ON THE INCLUSION PROPERTIES FOR MULTI-LEVEL CACHE HIERARCHIES

Jean-Loup Baer and Wen-Hann Wang

Department of Computer Science
University of Washington
Seattle, WA 98195

Abstract

The inclusion property is essential in reducing the cache coherence complexity for multiprocessors with multilevel cache hierarchies. We give some necessary and sufficient conditions for imposing the inclusion property for fully- and set-associative caches which allow different block sizes at different levels of the hierarchy. Three multiprocessor structures with a two-level cache hierarchy (single cache extension, multiport second-level cache, bus-based) are examined. The feasibility of imposing the inclusion property in these structures is discussed. This leads us to propose a new inclusion-coherence mechanism for two-level bus-based architectures.

1 Introduction

Caches have been used effectively to bridge the gap between fast processor cycle times and slow memory access times. Cache performance is mainly dependent on two factors: the hit ratio and the access time. The hit ratio is, to a great extent, a function of the size of the cache; the larger the cache, the higher the hit ratio. However, to increase the cache size naively in the hope of getting a better hit ratio can degrade the access time. First, a given cache size imposes a physical limitation on its speed: With the same technology, the larger the cache, the slower it will be. Moreover, some technologies(e.g., GaAs) impose restrictions on the size. In addition, optimizing the speed of a small cache is easier than optimizing the speed of a large one. Second, the virtual to real address translation and the tag matching cannot be done totally in parallel with the cache access when the cache size is large and the associativity is insufficient. This effect further reduces the speed.

In spite of the above problems, the use of large caches is still very appealing. On the performance side, large caches result in high hit ratios which in turn reduce memory traffic. This effect is especially valuable for multiprocessors. As to the cost side, because memories are getting cheaper each year, large caches become more affordable. Thus the design of efficient large caches is becoming an important problem. In [8], an expedient virtual address caching scheme is proposed which eliminates the need for address translation on a cache hit. In [6], an addressing strategy based on the most recently used(MRU) information is exploited to speed the address translation and cache access. Another promising scheme for dealing with large caches is based on the concept of multilevel cache hierarchies [3, 15, 16], in which smaller but faster caches are introduced to reduce the gap further between fast processor cycle times and relatively slow access times of large caches. The smaller first level caches are fast and the address translation can be done in parallel with data access if a virtual addressing caching scheme is not used. The large higher level caches provide higher overall hit ratios.

For multiprocessors, introducing cache hierarchies could aggravate the well-known cache coherence problem. Authors of [3, 15, 16] all suggest that the contents of the higher level caches be a superset of those of the lower level caches so that the higher level caches can shield the lower level caches from I/O and cache coherence interference. Otherwise some of the gain realized by the multilevel cache hierarchy would be lost by the unnecessary blind checks and invalidations percolated to lower level of the hierarchy. The mechanism for imposing this kind of inclusion without degradation in the performance is, nonetheless, not trivial. In [15], a weak form of inclusion is imposed through "blind" invalidations of the lower level caches. In [16], the inclusion property of the direct-mapped organization is considered [1]. A more comprehensive mechanism was given in [3], where a replacement algorithm was proposed and necessary and sufficient conditions for imposing the inclusion property were given and proved for fully associative caches. In this paper we extend the results in [3] to more general set-associative cache organizations. We also investigate the effect of different block sizes on imposing the inclusion property. These results are then used to examine the impact of imposing the inclusion property on cache coherence control for three different architectures.

The rest of paper is organized as follows: Section 2 briefly reviews previous results on fully associative caches and extends them to the case of different block sizes. Section 3 proves some constraints on imposing the inclusion property for set-associative cache hierarchies. Section 4 examines the impact of the inclusion property and cache coherence on three different architectures. Conclusions are drawn in section 5.

2 MultiLevel Inclusion(MLI) Properties for Fully Associative Caches

We shall use the same memory hierarchy model as in [3]. To make this paper self-contained, we briefly state the model and the previous results for fully associative caches with the same block size. We then extend the results to different block sizes.

[1] Unfortunately, the condition for imposing the inclusion property is given incorrectly.

2.1 Multilevel cache hierarchy and MLI

A multilevel cache hierarchy consists of h levels of caches, C_h, ...,C_1 and forms a tree, i.e., each cache at level C_{i+1} is shared by one or more caches at level C_i and a cache at level C_i is a direct child of only one cache at level C_{i+1} (see the examples of Figure 7). A processor reference is serviced by the cache closest to the processor that contains the data. (We assume that all references are for real addresses.) At the same time that cache provides information to the caches on the path between itself and the processor. This procedure might displace items present in those intermediary caches. On the basis of this model, we say that:
"A MultiLevel cache hierarchy has the inclusion property(MLI) if the contents of a cache at level $i+1$, C_{i+1}, is a superset of the contents of all its children caches, C_i, at level i." This definition implies that the write-through policy must be used for lower level caches. As we will assume write-back caches in this paper, the MLI is actually a "space" MLI, i.e., space is provided for inclusion but a write-back policy is implemented.

It has been shown [11] that in a single processor environment the MLI property cannot be maintained with a local LRU algorithm. Moreover, in a multilevel cache hierarchy for multiprocessors neither a local LRU nor a global LRU (where all references to a C_i cache are percolated to its parent for rearranging the LRU stack at the C_{i+1} level) can guarantee the MLI property even when the size, or capacity, (i.e., the number of data bytes) of every C_{i+1} cache is strictly greater than the summation of the sizes of all its children C_i caches. More formally, let m_{i+1} denote the size of a C_{i+1} cache and let $m_i(k)$ denote the size of the k^{th} C_i cache which is a child of the C_{i+1} cache. Then,

Theorem 1. Under a global LRU algorithm, the conditions that the block size of caches in the hierarchy are the same and that $m_{i+1} > \sum_{k=1}^{N} m_i(k)$, for $1 \leq i < h$, are not sufficient for the MLI property to hold.

A proof by counter-example is given in [3].

A mechanism ensuring the MLI property is presented in the following section.

2.2 Necessary and sufficient conditions for imposing MLI

The MLI property can be imposed by associating with each block in a C_{i+1} cache a counter that holds the number of C_i caches in which the block is also resident. This counter has $log_2(N+1)$ bits if the C_{i+1} cache has N children. When a block is loaded in a C_i cache from its parent C_{i+1} cache, the counter for the block in the C_{i+1} cache is incremented by one. When a block is replaced in a C_1 cache, the counter in the parent C_2 cache is decremented by one. A block is a candidate to be replaced if its associated counter has value 0 (called the overflow state). Then the replacement algorithm that is used as one of the conditions for the MLI to hold is:
Algorithm I (I stands for inclusion)
(a) Lowest level (C_1): Any replacement algorithm will do (e.g., local LRU). Notify the parent cache (C_2) of the block being replaced.
(b) Other levels : Replace the block which is in the overflow state. If there is more than one, choose one. If there is none (this won't happen when MLI is enforced), randomly choose one block. Notify the parent cache of the block being replaced.

When using the above mechanism, the necessary and sufficient condition for guaranteeing the MLI property is that the size of each C_{i+1} cache is not smaller than the sum of the sizes of its children C_i caches. More formally:
Theorem 2. While using the replacement algorithm I the MLI property holds iff $m_{i+1} \geq \sum_{k=1}^{N} m_i(k)$.

The proof is given in [4].

We now extend this result to the case where C_{i+1} and C_i have different block sizes. We assume that all C_i caches have the same block size B_i although their capacities can be different. We also assume that $B_{i+1} \geq B_i$ (the opposite would not make much sense from an architectural viewpoint), and that the ratio $B_{i+1}/B_i$is a power-of-two integer 2^s (this is reasonable since the B_i's are powers of 2). We denote by q_i^j the number of blocks in the j^{th} C_i cache; thus its capacity is $m_i(j) = q_i^j B_i$. The extension of Theorem 2 to caches with different block sizes is:

Theorem 3. While using the replacement algorithm I and if B_{i+1} is an integral power of two multiple of B_i, the MLI property holds iff $m_{i+1} \geq \sum_{k=1}^{N} m_i(k) \times \frac{B_{i+1}}{B_i}$.

Proof:
Let p be the number of blocks in C_{i+1} and let $q = \sum_{j=1}^{N} q_i^j$ be the total number of blocks in its children C_i caches. The inequality in the theorem statement can be restated as:

$$pB_{i+1} \geq qB_iB_{i+1}/B_i$$

$$\text{or } p \geq q$$

Let us denote the blocks in the C_{i+1} cache by $L_{i+1}^1, L_{i+1}^2, ..., L_{i+1}^p$. Each block L_{i+1}^y of size B_{i+1} consists of 2^s sub-blocks of size B_i , say $L_{i+1}^{y,1}, L_{i+1}^{y,2}, ..., L_{i+1}^{y,2^s}$.

We prove by contradiction that it is necessary that $p \geq q$. Assume that $p < q$ and that all caches are empty. Consider the sequence of references to the first bytes of memory addresses 0, B_{i+1} , $2B_{i+1}$,..., $(p-1)B_{i+1}$. The corresponding distinct blocks of size B_{i+1} will be loaded in C_{i+1} and the corresponding distinct (sub)blocks of size B_i will be loaded in the B_i caches. Let the next reference be to memory address pB_{i+1} . There is no room in C_{i+1} for the block starting at that address while we have empty slots in the C_i caches which can receive the corresponding sub-block. Since there is no block in overflow state, the MLI property as defined previously cannot hold and hence we must have $p \geq q$.

The sufficient part of the theorem can be proved easily. Since $p \geq q$, we have a one-to-one mapping from a block in C_i to a block in C_{i+1} and we are reduced to the case of Theorem 2. □

3 Multilevel Inclusion Properties for Set Associative Caches

In Section 2, we stated the necessary and sufficient conditions to impose MLI in fully associative caches. We now treat the more realistic, and slightly more complex, case of set associative caches. Let us denote as:

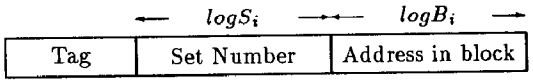

Figure 1: Cache addressing format

$Size(i)$, the capacity,

B_i, the block size,

S_i, the number of sets,

A_i, the set associativity

of a C_i cache. We restrict ourselves to the most commonly used set mapping; in this case the various fields of an address follow the scheme of Figure 1 and we have:

$$Size(i) = A_i B_i S_i$$

As before we assume that $B_i \leq B_{i+1}$ and that $B_{i+1} \ / \ B_i$ is an integer 2^s. Note that the degree of associativity of a fully associative cache is the number of blocks in the cache.

In Theorems 4 and 5 we treat the uniprocessor case. This is extended in Theorem 7 for multiprocessor architectures.

Theorem 4 considers the rather improbable case where the number of sets in the C_i cache is less than 2^s. A possible interpretation for this is to consider that C_i is a very small on-chip fully-associative translation look-aside buffer(TB) and that C_{i+1} is a back-up translation buffer with several entries (per block size B_{i+1}) being loaded at once. For example, one could have 4 entries in the on-chip TB ($S_i = 1, B_i = 1, A_i = 4$) backed up by a TB with $B_{i+1} = 8$. Theorem 4 states the conditions that must be imposed on A_{i+1} in order for MLI to hold.

Theorem 4.
If $S_i < \frac{B_{i+1}}{B_i}$, the MLI holds under replacement algorithm I iff $A_{i+1} \geq A_i \times S_i$.
Proof:

$A_{i+1} \geq A_i \times S_i$ implies that each set of a C_{i+1} cache alone already satisfies the condition of Theorem 3. The sufficient condition is readily proved.

Let us refer to Figure 2 for proving the necessary condition. From the mapping, we see that, in the worst case, S_i sets of C_i can all map to the same set of C_{i+1} . Since there are A_i blocks in a C_i set, there is a total of $S_i \times A_i$ blocks that can be mapped into the same set. If the blocks are far apart [2] and do not belong to a common C_{i+1} block, (an example of two far apart blocks is shown in Figure 2b) then the MLI cannot hold unless $A_{i+1} \geq A_i \times S_i$. □

From now on, we assume $S_i \geq B_{i+1}/B_i$.

Lemma 1.
If MLI holds then $A_{i+1} \geq A_i \times \frac{B_{i+1}}{B_i}$.
Proof:
We use Figure 3 to aid in the proof. We see that there is a total of 2^s sets which can be mapped to the same C_{i+1} set. This

[2] It is reasonable to assume that the number of tag bits are large enough to make this "far-apart" possible

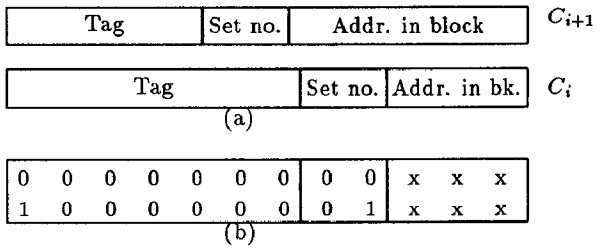

Figure 2: the case when $S_i < \frac{B_{i+1}}{B_i}$ (a) and an example of two far apart C_i blocks, which map to the same C_{i+1} set but different C_{i+1} blocks (b).

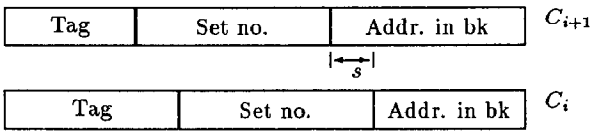

Figure 3: Situation for Lemma 1

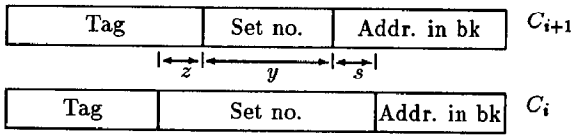

Figure 4: Situation for Lemma 2

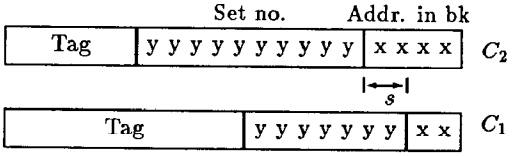

Figure 5: Example 1

implies that there are $A_i \times 2^s$ C_i blocks that can be mapped into the same C_{i+1} set. Again, as we stated in the proof of theorem 4, these blocks can be far apart and may not belong to a common C_{i+1} block. Thus, the MLI cannot hold unless $A_{i+1} \geq A_i \times \frac{B_{i+1}}{B_i}$. □

Lemma 2.
If MLI holds then $A_{i+1} \geq A_i \times \frac{S_i}{S_{i+1}}$.
Proof:
Case 1: $\frac{B_{i+1}}{B_i} \geq \frac{S_i}{S_{i+1}}$: true from Lemma 1.
Case 2: $\frac{B_{i+1}}{B_i} < \frac{S_i}{S_{i+1}}$.

This case is depicted in Figure 4. Again we see that there is a total of 2^{s+z} C_i sets which can be mapped to the same C_{i+1} set, where 2^{s+z} is $\frac{S_i}{S_{i+1}}$. This means that there are $A_i \times 2^{s+z}$ C_i blocks which can be mapped into the same C_{i+1} set. As stated above, these blocks can be far apart and may not belong to a common C_{i+1} block. Thus the MLI cannot hold unless $A_{i+1} \geq A_i \times \frac{S_i}{S_{i+1}}$. □

We show here two examples, one for each of the above lemmas, to clarify further the above necessary conditions.

Example 1 (cf. Figure 5)

C_1: $Size(1) = 512,\ B_1 = 4,\ S_1 = 128,\ A_1 = 1$

C_2: $Size(2) = 32K, B_2 = 16, S_2 = 1024, A_2 = 2$

We cannot have MLI since $A_2 < A_1B_2/B_1 = 4$. For example C_1 blocks at address 0, 16K+4 and 32K+8 map to set 0, 1, and 2 in C_1 and to the same set (set 0) in C_2. The latter needs to be at least 4-way set associative.

Example 2 (cf. Figure 6)

C_1: $Size(1) = 1024,\ B_1 = 4,\ S_1 = 256,\ A_1 = 1$

C_2: $Size(2) = 2048,\ B_2 = 16,\ S_2 = 32,\ A_2 = 4$

We cannot have MLI since $A_2 < A_1S_1/S_2 = 8$ although $A_2 = A_1B_2/B_1$. For example, C_1 blocks at address 0, 512, 1028, 1540 and 2056 map to set 0, 128, 1, 129 and 2 in C_1 and to the same set (set 0) in C_2. The latter needs to be at least 8-way set associative.

In general, the capacity of the second level cache will be much larger than the capacity of the first level cache and therefore the number of sets S_{i+1} will be greater than S_i. Therefore the situation arising in Lemma 1 will be the most common.

Theorem 5.
The MLI holds under replacement algorithm I iff $A_{i+1} \geq A_i \times K$, where K is $\max(\frac{B_{i+1}}{B_i}, \frac{S_i}{S_{i+1}})$.

Proof:
The "only if" part is proved directly from Lemmas 1 and 2. We prove the "if" part by considering two cases.

Case 1: $\frac{B_{i+1}}{B_i} \geq \frac{S_i}{S_{i+1}}$.
Suppose the MLI property does not hold; then there must be a block a which resides in C_i but not in C_{i+1} . According to the set mapping as shown in Figure 3, block a is mapped to set b of C_{i+1} where $b = \lfloor a/2^s \rfloor$. From this mapping, we know there are at most $2^s \times A_i$ blocks in C_i , including block a, that can be mapped to the C_{i+1} set b. Now the proof is reduced to a fully associative cache case, with set b of C_{i+1} corresponding to $2^s \times A_i$ C_i blocks. As the replacement algorithm I is used and $A_{i+1} \geq A_i \times \frac{B_{i+1}}{B_i}$, we know from Theorem 3 that block a can always be in C_{i+1} . This contradicts the assumption that a cannot be in C_{i+1} . Thus, the MLI property holds in this case.

Case 2: $\frac{B_{i+1}}{B_i} < \frac{S_i}{S_{i+1}}$.
Again suppose MLI does not hold; then there must be a block a that is in C_i but not in C_{i+1}. From Figure 4, we see there are at most $2^{s+z} \times A_i$ blocks including block a which are in C_i and are mapped to the same set b in C_{i+1} , where $b = mod(\lfloor a/2^s \rfloor, 2^y)$. Since $A_{i+1} \geq A_i \times \frac{S_i}{S_{i+1}} = A_i \times \frac{2^{s+y+z}}{2^y} = A_i \times 2^{s+z}$, set b has at least as many C_{i+1} blocks as the total corresponding C_i blocks. Again, set b itself can be treated as a fully associative cache and our previous result of theorem 3 shows that block a can always exist in C_{i+1} This is a contradiction; i.e., the MLI property holds. □

For purpose of completion, we state without proof the following theorem for the impractical case when $B_{i+1} < B_i$.

Theorem 6.
If $B_{i+1} < B_i$ then MLI holds under replacement algorithm I iff Size(C_{i+1}) $\geq$ Size(C_i) and $A_{i+1} \geq A_i$.

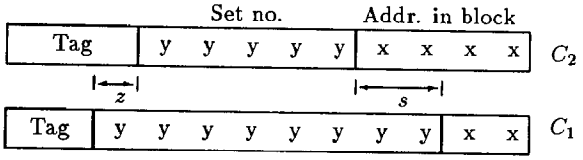

Figure 6: Example 2.

For multiprocessors where a C_{i+1} cache is shared by a number $Cmax$ of C_i caches, the above results can be easily generalized if we make the practical assumption that all caches on the same level are identical. They can be viewed as a single cache with a set associativity which is the summation of the set associativities of all caches. We include here the multiprocessor version of Theorem 5. Its proof is a straightforward extension of the proof of Theorem 5.

Theorem 7
The MLI holds under replacement algorithm I iff $A_{i+1} \geq \sum_{k=1}^{k=Cmax} A_i(k) \times K$, where K is $\max(\frac{B_{i+1}}{B_i}, \frac{S_i}{S_{i+1}})$.

As a final example that we will see later consider the organization of a level-two cache C_2 of capacity 256K bytes that is to be shared by C_1 caches of capacity 16K, direct mapped(A=1), and block size B_1=16. If we have B_2=B_1=16, then we can have 16 C_1 caches sharing C_2 as long as C_2 is 16-way set associative. If we have B_2=$4B_1$=64 and do not want to change C_2's set associativity, then we must restrict the sharing to 4 C_1 caches. Finally, if we want all 16 C_1 caches to share C_2 with B_2=$4B_1$, and keep MLI, then $C2$ must have a 64-way set associativity. It is necessary to look at alternative ways of imposing MLI(cf. Section 4.3).

In summary, in this section we have presented several conditions for having the inclusion property for set-associative cache hierarchies in which different blocks sizes are allowed at different levels. In practice we will have $Size(i+1) >> Size(i)$ (and hence $S_{i+1} \geq S_i$) and $B_{i+1} \geq B_i$. In this context, Theorem 7 states the most important result of this section, namely: In order to realize the inclusion property in a cache hierarchy, the degree of set associativity of a parent cache must be at least as large as the product of the number of its children, their set associativity, and the ratio of block sizes.

4 Impact of MLI and Cache Coherence on System Structure

The emergence of cache hierarchies and means to manage them in a reasonable manner have motivated the results of the previous two sections. Among the organizations that have been proposed, the three that we describe now seem to have attracted the most interest. Because all three limit themselves to two levels, we shall do the same here.

The first organization is simply to extend a single level cache to a two-level one. Many examples with on-chip read-only caches follow this paradigm. A more attractive architecture is to consider a shared-memory multiprocessor where each processor has a two-level cache hierarchy: A full-fledged small and fast instruction and data cache C_1 backed up by a large slower second-

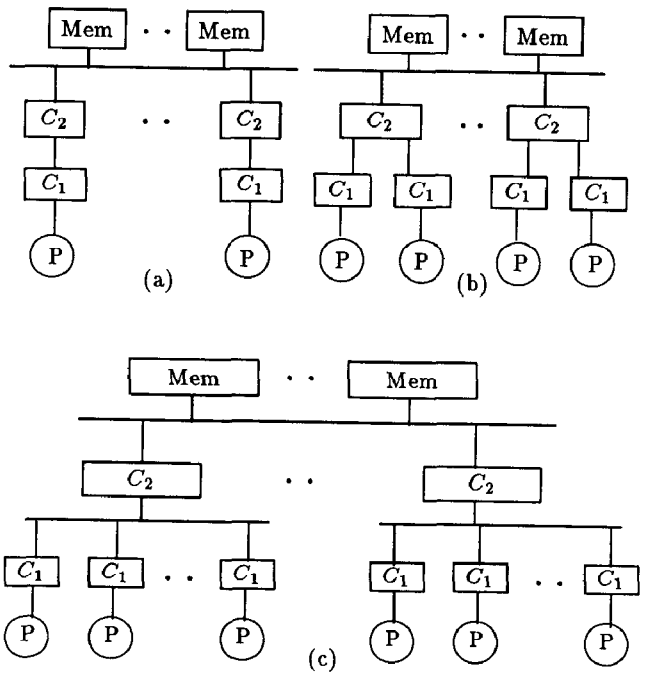

Figure 7: Three multiprocessor structures

level cache C_2 . A busing structure connecting the second-level caches leads to the multiprocessor shown on Figure 7a.

The second organization, exemplified by the Facom architecture [9] for the uniprocessor case, is what we call a multiport cache hierarchy. It consists of a second-level cache C_2 shared directly by a limited number (say at most 4) of first level caches C_1 as shown in Figure 7b. Of course, in principle this architecture can be extended to one where there would be several C_2 caches. Cache coherence between the C_1 caches is performed using a directory approach [5, 3].

The third organization, or bus-based hierarchy, consists of a very large second-level cache C_2 , "an order of magnitude larger than the sum of all the next lower caches" [15], being shared by up to two dozen first-level caches C_1 . Cache coherence between C_1 caches is achieved through some bus-based protocol [2, 14]. An extension of this architecture would consider a hierarchy of buses, i.e., the system would contain several clusters of second-level cache, first-level caches and associated processors with the clusters being connected by a common bus as shown in Figure 7c.

It is the consensus that all these existing or proposed architectures are practical only if some inclusion property is imposed (note that the hierarchy implemented on the Facom does not follow this rule and therefore it suffers from severe inefficiencies). Although we only have seen proposals where the block sizes at the two levels are the same, we show that this assumption is not always warranted if we wish to have cost-effective hierarchies. By the same token, the constraints to attain inclusion imposed in the previous section might have to be relaxed. We explore these various alternatives in the context of the three organizations of Figure 7.

4.1 Extension of a single level cache

The motivations for the replacement of a single level cache by a two-level hierarchy have been given in our introductory section. Here we examine the contradictory impacts of parameters such as cache sizes, block sizes, associativity and inclusion properties when the goals of the various components of the hierarchy are taken into account.

Consider first the benefits of a cache hierarchy. Using a trace-driven simulation (VAX traces of a mix of system and application programs), Short [13] reports a 15-20% decrease in execution time when backing up a 16K C_1 by a 256K C_2 . In his simulation for a uniprocessor, he assumes the cycle time for the second level and the memory access time to be respectively 4 and 15 times that of the first level. The same block size is assumed for C_1 and C_2 . More dramatic improvements are reported when the memory access time is 30 or 60 times that of C_1 .

Clearly, this increase in performance is not achieved without a cost – that of the C_2 cache. A better comparison, yielding approximately equal cost in a multiprocessor environment, would be to compare the above structure with a 32K C_1 that would have to include a "fast" snoopy-cache mechanism. In this case, the performance improvement, of the order of 6-8% [13], is still quite significant. Note that in the two-level hierarchy following the present organization, the snoopy mechanism needs to be implemented solely at the second level. Therefore, it does not have to be as optimized as if there were a single cache because the first level C_1 , and hence the processor, will most often be shielded by the C_2 cache from cache coherence and I/O effects. Coherence at the first level is very easy to obtain as shown below.

The improvements in performance require C_1 to be as fast as possible. This fact argues for a small, direct-mapped cache and hence a rather small block size B_1. For instance, a 16K cache with a 16 bytes block size could have a 40ns. cycle time matching a fast, but not superfast, processor. Consider now the 256K C_2 backing up this 16K C_1 cache. If we keep the same block size and allow a four-way set associativity, we would have a tag memory of (256K/16) tags of 19 bits (16 for address, one each for valid, clean/dirty, and inclusion) plus additional bits for protection and replacement algorithm purposes, i.e., of the order of 40 to 48Kbytes with most of it having to be duplicated for snooping purposes in a multiprocessor environment. Although memory is cheap, this cache is fast memory and hence not inexpensive. An alternative is to quadruple the block size in C_2 , since this change is still compatible with the previously stated conditions for inclusion, and use a sector (sub-block) organization [12]. Now the number of tags has decreased by a factor of 4 but the length of the tag is increased by 9 bits (3 bits for each of the 4 sub-blocks instead of 3 bits for the whole block) for validity, clean/dirty and inclusion states. Overall the tag memory size has been reduced by well over 50%.

Speed considerations have led us to a small C_1 cache and cost-effectiveness points towards a large sector organized C_2 cache. A simple coherence control at the C_1 level and a unified coherence mechanism between the two levels are two additional advantages that can be brought forth by the sector organization. Recall that in a sector cache [8], we can distinguish between three logical block sizes, namely:

- The block tag-size imposed by the formula:

  capacity = block tag-size × set associativity × number of sets

- The block coherence-size, i.e., the unit of size for which cache coherence is maintained. Naturally, the block tag-size is a multiple of, or equal to, the coherence-size.

- The block transfer-size, i.e., the amount of data fetched on a miss (of course the tag-size is a multiple of, or equal to, the transfer-size).

It is apparent that we should choose B_1 as the coherence-size; having a coherence-size larger than B_1 does not make sense and having it smaller complicates the logic in C_1 . Then the transfer-size will be a multiple of, or equal to, B_1. The first level cache misses, replacements, and requests for permissions to write clean blocks are directed to the second-level cache. The coherence problem between C_2 caches is solved on a sub-block (of size B_1) basis. It is only when a sub-block in C_2 has its inclusion bit set and has to be invalidated or written-back that the corresponding C_1 needs to be disrupted. The only action at the C_1 level is either to purge (write-back) or invalidate a given block. This can be implemented very simply since it requires only one line to widen the path to transmit the order and to provide a "cycle stealing" mechanism in the C_1 cache controller.

In summary, this organization is quite attractive. It combines a fast access to a small cache, a simple coherence mechanism, and the ability to have an economical large second-level cache. Its main weakness is that C_2 is completely allocated to a single processor. The other two organizations that we discuss now allow the sharing of C_2 but this raises new problems.

4.2 Multiport second-level cache

In the multiport two-level hierarchy, a large C_2 is shared by n C_1 caches. There are at least three reasons why this n must be small.

1. The multiport organization requires arbitration of access from the C_1 caches to the C_2 . The logic to do so is expensive and severely limits the value of n [7].

2. The cache coherence mechanism between the first level caches must be implemented according to a directory based scheme since the C_1 caches do not share a common bus. The amount of tag memory to do this is not trivial: It requires $n + 1$ bits in C_2 per coherence-size block if we want to avoid broadcasts [5] and 2 bits only [1] if we trade tag memory for slightly more complex protocols. In addition, the (hardware) coherence controller is not that simple.

3. n has a multiplicative effect with respect to the inclusion properties (recall Theorem 7). Thus the set associativity grows proportionally to n and this will limit very quickly the block tag-size since the set associativity is also proportional to B_2/B_1.

In order to illustrate these points, we assume the same parameters for the first-level caches as in the previous organization and the same total capacity for C_2 . That is, the size of C_2 is 256 Kbytes. Each C_1 is 16 Kbytes, direct mapped ($A_1 = 1$), has a 16 byte block size ($B_1 = 16$), and therefore has 1024 sets ($S_1 = 1024$). From Theorem 7, we know that the set-associativity of C_2 must be at least $A_2 \geq nA_1$. If we want as large a block tag-size as before ($B_2 = 64$), then we must have $A_2 \geq 4nA_1$. This gives a practical upperbound for n to be 4 resulting in a 16-way set associative C_2 .

This last figure clearly restricts the range of systems for which this organization can be practical to powerful mainframes with very tightly-coupled multiprocessing. If we want to increase the number of processors, we need to replicate the cache hierarchy and have some coherence mechanism between the second-level caches. This extension will add complexity to the level two coherence controller since the protocols will have to take into account not only level one coherence orders but also those coming from level two. Thus, the circuitry needed at the second level (extensive tagging, coherence mechanism for the two levels, very high set-associativity, multiport arbitration) becomes formidable.

This organization appears to have a limited appeal. The next organization will allow more extensive parallelism and a shared C_2 cache. However, the price to pay will be weaker inclusion properties.

4.3 Bus-based hierarchy

The bus-based hierarchy, or cluster, organization (cf. Figure 7c) can be seen as an extension of the previous two organizations. Like the first organization, the second level caches are bus-connected (inter-cluster bus); like the second one, each level two cache is shared by some level one caches and associated processors to form a cluster. The main difference, however, is the fact that we want a medium number, say 16 to 24, of C_1 caches sharing a C_2 . This requires that the C_1 caches themselves be connected by a shared-bus (intra-cluster bus). The flexibility to add C_1 caches in a cluster, or to augment the number of clusters, is not achieved without some overhead. In particular:

- As a relatively large number of C_1 caches are connected through the intra-cluster bus, a full-fledged (shared-bus) cache coherence protocol needs to be implemented at that level. The complexity of the implementation tends to slow down the cache [10] and impact on the processor cycle time.

- Imposing the inclusion property with conditions as defined in Section 3 becomes impractical since, even at equal block sizes, the set associativity of C_2 becomes too large. Since the inclusion property is still needed so that inter-cluster coherence can remain manageable, we must find an alternative to algorithm I.

A first solution to this problem, assuming equal block sizes at the two levels, is proposed by Wilson [15]. In his scheme, each time a block in a cache C_2 is to be replaced, an invalidation signal is sent on its intra-cluster bus (Wilson does not elaborate on

clean/dirty blocks and invalidate vs. purge in case of a write-back policy at the C_1 level). This policy clearly will enforce inclusion but generates unnecessary traffic on the intra-cluster bus when there is no copy of that block in any of the C_1 caches. Similarly, and maybe more importantly from the performance viewpoint, invalidations of this type might be percolated from the second to the first level to ensure the inter-cluster coherence.

In order to prevent these "blind" invalidations, we can associate an inclusion bit (IB) with each block in C_2 . IB is on if there is at least one C_1 cache which has a valid copy of the block. IB is set on any miss to a C_1 served by C_2 (this includes both hit and miss in C_2). It is reset (IB off) on a write-back from a C_1 that either invalidates all other intra-cluster copies or that is known to come from the only C_1 with a valid copy (this depends on the intra-cluster protocol).

Setting the clean/dirty (CD) bit in C_2 , where the clean/dirty property is with respect to other clusters and main memory, is not trivial. CD is set (clean) on a read miss served by C_2 and reset (dirty) on a write miss served by C_2 . Furthermore, C_2 needs to listen to the transactions on the intra-cluster bus to reset CD on any transaction that will modify the clean status of the block in any of the C_1 caches (this again depends on the level one protocol).

With the help of these IB and CD bits, we can now extend any of the MOESI protocols [14] for inter-cluster coherence with as little interference at the first level as possible. We sketch here how this could be done. In any MOESI protocol, we need at least three states: Invalid, Clean and Dirty (extra states are almost always included for improving performance). We now have a corresponding minimum of 5 states for a C_2 block, namely:

1. Invalid.
2. CleanNotI: Clean in C_2 only (IB off, CD on).
3. CleanI: Clean in C_2 and possibly some C_1 (IB on, CD on).
4. DirtyNotI: Dirty wrt main memory and other clusters but up to date (i.e., no valid copy in the C_1 's) (IB off, CD off)
5. DirtyI: Dirty and not up to date (IB on, CD off).

The inter-cluster references are then treated as follows (from the intra-cluster viewpoint).

- If the block in C_2 is in the Invalid, CleanNotI, or DirtyNotI, there is no action to be taken at the intra-cluster level.
- If the block in C_2 is in the CleanI state, no action is to be taken on a read transaction. A write or invalidate has to be percolated (as an invalidation) to level one and IB is reset. This change in IB is required since, for some inter-cluster protocols, the block in C_2 can still be valid.
- If the block in C_2 is in the DirtyI state, a purge will be sent to level one. The CD bit will be set and, depending on the type of inter-cluster transaction and the choice in implementation, the state of the block in C_2 will become either Invalid, CleanNotI or CleanI.

Finally, if a block in C_2 is to be replaced, then only when the IB bit is set will we have to send a purge or an invalidation on its intra-cluster bus.

This bus-based organization is certainly more flexible than the multiport cache one. When compared with the first organization it presents the advantage of the sharing of a C_2 cache and from this viewpoint is more economical. The drawback is the need for inclusion control. Performance studies to analyze if the first organization or if the bus-based (with or without the IB/CD control) is more cost-effective are in progress.

5 Conclusion

The multilevel cache hierarchy is a promising approach to the design of large caches. To ensure simple cache coherence protocols for systems with a multilevel cache hierarchy, the inclusion property should be imposed. In this paper, we have presented several conditions for imposing the inclusion property for fully- and set-associative cache hierarchies which allow different block sizes on different levels. Among our results, the most important one shows that to realize the inclusion property in a cache hierarchy, the degree of set associativity of a parent cache must be at least as large as the product of the number of its children, their set associativity, and the ratio of block sizes.

We have examined several organizations to study the feasibility of applying these results to ensure an efficient cache coherence control. We have found that it is feasible to satisfy the conditions in the organization which extends a single level cache in the shared-bus organization to a two-level cache. For a multiport second-level cache organization, the inclusion constraints seem to be too strict and this seriously limits the number of first-level caches. As for the bus-based hierarchy, satisfying the conditions as stated above is not practical. Instead, we have to resort to broadcast invalidations. We have presented a scheme that reduces the number of broadcasts to only those that are necessary.

Clearly some analytical and trace driven performance studies are in order to assess the usefulness and the system impacts of a multilevel cache hierarchy. We are in the process of performing such studies. As a final remark, it is interesting to note that our efforts in formally stating the conditions for multilevel inclusion can also be used in reducing the traces for trace-driven cache simulations.

Acknowledgment
This work was supported in part by NSF Grant DCR-8503250, CCR-8702915 and CCR-8619663.

References

[1] Archibald,J. and J.-L. Baer. An economical solution to the cache coherence problem. In *Proc. 11th Symposium on Computer Architecture*, pages 355–362, 1984.

[2] Archibald,J. and J.-L. Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM TOCS*, 4(4):273–298, November 1986.

[3] Baer, J.-L. and W.-H. Wang. Architectural choices for multi-level cache hierarchies. In *Proc. 16th International Conference on Parallel Processing*, pages 258–261, 1987.

[4] Baer,J.-L. and W.-H.Wang. Architectural choices for multi-level cache hierarchies. Technical Report TR 87-01-04, University of Washington, January 1987.

[5] Censier,M. and P.Feautrier. A new solution to coherence problems in multicache systems. *IEEE TC*, C-27(12):1112–1118, December 1978.

[6] Chang, J.H., H. Chao and K. So. Cache design of a sub-micron cmos system/370. In *Proc. 14th Symposium on Computer Architecture*, pages 208–213, 1987.

[7] Enslow Jr., P.H. Multiprocessor organizations – a survey. *Computing Surveys*, 9(1):103–129, March 1977.

[8] Goodman, J. Coherency for multiprocessor virtual address caches. In *Proc. Architectural Support for Programming Languages and Operating Systems(ASPOLS-II)*, pages 72–81, 1987.

[9] Hattori,A., Koshino,M. and S.Kamimoto. Three-level hierarchical storage system for FACOM M-380/382. In *Proc. Information Processing IFIP*, pages 693–697, 1983.

[10] Katz, R., Eggers, S., Wood, D., Perkins, C. and R.G. Sheldon. Implementing a cache coherence protocol. In *Proc. 12th Symposium on Computer Architecture*, pages 276–283, 1985.

[11] Lam,C-Y. and S.Mudnick. Properties of storage hierarchy systems with multiple page sizes and redundant data. *ACM TODS*, 4(3):345–367, September 1979.

[12] Liptay, J. S. Structural aspects of the System/360 model 85 part II - the cache. *IBM System Journal*, 7(1):15–21, 1968.

[13] Short, R. T. *A Study of Multilevel Cache Memories.* Master's Thesis, University of Washington, 1987.

[14] Sweazey, P. and A.J. Smith. A class of compatible cache consistency protocols and their support by the IEEE futurebus. In *Proc. 13th Symposium on Computer Architecture*, pages 414–423, 1986.

[15] Wilson Jr., A.W. Hierarchical cache/bus architecture for shared memory multiprocessors. In *Proc. 14th Symposium on Computer Architecture*, pages 244–252, 1987.

[16] Winsor, D.C. and T.N. Mudge. Crosspoint cache architectures. In *Proc. 16th International Conference on Parallel Processing*, pages 266–269, 1987.