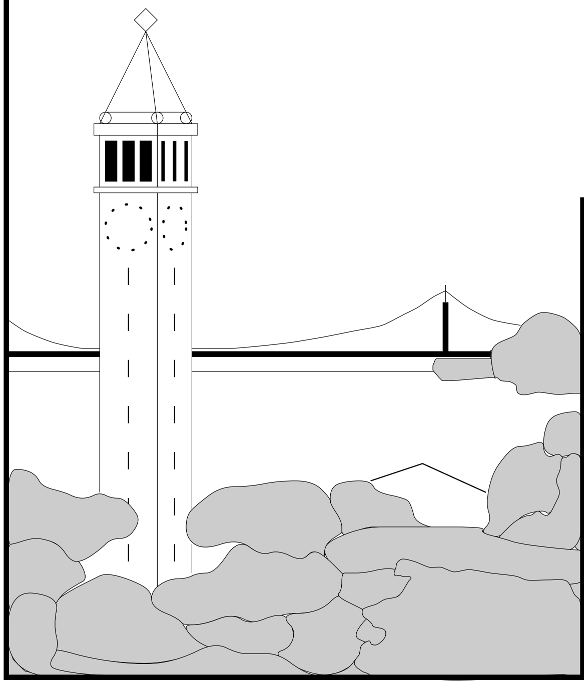
Implementation Issues in Modern Cache Memory

Jih-Kwon Peir
Windsor W. Hsu
Alan Jay Smith



Report No. UCB/CSD-98-1023

November 1998

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Implementation Issues in Modern Cache Memory

Jih-Kwon Peir Member, IEEE, Windsor W. Hsu Student Member, IEEE, and Alan Jay Smith, Fellow, IEEE

Abstract— **As the performance gap between processors and main memory continues to widen, increasingly aggressive implementations of cache memories are needed to bridge the gap. In this paper, we consider some of the issues that are involved in the implementation of highly optimized cache memories and survey the techniques that can be used to help achieve the increasingly stringent design targets and constraints of modern processors. In particular, we consider techniques that enable the cache to be accessed quickly and still achieve a good hit ratio. We also consider issues such as area cost and bandwidth requirements. Trace-driven simulations of a TPC-C-like workload and selected applications from the SPEC95 benchmark suite are used in the paper to compare the performance of some of the techniques.**

Keywords— **Cache memory, cache access mechanism, address translation, cache area and bandwidth.**

I. Introduction

Cache memories are small fast memories used to temporarily hold the contents of portions of main memory that are (believed to be) likely to be used. The basic concepts of using cache memories to improve processor performance have been well studied and understood. See for example [66], [67]. Today, caches have become an integral part of all processors. However, as the performance gap between processor and main memory continues to widen, increasingly optimized implementations of caches are needed. In this paper, we consider some of the issues in implementing aggressive cache memories and survey the techniques that are available to help meet the increasingly rigorous design targets and constraints of modern processors.

The ability of caches to bridge the performance gap is determined by two primary factors — the time needed to retrieve data from the cache and the fraction of memory references that can be satisfied by the cache. These two factors are commonly referred to as *access (hit) time* and *hit ratio* respectively [66], [67]. The access time is especially critical for first level (L_1) caches because a longer access time typically implies a slower processor clock rate and/or more pipeline stages. In order to minimize access time, cache access should be triggered as soon as the address of the memory reference is available. However, the virtual memory architecture, by imposing a potentially many-to-one mapping of virtual to physical addresses, places constraints on how this can be achieved. The hit ratio is also critical, both because misses impose delays, and because off-chip bandwidth, especially when there is a shared bus, is a very limited resource.

A. Cache Fundamentals

CPU caches are normally associative memories; the key is a (real or virtual) memory address. Because of the difficulties of building highly associative memories, most CPU cache memories are organized as two-dimensional arrays. The first dimension is the *set*, and the second dimension is the *set associativity*. The *set ID* is determined by a function of the address bits of the memory request. The *line ID* within a set is determined by matching the address tags in the target set with the reference address. Caches with set associativity of one are commonly referred to as *direct-mapped caches* while caches with set associativity greater than one are referred to as *set-associative* caches. If there is only one set, the cache is called *fully-associative.*

Each cache entry consists of some data, and a tag that identifies the main memory address of that data. Whether a memory request can be satisfied by the cache is determined by comparing the requested address with the address tags in the tag array. There are thus two parts to a cache access. One is to access the tag array and perform the tag comparison to determine if the data is in the cache. The other is to access the data array to bring out the requested data. For a set-associative cache, the results of the tag comparison are used to select the requested line from within the set driven out of the data array.

In most computers, caches are accessed on the real memory address, whereas the ALU generates the virtual memory address. To speed up the translation process (and to not have to access the main memory page tables), another cache, one for the page tables, is used. The page table cache is most commonly known as the *Translation Lookaside Buffer (TLB)* [4]. The need to translate the virtual address to the real address may further delay cache access.

B. Performance Evaluation

Trace-driven simulation is the standard methodology for the study and evaluation of cache memory design; some trace driven simulation results appear later in this paper. Trace driven simulation is a form of event driven simulation, in which the events consist of those collected from a real system rather than those generated randomly. For cache memory studies, the traces consist of sequences of memory reference addresses. Traces may be collected by a variety of hardware and/or software methods. A comprehensive discussion of this technique and its strengths and weaknesses is in [68].

This work is supported in part by the National Science Foundation under grants MIP-9624498, MIP-9116578 and CCR-9117028, by the State of California under the MICRO program, and by IBM, Intel, Sun Microsystems, Fujitsu Microelectronics, Toshiba America, Cirrus, Sony Research Laboratories, Microsoft and Quantum.

Jih-Kwon Peir is with Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL 32611, peir@cise.ufl.edu.

Windsor W. Hsu and Alan Jay Smith are with Computer Science Division, Department of Electrical Engineering and Computer Sciences, University of California Berkeley, CA 94720, {windsorh,smith}@cs.berkeley.edu. Windsor W. Hsu is also with Computer Science Department, IBM Almaden Research Center, San Jose, CA 95120, windsor@almaden.ibm.com.

Among the traces used in this paper is a trace of the server side of a workload similar to the Transaction Processing Performance Council's benchmark C (TPC-C). This was collected with a software tracing tool on an IBM RISC System/6000 system running AIX. Our other traces consist of five integer-intensive programs *(Compress, Gcc, Go, Li, and Vortex)* and three floating-point intensive applications *(Apsi, Su2cor, and Turb3d)* from the SPEC95 benchmark suite. These traces were collected with the Shade tool on SUN Sparc Systems running Solaris [72]. In our simulations, the first 50 million instructions of each trace are used for cache warm up purposes.

II. Cache Implementation Issues

A. Addressing Constraint

In order to minimize effective memory access time, the access should be triggered as soon as the effective address of the memory reference becomes available. In most computers, however, caches are addressed, as noted above, with the physical address, and thus there is a delay for translation. This delay can often be partially overlapped, but it is hard to avoid completely. Virtually addressed caches do not require address translation during cache access, but the fact that multiple virtual pages may be mapped to the same physical page greatly complicates their design.

A.1 Physical Address Cache

As described earlier, caches are organized as 2-dimensional arrays and are accessed in a two phase cycle. In the first phase, a cache set is selected by using a portion of the address known as the *index* bits. In the second phase, the remaining part of the address is used to make a further selection from within this cache set to yield either a cache miss determination or the requested data. This two phase access cycle means that only a portion of the address needs to be available at the onset of cache access.

There are various techniques that exploit this two phase access cycle to enable a physically addressed cache to be accessed without requiring an extra address translation cycle. Because only the page number bits need to be translated, the untranslated bits are immediately available to start the access. For example, if the untranslated bits can be used to select the set, then for a 2^K page size, we can overlap set selection with translation, and then do a J-way associative search among the elements of the set, for a cache of size $J \times 2^K$. However, there is a practical limit to this approach because increasing the set-associativity provides only a diminishing return on cache hit ratio but adds hardware complexity and adversely impacts the access time [23].

Rather than limiting the index bits to within the page offset, another approach is to increase the number of address bits available before address translation. A straightforward way to achieve this is to increase the page size [27]. For many computer architectures, however, the page size is typically fixed and enlarging it would require substantial changes to both the architecture and the system software. In addition, it may lead to more memory fragmentation. In some newer architectures, however, the page size may be variable. A technique that can be used to increase the number of address bits available before address translation is to restrict the virtual to physical page mapping so that the low-order bits of the physical and virtual page numbers are identical [25]. This amounts to implementing a set-associative rather than fully-associative main memory. This technique, which has also been referred to as page coloring, may increase the number of page faults [13], although with associativities of 8 and larger (in the main memory), the effect is likely to be negligible.

Another way to make more address bits available before address translation is to predict the additional address bits. An example of a good predictor is the content of the base register that is used to compute the effective address [24]. At the address generation stage, the low-order page address bits in the base register are used to index into a history-based prediction table to obtain the needed physical address bits. The base register content is an accurate predictor for the needed physical index bits because the displacement for computing the effective address is usually small and recently used pages tend to be re-referenced, i.e. there is locality of reference. Results reported in [24] show that a 128-entry direct-mapped prediction table can achieve accuracy exceeding 98% for commercial workloads. Similar prediction schemes are presented in [13], [6]. The MIPS R6000 [74] implements a TLB-slice which is essentially a predictor based on the low-order virtual page address.

A.2 Virtual Address Cache

Instead of using bits from the virtual address as a predictor for the physical address, a different approach is to use the virtual address to directly access the cache [55], [66], [83]. This avoids the delay for translation; other TLB functions such as page protection, update, and reference information can be resolved in parallel with cache access [84]. In addition, all the addresses must be tagged with an address space identifier or else the cache must be purged on every task switch [66].

The most serious drawback of the virtual address cache is that multiple virtual addresses may be mapped to the same physical address, i.e. *synonyms* may occur. Synonyms occur when code or data is shared. In addition, on some systems, the supervisor and its data structures exist in all the address spaces. Thus, in a virtual address cache, the absence of an address tag does not imply a miss because the data may be located elsewhere in the cache under a different virtual address. This is known as the *synonym problem* [5], [60], [66].

The usual approach to handling synonyms is to prevent them from being present in the cache at the same time. Such an approach has the nice property that it adds no overhead to the frequent case of a cache hit. The only penalty comes in the less frequent case of a cache miss during which the cache has to be searched for any synonym entry. The way to detect synonyms is to map the requested virtual address into its physical address and see if any of the virtual addresses in the cache maps into the same physical address. In general, a reverse translation buffer (*RTB*) is needed in order for this approach to be feasible [66]. Such a buffer is accessed with the physical address and indicates the cache line that is associated with that physical address [5], [60], [66], [20]. This in effect maintains a physical tag for each and every line present in the virtual address cache, meaning that a cache line has to be invalidated whenever its physical tag is replaced.

One way to reduce the complexity in handling synonyms is to make sure that the index bits used to select the cache set are the same for both the physical and virtual addresses [25], [10], [66], [74]. In this case, the reverse mapping of cache lines can be implemented by simply associating both the virtual and physical address tags with each cache line. However, as mentioned earlier, this restricted page mapping may result in more page faults [13]. Since the index bits are the most critical for cache access, a hybrid approach is to use virtual indices with physical tags. In this case, synonyms are mapped to a small number of sets determined by the number of index bits beyond the page offset. Software approaches have also been proposed to eliminate synonyms. See for instance [83], [35], [80].

A.3 Interesting Implementations

Typically, a combination of the above techniques and other engineering solutions are used to circumvent the addressing constraint. For instance, the IBM-3090 has a unified 128KB L_1 cache with physical tags [45], [75]. The tag array is organized as 32-way set-associative to limit the index bits to within the 4-KB page offset. The data array has a virtually-indexed 4-way set-associative design to avoid fetching all 32 double words out of the data array simultaneously. As a result, there is a primary set indexed by including the low-order 3 bits (bits 17-19) of the virtual page address and seven synonym sets indexed by using other combinations of the 3 bits. In case of a synonym hit, the correct data location is available from the tag comparison and the data can be fetched out in the next cycle. The synonym line is moved to the primary set afterwards. This solution becomes prohibitively expensive with larger caches because of the higher set associativity of the tag array.

The recently announced IBM S/390-G4 CMOS mainframe processor has a 64KB 4-way set-associative unified L_1 cache [79]. With 4KB pages, two of the address bits (18:19) needed to index the cache are subject to translation. These two address bits are predicted prior to the cache access cycle [24], [79]. An Absolute Address History Table (AAHT) is used to maintain the recent associations between the physical address bits (18:19) and the values in the base and the index registers. At the address generation cycle, AAHT is accessed to obtain the predicted bits (18:19) for accessing the cache in the following cycle.

The UltraSPARC-IIi [73] has 16KB L_1 instruction and data caches. The instruction cache is 2-way set-associative, physically indexed and tagged. However, for fast access time, the data cache is direct-mapped with virtual index bits and physical tags. The approach taken in the UltraSPARC-IIi is to rely on software to force synonym lines to have the same index bits. In cases where the synonyms cannot be mapped to the same index bits, the software either flushes the data cache or turns off caching for the synonym pages.

In a two-level cache design, a natural way of incorporating both the virtual and physical cache designs is to have a first-level virtual cache for fast access and a second-level physical cache for resolving synonyms [77]. In this case, the tag array of the L_2 cache acts as a means of locating L_1 cache lines via their physical addresses. For instance, the MIPS R10000 [85] has 32KB 2-way set-associative L_1 instruction and data caches. Both caches are virtually indexed and physically tagged. The L_2 cache is physically indexed and tagged to detect synonyms and to handle cache coherence requests. In order to support a 4 KB page size, two virtual bits are needed to index the L_1 cache. The L_2 tag array maintains these bits to enable L_1 cache lookup.

B. Access Time and Miss Ratio Targets

The performance of a cache is determined both by the fraction of memory requests it can satisfy (*hit/miss ratio*) and the speed at which it can satisfy them (*access time*). There have been numerous studies on cache hit/miss ratios with respect to the cache and line sizes, and the set associativity [66], [67], [23]. In general, larger caches with higher set associativity have higher hit ratios. Unfortunately, such cache topologies tend to incur longer access times, because in a set-associative cache, after the tags for the lines in the set are read out, a comparison is performed (in parallel) and then a mux is used to select the data corresponding to the matching tag. For instance, results from the on-chip timing model, *cacti*, suggest that a 16KB direct-mapped cache with 16-byte lines is about 20% faster than a similar 2-way set associative cache [82]. As addresses become longer, the tag comparisons are slower. A general strategy for simultaneously achieving fast access time and high hit ratio is to have a fast and a slow access path. The fast path achieves fast access time for the majority of memory references while the slow path boosts the effective hit ratio. We refer to these two cases as the *fast access* and the *slow access* respectively. Techniques for achieving fast cache access while maintaining high hit ratios can be broadly classified into the following four categories:

- *Decoupled cache*: The data array access and line selection are carried out independently of the tag array access and comparison so as to circumvent the delay imbalance between the paths through the tag and data arrays.
- *Multiple-access cache*: A direct-mapped cache is accessed sequentially more than once in order to achieve the access time of a direct-mapped cache for the fast access and the hit ratio of a set-associative cache as a whole.
- *Augmented cache*: A direct-mapped cache is augmented with a small fully-associative cache to improve the overall hit ratio without lengthening the access time.
- *Multi-level cache*: A small and fast upstream cache is used for the fast access while one or more larger and slower downstream caches are used to capture the fast-access misses with minimal penalties.

B.1 Decoupled Caches

The defining characteristic of decoupled caches is that the data can be fetched without any dependency on the results of tag comparison. This is trivially true in the direct-mapped case because in such a cache, there is only one cache line in each cache set. However, it tends to have an inferior hit ratio due to conflict misses [22]. Direct-mapped designs also have a cost advantage over set-associative designs because they require fewer comparators and less bandwidth out of the data array. This latter fact makes the direct-mapped design especially suitable for large off-chip caches that are implemented using commodity SRAMs with limited pin bandwidth [22], [53].

For a set-associative cache, the line ID has to be known before the requested data can be delivered from the cache. In the straightforward implementation, the line ID is obtained from the results of the tag comparison. However, this dependency on the tag comparison tends to be on the critical path. A general strategy to circumvent this bottleneck is to predict the line ID. For instance, the Most-Recently-Used (MRU) line selection scheme predicts that the requested cache line is the most recently used line in the target set [69]. The MRU entries of a n-way set-associative cache is essentially equivalent to a direct-mapped cache of $1/n$ the size. Depending on the locality of reference, this simple MRU prediction scheme can be very effective.

A generalization of this technique is to predict the line ID using a history table [42]. The history table is able to record the line IDs for *more* recently referenced cache lines beyond the MRU lines to increase the prediction accuracy. As the size of the history table is increased for higher prediction accuracy, the history table tends to become sparse. The PowerPC 620 uses a clever implementation of history table prediction that avoids building a large table [41], [37]. The PowerPC 620 has 32KB, 8-way "semi-associative" instruction and data caches. The 8 lines in each set are content-addressable based on the 8 low-order virtual bits (bits 44-51 in a 64-bit address) of the address tag. This is equivalent to determining the line ID by matching only a portion (8 bits) of the address tag. The actual cache hit/miss is verified through the normal tag path. Matching the 8 low-order tag bits is essentially the same as implementing a history table that can record 256 line IDs per cache set. However, in order to avoid any ambiguity in the content-addressable access, the partial tag must be unique among all the lines in each cache set. In other words, this design has only the fast access and thus may adversely affect the cache hit ratio as the name "semi-associative" implies.

Figure 1 shows the prediction accuracy of the MRU, history table, and partial-tag methods for the L_1 data cache. In this set of simulations, we consider 16 KB 4-way set-associative caches with 32-byte lines. We assume that the number of entries in the history table is equal to the total number of cache lines, and thus for each cache set, 4 line ID predictions can be recorded and indexed by the 2 low-order tag bits. We also simulate the partial-tag scheme which matches either 6 or 8 lower tag bits to predict the line ID. This is similar to using a history table that records

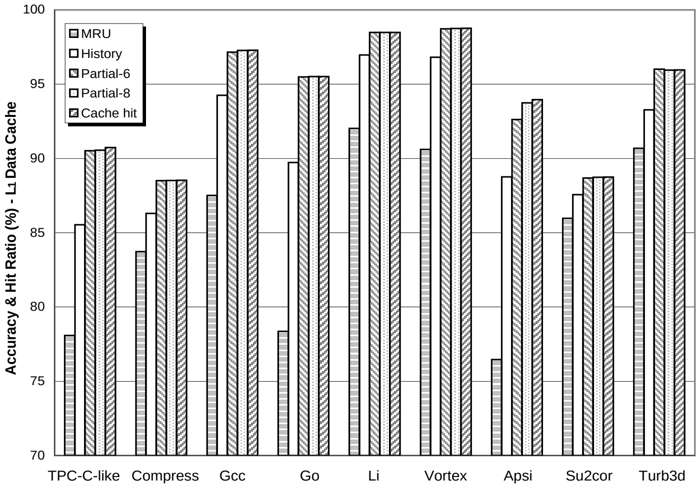


Fig. 1. Comparison of the Three Line-ID Prediction Methods.

64 and 256 line IDs respectively for each cache set.

The results are generally as expected. For example, the history-table scheme has a much higher prediction accuracy than the simple MRU scheme for all the simulated applications. Furthermore, the partial-tag matching technique is much more accurate than the history table. A partial tag of 6 bits is able to identify almost all the cache hits. This is due to the fact that within the working set of these applications the high-order tag bits tend to change infrequently. As a result, there are very few collisions among these partial tag bits for the lines located in each set.

The history table prediction scheme uses the low-order address bits to predict the line ID. In the special case of a 2-way set-associative cache, there is an efficient implementation known as the difference-bit cache that effectively uses all the address bits without requiring a huge history table [30]. In a 2-way set-associative cache, the address tags in the same set must differ by at least one bit. For each set, the difference-bit cache keeps track of the position of the least-significant bit that the two tags differ and the value of this bit in the first tag. The access time of this scheme depends on the delay in accessing the recorded information, decoding the difference-bit position and comparing the difference bits.

Recall that in the IBM-3090, the address synonym problem is resolved by decoupling the cache tag and data arrays. The tag array has a high set-associativity in order to confine the index bits within the page offset. The data array has a lower set-associativity and is accessed by some virtual address bits. This scheme can also be applied to shorten the cache access time. For instance, in the Direct-mapped Access Set-associative Check (DASC) cache [63], the data array is direct-mapped to allow the data to be fetched out of the cache without waiting for the results of tag comparison. In case of a miss to the direct-mapped location, the set-associative tag array is able to identify cache hits to other locations within the set. To increase the proportion of fast access hits, recently used cache lines are swapped into the direct-mapped locations.

The decoupled caches described so far achieve fast cache access by allowing the data to be fetched out of the cache without waiting for the full tag comparison. To further improve access time, the processor pipeline may start consuming the data speculatively once they become available. One obvious difficulty in supporting such optimistic use of cache data is that the CPU must be able to back out of execution begun with the incorrect data. A straightforward way to avoid the speculative use of data is to perform tag comparison on only a subset of the tag array during the fast access. The path balancing technique proposed in [49] combines a small line ID history table with a subset of the cache tag array into a path balance table (PBT). The history table is used to predict the line ID to avoid the late-select bottleneck while the subset of the cache tag array is used to determine whether the prediction is correct. Together, they non-speculatively improve the cache access time. A generalized scheme to further improve the fast access hit ratio is presented in [50].

B.2 Multiple-Access Caches

A multiple-access cache is essentially a direct-mapped cache that may be accessed more than once (usually twice), each time with a different mapping (hash) function, to satisfy a memory request. A fast access time is achieved when the requested data is located at the *primary* location, i.e. the location determined by the primary hash function. When a miss occurs, the cache is accessed again by an alternative or secondary hash function. A small penalty is paid if the data is located at the *secondary* location. Data swapping between the primary and secondary locations is necessary to increase the hit ratio to the primary location. In a multiple-access cache, the LRU replacement scheme becomes more complicated when the requested data is not found in either location because the primary/secondary locations for one reference could be reversed for another reference. In addition, data swapping requires extra cache ports to avoid contention with normal cache access.

In [2], a simple rehashing function based on flipping the highest-order index bit is used. Upon a hit to a secondary location, the lines located in the primary and secondary locations are swapped. When a cache miss occurs, the requested line is fetched into the primary location, the line in the primary location is moved to the secondary location, and the line in the secondary location is evicted. Although this scheme appears similar to a 2-way set-associative LRU cache, it has inferior hit ratio. This is because the primary location for a reference may be the secondary location for another reference and vice versa. In other words, this scheme does not accurately maintain the LRU information between the primary and secondary locations.

The column-associative cache [3] improves the replacement algorithm by maintaining a *rehash bit* for each line to indicate whether it has been accessed using the alternative hash function. When a primary access misses a line with the rehash bit turned on, there is no need to search the secondary location and the line in the primary location is simply replaced. The rehash bit helps to identify the correct primary/secondary sequence. However, the MRU/LRU information is still missing when both locations are used as the primary location. This deficiency can be corrected by using the rehash bit as a LRU bit to indicate whether the primary or secondary location has been more recently referenced. The search of the secondary location is needed only when the primary location has been more recently accessed. This improved scheme not only matches precisely the 2-way set-associative cache hit ratios, but also eliminates unnecessary searches to the secondary location [14].

Because of locality, memory references are not uniformly distributed across the sets of a cache. This skew reduces the effectiveness of a cache because it results in the caching of a considerable number of less-recently-used lines, which are less likely to be re-referenced before they are replaced. Results in [51] show that during the execution of a TPC-C-like workload, the direct-mapped data cache may contain more than 40% such less-recently used lines and the hit ratio to these lines is typically less than 1%.

The group-associative cache [51] dynamically identifies the underutilized cache frames in a direct-mapped cache and effectively uses them to store the data that are more likely to be re-referenced. In this technique, a Set-Reference History Table (SHT) is used to track the cache lines that have been referenced recently. When a cache miss occurs and the line being replaced has been referenced recently, it is moved into an alternate location within the cache. The alternate location is selected from among those that have not been accessed in the recent past. A small out-of-position directory (OUT) is used to keep track of the more-recently used lines that have been so displaced. In this design, the possible locations that a line can reside in is not predetermined as is the case in a set-associative cache. Instead, the cache is dynamically partitioned into groups of cache lines with the same direct-mapped index bits. The total number of groups and the individual group associativity are able to adapt to the reference pattern to more closely approximate the global LRU scheme.

One major disadvantage of the multiple-access cache is the need to swap data between the primary and secondary locations. This need can be eliminated by dynamically determining the primary set. For instance, an MRU bit can be maintained for each pair of primary and secondary locations to indicate the location that should be probed first [31]. A concern with this approach is the delay in obtaining the MRU information. Unlike the MRU line prediction scheme where the ID of the MRU line is needed only when the correct line is to be selected, the MRU bit in the multiple-access cache must be available before the cache is accessed. In addition, as is the case with the MRU line prediction scheme, the hit ratio to the primary location may suffer due to the fact that there is only one MRU line in each set. In [12], the MRU-bit scheme is extended by using a steering-bit table to indicate the location where the cache access should probe first. Such a table is equivalent to the line ID history table and can be many times bigger than the number of sets to improve the accuracy of accessing the primary location.

The multiple-access scheme based on hash-rehash functions has been applied to manage the page table access in the PowerPC architecture [43]. A variation of this scheme has also been proposed for unifying the L_1 instruction and data caches [15].

B.3 Augmented Caches

As discussed above, the direct-mapped cache has a fast access time but a poor hit ratio due to conflict misses. The augmented cache approach attempts to improve the overall cache hit ratio by augmenting the direct-mapped cache with a small fully-associative cache. Both caches are accessed in parallel and due to the small size of the fully-associative cache, an overall access time that is comparable to that of the direct-mapped cache can be achieved. The contents of the two caches are normally disjoint so as to improve the overall hit ratio.

Several variations of this general approach have been proposed. They differ primarily in the way they use and manage the two caches. The *victim cache* is based on the idea that when a line is replaced in the direct-mapped cache, it should be moved into the fully-associative cache in case it will be re-referenced again shortly [28]. In other words, the fully-associative cache is used to hold the victims of replacement in the direct-mapped cache so as to convert some of the conflict misses into hits in the victim cache.

Another way to use the fully-associative cache is based on the observation that very recently used data have a strong tendency to be reused again shortly. Therefore, they should be kept in the fully-associative cache to ensure that they will not be quickly replaced. The *assist cache* as implemented in the HP-PA7200 is one such example [34]. The L_1 cache of the PA7200 consists of a small fully-associative on-chip assist cache and a large direct-mapped off-chip cache. The assist cache is managed as a simple FIFO buffer. The cache line, upon a miss, is first moved into the assist cache; it is moved into the off-chip direct-mapped cache upon replacement from the assist cache. In contrast to the victim cache which is placed "after" the direct-mapped cache, the assist cache is placed "before" the direct-mapped cache to ensure that the recently used data will not be susceptible to conflict misses.

The third strategy attempts to reduce conflict misses by using the direct-mapped cache solely for holding hot data, i.e. data which are very likely to be reused again. The data which are less likely to be reused are prevented from entering the direct-mapped cache to replace the hot data and cause conflict misses. Instead, they are placed in the small fully-associative cache. The decision of where to cache a piece of data can be made in several ways. In the PA7200, the data always goes to the assist cache first but the load and store instructions can carry a "use-once hint" inserted by software to indicate that the data will be used only once. When such data are replaced from the assist cache, they will bypass the off-chip cache.

Instead of relying on software hints of usage patterns, another approach to allocating data to the two caches is to keep track of the past reference behavior. For instance, a reference bit can be maintained for each word in each cache line in the direct-mapped L_1 cache to record whether the line exhibits temporal locality [56]. In this scheme, temporal locality is defined as repeated accesses to any word in a cache line. When a word is accessed more than once, the corresponding reference bit is set. The line will be loaded to the direct-mapped cache upon a miss if it exhibits temporal locality during the last time that the line is in the direct-mapped cache. Otherwise, it will be moved into the fully-associative cache which is referred to as the Non-Temporal (NT) buffer. Whenever a line is replaced from the direct-mapped cache the locality indicator is saved at the L_2 entry where the replaced line is located.

In [26], a *Memory Access Table (MAT)* is used to record reference behavior at the granularity of memory blocks that are several times bigger than the cache line. Each MAT entry is associated with a memory block and contains a counter to keep track of the frequency of access to that particular block. This counter is decremented when a cache line in the block is the target for replacement. When a miss occurs in the direct-mapped cache, the counter of the memory block where the requested line is located is compared with the counter of the memory block where the replaced line is located. The replacement happens only when the requested block counter has a larger value. Otherwise, the requested line is moved into the fully-associative cache which is called the By-Pass (BP) buffer.

Figure 2 compares the hit ratio of the direct-mapped cache and several augmented and multiple-access caches including the victim cache, assist cache, BP-buffer, NT-buffer, column-associative cache, and group-associative cache. In this set of simulations, we assume that the small fully-associative cache contains 32 lines. The L_1/L_2 cache configurations are the same as those used in Figure 1 except that the L_1 caches are direct-mapped. For the BP-buffer scheme, we use a 1K-entry MAT with 1KB memory blocks. The rehash-bit is used in the column-associative cache. For the group-associative cache, we assume that the SHT can record three-eighths of the more-recently-used lines and the OUT directory can locate a quarter of the cache lines. In general, except for the NT-buffer scheme, all the other caching schemes have overall hit ratio that is higher than that of the direct-mapped cache. Among these schemes, the group-associative shows superior overall hit ratio for all the workloads. The victim cache has the highest primary hit ratio, which is essentially that of the direct-mapped cache. All the other schemes suffer some adverse impact on the primary hit ratio, especially the assist cache, BP-buffer and NT-buffer. Because the group-associative cache has the most flexible alternative locations, it suffers the least impact. The column-associative cache shows overall hit ratio comparable to that of the victim cache. The assist cache with its simple FIFO design performs reasonably well, especially for the floating-point intensive applications.

As a whole, the results of selectively allowing cache lines to bypass the direct-mapped cache based on their past reference behavior do not look encouraging. The hit ratio of the NT-buffer scheme is markedly worse than that of the direct-mapped cache for the TPC-C-like workload and

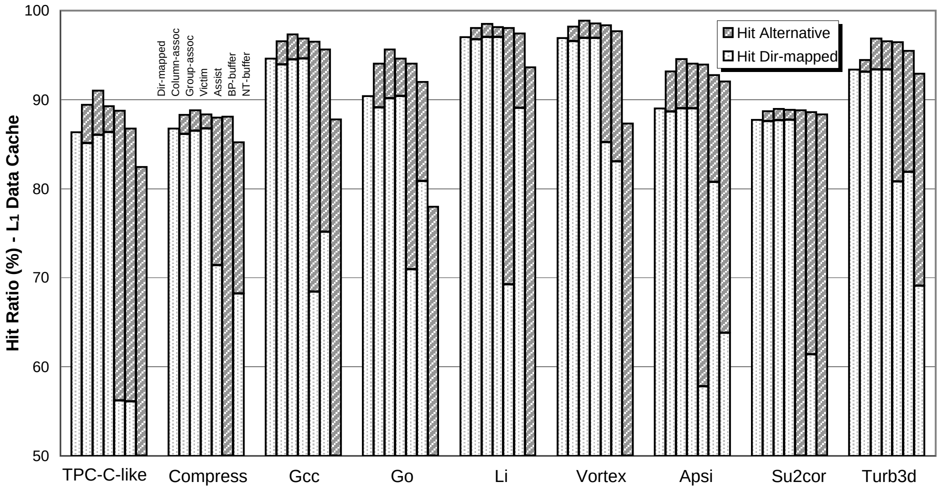


Fig. 2. Comparison of Multiple-access and Augmented Caches.

L1/L2 Set-associativity		L1 Data Miss Ratio (TPC-C-like)	16/128	16/256	16/512	16/1024
	Inclusion	1w/1w	14.38	13.91	13.78	13.70
		1w/4w	13.75	13.65	13.64	13.64
		4w/1w	10.65	9.85	9.70	9.62
		4w/4w	9.58	9.31	9.27	9.26
	No Incl.	1w/-	13.64	13.64	13.64	13.64
		4w/-	9.26	9.26	9.26	9.26

(Column heads 16/128 to 16/1024 are under the spanning head "L1/L2 Size (KB)".)

Fig. 3. Effect of Maintaining Inclusion Property.

the five integer intensive programs. Even the BP-buffer scheme, which maintains counters to track reference patterns, has slightly lower overall hit ratios than the other four schemes for some of the applications. These results are a reflection of the fact that, because of locality of reference, it is generally much easier to predict the memory locations that are likely to be used than those that are not.

B.4 Multi-level Caches

The organization and performance of multi-level caches have been studied extensively [70], [17], [7], [8], [53]. In order to overcome the growing performance gap between processor and DRAM and still satisfy chip area constraints, most of today's processors have multiple levels of cache consisting of small and fast on-chip (on-die) caches and larger and slower off-chip (off-die) caches. The access time, hit/miss ratio and chip area tradeoffs of two-level on-chip caches are discussed in [29]. The results suggest that when there is sufficient chip area to support L_1 caches of up to 32 KB, a two-level on-chip cache organization should be considered. For example, the Alpha 21164 [16] has 8KB L_1 instruction and data caches with a unified 96KB on-chip L_2 cache. On the other hand, in the Alpha 21264 [36], the 2-level on-chip caches are replaced with 64KB single-level, 2-way set-associative caches, which improve the hit ratio at the cost of slower (2 cycles) access.

Although multi-level caches are conceptually simple, some additional issues have to be considered. For example, the reference locality to the downstream caches is much weaker because the high-locality requests can often be satisfied by the upstream caches. Therefore, in order to be able to capture a majority of the L_1 cache misses, the L_2 cache usually has to be more than an order of magnitude larger than the L_1 cache. In addition, it is difficult to maintain precise program locality at the L_2 cache because the memory requests that are satisfied by the L_1 cache are typically not issued to the L_2 cache in order to avoid excessive bandwidth requirements at the large L_2 cache. As a result, L_2 cache replacement is based on a Least Recently Missed (LRM) scheme rather than LRU. Mechanisms to supply the L_2 cache with L_1 cache hit information are evaluated in [40], [49]. A small performance improvement for *xlisp* of the SPEC92 benchmark suite is reported for certain cache configurations [40].

Another issue to consider in multi-level caches is that the miss penalty is effectively increased unless a cache miss is triggered before cache hit/miss is determined. Techniques for early triggering L_1 misses are discussed in [49]. In addition, having multiple levels of cache also delays the handling of cache coherence requests initiated by other processors or I/O units because the various levels of cache may have to be searched. As mentioned before, in two-level cache designs, the L_2 cache is usually several times bigger than the L_1 cache. Therefore, when the requested line is absent from the L_2 cache, it is not likely to be in the L_1 either. However, for correctness, when this situation is encountered, the search of the L_1 cache cannot be omitted.

One way to reduce the delay in satisfying cache coherence requests and prevent unnecessary interruptions to the processor is to maintain the *inclusion property* between adjacent cache levels [8]. This property simply states that the contents of the higher-level cache is included in the lower-level cache. In other words, a line absent from the L_2 cache cannot be valid in the L_1 cache. However, imposing the inclusion property in multi-level caches may reduce cache performance. First, whenever a line is replaced from the L_2 cache, the corresponding line in the L_1 cache must be invalidated. Multiple invalidations may be necessary when the L_1 and L_2 line sizes are different. Second, the effective L_2 cache size is reduced.

The impact of the invalidations on L_1 cache hit ratio for the TPC-C-like workload is summarized in Figure 3. In these simulations, we fix the L_1 cache size at 16KB and vary the L_2 cache size from 128KB to 1MB. We also vary the set-associativity from 1 to 4 for both caches. The invalidations have a significant impact when the L_2 cache is small or direct-mapped. This impact is further heightened with a set-associative L_1 cache.

The inclusion property ensures that there is maximum overlap of cache contents between the L_1 and L_2 caches.

On the other hand, in order to maximize the effective size of the L_2 cache, the cache contents should be disjoint or exclusive, especially when the L_2 cache is not significantly bigger than the L_1 cache [39], [40], [29]. However, maintaining an exclusive multi-level cache requires higher L_1/L_2 bandwidth. In addition, the replacement becomes complicated when the line size and the number of sets are different between the two caches.

B.5 Other Related Techniques

There are many other techniques that focus primarily on improving the cache hit ratio. One general approach is to reduce cache pollution by being selective about what gets cached [44], [1], [76], although we are very skeptical about the effectiveness of this type of scheme. Another approach is to have two separate caches, one designed for capturing temporal locality and the other, spatial locality [18], [46], or to have variable fetch sizes to handle poor spatial locality [54], [26], [33]. A third class of techniques strives to reduce cache conflict misses through the use of better address hash functions and more sophisticated page mapping techniques [25], [74], [80], [32], [61], [11], [19].

C. Area and Bandwidth Constraints

In order to bridge the growing performance gap between processor and memory, more and more silicon area is being dedicated to the on-chip caches. For example, the Intel Pentium Pro consists of a pair of 8KB on-die instruction and data L_1 caches and an on-module 512KB L_2 cache. Together these caches occupy 65% of the total die area and account for 88% of the total number of transistors [48]. The size of the caches is only part of the reason the cache hierarchy takes up so much die area in today's processors. The other reason is that the caches must be able to satisfy the enormous memory bandwidth demanded by aggressive multiple-issue dynamic processors which are capable of issuing multiple instruction and data references per cycle.

There are several approaches to increasing cache bandwidth. A straightforward way is to have separate instruction and data caches so that the instruction and data references can be handled simultaneously. However, as processors become increasingly superscalar, this approach by itself is not sufficient. Because the instruction reference stream is highly sequential, the instruction bandwidth required can typically be satisfied by using a wide instruction cache port (e.g. 16 bytes) and/or an instruction buffer [21] to deliver multiple instructions per cycle. However, due to frequent branches, the instruction cache often suffers incomplete fetches. The trace cache [52], [57] alleviates this problem by storing the logically contiguous instructions in a physically contiguous block in a separate cache.

The data reference stream is rarely so well behaved and therefore requires more aggressive designs to handle multiple requests per cycle. A general technique to enable concurrent memory or cache access is to divide the cache or memory into banks that can be independently accessed [9], [71]. For instance, the MIPS R10000 relies on cache banks that are 2-way interleaved to handle up to two concurrent data cache accesses [85]. The drawback of this approach is that there may be contention for the same bank which will reduce the effective bandwidth. In the Alpha 21164, the data cache is duplicated to achieve approximately the performance of a true dual-ported cache at the expense of more than doubling the chip area [16]. The Alpha 21264 [36] achieves high cache bandwidth by phase-pipelining the access so that one index can be supplied on every clock edge. This fast pipelined access avoids bank conflicts without requiring duplicated cache arrays. The Amdahl 470V machines also used a pipelined cache [65].

An effective way to increase cache bandwidth without incurring a huge area cost is to use a small multiple-ported buffer to retain recently fetched data [81]. The buffer is searched when a memory request is waiting for an available port to access the cache. If the requested data is found in the small buffer, the normal cache access is canceled.

A different approach to reducing the real estate taken up by the cache hierarchy is to reduce the size of the cache tag array. A popular technique is to associate each cache tag with a *sector* consisting of a fixed number of cache subsectors [38], [47]. This effectively increases the line size from the standpoint of cache management. In order to avoid excessive memory and bus traffic, a cache miss will only bring in the requested line and not the entire sector. One major drawback of the sector cache is that the allocation of cache space is done on a sector basis, even though only some of the subsectors of that sector may be in use. A recent work [58] confirms that a one-level sector cache usually does not perform as well as a standard one-level set associative cache. Nevertheless, the sector cache is useful when there is a need to integrate the cache tag array of a large off-chip cache into the processor. The decoupled sector cache is an attempt to improve cache space utilization by letting several sectors share a common pool of cache locations. In the design proposed in [62], each cache set may contain more than one sector and the lines within a sector can be stored in any location within the cache set.

Another approach to reducing the area requirement of the tag array is to avoid duplication of address tags. For instance, due to locality of reference, the high-order bits of the address tags tend to change less frequently than the low-order bits [78]. Therefore, some area may be saved by keeping these unique high-order address tags in a small table and replacing the high-order address tags in the cache tag array with pointers to this small table. Another source of tag duplication lies in the fact that the TLB, cache tag array and possibly branch target buffer all maintain some form of address tags [64]. Therefore, it is more space-efficient to implement a unified tag array to save a single copy of the active address tags. Whether these two proposals can be implemented efficiently is not clear.

III. Challenges Ahead

During the past decade the performance of processors has improved by almost 60% each year. Although the capacity of DRAM has doubled every 18 months during this same period, its performance has improved by less than

10% per year. Such a trend is expected to continue in the foreseeable future. Bridging this ever-growing performance gap between the processor and memory in a cost-effective manner will require novel cache designs and increasingly aggressive implementations of cache memories.

Current trends in the industry suggest that in the future, it may become economically feasible to integrate a processor on the same die as the DRAM [59]. Such an integration has the potential to reduce system cost and improve both DRAM latency and available bandwidth [48]. Although these improvements may be substantial, the inherently slow DRAM access still presents a significant gap with respect to the speed of the processor. For general purpose computing, cache memories will continue to play a crucial role in bridging the processor-DRAM performance gap.

References

[1] S. Abraham, et al., "Predictability of Load/Store Instruction Latencies," *MICRO'26*, Dec. 1993, pp. 139–152.

[2] A. Agarwal, J. Hennessy, M. Horowitz, "Cache Performance of Operating Systems and Multiprogramming," *ACM Trans. Computer Systems*, Vol. 6(4), Nov. 1988, pp. 393–431.

[3] A. Agarwal, S. Pudar, "Column-Associative Caches: A Technique for Reducing the Miss Rate of Direct-Mapped Caches," *20th ISCA*, May 1993, pp. 179–190.

[4] T. Ahearn, et al., "Virtual Memory System," US Patent No. 3781808, Dec. 25, 1973.

[5] J. Alvarez, R. Barner, "Memory System with Logical and Real Addressing," US Patent No. 3723976, March 27, 1973.

[6] T. Austin, G. Sohi, "High-Bandwidth Address Translation for Multiple-Issue Processors," *23rd ISCA*, May 1996, pp. 158–167.

[7] J. Baer, W. Wang, "Architectural Choices for Multilevel Cache Hierarchies," *14th ISCA*, June 1987, pp. 258–261.

[8] —, "On the Inclusion Property for Multi-Level Cache Hierarchies," *15th ISCA*, May 1988, pp. 73–80.

[9] F. Baskett, A. Smith, "Interference in Multiprocessor Computer Systems with Interleaved Memory," *Communications of the ACM*, June 1976, Vol. 19(6), pp. 327–334.

[10] S. Bederman, "Cache Management System Using Virtual and Real Tags in The Cache Directory," *IBM Tech. Disc.*, 21(11), April 1979, pp. 4541.

[11] B. Bershad, D. Lee, T. Romer, J. Chen, "Avoiding Conflict Misses Dynamically in Large Direct-Mapped Caches," *6th ASPLOS*, Oct. 1994, pp. 158–170.

[12] B. Calder, D. Grunwald, J. Emer, "Predictive Sequential Associative Cache," *2nd Symp. on High-Perf. Comp. Arch.*, Jan. 1996, pp. 244–253.

[13] T. Chiueh, R. Katz, "Eliminating the Address Translation Bottleneck for Physical Address Cache," *5th ASPLOS*, Sep. 1992, pp. 137–148.

[14] B. Chung, J. Peir, "LRU-Based Column Associative Caches," *ACM SIGARCH Comp. Arch. News*, Vol. 26(2), May 1998, pp 9–17.

[15] N. Drach, A. Seznec, "Semi-Unified Caches," *1993 Int'l Conf. Parallel Processing,* Aug. 1993, pp. I 25–28.

[16] J. Edmondson, P. Rubinfeld, R. Preston, V. Rajagopalan, "Superscalar Instruction Execution in the 21164 Alpha Microprocessor," *IEEE Micro*, Vol. 15(2), April 1995, pp. 33–43.

[17] R. Fletcher, "Second Level Cache Replacement Method and Apparatus," *US Patent No. 4,464,712*, Aug. 1984.

[18] A. Gonzales, C. Aliagas, M. Valero, "A Data Cache with Multiple Caching Strategies Tuned to Different Types of Locality," *1995 Int'l Conf. Supercomputing*," 1995, pp. 338–347.

[19] A. Gonzalez, M. Valero, N. Topham, J. Parcerisa, "Eliminating Cache Conflict Misses Through XOR-Based Placement Functions," *11th Int'l Conf. Supercomputing*, 1997, pp. 76–83.

[20] J. Goodman, "Coherency for Multiprocessor Virtual Address Caches," *2nd ASPLOS*, Oct. 1987, pp. 72–81.

[21] G. Grohoski, C. Moore, "Instruction Buffer to Support Multiple Fetches and Dispatches," *IBM Tech. Disc.*, Vol. 32(4B), Sep. 1989, pp. 30–31.

[22] M. Hill "A Case for Direct-Mapped Caches," *IEEE Computer*, Vol. 21(12), Dec. 1988, pp. 25–40.

[23] M. Hill, A. Smith, "Evaluating Associativity in CPU Caches," *IEEE Trans. Computers*, Vol. 22(12), Dec. 1989.

[24] K. Hua, et al., "Early Resolution of Address Translation in Cache Design," *Int'l Conf. Comp. Designs*, Oct. 1990, pp. 408–412.

[25] K. Inoue, H. Nonogaki, T. Urakawa, K. Shimizu, "Plural virtual address space processing system," US Patent No. 4145738, March 20, 1979.

[26] T. Johnson, W. Hwu, "Run-Time Adaptive Cache Hierarchy Management via Reference Analysis," *24th ISCA*, June 1997, pp. 315–326.

[27] N. Jouppi, "Architectural and Organizational Trade-Offs in the Design of the MultiTitan CPU," *15th ISCA*, May 1988, pp. 281–289.

[28] —, "Improving Direct-Mapped Cache Performance by the Addition of A Small Fully-Associative Cache and Prefetch Buffers," *17th ISCA*, May 1990, pp. 364–373.

[29] N. Jouppi, S. Wilton "Tradeoffs in Two-Level On-Chip Caching," *21st ISCA*, April 1994, pp. 34–45.

[30] T. Juan, T. Lang, J. Navarro, "The Difference-bit Cache," *23rd ISCA*, May 1996, pp. 114–120.

[31] R. Kessler, R. Jooss, A Lebeck, M. Hill, "Inexpensive Implementation of Set-Associativity," *16th ISCA*, May 1989, pp. 131–139.

[32] R. Kessler, M. Hill, "Page Placement Algorithm for Large Real-Indexed Caches," *ACM Trans. Computer Systems*, Vol. 10(4), Nov. 1992, pp. 338–359.

[33] S. Kumar, C. Wilkerson, "Exploiting Spatial Locality in Data Caches using Spatial Footprints," *25th ISCA*, June 1998, pp. 357–368.

[34] G. Kurpanek, et al., "PA7200: A PA-RISC Processor with Integrated High Performance MP Bus Interface," *CompCon'94*, Feb. 1994, pp. 375–382.

[35] R. Lee, "Precision Architecture," *IEEE Computer*, Vol. 22(1), Jan. 1989, pp. 78–91.

[36] D. Leibholz, R. Razdan, "The Alpha 21264: A 500 MHz Out-of-Order Execution Microprocessor," *CompCon'97*, Feb. 1997, pp. 28–36.

[37] D. Levitan, T. Thomas, P. Tu, "The PowerPC 620 Microprocessor: A High Performance Superscalar RISC Microprocessor," *CompCon'95*, Mar. 1995, pp. 285–291.

[38] J. Liptay, "Structural Aspects of the System/360 Model 85, Part II: The Cache," *IBM Systems Journal*, Vol. 7, 1968, pp. 15–21.

[39] L. Liu, "Vertical Partitioning in Cache Hierarchies," *IBM Tech. Disc.*, Vol. 30(8), Jan. 1988, pp. 33.

[40] —, "Managing Coherence for Multi-Level Caches," IBM Research Report, RC 18947, May 1993.

[41] —, "Cache Designs with Partial Address Matching," *MICRO'27*, Dec. 1994, pp. 128–136.

[42] —, "History Table for Set Prediction for Accessing a Set-Associative Cache," *US Patent No. 5,418,922*, May 1995.

[43] "The PowerPC Architecture," Edited by C. May, E. Silha, R. Simpson, H. Warren, Morgan-Kaufmann, May 1994.

[44] S. McFarling, "Cache Replacement with Dynamic Exclusion," *19th ISCA*, May 1992, pp. 191–200.

[45] B. Messina, W. Silkman, "Cache Synonym Detection and Handling Mechanism," US Patent No. 4332010, May 25, 1982.

[46] V. Milutinovic, M. Tomasevic, B. Markovic, M. Tremblay, "The Split Temporal/Spatial Cache: Initial Performance Analysis," *SCIzzL-5*, March 26, 1996.

[47] H. Olnowich, "Set associative sector cache," US Patent No. 4493026, Jan. 08, 1985.

[48] D. Patterson, et al., "A Case for Intelligent RAM," *IEEE Micro*, Vol. 17(2), Mar/Apr 1997, pp. 34–44.

[49] J. Peir, W. Hsu, H. Young, S. Ong, "Improving Cache Performance with Balanced Tag and Data Paths," *7th ASPLOS*, Oct. 1996, pp. 268–278.

[50] —, "Fast Cache Access with Full-Map Block Directory," *Int'l Conf. Comp. Designs*, Oct. 1997, pp. 578–586.

[51] J. Peir, Y. Lee, W. Hsu, "Capturing Dynamic Memory Reference Behavior with Adaptive Cache Topology," *8th ASPLOS*, Oct. 1998, pp. 240–250.

[52] A. Peleg, U. Weiser, "Dynamic Flow Instruction Cache Memory Organized Around Trace Segments Independent of Virtual Address Line," US Patent No. 5381533, 1994.

[53] S. Przybylski, M. Horowitz, J. Hennessy, "Characteristics of Performance-Optimal Multi-Level Cache Hierarchies," *16th ISCA*, May 1989, pp. 114–121.

[54] S. Przybylski, "The Performance Impact of Block Sizes and Fetch Strategies," *17th ISCA*, May 1990, pp. 160–169.

[55] F. Reiley, J. Richcreek, "Parallel Addressing of A Storage Hierarchy in A Data Processing System Using Virtual Address," US Patent No. 3693165, Sep. 19, 1972.

[56] J. Rivers, E. Davidson, "Reducing Conflicts in Direct-Mapped Caches with A Temporality-Based Design," *1996 Int'l Conf. Parallel Processing*, Aug. 1996, pp. 151–162.

[57] E. Rotenberg, S. Bennett, J. Smith, "Trace Cache: A Low-Latency Approach to High-Bandwidth Instruction Fetching," *MICRO'29*, Dec. 1996, pp. 24–34.

[58] J. Rothman, "Analysis of Sector Caches for Single Processor Systems," In preparation, University of California, Berkeley.

[59] A. Saulsbury, F. Pong, A. Nowatzyk, "Missing the Memory Wall: The case for Processor/Memory Integration," *23 ISCA*, May 1996, pp. 90–101.

[60] G. Schmidt, J. Schnell, "Dynamic Address Translation Reversed," US Patent No. 3786427, Jan. 15, 1974.

[61] A. Seznec, "A Case for Two-Way Skewed-Associative Caches," *20th ISCA*, May 1993, pp. 169–178.

[62] —, "Decoupled Sectored Caches: Conciliating Low Tag Implementation cost and Low Miss Ratio," *21st ISCA*, April 1994, pp. 384–393.

[63] —, "DASC Cache," *1st Symp. on High-Perf. Comp. Arch.*, Jan. 1995, pp. 134–143.

[64] —, "Don't use the page number, but a pointer to it," *23rd ISCA*, May 1996, pp. 104–113.

[65] A. Smith, "Sequential Program Prefetching in Memory Hierarchies", *IEEE Computer*, Vol. 11(12), Dec., 1978, pp. 7-21.

[66] —, "Cache Memories," *Computing Surveys*, Vol. 14(4), Sep. 1982, pp. 473–530.

[67] —, "Cache Memory Design: An Evolving Art," *IEEE Spectrum*, Dec. 1987, pp. 40–44.

[68] —, "Trace Driven Simulation in Research on Computer Architecture and Operating Systems," *Conf. New Directions in Simulation for Manufacturing and Comm.*, Aug. 1994, pp. 43–49.

[69] K. So, R. Rechtschaffen, "Cache Operations by MRU Change," *Int'l Conf. Comp. Designs*, Oct. 1986, pp. 584-586.

[70] F. Sparacio, "Data Processing System with Second Level Cache," *IBM Tech. Disc.*, 21(6), Nov. 1978, pp. 2468–2469.

[71] P. Stanley, R. Brown, A. Peters, "Odd/Even Bank Structure for A Cache Memory," US Patent No. 4424561, Jan. 03, 1984.

[72] Sun Microsystems, "Introduction to Shade," April 1993.

[73] —, "UltraSPARC-IIi User's Manuel," 1997.

[74] G. Taylor, P. Davis, M. Farmwald, "The TLB Slice - A Low-Cost High-Speed Address Translation Mechanism," *17th ISCA*, May 1990, pp. 355–363.

[75] S. Tucker, "The IBM 3090 System: An Overview," *IBM System Journal*, Vol. 1(25), 1986.

[76] G. Tyson, M. Farrens, J. Matthews, A. Pleszkun, "A Modified Approach to Data Cache Management," *MICRO'28*, Nov. 1995, pp. 93–103.

[77] W. Wang, J. Baer, H. Levy, "Organization and Performance of a Two-Level Virtual Real Cache Hierarchy," *16th ISCA*, May 1989, pp. 140–148.

[78] H. Wang, T. Sun, Q. Yang, "CAT - Caching Address Tags, A Technique for Reducing Area Cost of On-chip Caches," *22nd ISCA*, June 1995, pp. 381–390.

[79] C. Webb, J. Liptay, "A High-Frequency Custom CMOS S/390 Microprocessor," *Int'l Conf. Comp. Designs*, Oct. 1997, pp. 241–246.

[80] B. Wheeler, B. Bershad, "Consistency Management for Virtually Index Caches," *5th ASPLOS*, Oct. 1992, pp. 124–136.

[81] K. Wilson, K. Olukotun, M. Rosenblum, "Increasing Cache Port Efficiency for Dynamic Superscalar Microprocessors," *23rd ISCA*, May 1996, pp. 147–157.

[82] S. Wilton, N. Jouppi, "An Enhanced Access and Cycle Time Model for On-Chip Caches," DEC WRL Research Report 93/5, July 1994.

[83] D. Wood, et al., "An In-Cache Address Translation Machanism," *13th ISCA*, June 1986, pp. 358–365.

[84] D. Wood, R. Katz, "Supporting Reference and Dirty Bits in SPUR's Virtual Address Cache," *16th ISCA*, May 1989, pp. 122–130.

[85] K. Yeager, "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, Vol. 16(2), April 1996, pp. 28–40.

Jih-Kwon Peir received his B.S.E. degree from National Cheng-Kung University, Taiwan, M.S. degree from University of Wisconsin-Milwaukee, and Ph.D. degree from University of Illinois, Urbana-Champaign, both in computer science. He was involved in mainframe processor design as a research staff member at IBM T.J. Watson Research Center (1986-92). During 1992-93, he served as deputy director in charge of the development of Pentium-based SMP systems in the Computer Technology Division of Taiwan's Industrial Technology Research Institute. He is currently an Associate Professor in the Computer and Information Science and Engineering Department at University of Florida. Dr. Peir's research interests include computer architecture and performance evaluation. He received an IBM Faculty Development Partnership Award in 1995 and an NSF Faculty Early Career Development Award in 1996. He serves as a subject area editor of the Journal of Parallel and Distributed Computing and is on the editorial board of the IEEE Transactions on Parallel and Distributed Systems.

Windsor W. Hsu was raised in the city state of Singapore. He received the B.S.(highest honors) in electrical engineering and computer sciences from the University of California, Berkeley. He is currently a Ph.D. student in the Computer Science Division of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley. Since 1996, Windsor has also been an engineer with the Storage Systems Department at the IBM Almaden Research Center. Windsor's research interests include computer architecture and performance analysis and modeling of computer systems. Windsor was a UC Regents' Scholar (1991-94), a UC Chancellor's Scholar (1993-94), a UC Regents' Fellow (1994-95) and an IBM Cooperative Fellow (1995-98).

Alan Jay Smith was raised in New Rochelle, New York. He received the B.S. degree in electrical engineering from the Massachusetts Institute of Technology, and the M.S. and Ph.D. degrees in computer science from Stanford University. He was an NSF Graduate Fellow. He is currently a Professor in the Computer Science Division of the Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, where he has been on the faculty since 1974; he was vice chairman of the EECS department from July, 1982 to June, 1984. His research interests include the analysis and modeling of computer systems and devices, computer architecture, and operating systems. He has published a large number of research papers, including one which won the IEEE Best Paper Award for the best paper in the IEEETC in 1979. He also consults widely with computer and electronics companies.

Dr. Smith is a Fellow of the Institute of Electrical and Electronic Engineers. He is on the Board of Directors (1993-99), and was Chairman (1991-93) of the ACM Special Interest Group on Computer Architecture (SIGARCH), was Chairman (1983-87) of the ACM Special Interest Group on Operating Systems (SIGOPS), was on the Board of Directors (1985-89) of the ACM Special Interest Group on Measurement and Evaluation (SIGMETRICS), was an ACM National Lecturer (1985-6) and an IEEE Distinguished Visitor (1986-7), was an Associate Editor of the ACM Transactions on Computer Systems (TOCS) (1982-93), is a subject area editor of the Journal of Parallel and Distributed Computing and is on the editorial board of the Journal of Microprocessors and Microsystems. He was program chairman for the Sigmetrics '89 / Performance '89 Conference, program co-chair for the Second (1990) Sixth (1994) and Ninth (1997) Hot Chips Conferences and has served on numerous program committees.