Sub-tagged Caches: Study of Variable Cache-Block Size Emulation

Siddhartha V Tambat   Sriram Vajapeyam   S Muthulaxmi

IISc-CSA-TR-2001-4
http://archive.csa.iisc.ernet.in/TR/2001/4/

Computer Science and Automation
Indian Institute of Science, India

July 2001

Sub-tagged Caches: Study of Variable Cache-Block Size Emulation

Siddhartha V. Tambat† Sriram Vajapeyam†‡ S. Muthulaxmi‡*

†Dept. of Computer Science & Automation ‡Supercomputer Education & Research Centre
Indian Institute of Science
Bangalore, INDIA 560012

{svtambat,sriram}@csa.iisc.ernet.in, muthu@synergy-infotech.com

July 31, 2001

Abstract

The performance of a cache of a given size depends significantly on the cache block size. Previous studies have shown that insufficient memory bandwidth, rather than raw memory latencies, will be the major obstacle to improving system performance. The cache block size has a large and varying impact on memory traffic and hit rate across programs, suggesting the need for variable block size caches. We propose sub-tagged caches, which extend sub-blocked caches by breaking up the address tag into a common block tag shared by all sub-blocks and a sub-tag unique to the specific sub-block. Sub-tagged caches allow words from different memory blocks to co-reside in a cache block, without changing the critical hit access path, and thus emulate variable address block sizes. Typically only a few address tag bits differ between the blocks involved in a cache conflict, thus motivating sub-tagged caches. Performance evaluation with SPEC2000 benchmarks shows that the miss ratio and memory traffic ratio of a column-associative sub-tagged cache is comparable to that of a four-way set-associative sub-blocked cache, while its average memory access time is 18% better on average.

Dynamic sub-block prefetching in conjunction with a sub-tagged cache emulates variable address and fetch block sizes. A comparative study of different variable block caches shows that while no single cache organization is best across all benchmarks, variable block size caches outperform fixed block size caches for seven of the twelve benchmarks. Among the different variable block size caches, the column-associative sub-blocked and sub-tagged caches seem to perform the best. Column-associative sub-tagged caches uniformly outperform direct-mapped decoupled sectored caches, while direct-mapped sub-tagged caches perform better than decoupled sectored caches for nine of the twelve benchmarks.

{This document was submitted for review to
the 8th International Symposium on High Performance Computer Architecture (HPCA-8)
on July 20, 2001.}

*S. Muthulaxmi is currently a software engineer at Synergy Infotech Pvt. Ltd., Bangalore. Her contribution to this work was part of her M.Sc.(Engg.) thesis work [17] done at IISc during 1995-97.

1. Introduction

The importance of effective uniprocessor caches is increasing given the growing disparity between processor and memory cycle times. Goodman et al [3], argue that a majority fraction of the processor cycles spent waiting for memory will continue to be due to insufficient memory bandwidth rather than due to raw memory latencies, and emphasize the importance of optimizing memory traffic. They demonstrate a gap exceeding two orders of magnitude between traffic generated by current cache organizations and that by ideal, minimal-traffic caches. The cache block size (more specifically, the transfer block size) is shown to have very large and varying impact on memory traffic across programs, leading them to suggest the need for supporting variable cache block sizes that can be dynamically tuned to individual applications.

Typical caches today employ fixed size, multi-word cache blocks to exploit spatial locality (via implicit prefetching of the entire block on a word miss) and to amortize the cost of cache tags across multiple words. However, since the spatial locality exhibited by applications for a given cache size varies significantly, different block sizes yield the best performance for the different applications. So, a compromise block size that performs reasonably well across a wide range of applications has to be chosen. In addition, other considerations like reducing the amount of tag storage (which requires larger address blocks) and reducing the amount of false sharing on multiprocessors (which requires smaller address blocks) influence the selection of the "best" block size.

Fixed block size caches generate considerable unnecessary memory traffic for programs for which smaller cache blocks are optimal. Additional consequences of multi-word blocks are potentially higher conflict misses and poorer cache utilization since unused words within a cache block hurt performance by polluting the cache. Previous studies by Goodman et al [3] and others have shown that less than half of the data brought into the cache gets used before eviction. In general, single-word block size caches will generate the least traffic when other cache parameters are held appropriately fixed, but yield poorer miss-rates (due to lack of spatial locality) and have a larger number of tag bits. Single-word blocks can be augmented to some extent with prefetching techniques such as one-block lookahead prefetch to simulate larger transfer blocks, while continuing to allow word-level replacement to reduce conflict misses. However, larger blocks (e.g. two-word blocks) are preferable to such a static prefetch scheme, since the benefits of word-level replacement might not be commensurate with the additional tag costs. Tagging a cache with small block sizes at cache block granularity requires bigger tag arrays and longer access times. This could be significant despite the huge on-chip transistor budgets, given the move to larger on-chip primary (L1) caches and 64-bit architectures.

Alternatives to fixed block size caches have been proposed. Simple sub-blocked caches employ different address and transfer block sizes, to save tag space. Decoupled sectored caches emulate variable size address blocks by allowing sub-blocks in a cache block to choose among several tag locations dynamically, and thus increase cache block utilization. Adjustable block size and spatial footprint caches use variable size transfer (fetch) blocks to dynamically adapt to the spatial locality exhibited by the application, and so do not incur the memory traffic overhead of fixed block size caches.

In this paper, we propose a sub-tagged cache, which emulates variable size address blocks by associating a small portion of the block's address tag with each individual sub-block. This allows words from different memory blocks to co-reside a cache block, thus improving cache utilization without increasing cache hit time. By employing history-based selective sub-block prefetching in conjunction with a sub-tagged cache, we can emulate a variable block size cache.

We present a comparative study of the different approaches to emulating variable block sizes, especially focusing on sub-tagged caches and decoupled sectored caches. We use column associativity rather than set associativity as a base feature for caches where appropriate, so as to improve overall hit rate. Performance evaluation of L1 caches using trace-driven simulations of SPEC2000 integer benchmarks shows that no single fixed block size or variable block size cache performs best across all our benchmarks. However, column-associative variable block size caches outperform column-associative fixed block size caches for six of the twelve benchmarks, and they outperform direct mapped fixed block size caches for all benchmarks. Further, column-associative sub-tagged caches outperform decoupled sectored caches for all benchmarks, while direct mapped sub-tagged caches outperform decoupled sectored caches for nine of the twelve benchmarks. Finally, we find a need for improved sub-block prefetching schemes that can quickly detect their own poor performance and bail out to default fixed-size sub-block prefetch, i.e., emulate fixed cache blocks.

The rest of the paper is organized as follows. We present a detailed description of previously proposed variable block size caches in Section 2. We propose sub-tagged caches in Section 3. We then discuss the choice of sub-tag bits in a sub-tagged cache in Section 4. We present our performance comparison of the different variable block size caches in Section 5. We summarize the paper in Section 6.

2. Background: Cache Block Size

Early on, Goodman recognized the importance of a simple memory hierarchy for reducing memory bandwidth, particularly in a multiprocessor environment [7]. A cache block was recognized as encapsulating two notions: an *address block* that determines the cache tag overheads, and a *transfer block* that determines the unit of memory-cache transfer (the fetch size). Several studies [9, 23, 19] subsequently measured the trade-offs between miss ratio and traffic ratio by varying block and sub-block sizes. One of the studies [19] also investigated the effect of varying the fetch size independent of the block size and concluded that the optimal statically determined fetch size was generally twice the block size. McNiven and Davidson looked at reducing traffic between adjacent levels of the memory hierarchy [16]. Huang and Shen studied minimal required bandwidths for processors [10] by considering program-generated values, but without quantifying actual program address behaviour. Goodman et al [3] argue that memory bandwidth, rather than latency, will be the limiting memory performance factor. They demonstrate a gap exceeding two orders of magnitude between traffic generated by current cache organizations and that by ideal, minimal-traffic caches. The cache block size was shown to have a very large and varying impact on memory traffic across programs, leading them to suggest the need for supporting variable cache block sizes.

2.1. Variable Transfer/Fetch Block Sizes

Kumar and Wilkerson [14] describe a variable fetch-block scheme where spatial footprints (bitmaps) are associated with every sub-block in a cache. These spatial footprints record access patterns for every block, and are used for prefetching sub-blocks selectively on a miss for that cache block. Thus, the spatial footprint (SFP) cache selects the fetch block size dynamically based on the spatial locality exhibited by the application. The footprints are stored in a table that is indexed using a combination of bits from the instruction address as well as the data address.

The Spatial Locality Detection Table [12] attempts to dynamically pick the right fetch size based on the memory reference. However, it does it at a coarse granularity, and does not address the problem of increased L1 cache access latency associated with small cache blocks.

Dubnicki and LeBlanc proposed adjustable block size caches [5] to prevent false sharing in shared memory multiprocessor systems. The scheme is fairly complex to implement for a direct-mapped cache organization, and also has the disadvantage that it looks for spatial locality only in immediate neighbours.

2.2. Variable Address Block Sizes

Sub-blocked caches reduce the cost of tagging a cache with small blocks at cache block granularity by associating a single tag with a cache block consisting of multiple sub-blocks, but valid, dirty and coherence bits with each sub-block. They reduce memory traffic by transferring only a single sub-block on a cache miss. However, the non-fetched sub-blocks within a cache block result in poor utilization of the cache and poor hit ratios.

Seznec [21] proposed decoupled sectored caches (DSC) as a way to reconcile low tag implementation cost with low miss ratio. In a decoupled sectored cache, instead of the static association between the tag and data, the association is determined dynamically at run time. The address tag location associated with a cache sub-block is dynamically chosen among several possible tag locations. This mapping is dynamically maintained by using tag selection bits for every cache sub-block to specify the location of the tag. It also maintains the coherence and validity bits on a per sub-block basis. Since a tag can be associated with one or more sub-blocks in a single block, the DSC emulates variable address block sizes. On a cache reference, the tag match is slightly more complicated than in traditional caches and, therefore, takes longer. However, since the data array access is not affected, a way-predictor [15] can be employed to optimistically use the data without waiting for the tag matching to complete [21].

2.3. Caches for Best Hit Performance

While direct-mapped caches can exhibit a large number of conflict misses, a set-associative cache always displays a higher hit access time than a direct-mapped cache of the same total size because an associative search of a set is required during each reference, followed by a multiplexing of the appropriate data word to the processor. Other techniques access a direct-mapped cache sequentially more than once in order to achieve a fast access time for the first access and a high hit ratio as a whole. Examples of such techniques include the hash-rehash cache [1], the column-associative cache [2], the Direct-mapped Access Set-associative Check (DASC) cache [22], and the group-associative cache [18]. Upon a cache conflict, the hash-rehash and column-associative caches dynamically choose an alternate hashing function to store the conflicting block in an alternative location within the same cache. A rehash bit is added to each cache block to indicate whether that block stores data that is referenced by the alternate hashing function. A way to extend the column-associative cache to include multiple alternative locations is described in [4, 24]. The DASC cache and the group-associative cache combine an associative tag memory with a direct-mapped data memory. Both speculatively access the data array using the address as an index and potentially re-access the array if the first access fails but the requested data is present in a secondary location. In the DASC cache, a block can reside in only a limited number of locations. The group-associative cache maintains a small, highly associative tag directory for blocks not in their primary locations; these tags contain an index into the data array so that a block may reside in any location. However, the number of tag directory entries is only a fraction of the number of cache blocks, so that only a limited number of blocks can be located in other than their primary location.

A general approach to improving direct-mapped cache hit ratio is Jouppi's victim cache [13]. A victim cache is a small entirely separate fully-associative cache that provides some extra cache blocks for data that is replaced from the direct-mapped cache due to misses. In [11], a small fully-associative buffer is proposed for holding the blocks that exhibit poor temporal locality so as to prevent them from entering and polluting the primary direct-mapped cache. Another approach to reducing conflict misses is to use better hashing or mapping functions [20, 6].

3. Sub-tagged Caches

We propose *subtagged caches*, a novel way of emulating variable size cache blocks. The sub-tagged cache enhances sub-blocked caches by storing a small subset of a cache block's tag (say, 4 bits) with each sub-block. The address tag is broken up into two components: the block tag that is common to an entire cache block, and a sub-block tag (sub-tag for short) that is unique to each sub-block. Figure 1 compares a sub-tagged cache having 4 sub-blocks per block with a sub-blocked cache and with a Decoupled Sectored Cache (DSC) of similar configurations.

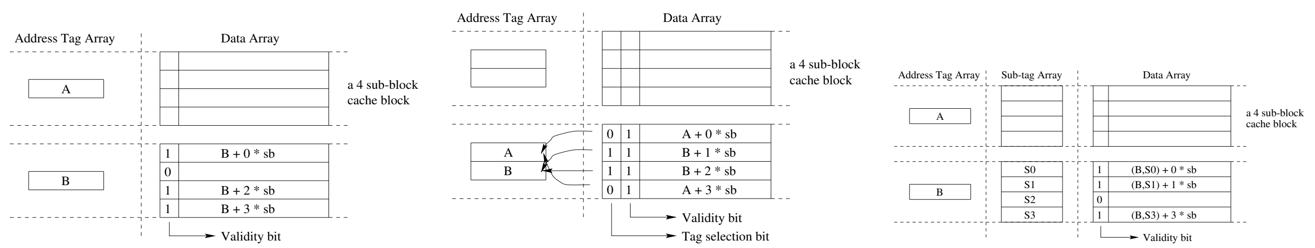
The motivation behind this organization is that the tags of conflicting memory blocks often differ in only a few bits (we present data for this in the next section). Sub-tagging allows words that have the same block tag but different sub-tags to co-reside in a cache block, provided they occupy different sub-blocks. This organization emulates a variable address-block size with low tag overhead: a cache block can be occupied by a single memory block or by words from different memory blocks, depending on access patterns; the additional tag overhead comes from the unique sub-tag for each sub-block. In combination with selective prefetching of sub-blocks, a sub-tagged cache block emulates a variable size cache block, i.e. both address and fetch block sizes are adaptable.

Sub-tagged caches have the same access time as direct-mapped caches while having better hit rate than DSCs for several applications. Neither subtagged caches nor DSCs uniformly demonstrate better hit rates for all applications, subtagged caches have advantages over DSCs in terms of both access time and memory traffic.

The hit logic [8] for a direct-mapped sub-tagged cache is illustrated in Figure 2. The delay through this logic determines cache hit time and directly affects effective access time. The hit logic for a sub-tagged cache using bit selection to select the set (block) of the reference has four components. The first component, the data RAM, holds the cached data. The second component, the tag RAM, holds the block tags associated with cache blocks; the third component, the sub-tag RAM, holds the state bits and address sub-tags associated with cached sub-blocks. The last component, the match logic, produces a single bit indicating whether the referenced sub-block is present.

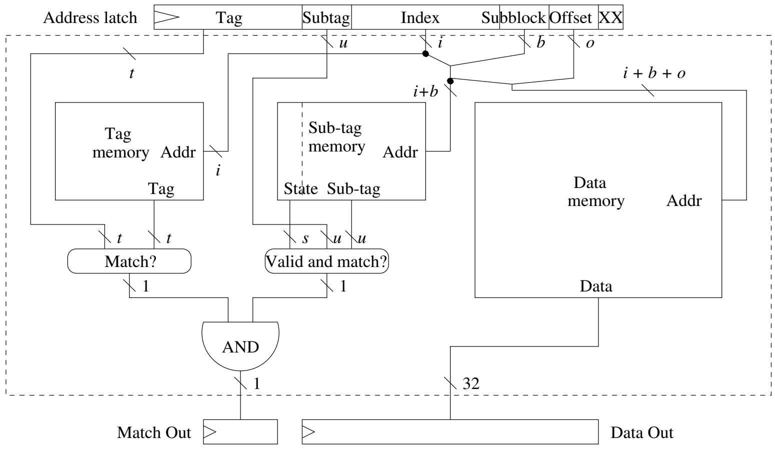
A cache hit consists of obtaining a match on both the tag and the sub-tag. The tag memory and the sub-tag memory are both direct-mapped (like the data blocks) and can be accessed in parallel. The tag memory is addressed by the index field. The sub-tag memory can be organized conceptually as if it were sub-tag wide and accessed with an address formed by concatenating the index (i bits) and sub-block (b bits) fields of the address. If it is implemented as a wider memory, some or all of the bits in the sub-block field will be used to select a sub-tag after the sub-tag memory access. The sub-tag access might take a little longer than the tag access in the latter case since it involves additional demultiplexing of the address from the block to the sub-block level, even though sub-tag selection could be done in parallel with the parallel comparison of the multiple sub-tags of a cache block with the address sub-tag. However, since the data array access is not affected, the hit time of a sub-tagged cache can be made the same as that of an ordinary direct-mapped cache by optimistic use of the required word. If bits from arbitrary parts of the address tag are used to form the sub-tag, the tag and sub-tag bits may need to be shifted when read from or written to the tag and sub-tag memory respectively. This shifting can be done in the address computation (EX) stage of the pipeline and may be completed by the time the tag (sub-tag) is fetched from the tag (sub-tag) memory. Thus, the critical hit access path would not be affected. The match-logic produces a single bit indicating whether the referenced sub-block is present. This bit is asserted only if the tag read from the tag memory is equal to the tag field of

Figure 1 Comparison of sub-blocked, DSC, and sub-tagged caches of equal size. There are 4 sub-blocks in a block while the size of a cache sub-block is *sb*.



(a) Sub-blocked cache (b) Decoupled sectored cache (c) Sub-tagged cache

Figure 2 Hit access logic for a sub-tagged cache.



Legend: o = log2(number of words per sub-block) bits; b = log2(number of sub-blocks per block) bits; i = log2(number of blocks) bits; t = 32 - u - i - b - o -2 bits

the address, the sub-tag read from the sub-tag memory is equal to the sub-tag field of the address, and the state read from the sub-tag memory is valid.

A cache miss in a sub-tagged cache can be either a block miss, caused by a mismatch of the block tag, or a sub-block miss caused by a mismatch of just the sub-tag (i.e. there is a match with the block tag). A block miss results in the entire current block being displaced (with its dirty sub-blocks written back to memory) and the requested sub-block (with the new block tag) being fetched. A sub-block miss results in only the sub-block (if any) in the accessed location being displaced. The requested sub-block is fetched into the location and only the corresponding sub-tag is updated.

Sub-tagging can just as easily be applied to set-associative caches as well. However, the set-associative sub-tagged cache has the same higher access time as a set associative cache. Instead, we use column-associativity [2] to boost the hit ratio of the sub-tagged cache without unduly increasing its access time.

A column-associative sub-tagged cache (CASTC) resolves conflicts in the (block) tag by choosing alternate locations in the cache in which the conflicting data can reside. As described in [2], a rehash bit is used with each cache block to guide the search and replacement. For simplicity and for speed, the first-time access is performed using bit selection, and bit flipping (bit selection with the highest order bit of the index inverted) is used as the second hashing function. Notice that the column-associative subtagged cache's access time for a first-time hit remains unchanged from that of the sub-tagged cache. Because first-time hits are much more frequent than misses, the extra cycles required to implement the column-associativity on a miss can be easily balanced by the decrease in number of accesses to main memory, resulting in a smaller average memory access time when compared to a sub-tagged cache without column associativity. The use of bit flipping as a second hashing function requires appending the high-order bit of the index field to the block tag. This bit can potentially be used, along with other tag bits, in the sub-tag instead. However, as discussed later, we found that using the least significant bits of the "original" tag in

Table 1 Tag cost (number of bits) per word for various caches. The block size for the traditional cache is 4 bytes for 8KB, 16KB caches and 8 bytes for 32KB, 64KB caches. For other caches, the block size is 32 bytes for 8KB, 16KB caches and 64 bytes for 32KB, 64KB caches, while there are 8 sub-blocks in a cache block.

Cache Organization	Cache Size			
	8KB	16KB	32KB	64KB
Traditional	19.0	18.0	8.50	8.00
Sub-blocked	2.38	2.25	1.06	1.00
DSC: N=2,1-way	5.50	5.25	2.50	2.38
DSC: N=4,1-way	10.5	10.0	4.75	4.50
Sub-tagged: 4 sub-tag bits	5.88	5.75	2.81	2.75
Sub-tagged: 8 sub-tag bits	9.38	9.25	4.56	4.50
CASTC: 4 sub-tag bits	6.13	6.0	2.94	2.88
CASTC: 8 sub-tag bits	9.63	9.5	4.69	4.63

the sub-tag resulted in better performance.

3.1. Subtagged Caches vs. Decoupled Sectored Caches

As pointed out before, both the sub-tagged cache and the DSC emulate variable block size caches by associating different address tags with different sub-blocks in the same cache block. The access time for the DSC is likely to be slightly higher than the sub-tagged cache. Moreover, the control logic for invalidating sub-blocks selectively on a tag mismatch in the DSC is more complex than in sub-tagged caches. In terms of space overhead, the tag requirements of a DSC can be much higher. Consider a 16KB cache with 32 byte blocks and 18-bit address tags. Using a 4 bit sub-tag per 4 byte word results in a 14 bit main tag plus 32 bits of sub-tags for a cache block. This can, in the best case, allow each sub-block to have its own sub-tag. A similar investment in a DSC will allow only two 18 bit tags to be shared by the 8 sub-blocks. Of course, these two tags can differ in any number of bits, whereas in a sub-tagged cache only the sub-tag bits of the different sub-blocks can differ. Table 1 compares the tag implementation cost of sub-tagged cache and DSC. The tag space overhead of traditional and sub-blocked caches is also shown for perspective. For the DSC, the decoupling degree (i.e., the number of address tags per cache block) and the associativity degree of the tag array are indicated. For example, "N=2,1-way" denotes a DSC with a decoupling degree of two and a direct-mapped address tag array. Only tags needed for retrieving the address of the data word stored in a cache block are considered here, i.e., address tags for traditional and sub-blocked caches, address tags and selection tags for DSC, and address tags and sub-tags for sub-tagged caches. (The cost of the rehash bit in CASTC is also considered.) Sub-tagged caches have lesser implementation complexity and better access time than DSC and comparable tag space overhead, but achieve better overall performance as discussed later.

4. Tag-bit Conflicts and Sub-tag Selection

The motivation for the sub-tagged cache lies in the small number of tag bits that participate in a cache conflict miss. We conduct simple experiments to determine the participation of individual tag bits in cache conflicts. For a given cache organization, whenever a valid cache (sub-)block is replaced, each tag bit that differs between the replaced block and the requested block is considered to participate in that conflict. Figure 3 plots the fractional contribution to conflicts of different tag bits, and Figure 4 shows the cumulative fractional contribution to conflicts of different number of tag bits for the SPEC2000 integer benchmarks. We used a 16KB direct-mapped data cache with 4 byte sub-blocks and 8 sub-blocks for this study. The figures show that just a few tag bits contribute to a large fraction of the conflicts, and that these bits are typically the low-order tag bits. For example, 5 bits contribute to about 98% of the conflicts in gzip, and these are bits 14-18. Typically 4-6 bits contribute to over 50% of the conflicts on average across all benchmarks.

We considered several variants for selecting sub-tag bits. For example, we tried using combinations of a few least significant and a few most significant address tag bits as the sub-tag, or using just a one or two bit wide sub-tag. While different sub-tags performed differently for different programs, using the four or eight least significant address tag bits as the sub-tag

Figure 3 The fraction of total conflicts that each tag bit participates in for the SPECint2000 benchmarks.

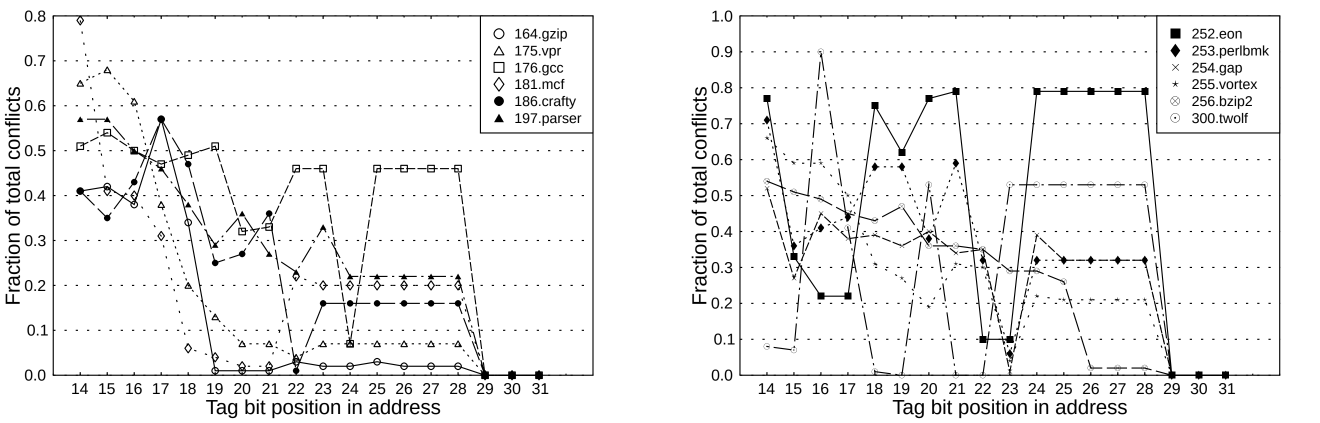
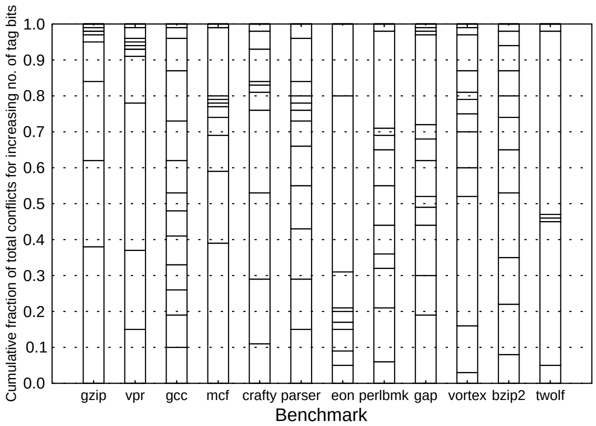


Figure 4 The cumulative fraction of total conflicts for increasing number of tag bits for the SPECint2000 benchmarks.

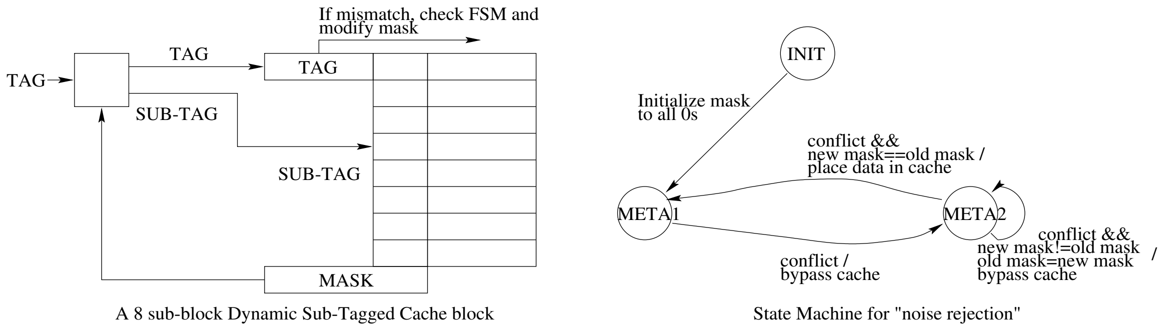


provided good overall performance for a moderate increase in tag cost (Table 1). Therefore, we choose the four or eight least significant tag bits to form the sub-tag in the experiments described in the rest of this paper.

4.1. Stack Relocation

Figure 3 shows that the higher order bits of the tag, bits 22 - 28, contribute uniformly to conflicts for all the benchmarks. The reason for this is that all stack and heap addresses differ exactly in these bits, due to the starting addresses for the stack and heap regions used by the compiler. (The start of the stack is set at link-time or load-time. For stack and heap variables, addresses are assigned at run time, when the dynamic storage is allocated.) Whenever a stack and heap address conflict in the data cache, all these bits of the tag will conflict, resulting in a block miss for the sub-tagged cache. However, this conflict behaviour is an artifact of the choice made by the compiler for the base addresses of the stack and heap. A cache-conscious compiler could easily pick appropriate base addresses to alleviate this problem for the sub-tagged cache. We consider this scenario by changing the base address of the stack in our benchmark traces such that the stack and heap addresses conflict only in bit 28. Specifically, we relocate the base address of the stack from 0x2ff2ffff to 0x2000ffff. This change amounts to a simple relocation of the stack segment to a better location from the perspective of the sub-tagged cache. We then include the single bit of the tag (bit 28 in this case), that conflicts with the other regions, in the sub-tag. Since relocation of the stack segment affects only the conflicts in the tag, it does not affect the performance of the other cache organizations we consider in this paper.

Figure 5 Dynamic sub-tagged cache organization with a tag-sized mask and finite state machine per cache block.



4.2. Dynamic Choice of Sub-tag Bits

The sub-tagged cache uses the idea of capturing several memory blocks in the same cache block by storing the few conflicting tag bits in the sub-tags while retaining the rest in a common block tag. However, different tag bits may conflict frequently for different applications, and using a "static" (i.e., hard-coded) selection of tag bits (for example, the four least significant bits) to form the sub-tag may not be an optimal solution. To mitigate this problem, we propose the Dynamic Sub-Tagged Cache (DSTC), wherein the bits to be extracted from the tag for the sub-tag are decided dynamically. We do this by associating a sub-tag mask with each cache block. A certain number of bits are set in this mask – these bits indicate the sub-tag bits of the corresponding address tag. The mask is used to extract the sub-tag from the address tag presented during cache lookup.

When a tag mismatch takes place, the mask is recomputed. For a cache block consisting of eight sub-blocks, if $T_1, T_2,...,T_8$ are the tags of the data words in the sub-blocks and T_i is the tag of the requested datum, then the new mask is computed as (T_1 OR T_2 OR ... OR T_8 OR T_i) AND ((NOT T_1) OR (NOT T_2) OR ... OR (NOT T_8) OR (NOT T_i)), where the AND, OR and NOT are performed bitwise. The tag of the data word that can be potentially replaced from the cache by the requested datum is not used in the computation. This computation causes the conflicting tag bit positions to be set in the mask. If the number of bits set in the mask is less than or equal to the sub-tag size, then a tag mismatch causes a sub-block miss; the requested datum is fetched into the corresponding sub-block, and the sub-tags of the remaining sub-blocks are recomputed from the tag using the new mask. Otherwise, a tag mismatch causes the block to be replaced from the cache, the mask is reset, and the requested datum is fetched into the cache. We use a simple "noise rejection" filter to prevent spurious mask modifications – the mask is not recomputed until the same tag bit conflicts are seen twice. This requires additional logic to implement cache bypass when the mask is not modified and the requested datum cannot co-reside with other data words in the cache block. Figure 5 depicts the organization of a single cache block of the DSTC along with a two-bit finite state machine (FSM) used as a noise rejection filter. The FSM verifies that the same tag bit conflicts are seen twice before a new mask is computed. Though this scheme incurs the additional overhead of storing a tag-sized mask and a two-bit FSM with each cache block, our performance evaluation shows that it achieves a hit ratio that is consistently better than that of a static sub-tagged cache. Moreover, the DSTC has the same hit access time as that of a sub-tagged cache since the mask modification is done only on a cache miss, and so can be overlapped with the servicing of the miss.

5. Performance of Variable Block Size Caches

We now report a comparative study of different cache organizations that try to emulate variable cache block sizes. In particular, we compare the performance of the sub-tagged cache and the decoupled sectored cache, especially in the presence of column associativity. We introduce sub-block prefetching to the sub-tagged cache so as to emulate both variable-size address blocks and variable-size transfer blocks. For fair comparison, sub-block prefetching is used in the decoupled sectored cache as well. Our sub-blocked cache with sub-block prefetching is equivalent to the spatial footprint cache without its default predictor mechanism [14]. Our comparisons are based on both memory traffic and hit rate.

Before presenting the results of the study, we first describe the performance metrics used, the experimental methodology, and the prefetching mechanisms employed. We first compare the different cache organizations without any prefetching, and then study their relative performance in the presence of prefetching.

5.1. Cache Performance Metrics

We use three cache performance metrics in our results – the miss ratio, the memory traffic ratio, and a very simple formulation of the average memory access time. The *memory traffic ratio* is the ratio of the total traffic between the cache and lower levels of the memory hierarchy to the number of data references. (We consider only level one (L1) caches in this study.) This metric is useful in determining the cache-to-memory bandwidth requirements of the benchmarks for the various caches.

We use the simplest model for the *average memory access time*, and do not attempt to model such features as overlapping memory references, non-blocking caches, etc. For direct-mapped caches, the cache hit time is one cycle for hits, and one plus MM for misses. Thus, for R references, of which H_1 are hits,

$$t_{avg} = \frac{1}{R}[H_1 + (MM + 1)(R - H_1)]$$

We assume a hit in a conventional direct-mapped L1 cache takes a single cycle, and the memory access delay MM is 100 cycles. For set-associative caches, we assume that a hit takes 2 cycles.

For victim caches and column-associative caches, an extra delay is encountered when the requested data is present in an alternative location. We assume that this extra delay is 2 cycles. Note that the search of the alternative locations does not add to the cache miss penalty because the cache miss can be triggered once the requested data is not present in the primary location. In other words, the delay of searching the alternative locations can be overlapped with the cache miss penalty. Similarly, the extra delay incurred in scanning several selection and valid tags on a block miss in a decoupled sectored cache is hidden by the servicing of the cache miss. For victim caches, the access time is one for first-time hits, and three for the H_2 hits during a lookup of the fully-associative cache. References to the fully-associative cache that miss suffer a penalty of $(MM + 1)$ cycles. Thus,

$$t_{avg} = \frac{1}{R}[H_1 + 3H_2 + (MM + 1)(R - H_1 - H_2)]$$

where H_1 is the total number of hits on a first-time access and H_2 is the total number of hits on a second-time access.

For column-associative caches, we need an additional parameter R_2, which is the total number of second-time accesses. Thus the access time is one for first-time hits, and three for the H_2 hits during a rehash attempt. If a rehash is not attempted, then $(MM + 1)$ cycles are spent. Rehash attempts that miss suffer a penalty of $(MM + 1)$ cycles. Therefore,

$$t_{avg} = \frac{1}{R}[H_1 + 3H_2 + (MM + 1)(R - H_1 - R_2) + (MM + 1)(R_2 - H_2)]$$

The simulator described in the next section measures R, R_2, H_1, and H_2 for each of the cache types, and it derives average memory access times from the above equations.

5.2. Reference Traces and Cache Models

We study the performance of sub-tagged caches for all the twelve SPEC2000 integer benchmarks. Integer (non-numeric) benchmarks are known to stress the memory system more than floating-point (numeric) benchmarks because of their irregular data access patterns. The benchmarks were compiled using the native C and C++ compilers on an IBM RS/6000 AIX 4.3 workstation with the standard SPEC -O3 -qarch=ppc optimization flags. We used the training inputs for all benchmarks except *gap* in which case we used the test input. All traces are for user-level activity only including any activity in the statically linked libraries. The traces include all instruction and data addresses generated by the program but specify no other instruction information. Double-word memory accesses are handled by consecutively issuing the two adjacent single-word addresses. We simulate each benchmark for 200M instructions after discarding the appropriate number of instructions in the start-up phase for each program. We observed that such a 200M-instruction simulation window, chosen from later in the program's execution, reliably gives representative results for the SPEC2000 integer programs for reasonable simulation times [1]. Table 2 lists the inputs that we used to generate the traces for each benchmark. It also lists the number of memory references that we simulated (in millions), the percentage of references that are reads and writes, and the data set sizes for each benchmark.

[1] We also simulated traces of 2 billion instructions for each benchmark, and found the miss ratio and memory traffic for 200M and 2B instruction traces to be within 1% of each other for each benchmark.

Table 2 SPECint2000 benchmark characteristics.

Benchmark	Input	Number instrns (M)	Number refs (M)	% reads	% writes	Data set sizes (MB)
164.gzip	input.combined	200	118.54	62.98	37.02	2.42
175.vpr	net.in,arch.in	200	92.21	61.91	38.09	0.19
176.gcc	cp-decl.i	200	82.64	74.54	25.46	1.43
181.mcf	inp.in	200	133.86	54.93	45.07	93.06
186.crafty	crafty.in	200	107.87	57.58	42.42	1.82
197.parser	train.in	200	87.44	73.93	26.07	2.5
252.eon	chair.control.cook	200	84.35	71.21	28.79	0.16
256.perlbmk	perfect.pl	200	83.38	43.88	56.12	0.27
254.gap	test.in	200	93.61	70.43	29.57	64.1
255.vortex	bendian.raw	200	119.18	63.30	36.70	7.31
256.bzip2	input.compressed	200	108.87	57.39	42.61	5.54
300.twolf	train	200	83.43	62.56	37.44	0.23

Table 3 L1 cache parameters for some current microprocessors.

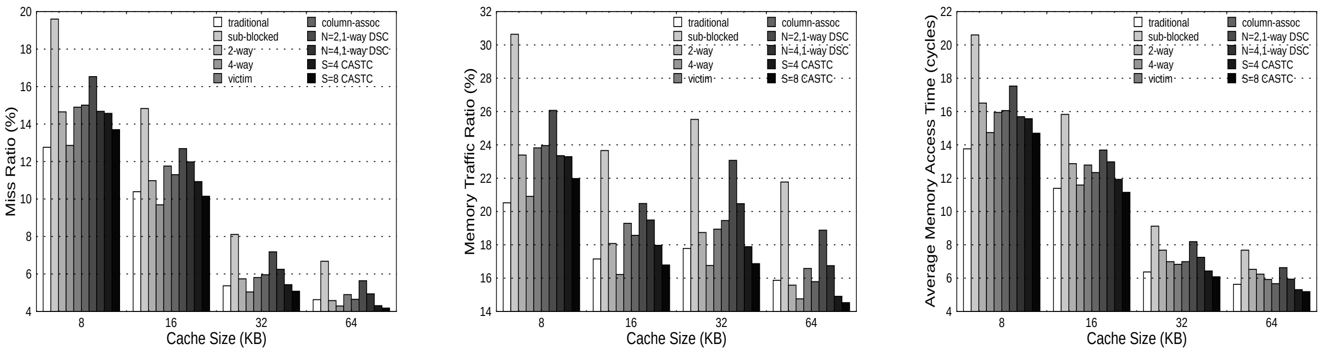
Processor	L1 cache size (KB)	L1 block size (bytes)	L1 assoc	L1 access time (cycles)	Tag cost/ word (bits)
PowerPC	64	64	4-way	2	1.12
21264	64	64	2-way	3	1.06
R10000	32	32	2-way	2	2.25
PIII	16	32	4-way	2	2.5

We study sub-tagged caches for L1 data caches of sizes 8KB, 16KB, 32KB, and 64KB. The block size is 32 bytes for 8KB and 16KB caches, and 64 bytes for 32KB and 64KB caches, while each block contains 8 sub-blocks. We chose the cache, block, and sub-block sizes based on the cache configurations of some current microprocessors, as shown in Table 3. We compiled this data by hand, using the processors' original manuals. The caches are copy-back and write-allocate. We modified DineroIII [9], a trace-driven cache simulator, to simulate direct-mapped, set-associative, victim, column-associative, decoupled sectored, and sub-tagged caches. We simulate fully associative victim caches of 16 entries. This choice is based on simulation data which suggests that the removal of conflicts quickly saturates beyond this size. In addition, remember that each victim cache entry is a complete cache block, storing the tag, status bits, and the data block, which contains 32 bytes (for 8KB and 16KB caches) and 64 bytes (for 32KB and 64KB caches) in these simulations. For the column-associative cache, we determine the secondary location by flipping the highest-order index bit. As described in [2], we include a *rehash* bit with each entry of the tag array to guide the search and replacement. We simulate decoupled sectored caches with decoupling degrees of two and four having direct-mapped address tag arrays (denoted by N=2,1-way and N=4,1-way respectively) which have roughly the same tag costs as that of sub-tagged caches with four and eight sub-tag bits respectively (Table 1). The cache simulator measures the miss ratio, the memory traffic ratio, and the average memory access time. We calculate traffic ratios by dividing the total traffic by the number of reads and writes issued. "Total traffic" in this case includes write-back traffic but not request traffic (i.e., addresses). We also flush the cache upon program completion, writing back all dirty data. We include these flushed write-backs in our traffic measurements. We do not model the timing of the cache-to-main memory interface in order to provide an upper bound on the impact of sub-tagging, and thus to focus on the effects of sub-tagging alone. This also helps to keep the simulator fairly simple and the design space tractable.

5.3. Sub-block Prefetching

We use a simple intra-block prefetch scheme to dynamically determine the block size needed by an application, and prefetch only the predicted sub-blocks on a cache block miss. Such a prefetch scheme in conjunction with sub-blocked,

Figure 6 Miss ratio, memory traffic ratio, and average memory access time vs. cache size, averaged over all twelve SPECint2000 benchmarks.



decoupled sectored, or sub-tagged caches provides a way of emulating caches with variable transfer block sizes. The predictor is fairly simple to implement. A bit mask is associated with each memory region of, say, 128 bytes. On every reference to a sub-block within this memory region, the corresponding mask bit is set. This is done on every hit or miss to a sub-block. On a block miss, the mask is used to selectively prefetch sub-blocks for which the mask bit are set, and the mask is reset. For example, a mask value of 11000100 on a cache miss to the second sub-block of a cache block will result in prefetch of sub-blocks 1, 2, and 6 of that cache block. Using more history (for example, using the previous two masks for a cache block) allows additional sub-block fetches to be traded for fewer missed sub-blocks. When the history consists of more than one mask, a bitwise OR is used to generate the predicted mask. By using a bitwise OR to combine the masks, we bias the predictor towards reducing the number of sub-blocks missed.

The mask prefetch scheme requires a hardware table to hold the per-memory-region mask registers (referred to as predictor registers). The desirable size of this table is determined by the size of the memory region that each predictor register is associated with. We experimented with different memory region sizes ranging from 32 bytes to 4KB, and found fairly similar performance for the different sizes. (An objective in varying the memory region size was to attempt to associate different predictor registers with different data structures to capture access patterns unique to individual data structures. A data structure is likely to have a fixed access pattern, e.g. accessing array elements with a fixed stride or repeatedly traversing a linked list, and since data structures in typical applications are large, access patterns are likely to remain fixed over large memory regions.) Each predictor register consists of a tag to resolve conflicts among the entries and one or more of the previously recorded masks depending on the amount of history used by the predictor. The table is indexed by the block address of the reference. In this paper, we use infinite sized tables for predictor registers in order to determine an upper bound on the performance of sub-block prefetching. Similar hardware tables for holding per-memory-region information (e.g. to determine replacement choice as per frequency of access [11]) have been proposed in the literature; the hardware table size was not found to be a major impediment in their study, for their use of per-memory-region information. We expect similar behaviour for our scheme; we do not model interference in limited size hardware tables. The predictor table is looked up on every cache access so that the mask is available on a miss. No extra cycles are added to the miss penalty.

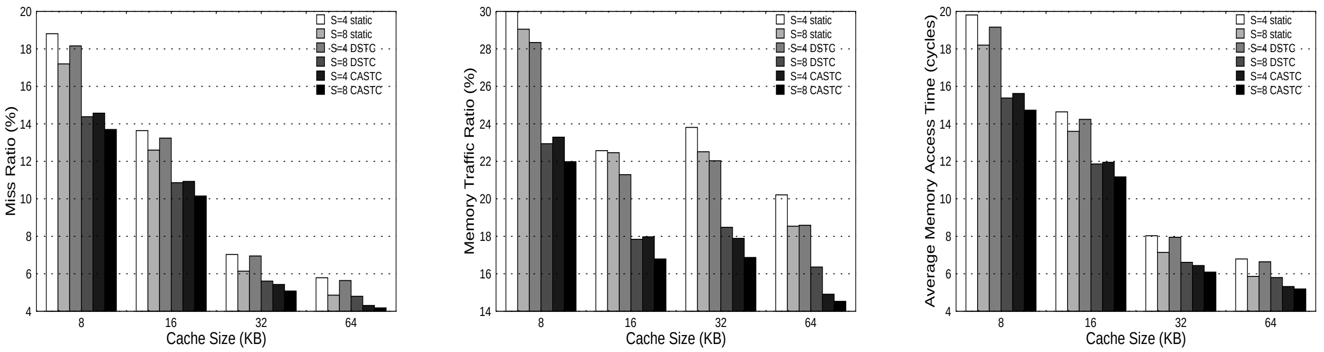
5.4. Performance of Variable-Size Address Blocks

We first compare the different cache organizations of interest (Table 4) without employing prefetching. This allows us to study the emulation of variable-size address blocks. Figure 6 compares the miss ratio, memory traffic ratio, and average memory access time for the different sub-blocked cache organizations – direct-mapped, set-associative, victim, column-associative, DSC, and column-associative sub-tagged cache (CASTC). Measurements for a traditional non sub-blocked cache (with block size equal to the sub-block size in the sub-blocked caches) are also shown for perspective. The numbers are shown for cache sizes of 8KB, 16KB, 32KB, and 64KB, and are averaged over all the twelve SPEC2000 integer benchmarks. It is evident that all of the cache designs exhibit much better performance than the direct-mapped sub-blocked cache. The plots show that the four-way set-associative cache achieves the best miss ratio and lowest memory traffic for 8KB and 16KB caches, while the miss ratio and memory traffic of the four-way set-associative cache and the S=8 CASTC are the lowest for 32KB and 64KB caches. The S=8 CASTC has much lower average memory access times than the other cache designs

Table 4 Acronyms and parameters used for the various caches. (For all caches, *PF* denotes sub-block prefetch and *h* denotes the history size used by the prefetcher.)

Acronym	Acronym description	Parameter	Parameter description
CAC	Column-Associative Cache	-	-
DSC	Decoupled Sectored Cache	N, 1-way	decoupling degree, associativity of address tag array
CASTC	Column-Associative Sub-Tagged Cache	S	number of sub-tag bits
DSTC	Dynamic Sub-Tagged Cache	S	number of sub-tag bits

Figure 7 Miss ratio, memory traffic ratio, and average memory access time for static sub-tagged, CASTC, and DSTC vs. cache size averaged over all twelve SPECint2000 benchmarks.



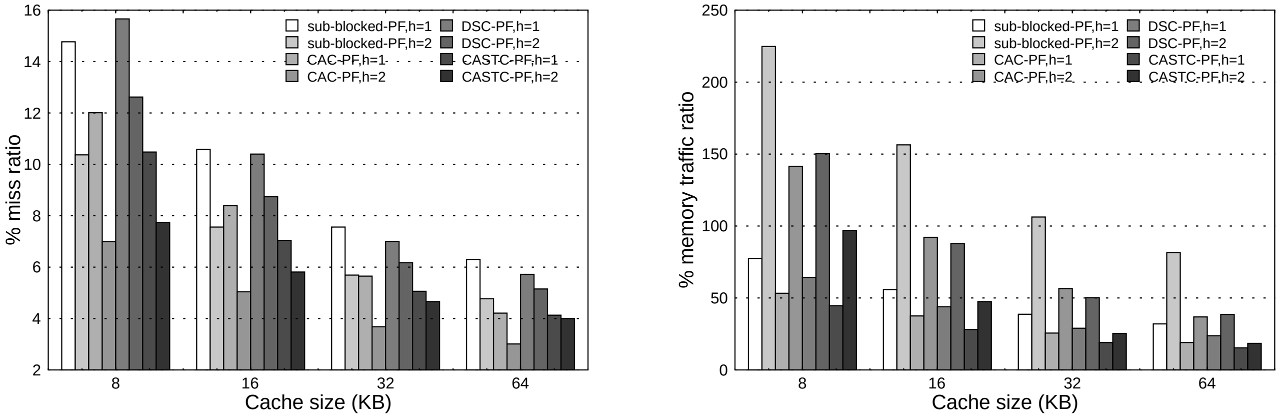
for all the cache sizes we considered. (The direct-mapped sub-tagged cache outperforms the DSC for four of the twelve benchmarks.)

The four-way set-associative cache has a miss ratio that is better than that of the S=4 and S=8 CASTC for 8KB and 16KB caches. However, the miss ratio of the S=8 CASTC is comparable to that of the four-way set-associative cache for 32KB and 64KB caches. The memory traffic ratio of these two caches shows the same characteristic. The average memory access time of the S=8 CASTC is comparable to that of the four-way set-associative cache for a 8KB cache, but the CASTC saves about 0.44 cycles, 0.91 cycles and 1.05 cycles over the set-associative cache for cache sizes of 16KB, 32KB and 64KB respectively. The savings are thus larger for large level-one (L1) caches.

The miss ratio of the S=4 and S=8 CASTC is better than that of the N=2,1-way and N=4,1-way DSC for all cache sizes. For example, the miss ratio of the S=4 CASTC is better by 12%, 14%, 24.4% and 23.6% over the N=2,1-way DSC for cache sizes of 8KB to 64KB. The memory traffic generated by the S=4 and S=8 CASTC is also lower than that of the N=2,1-way and N=4,1-way DSC for all cache sizes. As an illustration, the S=4 CASTC achieves a memory traffic ratio that is lower by 10.6%, 12.3%, 22.5% and 21% than that of the N=2,1-way DSC for cache sizes of 8KB to 64KB. The CASTC achieves significantly lower average memory access times than the DSC for all cache sizes. For example, the S=4 CASTC saves about 1.96 cycles, 1.76 cycles, 1.76 cycles and 1.32 cycles over the N=2,1-way DSC for cache sizes of 8KB to 64KB.

Figure 7 shows the miss ratio, memory traffic ratio, and average memory access time for static sub-tagged caches, CASTC, and DSTC averaged over all the twelve SPEC2000 integer benchmarks. The numbers are shown for caches of size 8KB, 16KB, 32KB, and 64KB. The plots show that both the CASTC and DSTC are superior in performance over the static sub-tagged cache, and that the CASTC significantly outperforms the static sub-tagged cache and the DSTC in terms of miss ratio, memory traffic ratio, and average memory access time. For example, the miss ratio of the CASTC for a 8KB cache is better by 20% over that of the static sub-tagged cache and by 4.7% over that of the DSTC. The memory traffic ratio of the CASTC is better by 24% over that of the static sub-tagged cache and by 4% over that of the DSTC, while the improvement in average memory access time over the static sub-tagged cache is about 3.5 cycles and over the DSTC is about 0.65 cycles. We note that it is possible, of course, to use column associativity with DSTC. However, we isolate DSTC from CA in this study to focus on the benefits of dynamic sub-tag determination.

Figure 8 Miss ratio and memory traffic ratio vs. cache size for sub-blocked, column-associative, DSC, and CASTC caches with sub-block prefetching using a history of one and two. All numbers are averaged over all twelve SPECint2000 benchmarks.



5.5. Performance of Variable-Size Cache Blocks

In this section, we evaluate the performance of different caches that use sub-block prefetching in conjunction with small transfer blocks to emulate variable cache block sizes. In this study, we associate each prefetch predictor register with a memory region of size equal to the cache block size. We also experimented with storing the first-referenced sub-tags or evicted sub-tags for each cache block with the corresponding bit in the mask, to enable the prefetcher to fetch sub-blocks from different memory blocks on a cache miss. However, we found the performance of these variants to be poorer than fetching only adjacent sub-blocks.

We study the performance of the CASTC with sub-block prefetching (denoted by CASTC-PF) vis-a-vis sub-blocked, column-associative sub-blocked, and DSC caches using sub-block prefetching (denoted by sub-blocked-PF, CAC-PF and DSC-PF respectively). Figure 8 presents the miss ratio and memory traffic ratio for these caches. The sub-block size is 8 bytes while there are 16 sub-blocks in a block. (We use a sub-block size of 8 bytes instead of 4 bytes in this study since we found the performance benefits of sub-block prefetching to be significantly better for this choice.) The DSC-PF has a decoupling degree of four and a direct-mapped address tag array. The width of the sub-tag in the CASTC-PF is 8 bits. We show results for a history size of one and two used by the prefetcher.

The first plot shows that CAC-PF achieves the best miss ratios. While CAC-PF outperforms CASTC-PF for a history of two, the miss ratio of CASTC-PF is better than that of CAC-PF for a history of one. For example, for a cache size of 32KB, the miss ratio of CAC-PF is better by 21% than that of CASTC-PF for a history of two, while the miss ratio of CASTC-PF is better by 10.4% than that of CAC-PF for a history of one. CASTC-PF outperforms both sub-blocked-PF and DSC-PF. For example, for a cache size of 32KB, the miss ratio of CASTC-PF is better by 33% and 18.1% over sub-blocked-PF and by 27.7% and 24.5% over DSC-PF for a history of one and two respectively.

The second plot also shows that the memory traffic of CASTC-PF is significantly lower than that of sub-blocked-PF, DSC-PF, and CAC-PF. For example, for a cache size of 32KB, CASTC-PF outperforms sub-blocked-PF, CAC-PF, and DSC-PF by 50.8%, 25.7%, and 34.3% respectively for history of one. For history of two, the corresponding numbers are 76.2%, 55.2%, and 49.5% respectively. Comparing the numbers for a history of one and two shows that, on average, the miss ratio is decreased by 30%, while the increase in memory traffic is quite significant.

Performance comparison of the variable block size caches vis-a-vis traditional caches having fixed-size blocks shows that, on average, the miss ratio and memory traffic of the variable block size caches is comparable to that of the fixed block size caches. However, we observed that the variable block size cache performed better than fixed block size caches for several individual benchmarks in terms of both miss ratio and memory traffic. For the other benchmarks, we conclude that we need a bail-out mechanism that dynamically recognizes the poorer performance of selective prefetching and defaults to fixed-size prefetching.

Table 5 presents the best value achieved for miss ratio and memory traffic for each of the benchmarks, and the block size responsible for producing that value. The results are shown for a cache size of 32KB. We observe that in seven of the twelve

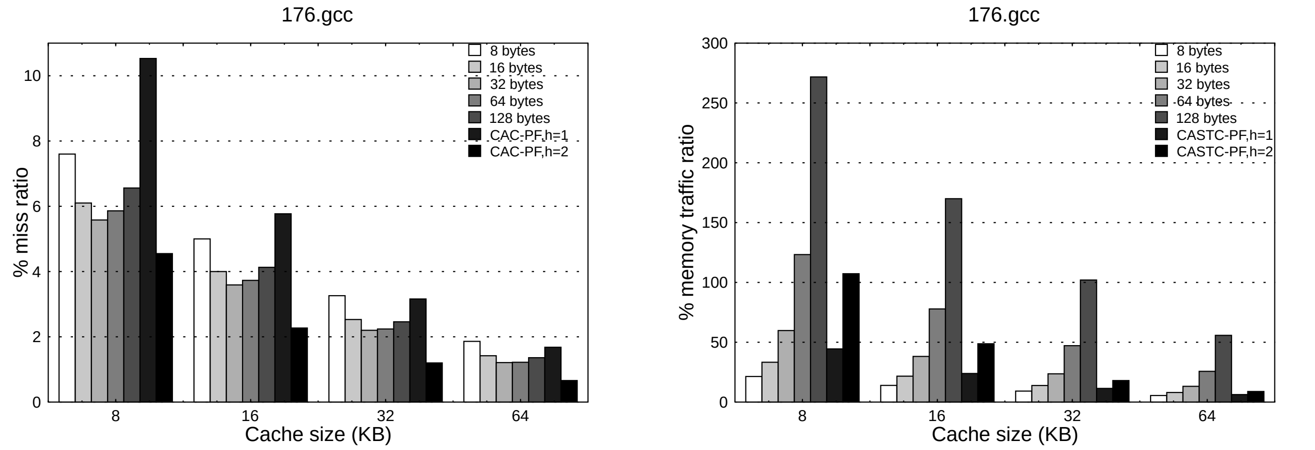
Table 5 The best cache organization for each benchmark and metric. Statistics are shown for a 32KB cache.

Benchmark	Lowest Miss Ratio			Lowest Memory Traffic Ratio			Best Fixed
	value	block size	Best Fxd/Variable	value	block size	Best Fxd/Variable	
gzip	2.88	CAC-PF,h=2	1.24	13.45	8	0.99	16,8
vpr	4.29	CAC-PF,h=2	1.08	17.06	8	0.91	64,8
gcc	1.2	CAC-PF,h=2	1.83	9.18	8	0.81	32,8
mcf	1.92	128	0.13	59.58	CASTC-PF,h=1	1.04	128,8
crafty	2.88	8	0.91	7.75	8	0.92	8,8
parser	1.52	CAC-PF,h=2	1.66	15.65	8	0.89	64,8
eon	0.62	CAC-PF,h=2	1.42	3.28	8	0.54	32,8
perl	1.86	CAC-PF,h=2	1.22	6.95	8	0.53	64,8
gap	4.55	128	0.52	48.71	CASTC-PF,h=1	1.05	128,8
vortex	2.02	32	0.85	7.99	8	0.73	32,8
bzip2	2.8	128	0.99	16.03	8	0.81	128,8
twolf	0.08	CAC-PF,h=2	8.75	0.48	CASTC-PF,h=1	5.13	8,8

benchmarks, CAC-PF,h=2 produced the lowest miss ratio, while the fixed block size cache achieved the best miss ratio for the remaining five benchmarks. A single fixed block size (8 bytes) produced the lowest memory traffic for nine out of twelve benchmarks, while CASTC-PF,h=1 has the lowest memory traffic for the other three benchmarks. The last column lists the best fixed block size for each of the two metrics. Where two sizes are given, the first produces the best miss ratio, the second produces the lowest memory traffic ratio. Examining this column, we see that no single block size is best across all our benchmarks. The best fixed block size varied between 8 and 128 bytes for the first metric. For the second metric, the best fixed block size was 8 bytes for all our benchmarks, of course. Even within a single application, no one block size is best, as evidenced by the performance of the variable block size organizations. CAC-PF,h=2 achieves the lowest miss ratio and CASTC-PF,h=1 achieves the lowest memory traffic ratio among the variable block size caches.

We next study the impact of varying the cache size on the performance of the variable block size caches vis-a-vis fixed block size caches.

Figure 9 Miss ratio and memory traffic ratio vs. cache size for traditional direct-mapped caches with block size of 8, 16, 32, 64 and 128 bytes, and CAC-PF and CASTC-PF with a history of one and two. The bar graphs represent just the gcc results.



As an illustration, Figure 9 shows the miss ratio and memory traffic for fixed and variable block size caches for gcc. The first five bars for each cache size in each plot is for traditional caches with block size varying from 8 to 128 bytes. The last two bars correspond to CAC-PF in the first plot and CASTC-PF in the second plot with 8 byte sub-blocks and 16 sub-blocks per block, using sub-block prefetch with a history of one and two. The first plot shows that the miss ratio of CAC-PF,h=2 is better than the best fixed block size cache (with a block size of 32 bytes) for all the cache sizes. The second plot shows that the fixed block size (with a block size of 8 bytes) cache has the lowest memory traffic for all cache sizes; the memory traffic ratio of CASTC-PF,h=1 is fairly comparable at cache sizes of 32KB and 64KB.

To be fair, we also compared the performance of CAC-PF and CASTC-PF with column-associative fixed block size caches and that of the direct-mapped sub-tagged cache using sub-block prefetching with DSC-PF. CAC-PF outperformed the column-associative fixed block size cache for six of the twelve benchmarks in terms of miss ratio, while the memory traffic of CASTC-PF was lower than that of the column-associative fixed block size cache for three benchmarks. The direct-mapped sub-tagged cache with sub-block prefetching outperformed DSC-PF for nine of the twelve benchmarks in terms of both miss ratio and memory traffic.

Thus, overall, the column-associative sub-tagged cache outperforms the decoupled sectored cache both in terms of hit rate and memory traffic. While the column-associative sub-blocked cache is the best in terms of hit rate, the column-associative sub-tagged cache is the best in terms of memory traffic. Given the presence of prefetching, we do not use our simplistic average memory access time metric to compare the different caches. Finally, variable block size caches outperform fixed block size caches for seven of the twelve benchmarks.

6. Summary and Conclusions

Today's caches use fixed size, multi-word blocks to exploit spatial locality exhibited by applications, via implicit prefetching of all the sub-blocks on a cache miss. However, unused words in a cache block result in poor utilization and considerable unnecessary memory traffic. The goal of this research has been to study caches that emulate variable block sizes - caches that use either address blocks or transfer blocks of variable sizes to boost the hit ratio and reduce memory bandwidth requirements. The main metrics used to evaluate cache performance in this study are the miss ratio, memory traffic ratio, and the simplest model of average memory access time.

We described the design of a *sub-tagged cache* that breaks up the address tag into two components: a block tag common to all the sub-blocks of a block, and small sub-block tags (sub-tags) with individual sub-blocks. This organization emulates a variable address-block size cache with low tag overhead. Further, we use history-based selective prefetching of the sub-blocks of a sub-tagged cache to emulate a variable cache block size. A dynamic sub-tagged cache can dynamically select sub-tag size and sub-tag bits to better adapt to the runtime behavior of the application. We use column-associativity and dynamic sub-tagging to boost the hit ratio of the sub-tagged cache without adversely impacting its access time.

We reported a comparative evaluation of different schemes for emulating variable block sizes. Apart from the sub-tagged cache, the decoupled sectored cache and a simple sub-blocked cache with selective sub-block prefetching emulate variable block sizes. The Spatial Footprint Cache employs a relatively sophisticated form of sub-block prefetching based on a history of spatial locality; we considered a suitable form of spatial locality based prefetching with the different cache organizations.

We first compared the different caches without pre-fetching. Simulations of different level-one (L1) caches with the SPEC2000 integer benchmarks show that the miss ratio and memory traffic of a column-associative sub-tagged cache (CASTC) is comparable to that of a four-way set-associative sub-blocked cache for larger L1 caches, while the latter is better for smaller L1 caches. CASTC's average memory access time is about 18% better on average than the 4-way set-associative sub-block cache for all L1 cache sizes considered. The column associative sub-tagged cache performs better than the equivalent decoupled sectored cache for all L1 cache sizes, whereas the direct-mapped sub-tagged cache outperformed DSC for four of the twelve benchmarks.

Sub-blocked caches that don't prefetch sub-blocks perform worse than fixed block size caches, so we next considered sub-block prefetching for the different caches. Our simulations show that no single fixed block size or variable block size cache performs best across all our benchmarks. The column-associative variable block size caches outperform direct-mapped fixed block size caches for seven of the twelve benchmarks, and outperform column-associative fixed block size caches for six benchmarks. Further, the column-associative sub-tagged cache uniformly outperformed the decoupled sectored cache, whereas the direct mapped sub-tagged cache outperformed DSC on average for nine of the twelve benchmarks.

A conclusion from our results is that the performance of variable block size emulation can be further improved by improving the sub-block prefetch algorithm. In particular, for those cases where a fixed block size cache outperforms a variable block size cache, the prefetch algorithm in the latter needs to quickly detect its poorer performance and bail out to fixed-size prefetch (i.e. emulate a fixed block size). Such a mechanism will ensure that variable block size caches become a superior choice always, for all benchmarks.

Acknowledgments

The simulations reported in this research were carried out on equipment donated by Intel Microprocessor Research Labs, USA. Matthew Jacob and K. V. Subramaniam answered numerous questions on the layout of program segments in memory. Arghya Mukherjee evaluated a preliminary design of the dynamic sub-tagged cache as a part of his M.E. project work. P. J. Joseph provided helpful advice and comments on earlier drafts of this document. Doug Burger provided useful feedback on Muthulaxmi's M.Sc. thesis in late 1997.

References

[1] A. Agarwal, J. Hennessy, and M. Horowitz. Cache peformance of operating systems and multiprogramming. *ACM Transactions on Computer Systems*, 6(4):364–373, November 1988.

[2] A. Agarwal and S. D. Pudar. Column-associative caches: A technique for reducing the miss rate of direct-mapped caches. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 179–190, May 17–19, 1993.

[3] D. Burger, J. R. Goodman, and A. Kägi. Memory bandwidth limitations of future microprocessors. In *ISCA23*, pages 78–89, May 22–24, 1996.

[4] B. Chung and J. Peir. Lru-based column associative caches. *Computer Architecture News*, 26(2):9–17, May 1998.

[5] C. Dubnicki and T. J. LeBlanc. Adjustable block size coherent caches. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 170–180, May 19–21, 1992.

[6] A. González, M. Valero, N. Topham, and J. M. Parcerisa. Eliminating cache conflict misses through XOR-based placement functions. In *Conference Proceedings of the 1997 International Conference on Supercomputing*, pages 76–83, July 7–11, 1997.

[7] J. R. Goodman. Using cache memory to reduce processor-memory traffic. In *Proceedings of the 10th Annual International Symposium on Computer Architecture*, pages 124–131, June 13–17, 1983.

[8] M. D. Hill. A case for direct-mapped caches. *IEEE Computer*, 21(12):25–40, December 1988.

[9] M. D. Hill and A. J. Smith. Experimental evaluation of on-chip microprocessor cache memories. In *Proceedings of the 11th Annual International Symposium on Computer Architecture*, pages 158–166, June 5–7, 1984.

[10] A. S. Huang and J. P. Shen. A limit study of local memory requirements using value reuse profiles. In *Proceedings of the 28th Annual International Symposium on Microarchitecture*, pages 71–81, November 29–December 1, 1995.

[11] T. L. Johnson and W. mei W. Hwu. Run-time adaptive cache hierarchy management via reference analysis. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 315–326, June 2–4, 1997.

[12] T. L. Johnson, M. C. Merten, and W. mei W. Hwu. Run-time spatial locality detection and optimization. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, pages 57–64, December 1–3, 1997.

[13] N. P. Jouppi. Improving direct-mapped cache performance by the addition of a small fully-associative cache and prefetch buffers. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 364–373, May 28–31, 1990.

[14] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*, pages 357–368, June 27–July 1, 1998.

[15] L. Liu. Cache designs with partial address matching. In *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 128–136, November 30–December 2, 1994.

[16] G. D. McNiven and E. S. Davidson. Analysis of memory referencing behavior for design of local memories. In *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pages 56–63, May 30–June 2, 1988.

[17] S. Muthulaxmi. A study of variable block size caches. Master's thesis, Supercomputer Education and Research Centre, Indian Institute of Science, October 1997.

[18] J.-K. Peir, Y. Lee, and W. W. Hsu. Capturing dynamic memory reference behavior with adaptive cache topology. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 240–250, October 3–7, 1998.

[19] S. Przybylski. The performance impact of block sizes and fetch strategies. In *Proceedings of the 17th Annual International Symposium on Computer Architecture*, pages 160–169, May 28–31, 1990.

[20] A. Seznec. A case for two-way skewed-associative caches. In *Proceedings of the 20th Annual International Symposium on Computer Architecture*, pages 169–178, May 17–19, 1993.

[21] A. Seznec. Decoupled sectored caches: Conciliating low tag implementation cost and low miss ratio. In *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pages 384–393, April 18–21, 1994.

[22] A. Seznec. DASC cache. In *Proceedings of the First International Symposium on High-Performance Computer Architecture*, pages 134–143, January 22–25, 1995.

[23] A. J. Smith. Line(block) size choice for cpu cache memories. *ACM Transactions on Computer Systems*, C-36:1063–1075, 1987.

[24] C. Zhang, X. Zhang, and Y. Yan. Two fast and high-associativity cache schemes. *IEEE Micro*, 17(5):40–49, September/October 1997.