the help of the compiler and operating systems, exploits dynamically this application-specific knowledge.

For example, if it is known in advance that there is not sharing data mapped to different address spaces, or that there is not sharing data at all in the target embedded system, then it is clear that no cache synonyms could exist. Figure 1 shows an example of two processes sharing the cache and not sharing physical pages. It can be seen that the first virtual pages from both processes have the same virtual address, but are mapped to different physical pages. In situations like this when no synonyms are possible, the combination of process ID and virtual address can serve as the single identifier to differentiate among all data addresses. Even though the virtual address of the two first pages from both address spaces are identical, the process identifier (PID) of each process which extends the tag would suffice to distinguish the two identical virtual addresses. Thus, no physical address is needed to locate data in the cache, and the TLB lookup step prior to accessing the cache could be avoided completely. Consequently, the cache for such references can be virtually indexed and virtually tagged with tags extended with process IDs. Only when physical memory needs to be accessed, in the cases of a cache miss or a cache write-back, is the virtual address translated into a physical address through the TLB. Additionally, *access control* (AC) bits and other status bits are associated with each cache block and obtained from the TLB when the data block is placed in the cache. For virtually-tagged caches it is inevitable that the cache line status bits need to be extended to contain access control for the cache line address. Since no TLB

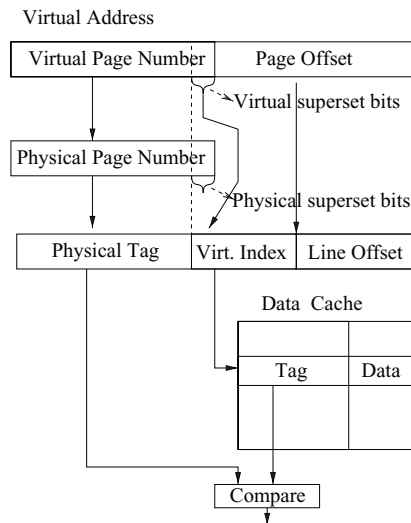


Fig. 2. Superset bits overlap between virtual index and VPN.

lookup is performed on cache access (and hit), the AC bits (usually from 2–4) need to be present in the cache.

In general, however, it is common that multiple processes working under the same application would need to share data in order to communicate with one another. While the majority of data accesses are typically nonshared data which can use virtual tags with the traditional for virtually-tagged caches extension of PID and AC (as shown in Figure 1), special care needs to be taken for the memory references mapped to shared physical pages.

A virtual address consists of a VPN and page offset. When used to access the cache, the same address is split into block offset, cache index field, and a tag. A *synonym group* is the set of VPNs from different processes which are mapped to the same physical page. In order for the cache to work properly it must be ensured that all synonyms from the group are mapped to the same cache location in order to access the shared physical page. If the cache index part of the address is completely contained within the page offset part, thus implying that the cache index from the virtual address is identical to the physical index, then no synonyms can occur if physical tags are used. When the virtual page size, however, is smaller than the cache size divided by the associativity, then a fraction of the most significant bits of the cache index field overlaps with the VPN. The intersection bits of cache index and VPN are called *superset bits*, or *color bits*. The set of synonyms are *aligned* if their superset bits are identical to the superset bits of the physical address, as shown in Figure 2. In this case, the virtual cache index is the same as the physical index, and consequently the physical tag is sufficient to differentiate among synonyms in the cache. If the superset bits are not identical and do not match with the corresponding bits from the physical address, which is often the case when no special care is taken, it becomes possible that the same location in physical memory is cached at two different cache locations. Furthermore, if synonyms from the same groups are

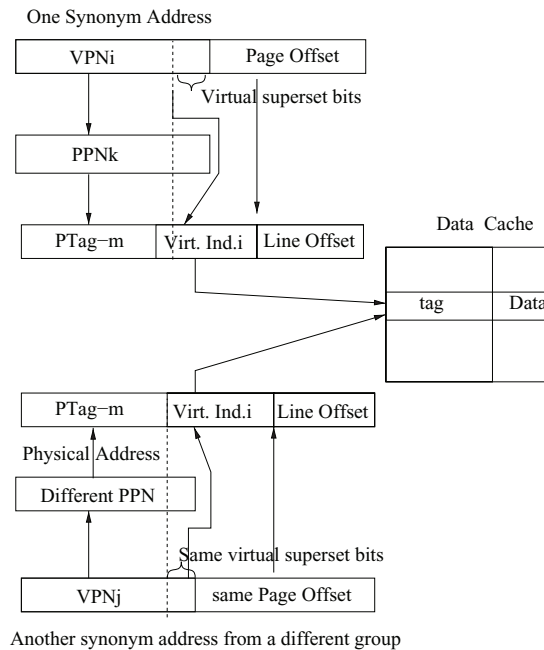


Fig. 3. Conflicting cache indices for the case of synonym groups exhibiting identical physical tags and virtual superset bits.

only aligned in the virtual address space, that is, the virtual superset bits are identical but do not match the physical superset bits, then it is possible that this synonym group may conflict in the cache with another memory location which happens to have the same virtual index, virtual superset bits, and physical tags, but different physical superset bits. Such a conflict will not be resolved in the cache with the physical tags, as they are identical. This situation is illustrated in Figure 3. Of course, this situation cannot happen if the synonym group is completely aligned (both in virtual and physical address space) or larger physical tags are used that overlap the superset bits.

Traditionally, general-purpose processors with virtual memory utilize virtually-indexed and physically-tagged caches. To avoid the synonym problem, the OS memory manager is required to align the set of all synonyms to the physical page frames to which they map, that is, is required to provide for a complete superset alignment in both virtual and physical address spaces. Such a requirement imposes a significant constraint in terms of physical memory utilization, as physical frames can be placed only in a subset of all possible page locations. This could impact the page fault rate in general-purpose systems, while in the case of an embedded system with limited physical memory it could result in cases where such alignment is simply not possible for the given working set unless physical frames are moved to secondary flash memory. Not only is such a requirement prohibitive for energy-efficient embedded systems, but also such a cache organization requires tag address translations each time the cache is accessed, resulting in significant power consumption.

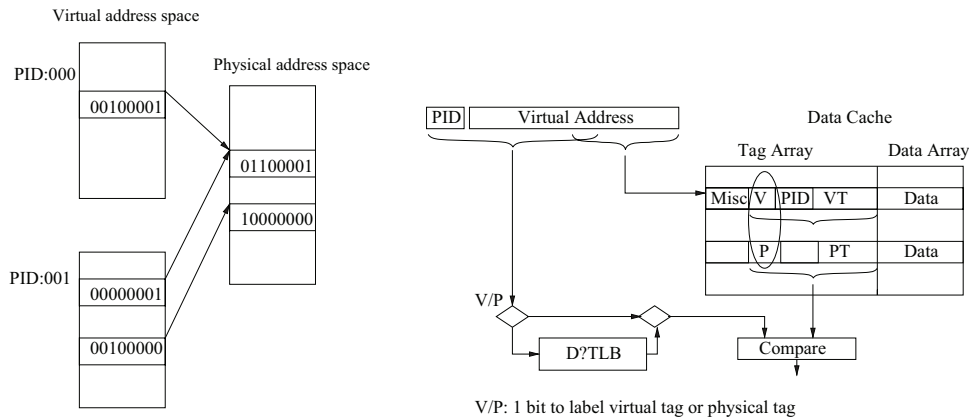


Fig. 4. Two processes sharing one data block: 1 aligned virtual addresses.

The technique that we propose in this work offers a cache architecture where both virtual and physical tags are utilized at the same time. Virtual tags are used for the majority of memory references to nonshared data, while physical tags are used only for references to shared memory. No restriction in terms of synonym alignment is imposed on the OS memory manager. A special mode bit is associated to each cache line to indicate whether a virtual or a physical tag is being used for that cache line. All cache lines are virtually indexed, with non-shared data tagged with virtual tags while shared data references are tagged with physical tags. Figure 4 illustrates an example where two processes, each having private data, share a data block. Shared blocks are identified in advance, in a way described in a subsequent section, and physically tagged when placed into the cache from physical memory. The private data blocks are tagged with extended virtual tags. The physical tag can be translated in parallel with the cache indexing. Thus, a significant amount of power for address translation for nonshared data references is saved, with practically no performance degradation.

As a virtually-tagged cache line can overlap with a physically-tagged cache line, the mode bit is used to differentiate between them, even if it happens that the virtual tag is identical to the physical tag; for all practical purposes the mode bit can be thought of as an extension to the tag. Consequently, when a sequence of references to private data is being generated by the processor and all of the data is in the cache, no address translations will be needed to access the cache. In the case of a cache miss, the address for the missed reference needs to be translated in order to access the physical memory. Furthermore, if the replaced cache line is virtually tagged but dirty (in the case of a write-back cache), it needs to be translated into a physical address before writeback to memory.

#### 4. WRITE-BACK AND WRITE-THROUGH CACHES

Write-back and write-through organizations need to be treated differently by the proposed methodology, as they treat differently the write accesses to

memory. For write-back caches, the proposed methodology works as explained so far. Cache lines containing data from shared memory regions utilize physical tags. For the rest of the memory, virtual tags are used for both read- and write lookups to the data cache.

When a cache miss occurs, one or two address translations are needed. In the case of read-only cache lines being replaced, only one translation is needed for the reference which is being brought to the cache. However, in the case of replacing a modified cache line, two address translations are needed. The first translation is to obtain the physical address for the read data and the second is for the physical address of the modified cache line that needs to be stored back to main memory, or to the lower level of the memory hierarchy. The translation for the missed reference, of course, has higher priority and needs to be performed first, as the processor is waiting for this data in a way identical to traditional virtually-tagged caches. The second address translation needs to be performed for the just-replaced cache line. As writing back to memory is typically not a time-critical operation for the processor, it is usually done on the backend of the processor by using a write buffer implemented as a queue; for instance, the XScale processor features an 8-entry write buffer. The entries in the write buffer are written back to memory when the memory bus is not occupied servicing processor reads. In the organization we propose, the write operations will enter the write buffer immediately with its virtual address and a single bit that specifies a virtual address (a write-buffer entry contains the address and the data to be written). The needed address translation will typically be performed during the next clock cycle, the only exception being that the TLB is needed to translate another cache miss in the subsequent cycle. In such a case, the write will remain in the write buffer until the TLB is available and its virtual address translated into a physical one. Having two cache misses in two consecutive clock cycles is an extremely rare situation and even in such a case, the write to memory will be delayed in the write buffer for a cycle, which will not impact processor performance. The only modification needed to the TLB controller is the simple logic that checks whether there is a write-buffer entry carrying a virtual address that needs an address translation. The typical size of a write buffer is 4 entries and the needed multiplexing for reading these entries already exists, as on cache miss the processor first checks the write-buffer entries before going to memory. For the proposed organization, checks in the write buffer are performed in the same way as in the cache, by using either a virtual or a physical tag. Consequently, the only hardware overhead to the TLB controller is the logic of, at the utmost, several gates that checks the mode bit for the write-buffer entry and subsequently replaces the write-buffer entry address with a physical one.

Write-through caches propagate each write to the lower level of the memory hierarchy. Because of this, in the case of memory-write the physical address is always needed, regardless of whether the memory location is shared. Read-memory references are handled in the same way as for the write-back cache organization. Consequently, the proposed technique will not achieve energy reductions as large as in the case of write-back caches. This effect is quantitatively evaluated in our experimental results. Note that this situation is different from

the case explained before, when two address translations are needed. In this case all writes which hit in the cache would need to be propagated to physical memory, thus requiring physical addresses (and the corresponding single TLB lookup), regardless of whether they are to private- or shared memory regions. In order to alleviate the write effect for write-through caches, we introduce a *physical page latch (PPL)* which stores the translated address for the most recent memory write. Together with this latch, we also introduce a register which holds the VPN for that most recently accessed physical page frame. This VPN serves as a tag which identifies whether the currently write-accessed memory page is the same as the one most recently written to page. Such a check is performed by simply comparing the VPN of the write operation with that stored in the register. In the case of a match, no TLB lookup needs to be performed, as the physical page number for this VPN is present in the PPL. In this way, a long series of writes to the same memory page will require only a single TLB lookup for the first write, while all subsequent writes will reuse it from the PPL. The only overhead associated with the PPL is the VPN comparator activated on a write to memory. The power needed by such a comparator, which is essentially a collection of XOR gates, is extremely smaller than the power taken by a TLB lookup. In our experimental results we evaluate in detail the utility of the PPL in achieving significant power reduction for write-through caches.

## 5. IDENTIFYING THE REFERENCES TO SHARED MEMORY

One of the important aspects of the proposed low-power cache-tagging organization is that it needs to be known in advance whether the address generated by the processor refers to shared or private data for that program. The proposed scheme performs address translations only for shared memory regions. It is important to understand what these shared regions are with respect to the program's address space and when the addresses of these locations are known. In the context of traditional physically-tagged/virtually-indexed caches and the proposed heterogeneously-tagged caches, a shared memory region is considered to be a physical memory page which is mapped by the OS memory manager into the virtual address spaces of at least two processes. Such shared physical memory pages exist for the purpose of communicating data between two different processes running in different virtual address spaces; it is controlled and provided by the OS in the form of *interprocess communication (IPC)* facilities, such as shared memory regions or shared buffers for message passing. The shared memory pages, following a request from the application program, are mapped to the virtual address space by the OS. Consequently, this interprocess sharing is always established and performed explicitly by the application process and controlled by the OS. The compiler can be easily made aware through special *#pragma* directives as to whether a particular data buffer from the application address space is to be treated as shared or private.

It is noteworthy that interprocess memory sharing, which requires special attention in terms of caching, is completely different from the data sharing that exists in multithreaded programs. Multithreaded applications are formed by running multiple lines of control (threads) that execute in the same address

space: the address space of the process where the threads are created. Consequently, the multiple threads belonging to this process share the address space and any communication between them is to be handled by the programmer with no intervention of the operating system. Such independence from the OS is the major advantage of lightweight threading, as switching between threads is usually performed entirely in user space, thus incurring minimal overhead. From the address space- and data-caching perspectives, however, all memory accessed by the multiple threads for sharing or nonsharing purposes amongst them is private for that address space and completely indistinguishable from the private memory of processes which do not carry multiple threads. No cache synonyms are possible for thread-level sharing because the threads use the same virtual addresses to access internally shared data. From the cache point-of-view, only memory pages, which are mapped across multiple address spaces and can thus be referred to by different virtual addresses, can result in cache synonyms.

Since the shared data buffers are explicitly defined by the application process, they can be easily identified by the compiler and OS. A mechanism is needed, however, to distinguish the virtual addresses referring to these interprocess shared pages from those referring to the rest of the virtual pages. One possibility is to use an extra space from the load/store instruction encodings in order to tag those memory instructions which refer to shared pages. For shared data, which is explicitly declared by the programmer and accessed directly, it is a trivial job for the compiler to tag the corresponding load/store instructions. However, if the program uses pointers to access such shared memory pages, it becomes significantly more difficult to determine which pointers can access shared pages; in most cases conservative assumptions need to be made and the majority of pointers marked as potential references to shared memory pages.

An alternative approach, which we have followed and recommend for our technique, is to distinguish those references to shared data through their virtual addresses. Modern embedded processors such as the Intel XScale and ARM9 feature a 32-bit virtual address space, which is very large for the demand of any embedded application. In order to distinguish references to shared pages through the virtual address, a portion of this address space may be easily reserved for such pages. For example, a small set of the most significant bits from the virtual address may be used to signify whether a shared page is being accessed. In this way, the interprocess shared buffers will be mapped by the OS into upper parts of the virtual address space and, as such, will be trivial to identify at runtime when generated by the processor. For instance, a value of “111” in the three most significant virtual address bits may signal that this is a reference to a shared buffer. The hardware needed for this check is a simple 3-input AND gate, which constitutes a zero overhead for all practical purposes.

## 6. LOW-POWER SYNONYM ALIGNMENT

Preventing memory synonyms to be cached at different cache locations, even in the case of physically-tagged caches, requires a special alignment in physical memory referred to as page coloring. The OS memory manager must ensure

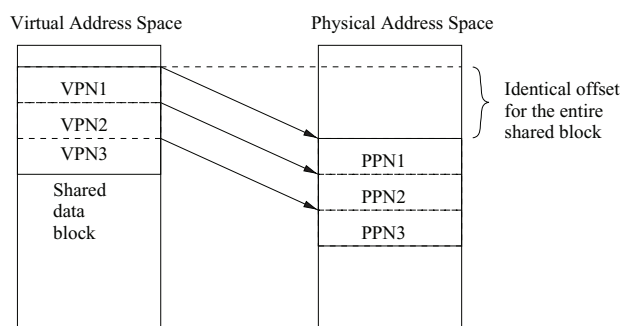


Fig. 5. Linear mapping from VPNs to PPNs.

that for each synonym group, the virtual and physical superset bits are identical. In embedded systems when the physical memory resource is limited, such alignment causes additional constraints to the memory management. In order to eliminate this constraint, we introduce a rapid translation for the superset bits, thus leaving the virtual and physical pages of shared memory regions unaligned. In such cases, the superset bits of each virtual page address are not necessarily identical to those of the physical address. Subsequently, the virtual cache index is no longer identical to the physical cache index; thus, it can potentially conflict with other virtual cache indices which do not belong to the same synonym group, as was shown in Figure 3.

In order to avoid such conflicts, the virtual superset bits need to be translated to the physical superset bits, with minimal cost. In embedded applications which are intensive on DSP and numerical computations, the shared data buffers are typically input/output buffers, coefficient tables, or just message passing buffers. For most cases, the shared buffer fits within a single page, and in those cases where it spans multiple pages it is common to allocate it in consecutive physical memory addresses and also to map it to consecutive virtual address pages. Such a consecutive property shows that their PPNs can be computed by adding an offset to their VPN. Moreover, the physical superset bits of each physical address can also be converted by adding an offset to the virtual superset bits, as shown in Figure 5. The offset can be determined by the OS when it loads the shared data into physical memory. Since the width of superset bits is  $\log_2(\text{cache size}/(\text{cache associativity} * \text{page size}))$ , only a few bits need to be manipulated. A fast parallel adder can translate the physical superset bits rapidly with little delay. Figure 6 shows an example of two processes as in the previous example, but with no alignment property. The physical superset bits are translated with the introduced superset offset adder, and the TLB for page translation is replaced with the page offset adder, which is significantly more power efficient. Multiple shared buffers could result in multiple offsets present. For such cases, the multiple offsets can be stored in a very small table (or a small set of registers/latches), the *synonyms offset table (SOT)*, and be retrieved before the add operation.

As we pointed out in the previous section, the interprocess shared memory regions are identified through their virtual addresses. Each such shared buffer

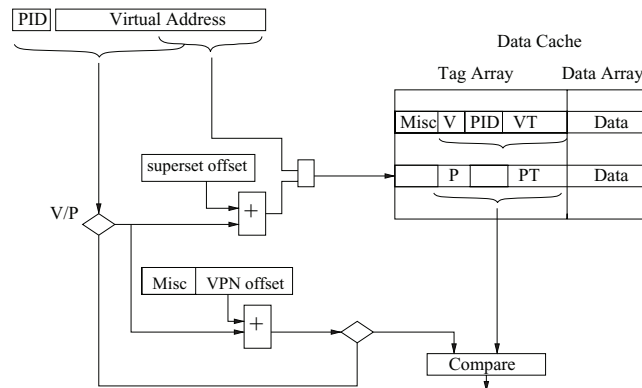


Fig. 6. Two processes sharing one data block; nonaligned shared virtual addresses.

is allocated in a prespecified region of the large virtual address space, which can be uniquely identified by a small subset of the most significant virtual address bits. For example, shared data buffers can be mapped only in the upper half of the virtual address space, each such buffer placed in a partition which is uniquely identified by the three bits to the right of the most significant address bit. Consequently, these identifying address bits can be easily used as a direct index into the synonyms offset table to obtain the offsets needed to compute the aligned superset bits, as well the physical page number of the shared memory page. The most significant virtual address bit in this example is used as an indication for whether the referenced data belongs to a shared memory page.

## 7. HARDWARE SUPPORT

The proposed methodology requires a specialized hardware support whose purpose is to perform the different cache access policies for synonym- and private references. Changes are needed in the cache line structure and the address translation path to the data cache.

Each data cache line is associated with an additional bit to indicate whether a physical tag for a synonym or virtual tag for a nonsynonym reference is being used. To accommodate virtual tags, the tag field is extended with a process ID (PID) and access control (AC) bits. The AC bits are transferred from the TLB when the cache line is placed from physical memory. Extending the cache tags with PIDs as well as associating the AC bits with the cache line is not a requirement freshly introduced by the proposed heterogeneously-tagged cache, but a general requirement of traditional virtually-tagged caches. Since the proposed technique effectively makes the cache function a virtually-tagged cache for many references to the cache with no TLB lookups, small PID and AC extensions to the cache line are required. In our experimental results we take into account the power overhead of these bits.

A very small table (or small set of registers), the synonyms offset table (SOT), is introduced to capture the few offset constants needed for the superset bit alignment and for address translation for the shared pages. As shown in Figure 7(a), each SOT entry includes a VPN offset field, a superset offset field,

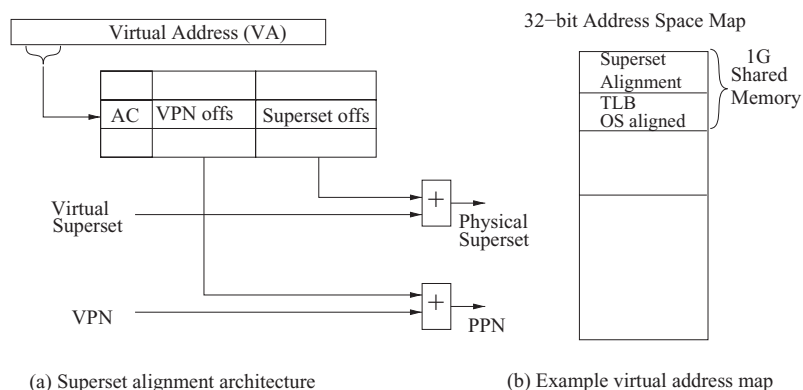


Fig. 7. The synonym offset table (SOT) and adders for converting superset bits and VPNs.

and the access control bits. Each SOT entry represents one shared data block. The SOT is directly indexed by a few of the most significant virtual address bits. For our experimental results we have assumed 8 entries/registers. As the number of SOT entries bounds the number of shared regions that can be aligned and translated through adders, it is important that there are enough SOT entries for this. However, because of the nature of this interprocess sharing, which typically corresponds to input/output data buffers and buffers for message passing, the number of shared regions is very small and almost always within the range of 2–4. Even though our benchmark did not require this many SOT entries, in order to be conservative in the evaluation of our approach we have accounted for the power overhead of an 8-entry SOT.

The number of SOT entries establishes an upper bound on the number of shared regions per process that can be handled with the adder-based synonym alignment. If the number of such regions exceeds the number of SOT entries, the remaining shared regions can be handled in the traditional way through the TLB and memory alignment. An example virtual address space layout is shown in Figure 7(b). The upper quarter of the address space is reserved for shared buffers, which are always mapped by the OS in this part of the address space. Depending on the number of SOT entries, a corresponding number of shared buffers are mapped into the upper half of the shared memory portions of the address space. The remaining shared buffers are mapped in the lower half of this space; thus are translated through the TLB and aligned by the OS in physical memory. With such a layout it becomes trivial to distinguish the address to the interprocess shared memory through a single 2-input AND gate connected to the two most significant bits of the virtual address. The third most significant bit is used to determine whether the reference will be aligned through the proposed superset alignment architecture, or will use the TLB. Even though the delay of the region identification logic and accessing the very small SOT is trivial, it can be completely hidden as well. Practically all high-end embedded processors support indexed addressing modes, where the value of an immediate field or index register is added to a base register in order to compute the effective virtual address. The value of the base register typically contains an address of

a memory segment where the accessed data is allocated. The most significant bit which define the segment's positions remain the same in this process, as the index register or the immediate field typically contains only an offset within this segment. In this way the most significant bits of the virtual address, which determine whether the reference is to a shared region, are available earlier in the pipeline before the actual effective address computation. As such, these most significant bits can be used to index the SOT and obtain the superset offset prior to entering the memory pipeline stage. Consequently, determining the type of memory address and accessing the SOT table do not introduce any performance overhead.

For all nonshared references, the virtual tag is extended with the PID and sent to access the data cache block. The access control bits field is an aggregate of the access control bits for each VPN in the shared memory block. This aggregation relaxes the access right granularity from page level to multiple consecutive pages of a shared memory block. However, it is very rare that different pages in the shared block have different access rights.

The two adders are used to rapidly and energy-efficiently translate the VPN and virtual superset for synonym addresses. The virtual superset bits are added to the superset offset to form an index to the data cache. The VPN tag is added to the VPN offset and the translated physical tag is sent to the tag comparator in parallel with cache indexing. The width of the VPN field is usually much larger than that of superset bits. For a 32-bit address and 4k page size with the 16k cache column size, the VPN is 20 bits and the superset is 2 bits. Consequently, the delay of the superset offset adder is much smaller than that of the VPN adder. As only the small (typically, 2-bit wide) superset adder is on the cache access path, the introduced delay is extremely small. The longer VPN to PPN adder, which replaces the traditional TLB lookup, is on a path parallel to the cache indexing, hence introducing no performance overhead. In any case, the adder delay is significantly smaller than the TLB delay which it replaces. In our experimental results we have taken into account the extra energy introduced by the superset and VPN adders. The entire hardware architecture of the proposed approach is presented in Figure 8.

## 8. EXPERIMENTAL RESULTS

In evaluating the proposed technique, we have performed a quantitative analysis and comparison between a baseline data cache with default TLB structure and the proposed architecture of a heterogeneously-tagged cache. We have evaluated both the adder-based translation and a traditional TLB-based translation for the shared memory regions. The baseline assumes aligned synonyms, thus the reported advantages would be even larger in reality, as handling non-aligned shared memory pages requires extra hardware or software support in the baseline architecture.

In our experimental study we have explored two major configurations. The first is representative for Intel XScale processors and consists of a 32K data cache and a 32-entry fully associative TLB; both direct-mapped and 4-way set-associative data cache organizations have been evaluated. The second

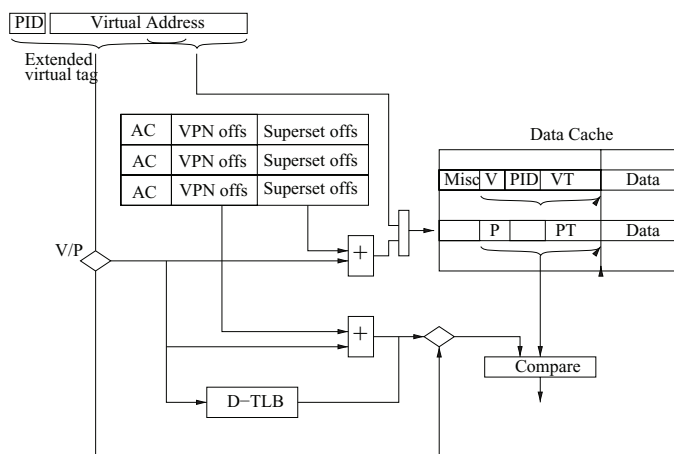


Fig. 8. Overall hardware organization.

Table I. Energy for the Baseline and Proposed D-Cache/TLB Architectures

	32-DM	32-4SA	16-DM	16-4SA	TLB-32	TLB-64
Dynamic (nJ)	0.06845/ 0.07072	0.29496/ 0.29825	0.06051/ 0.06084	0.18089/ 0.18516	0.08393	0.19222
Leakage (mW)	0.30440/ 0.31323	0.30446/ 0.31207	0.15280/ 0.16183	0.15935/ 0.16338	0.00933	0.01786

configuration, which corresponds to ARM 920T processors, consists of a 16K data cache and 64-entry fully associative TLB. Similarly to the first configuration, both direct-mapped and 4-way set-associative data cache organizations have been evaluated. The virtual page size is kept conventionally to 4K size. The energy for each access to data cache and D-TLB are estimated using the Cacti-4 [Tarjan et al. 2006] tool with process technology of  $0.18\mu m$ .

Table I shows the dynamic energy (per access) and leakage power for the baseline cache architectures, TLBs, and heterogeneously-tagged cache architecture. The baseline caches are physically tagged and have no modeling of special tag extensions beyond the standard ones, such as valid- and dirty-mode bits. For this configuration we have used the default Cacti-4 models. The proposed technique relies on a support for virtual tags, thus requiring the tags to be extended with process identifier- (PID) and access control (AC) bits. For this we have added 4 extra bits for PID and 2 bits for AC. The only extra bit that our methodology requires, which is in addition to the typical support for virtual tags, is the V/P bit that specifies the type of tag; we have included this bit in the total of 7 extra mode bits compared to the baseline cache. To model these bits, we have used the standard support for extending tags in Cacti-4, which is implemented through a special variable whose sole purpose is to define any tag extensions, if needed. Each entry in Table I depicts a pair of numbers: the first corresponds to the baseline cache, while the second is for the heterogeneously-tagged cache. The last two columns show the energy numbers for the 32- and 64-entry TLBs that we have used for our experiments. As the

Table II. Baseline D-TLB and Overall D-Cache+TLB Energy (in  $\mu\text{J}$ )

	adpcm	g721	gsm	epic	jpeg	mpeg	mp3
TLB-32	75	4051	4381	640	494	28666	26634
TLB-64	171	9279	10034	1465	1131	65654	60999
32K-DM/32	136	7356	7954	1161	897	52047	48357
32K-4SA/32	338	18290	19778	2887	2230	129410	120235
16K-DM/64	225	12200	13193	1926	1487	86322	80202
16K-4SA/64	332	18011	19476	2843	2196	127437	118401

proposed technique requires no modifications to the TLB, a single number is shown. The TLB energy numbers are similarly obtained through Cacti-4. In order to model a TLB structure, we have modified the number of address bits to match the VPN length of 20 bits (4K pages within a 32-bit address space). Since the proposed technique reduces dynamic power only, the numbers for leakage power show the impact of the technique on the leakage. As can be seen from the numbers, because of the slight increase in tag size, the leakage power is slightly increased. This increase, however, is extremely small; for instance, for the 32K caches the increase is around 2%, while for the 16K caches it is around 2.5%.

We have performed our experimental study on a set of widely used multimedia benchmarks from the Mediabench [Lee et al. 1997] set. The SimpleScalar [Austin et al. 2002] toolset is used as a simulation testbed. All input and output buffers used by these applications are considered shared memory. In a multi-tasking environment, these input and output data buffers (speech/image/video frames) would usually be communicated from one process to another. For simulation purposes we have captured the address ranges of these shared memory buffers and have provided them to the simulation environment, which was appropriately modified to model the proposed heterogeneous cache tagging. Through architectural simulations, the execution statistics, including the total number of TLB/cache accesses and the number of accesses to shared memory regions, are collected. The overall power consumption is computed by summing the energy needed for all data cache accesses and address translations, accounting for the entire application program.

Table II shows the baseline D-TLB characteristics. The first row in the table contains the benchmark name. The first six applications belong to the Mediabench set of benchmarks, while the seventh is a widely used open-source *mp3* encoder. The next two rows report the energy (in  $\mu\text{J}$ ) dissipated by the 32-entry and 64-entry fully associative TLBs only, respectively. The 32-entry TLB is used in combination with a 32KB cache as in Intel XScale, while the 64-entry is used in combination with a 16KB cache as in the ARM9 architecture. The subsequent two rows show the combined energy of a 32KB cache, direct-mapped and 4-way set-associative, and a 32-entry TLB. The last two rows report the energy of the 16KB cache, direct-mapped and 4-way set-associative, and a 64-entry TLB.

Table III shows some important execution statistics for each benchmark. All the numbers reported in this table are in thousands. The total number of memory accesses is reported in the first row, while the number of accesses to shared memory regions is shown in the second row. The third row (Writes) shows the number of memory writes, while the last row reports the number of memory

Table III. Benchmark Cache Access Atatistics (x1000)

	adpcm	g721	gsm	epic	jpeg	mpeg	mp3
Access	891	48272	52199	7621	5885	341549	317333
Shared	590	295	251	2018	134	93306	4335
Writes	373	11640	11784	841	1620	22924	82971
Private Writes	4	11640	11683	547	1588	21867	80960

writes to shared memory buffers. It can be seen, for example, that for *adpcm* the shared memory accesses constitute a large part of the total number of accesses, while for *gsm* they make up only a relatively small part of all accesses. This can be easily explained by the fact that *adpcm* is not computationally intensive and most of its work processes directly its input and output stream of speech frames. *Gsm* similarly works on a stream of speech frames, however, it is significantly more computationally intensive and for each frame it performs a large number of operations and accesses to the private state of the computation.

We first evaluate the effect on the data cache of using the proposed heterogeneously-tagged cache organization for a write-back cache policy. The physical address translation uses the default D-TLB. For the memory accesses to nonshared memory regions, the overhead is in extending the cache tag with access control bits and process ID bits; no tag translation is performed for such memory accesses. When there is a cache miss or a write-back of a cache line containing a virtual tag, the D-TLB is used to obtain the physical memory address. Note that in the case of replacing a modified cache line associated with a virtual tag, two address translations would be needed: one for the address of the new data which is being brought to the cache and one for the dirty cache line that needs to be stored back to memory. In our evaluation we have taken into account this situation and the corresponding number of address translations have been performed. For potential synonym memory references, namely memory references to shared memory regions, the cache access mechanism is identical to the traditional mechanism with the tag part of the address translated through the D-TLB.

Subsequently, we include the effect of the tag and superset bits translation through the introduced adder-based physical address computation logic. For private memory accesses, the energy per access is unchanged. However, for accesses to shared memory regions, the translation overhead includes the read from the synonyms offset table (SOT) in order to obtain the superset bits offset and VPN offset. It also includes the two adders to compute the physical superset bits and the PPN. In our experiments we have included the energy consumption of the SOT tables and the two adders, the small (2–3 bit) superset bit translation adder, and the wider (20 bit) VPN translation adder. We have modeled the SOT energy through Cacti-4 by specifying a small direct-mapped cache with a data array of 8 entries, each 24-bits wide. This is achieved by modifying the address bits and bitlines accordingly. The energy for this very small SRAM array has amounted to  $0.00237nJ$ . An alternative implementation for the SOT is simply to use a set of registers, which would consume a similar amount of power. It is evident that the SOT energy is orders of magnitude smaller than the energy

Table IV. Energy Consumption of Proposed Organization for Write-Back Caches

	adpcm	g721	gsm	epic	jpeg	mpeg	mp3
Enrg(32kD)	113	3441	3713	721	512	34006	24731
Red.(32kD)	33.7/17.1	99.3/53.2	99.5/53.3	71.6/37.9	80.5/42.9	65.6/34.7	91.4/48.9
Enrg(32k4)	315	14422	15590	2450	1851	111203	96545
Red.(32k4)	33.7/6.61	99.4/21.2	99.5/21.2	72.3/15.1	80.6/16.7	67.4/14.1	92.9/19.7
Enrg(16kD)	168	2999	3226	887	582	44319	25680
Red.(16kD)	33.7/25.5	99.3/75.4	99.5/75.6	71.1/54.0	80.2/60.9	64.2/48.7	89.6/68.0
Enrg(16k4)	278	8995	9714	1818	1312	84627	63354
Red.(16k4)	33.7/16.2	99.4/50.1	99.5/50.1	72.2/36.1	80.4/40.3	67.4/33.6	92.5/46.5
Het.-Tag & Adder Tr.:							
Enrg(32kD)	66	3415	3693	551	501	26679	23586
Red.(32kD)	95.7/51.2	100.0/53.6	100.0/53.6	98.2/52.6	82.8/41.1	91.2/48.7	95.7/51.2
Enrg(32k4)	269	14399	15570	2285	1840	104361	95766
Red.(32k4)	95.7/20.3	100.0/21.3	100.0/21.3	98.2/20.9	82.8/17.5	91.3/19.4	95.8/20.4
Enrg(16kD)	57	2939	3177	476	551	25924	21859
Red.(16kD)	98.1/74.5	100.0/75.9	100.0/75.9	99.2/75.3	82.9/62.9	92.2/70.0	95.8/72.7
Enrg(16k4)	168	8939	9666	1423	1283	68322	61258
Red.(16k4)	98.1/49.4	100.0/50.4	100.0/50.4	99.2/50.0	82.9/41.6	92.3/46.4	95.9/48.3

of the caches or TLBs. We have accounted for the power consumption of both adder circuits by using the data presented in Vratonjic et al. [2005]. In that research project the authors implemented and evaluated in terms of power and performance a number of different parallel adder architectures. They used a  $0.13\mu\text{m}$  process technology to implement the adders. For our study we have used the carry-select adder (CSA), which in terms of speed is very close to the fastest adders (SCL and KS) while exhibiting an area complexity close to that of the traditional ripple-carry adder. The paper reports that a 32-bit CSA adder was measured to dissipate  $1.78pJ$  per access. Using the Cacti formula for scaling process technologies (linear correlation in gate size and quadratic in voltage) we have estimated the CSA energy for the  $0.18\mu\text{m}$  technology process of our cache, TLB, and SOT components, and subsequently scaled it down (linearly) to a 20-bit and 3-bit adders. The energy numbers that we have computed thusly for our 20-bit and 3-bit adders are  $2.37pJ$  and  $0.1pJ$ , respectively.

Table IV shows energy dissipation for the proposed methodology for write-back cache organization. The table is split into two halves. The upper half reports the total energy for the four cache/TLB organizations enabled with heterogeneous tagging and the energy reductions for these configurations compared to baseline cache and TLB organization. The energy reductions reported are in percentage form and for each benchmark we include a pair of numbers. The first number shows the energy reduction of the address translation logic only. This only includes the energy dissipated by the TLB. The second number reports the total energy reduction for the data cache and TLB. Clearly, TLB reductions are directly proportional to the number of shared memory accesses and how large of a fraction they are to the total number of accesses. As can be seen from the table, the TLB energy reductions for *adpcm* are around 33%, only since this benchmark features a large number of accesses to shared memory.

Table V. Energy Consumption of Proposed Organization for Write-Through Caches

	adpcm	g721	gsm	epic	jpeg	mpeg	mp3
Enrg(32kD)	113	4417	4693	764	622	35682	31092
Red.(32kD)	33.3/16.9	75.2/39.9	77.1/41.0	64.8/34.2	58.2/30.6	59.8/31.4	67.5/35.7
Enrg(32k4)	316	15399	16570	2494	1961	112903	102926
Red.(32k4)	33.3/6.5	75.3/15.8	77.1/16.2	65.5/13.6	58.3/12.0	61.5/12.8	68.9/14.4
Enrg(16kD)	168	5236	5472	985	834	48132	40055
Red.(16kD)	33.3/25.2	75.2/57.1	77.1/58.5	64.4/48.8	58.0/44.0	58.3/44.2	66.0/50.1
Enrg(16k4)	279	11232	11959	1918	1563	88519	77931
Red.(16k4)	33.3/16.0	75.3/37.6	77.1/38.6	65.4/32.6	58.1/28.8	61.5/30.5	68.6/34.2
Het.-Tag & Adder Tr.:							
Enrg(32kD)	66	3479	3757	554	508	26789	24000
Red.(32kD)	95.7/51.2	98.4/52.7	98.5/52.8	97.7/52.3	81.4/43.3	90.8/48.5	94.1/50.4
Enrg(32k4)	269	14462	15634	2288	1847	104472	96181
Red.(32k4)	95.7/20.3	98.4/20.9	98.5/21.0	97.8/20.8	81.4/17.2	91.0/19.3	94.2/20.0
Enrg(16kD)	58	3002	3241	479	558	26032	22267
Red.(16kD)	98.1/74.5	99.3/75.4	99.3/75.4	99.0/75.1	82.3/62.5	92.0/69.8	95.1/72.2
Enrg(16k4)	168	9003	9730	1426	1290	68433	61672
Red.(16k4)	98.1/49.4	99.3/50.0	99.4/50.0	99.0/49.9	82.3/41.3	92.1/46.3	95.2/47.9

On the other extreme is *gsm*, which achieves 99% TLB energy reduction, as it is heavily dominated by local computations which do not access the shared input/output buffers. The other benchmarks span the range of 67%–92% TLB energy reductions. The total (cache and TLB) energy reduction depends on both the TLB reduction and also on the energy complexity of the particular cache organization. The reduction for a 4-way set-associative cache tends to be smaller than that for direct-mapped caches, as the former is significantly more power consuming. It is noteworthy that the proposed heterogeneously-tagged organization does not reduce the energy dissipated by the cache; it only eliminates the need for translating the tags for most accesses to the cache, thus achieving its power reductions from the significantly reduced TLB energy. We report the total (cache and TLB) energy reductions in order to evaluate the energy impact of the proposed technique to the entire cache system.

The lower half of Table IV reports the total energy and the energy reductions for the same cache/TLB configurations, but this time using adder-based translation for the superset bits and tags of the accesses to shared memory regions. The organization of rows is identical to the upper half of the table. Since the TLB lookups for shared memory regions are replaced with the much less energy-consuming adder operation, it can be seen from the table that the address translation energy reductions are in the range of 82%–99%. This energy reduction includes the energy overhead of the SOT table and both adder circuits.

Table V shows the energy dissipation and reductions for write-through caches. For this cache organization there is a cache write-back operation for each memory-write operation. Consequently, address translation is required for each memory-write. Therefore, for the case of write-through cache organization, the proposed methodology reduces the power on reads to private memory

Table VI. Energy Consumption of Proposed Organization for Write-Through Caches with a Latched Most-Recently Written Physical Page

	adpcm	g721	gsm	epic	jpeg	mpeg	mp3
Enrg(32kD)	82	3527	3756	718	517	34036	25558
Red.(32kD)	75.1/39.9	97.2/52.1	98.5/52.8	72.0/38.2	79.6/42.4	65.5/34.6	88.3/47.1
Enrg(32k4)	284	14509	15633	2448	1855	111257	97392
Red.(32k4)	75.0/15.8	97.2/20.7	98.5/21.0	72.7/15.2	79.7/16.8	67.2/14.0	89.7/19.0
Enrg(16kD)	97	3197	3325	880	592	44363	27380
Red.(16kD)	75.0/56.9	97.2/73.8	98.5/74.8	71.6/54.3	79.3/60.2	64.1/48.6	86.8/65.9
Enrg(16k4)	208	9193	9813	1812	1321	84750	65256
Red.(16k4)	75.0/37.5	97.2/49.0	98.5/49.6	72.6/36.3	79.5/39.8	67.2/33.5	89.3/44.9
Het.-Tag & Adder Tr.:							
Enrg(32kD)	64	3421	3696	551	501	26681	23640
Red.(32kD)	98.4/52.7	99.8/53.	99.9/53.5	98.2/52.6	82.8/44.1	91.2/48.7	95.5/51.1
Enrg(32k4)	267	14404	15573	2284	1840	104364	95821
Red.(32k4)	98.4/20.9	99.8/21.2	99.9/21.3	98.2/20.9	82.8/17.5	91.3/19.4	95.6/20.3
Enrg(16kD)	55	2944	3180	476	552	25925	21907
Red.(16kD)	99.3/75.4	99.9/75.9	100.0/75.9	99.2/75.3	82.9/62.9	92.2/70.0	95.7/2.7
Enrg(16k4)	166	8945	9669	1423	1283	68326	61312
Red.(16k4)	99.3/50.0	99.9/50.3	100.0/50.4	99.2/50.0	82.9/41.6	92.3/46.4	95.8/48.2

regions only; all write references as well as references to shared memory regions would need to be translated, as they either utilize physical tags or need a physical address in order to access the lower level of the memory hierarchy. The organization of Table V is identical to the previous table. It is similarly split into two halves, where the upper half reports on the basic heterogeneously-tagged architecture, while the lower half shows the impact on the adder-based superset alignment and tag translation. Write-through caches, as expected, result in less energy savings as compared to write-back caches due to the need to perform address translation for each write operation, regardless of whether it is to shared or private memory. The energy reductions for most benchmarks are 10%–25% less than those for write-back caches.

Table VI shows the energy dissipation and reductions for a write-through cache with the introduction of physical page latch (PPL) caching of the last address translation as described in Section 4. For this architecture, the physical address for the last write memory operation is stored into a special output address latch, accompanied by the VPN, which serves as a tag. Thus, if two subsequent writes access the same physical page frame, no address translation need be performed for the writes after the first one. The translated physical address is latched and reused for any subsequent writes to the same memory page. The organization of this table is identical to the previous two tables. It is clear from the data in this table that PPL optimization helps significantly for write-through caches and brings back the energy reductions close those for write-back caches. Interestingly, for the *epic* benchmark, write-through caches with PPL optimization achieve slightly better energy reductions than for the write-back cache. This can be attributed to a series of write-backs to the same memory page that can benefit from the PPL and avoid address translations.

## 9. CONCLUSION

In this article we have proposed a novel cache architecture for low-power embedded processors with virtual memory support. Application knowledge regarding the nature of data memory references is used to distinguish references to private data, and consequently to handle them in a more energy-efficient way. For private memory references, virtual tags are stored and only used when accessing the cache, thus completely eliminating the power-consuming address translation step. References to shared data regions, however, utilize physical tags and superset bits in order to avoid cache synonym problems. Furthermore, due to the typical size and allocation of such shared buffers, the address translation for them can be performed through a simple arithmetic operation instead of associative TLB lookup, which additionally reduces the total energy. The proposed hardware architecture efficiently captures the information regarding memory references in a reprogrammable way, and does not introduce significant area and performance overheads. We have demonstrated the significant power benefits of the proposed heterogeneously-tagged cache architecture for a set of widely used embedded applications.

## REFERENCES

- ARM, LTD. 1995. *ARM920T Technical Reference Manual*. ARM, Ltd.
- AUSTIN, T., LARSON, E., AND ERNST, D. 2002. SimpleScalar: An infrastructure for computer system modeling. *IEEE Comput.* 35, 2 (Feb.), 59–67.
- BENINI, L., MACII, A., AND PONCINO, M. 2003. Energy-Aware design of embedded memories: A survey of technologies, architectures, and optimization techniques. *ACM Trans. Embed. Comput. Syst.* 2, 1, 5–32.
- BENINI, L., MENICHELLI, F., AND OLIVIERI, M. 2004. A class of code compression schemes for reducing power consumption in embedded microprocessor systems. *IEEE Trans. Comput.* 53, 4, 467–482.
- CALDER, B., KRINTZ, C., JOHN, S., AND AUSTIN, T. 1998. Cache-Conscious data placement. In *Proceedings of the 8th International Conference on Architectural Support for Programming Languages and Operating Systems*, 139–149.
- CEKLEOV, M. AND DUBOIS, M. 1997. Virtual-Address caches. Part 1: Problems and solutions in uniprocessors. *IEEE Micro.* 17, 5 (Sept.), 64–71.
- CHILIMBI, T. M., HILL, M. D., AND LARUS, J. R. 1999. Cache-Conscious structure layout. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 1–12.
- EKMAN, M., DAHLGREN, F., AND STENSTROM, P. 2002. TLB and snoop energy-reduction using virtual caches in low-power chip-microprocessors. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED)*, 243–246.
- INTEL CORP. 2007. Intel XScale microarchitecture. Intel Corporation.
- JACOB, B. AND MUDGE, T. 1998. Virtual memory: Issues of implementation. *IEEE Comput.* 31, 6 (Jun.), 33–43.
- JUAN, T., LANG, T., AND NAVARRO, J. J. 1997. Reducing TLB power requirements. In *Proceedings of the International Symposium on Low-Power Electronics and Design (ISLPED)*, 196–201.
- KADAYIF, I., NATH, P., KANDEMIR, M., AND SIVASUBRAMANIAM, A. 2004. Compiler-Directed physical address generation for reducing DTLB power. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 161–168.
- KADAYIF, I., SIVASUBRAMANIAM, A., KANDEMIR, M., KANDIRAJU, G., AND CHEN, G. 2002. Generating physical addresses directly for saving instruction TLB energy. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 185.

- KANDEMIR, M., KADAYIF, I., AND CHEN, G. 2004. Compiler-Directed code restructuring for reducing data TLB energy. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES and ISSS)*, 98–103.
- KIM, J., MIN, S., JEON, S., AHN, B., JEONG, D., AND KIM, C. 1995. U-Cache: A cost-effective solution to synonym problem. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 243–252.
- KULKARNI, C., GHEZ, C., MIRANDA, M., CATTHOOR, F., AND MAN, H. D. 2005. Cache conscious data layout organization for conflict miss reduction in embedded multimedia applications. *IEEE Trans. Comput.* 54, 1, 76–81.
- LEE, C., POTKONJAK, M., AND MANGHONE-SMITH, W. H. 1997. Mediabench: A tool for evaluating and synthesizing multimedia and communications systems. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 330–335.
- LEE, J. H., LEE, J. S., JEONG, S., AND KIM, S. 2001. A banked-promotion TLB for high performance and low power. In *Proceedings of the IEEE International Conference on Computer Design (ICCD)*, 118–123.
- MIDDHA, B., SIMPSON, M., AND BARUA, R. 2005. MTSS: Multi task stack sharing for embedded systems. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, New York, 191–201.
- PANDA, P. R., CATTHOOR, F., DUTT, N. D., DANCKAERT, K., BROCKMEYER, E., KULKARNI, C., VANDERCAPPELLE, A., AND KJELDSBERG, P. G. 2001. Data and memory optimization techniques for embedded systems. *ACM Trans. Des. Autom. Electron. Syst.* 6, 2, 149–206.
- PETROV, P., TRACY, D., AND ORALOGLU, A. 2005. Energy-Efficient physically tagged caches for embedded processors with virtual memory. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*, 17–22.
- QIU, X. AND DUBOIS, M. 2001. Towards virtually-addressed memory hierarchies. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA)*, 51–62.
- SIMPSON, M., MIDDHA, B., AND BARUA, R. 2005. Segment protection for embedded systems using run-time checks. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, New York, 66–77.
- TARJAN, D., THOZIYOOR, S., AND JOUPPI, N. 2006. Cacti 4.0: An integrated cache timing, power and area model. Tech. Rep., HP Laboratories, Palo Alto, California, June.
- VRATONJIC, M., ZEYDEL, B., AND OKLOBDZIJA, V. 2005. Low- and ultra low-power arithmetic units: Design and comparison. In *Proceedings of the International Conference on Computer Design (ICCD)*, 249–252.
- WOO, D., GHOSH, M., OZER, E., BILES, S., AND LEE, H.-H. 2006. Reducing energy of virtual cache synonym lookup using bloom filters. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES)*, 179–189.

Received April 2007; accepted December 2007