A study on the performance of two-level exclusive caching

Enric Musoll and Mario Nemirovsky

XStream Logic, Inc.
San Jose, CA 95129
{enric,mario}@xstreamlogic.com

Abstract—

This work presents a study on the performance of a level-two cache configured as a victim storage for the evicted lines of the level-one cache. This two-level cache configuration, known as *exclusive caching*, is evaluated for a wide range of level-one and level-two sizes and associativity degrees, and the miss ratios of both levels are compared to those of the two-level inclusive caching.

Although the two-level exclusive strategy has lower miss ratios than the inclusive one by increasing the effective associativity and capacity, the replacement policy of the exclusive caching organization forces the invalidation of entries in the level-two cache, which reduces the benefits of having a victim level-two cache. The effect of these invalidations on the overall performance of a level-two exclusive caching organization is evaluated. For typical two-level cache configurations in which the level-two cache is direct-mapped, the performance of the exclusive caching is as much as 60% better for code fetches and as much as 75% for data accesses.

Keywords—**Exclusive caching, Inclusive caching, Level-two cache**

I. Introduction

Bridging the gap between fast processor cycle times and slow off-chip memory access times is still an important issue to tackle when designing a high-performance microprocessor [HP95]. Caches help in reducing this speed mismatch. With increasing levels of integration, designers have more die space for caches. Since a large cache pays off in performance, it is usual to see the cache space to be doubled from one processor generation to the next.

However, increasing the size of a cache does not come for free; the larger a cache is, the larger its access time. Thus, organizing the available cache space in two or more levels of cache is a popular technique for obtaining both a fast access and a large cache capacity [Prz90]. This is accomplished today by having a small level-one (L1) cache that can be accessed usually within one processor cycle, and a larger level-two (L2) cache that may take several cycles to access [SL88], and both levels are integrated along with the processor in the same die [Gwe98]. As performance requirements and transistor budgets increase, more levels of cache hierarchy may be seen in future generations of micro-processors.

Taking advantage of the spatial and temporal locality of the memory references, the most recently used data will be stored in the L1 cache, thus minimizing the latency of future accesses to this data. The question that arises now is what to store in the L2 cache. The answer will define which type of two-level caching strategy to use.

The most widely accepted multi-level caching strategy is called *fully inclusive*. Under an inclusive organization, the contents of the level i cache are at all time guaranteed to reside somewhere in the level $i + 1$ cache. This inclusion property imposes some restrictions on the associativity degree and number of sets of the levels of the cache hierarchy for efficiently using the space in the cache levels [BW88]. For example, when a line is evicted from level $i + 1$, it has to be invalidated in level i to maintain the inclusiveness property. These invalidations in the level i cache do not occur under certain number of sets and associativity degrees of levels i and $i + 1$. When the inclusion property is not imposed by the multi-level cache controller, the multi-level cache system becomes *partially inclusive*. These type of invalidations may degrade the performance of the level i cache. Thus, the multi-level cache controller could decide not to perform these invalidations, in which case the system becomes *partially inclusive*.

In a partially inclusive organization, thus, some data may reside in level i but not in level $i + 1$. This implies that, in a multi-processor environment, levels i and $i + 1$ will have to be snooped when another processor checks for the presence of a particular data since it can be in any of both levels. Some caches dedicate a read port just for snooping since a snoop reply usually needs to be provided right after the snoop request was issued. This extra read port increases the area and access time of the cache. In those caches that do not have a dedicated snoop port, the snoop traffic may hinder the performance of the cache since one of the read ports may be allocated by a snoop access and, therefore, it can not be used by a pending cache request issued by the processor core.

In a fully inclusive scenario, however, the rest of the processors just need to snoop the outer-most level of cache since this level contains all the data within the rest of the levels. Thus, from the point of view of simplicity of handling snoop accesses, a *fully inclusive* strategy makes more sense than a *partially inclusive* one.

Another strategy for two-level caching is called *exclusive* [JW93]. The objective of this strategy is to mitigate the problem of duplication of data that the fully inclusive cache organization has. This is accomplished by a mechanism that

guarantees that no duplicated lines are present in the cache levels. As in the inclusive strategies, most recently accessed lines are kept in the faster level-one cache. The exclusive caching has two advantages over the fully-inclusive organization:

- the effective associativity degree of the cache space is increased; and
- the capacity of the cache space is better utilized since there is no duplication of data.

However, some drawbacks arise when implementing an exclusive organization:

- the L1 cache needs to be snooped to guarantee the cache coherency; and
- the algorithm that guarantees the exclusion property is more complex than the one forcing the inclusion property. As it will be seen later, the exclusion strategy requires more accesses to both levels to guarantee the exclusion property.

Moreover, the replacement policy of the exclusive caching organization forces the invalidation of entries in the L2 cache, which diminishes the advantage of a better utilization of the cache space. The effect of line invalidations in the L2 cache on the overall performance if the exclusive caching will be evaluated.

In the sequel, this work will compare the performance of the fully-inclusive and exclusive two-level caching strategies. Having already stated the implementation differences between both strategies, the metrics that will be used to compare them will be the code fetch/data miss ratios for both levels. Moreover, the term *L2 performance* will be used to designate the performance of the two-level cache system. For the inclusive caching, the performance of the L2 is the performance of the whole cache hierarchy, but it is not for the exclusive strategy since the performance of the L1 needs to be added to the L2 one to obtain the overall cache system performance.

This work extends the study done in [JW93] with a more in-depth analysis of the miss ratios of the L1 cache plus the study of the effect that the invalidated lines in the L2 cache forced by the exclusive caching strategy have in the performance.

In Section II, the detailed algorithms used in this work for maintaining the inclusiveness and exclusiveness property in a two-level cache hierarchy are explained. Section III explains the case in which the exclusive caching strategy forces a line invalidation in the L2 cache. Section IV presents the results obtained by simulating three integer and three floating-point SPEC95 programs. Finally, Section V wraps up this work with the conclusions.

II. Algorithms to guarantee the fully inclusiveness and exclusiveness properties

This Section explains the algorithms used in this work to maintain the fully inclusiveness and exclusiveness properties in a two-level cache system. For each strategy, the actions taken in each of the four possible scenarios (L1 hit & L2 hit, L1 hit & L2 miss, L1 miss & L2 hit, and L1 miss & L2 miss) will be detailed.

In a two-level cache system, two alternatives exist for the time at which the L2 cache is accessed:

- access the L2 cache simultaneously with the L1 cache, irrespectively of whether there will be a miss in the L1. This reduces the effective access time of the L2 when there is a miss in the L1 cache; and
- access the L2 cache only when there is a miss in the L1 cache. This results in a larger effective access time of the L2 cache (the L2 latency plus the latency of determining the L1 hit/miss outcome).

The second alternative has the advantage of a potential lower power consumption since the L2 cache may be disabled (thus not consuming power) while determining the L1 hit/miss outcome. The first approach, however, consumes useless power when there is an L1 hit (which happens quite often if the L1 cache is reasonable large).

The two access strategies may also have a different behavior on the performance (i.e. the miss ratios) of the L2 cache. The reason is that when a cache is accessed, the state of the replacement policy may be modified. This modification may affect how future lines will be replaced. If the cache is not accessed (and, therefore, the replacement state is not modified), the sequence of lines to be henceforth replaced may be different compared to the previous case. If, however, the L2 cache is direct-mapped or the replacement algorithm does not depend on whether the cache has been accessed or not, then there is no performance difference between both L2 cache access strategies.

In this work, the simultaneously L2 cache access strategy is implemented for both inclusive and exclusive caching. Moreover, the replacement policy considered is LRU.

A. *Fully-inclusive caching algorithm*

- L1 hit & L2 hit
  - The requested data is obtained from the L1 cache.
- L1 hit & L2 miss (this case does not happen in a fully inclusive cache system)
- L1 miss & L2 hit
  - The requested data is obtained from the L2 cache.
  - The line containing the requested data is transfered from the L2 cache to the L1 cache.
  - If a line has to be evicted from the L1 due to the L2 cache line coming in, then the evicted line is just thrown away. If a write-though policy mechanism is implemented in the L1 cache (the most common case

in a fully-inclusive scenario), the evicted line will not be dirty.

- L1 miss & L2 miss
  - The requested data is obtained from the line brought from the third level of the memory hierarchy (a third level of cache or main memory).
  - The incoming line is placed in both the L1 and L2 caches.
  - If a line has to be evicted from the L2 cache, it may occur that it is dirty. In this case, it has to be updated into the third level of memory hierarchy; otherwise it is thrown away.
  - If a line has to be evicted from the L2 cache, the L1 cache has to be looked-up to check whether the evicted line from the L2 is present. If this is the case, this line is invalidated in the L1 cache.

B. Exclusive caching algorithm

- L1 hit & L2 hit (this case does not happen in an exclusive cache system)
- L1 hit & L2 miss
  - The requested data is obtained from the L1 cache.
- L1 miss & L2 hit
  - The requested data is obtained from the L2 cache.
  - The line containing the requested data is invalidated from the L2 cache and transfered to the L1 cache.
  - If a valid line has to be evicted in the L1 due to the L2 cache line coming in, the evicted line is placed in the L2 cache. In this case, if a line has to be evicted from the L2 cache to make room for the incoming line, it may occur that it is dirty, in which case it has to be updated into the third level of memory hierarchy; otherwise it is thrown away.
- L1 miss & L2 miss
  - The requested data is obtained from the line brought from the third level of the memory hierarchy.
  - The incoming line is placed in the L1 cache.
  - If a valid line has to be evicted in the L1 due to the line coming in, the evicted line is placed in the L2 cache. In this case, if a line has to be evicted from the L2 cache to make room for the incoming line, it may occur that it is dirty, in which case it has to updated into the third level of memory hierarchy; otherwise it is thrown away.

Table I shows, for each of the four scenarios and for both caching strategies, the number of cache accesses to both levels. An access is defined to be any look-up, invalidation or update of a cache (at least two accesses in each case occur, since both levels of cache are looked-up in parallel). The Table shows that, for the most common scenario (L1 hit & L2 hit for the fully inclusive strategy, and L1 hit & L2 miss for the exclusive one), the number of accesses is the same. However, for the second most common case (L1 miss & L2 hit for both), the exclusive strategy presents two more cache accesses. This shows that the algorithm to force the exclusiveness property is more complex than the one to impose the fully inclusiveness property.

TABLE I
NUMBER OF L1 AND L2 CACHE ACCESSES.

Scenario	Caching Strategy	
	Fully Inclusive	Exclusive
L1 hit & L2 hit	2	NA
L1 hit & L2 miss	NA	2
L1 miss & L2 hit	3	4
L1 miss & L2 miss	5	4

The last scenario in Table I comprises, for example, all the cold misses. In this scenario, forcing the inclusiveness property requires one more access to the cache system.

III. L2 INVALIDATIONS IN THE EXCLUSIVE CACHING

In the previous Section it has been shown that, for the exclusive caching strategy, a line in the L2 cache (henceforth named $\texttt{line}_2$) will always be invalidated in the rather common L1 miss & L2 hit scenario. This invalidated line will be placed in the L1 cache, which may cause a different, valid line from the L1 cache (henceforth named $\texttt{line}_1$) to be evicted and sent to the L2 cache. At this point, two situations may happen (let $L2Set(\texttt{line}_j)$ represent the L2 set index in the L2 cache for line $\texttt{line}_j$):

(a) $L2Set(\texttt{line}_1) \neq L2Set(\texttt{line}_2)$. $\texttt{line}_1$ will not be placed into the same location where the invalidated $\texttt{line}_2$ originally was. Therefore, a potential valid line from the L2 cache (henceforth named $\texttt{line}_{2E}$) will have to be evicted to make room for the incoming $\texttt{line}_1$.

(b) $L2Set(\texttt{line}_1) = L2Set(\texttt{line}_2)$. $\texttt{line}_1$ will be placed, most likely, in the same location where the invalidated $\texttt{line}_2$ originally was because the replacement algorithm tries to find an invalidated spot to place the incoming line.

The only case in which it may not go to the same place is when there are more invalid spots (in other ways) in the same set. In this case, the replacement algorithm may choose any of the invalidated spots to place the incoming line. This case, however, is not very common since the cache is usually filled up with valid lines.

Therefore, when situation *(a)* happens, a "hole" (i.e. an invalidated spot) is created in the L2 cache. Let this situation be henceforth named as an *non-exact swap* between L1 and L2. Although the original content of this spot (i.e. $\texttt{line}_2$) is not lost (since it was placed in the L1 cache) a line in the L2 ($\texttt{line}_{2E}$) could have been evicted out of the two-level cache system[1]. However, if $\texttt{line}_{2E}$ is referenced in the near future

[1] However, not all the holes have a negative impact on the L2 cache performance.

it will result in a miss to the cache system. In situation *(b)*, however, no hole is created since $\texttt{line}_{2E}$ fills the temporarily created hole. Let this situation be henceforth named as an *exact swap* between L1 and L2.

The amount of non-exact swaps may degrade the performance of the exclusive caching versus the case in which only exact swaps occur. To evaluate this loss of performance, a modified exclusive caching algorithm will be used in which only exact swaps occur. By comparing this modified algorithm with the original exclusive caching algorithm, the increase in L2 miss ratio due to the non-exact swaps is obtained. The modified exclusive algorithm (henceforth named *no-holes*) performs a swap only if this swap is exact. This algorithm, however, has the drawback of a larger L1 miss ratio since the last referenced line that hit in the L2 is not guaranteed to be placed into the L1.

IV. Results

This Section presents the results obtained by simulating three integer (`compress`, `gcc` and `li`) and three floating-point (`fpppp`, `tomcatv` and `vortex`) SPEC95 programs. In the sequel, the results are given for the average of all six programs.

The L1 sizes considered in this work are 8K, 16K and 32K, and with associativity degrees of 1, 2 and 4. For the L2 cache, the sizes chosen are 32K, 128K and 512K, with associativities of 1, 2 and 8. In all the cases, a line size of 32 bytes is considered. Moreover, L1 and L2 are both unified.

First, the performance of the fully-inclusive and exclusive two-level caching strategies is compared. The metrics that will be used to compare them will be the code fetch and data miss ratios for both levels. For the exclusive scenario, the L2 miss will denote the whole cache system miss ratio; for the fully-inclusive case, the L2 miss ratio already corresponds to the whole cache system miss ratio. In this work, the code fetch miss ratio is defined per line of instructions, i.e., the cache is accessed every time a new line of instructions is required.

Afterwards, the exclusive and no-holes are compared to evaluate the variation in the L2 miss rate due to the non-exact swaps that the exclusive caching presents.

In all the plots shown in this Section, the `x` and `y` axes are labeled as `R_S_T_U`, where `R` is either the L1 or the L2 cache, `S` is the size of the cache (in KBs), and `T` is the associativity degree. For the L2 case, `U` denotes whether the two-level cache system is either fully-inclusive (I), exclusive (E) or no-holes (N).

A. Fully-inclusive vs. Exclusive caching

Figure 1 shows the miss ratio, for code and data, of all the design points considered. For the exclusive caching, the L1 miss ratio does not depend on the associativity degree of the L2 cache. However, for the fully-inclusive strategy, the L1 performance is affected since evicted lines from the L2 will have to be invalidated in the L1 to guarantee the inclusiveness property.

Figure 1 shows that, for a direct-mapped 32K L2 cache, incrementing the associativity of a 32K L1 cache does not decrease the miss ratio of the L1 cache. This is due to the fact that the L1 cache behaves approximately as a direct-mapped cache independently of its associativity since the L1 cache can not have better performance than its L2 cache in a fully-inclusive scenario. Additionally, in the 2-way L1, a degradation on the performance is observed (compared to the direct-mapped L1) due to the fully-inclusiveness algorithm used in this work[2].

The performance of the L1 is always better for the exclusive caching than for the fully-inclusive. The less associativity the L2 has, the worse is the L1 performance for the fully-inclusive case. Moreover, the larger the L2 cache is, the closer is the performance of the L1 for both strategies. For the particular case of a 16KB 2-way L1 and 128KB direct-mapped L2, the data miss ratio of the L1 is reduced by around 12% in the exclusive caching. For code fetches, the difference is less pronounced than for data accesses.

Figure 2 shows the study for the L2 cache. For the fully-inclusive caching, the performance of the L2 is independent on the configuration of the L1. The exclusive caching presents always smaller miss ratios than its inclusive counterpart. The difference is larger for smaller and less associative L2 caches.

There are two reasons for which the performance of the L2 cache in the exclusive caching is better than in the inclusive one: the increase in the effective associativity and the increase in capacity. To illustrate how the increase in effective associativity affects the performance, consider the case [L1: 8KB,DM; L2: 512KB, DM]. The increase in capacity is very small (1.6%); however, the miss ratio for data accesses decreases about 25% in the exclusive caching. This improvement is almost entirely due to the increase of the effective associativity of the exclusive caching.

To illustrate the effect of the increase in capacity, consider the following two cases: [L1: 8KB,DM; L2: 32KB, 8W] and [L1: 32KB,DM; L2: 128KB, 8W], which have the same increase in capacity (25%) and the L2 has a large associativity degree (and, therefore, the extra associativity that the L1 provides is unlikely to significantly improve the L2 performance). The miss ratio decreases about 24% for the first case and about 17% for the second. Note that the larger the L2 cache is, the smaller is the increase in the L2 performance. The increase of performance comes from the large increase in capacity.

[2]The incoming line from the third level of cache is placed in parallel in both the L1 and the L2 caches. Another implementation could wait to place the line in the L1 until the evicted line of the L2, if any, is known; in this case, if the evicted line in the L2 causes an invalidation in the L1, the incoming line will be placed in this invalidated spot in the L1.

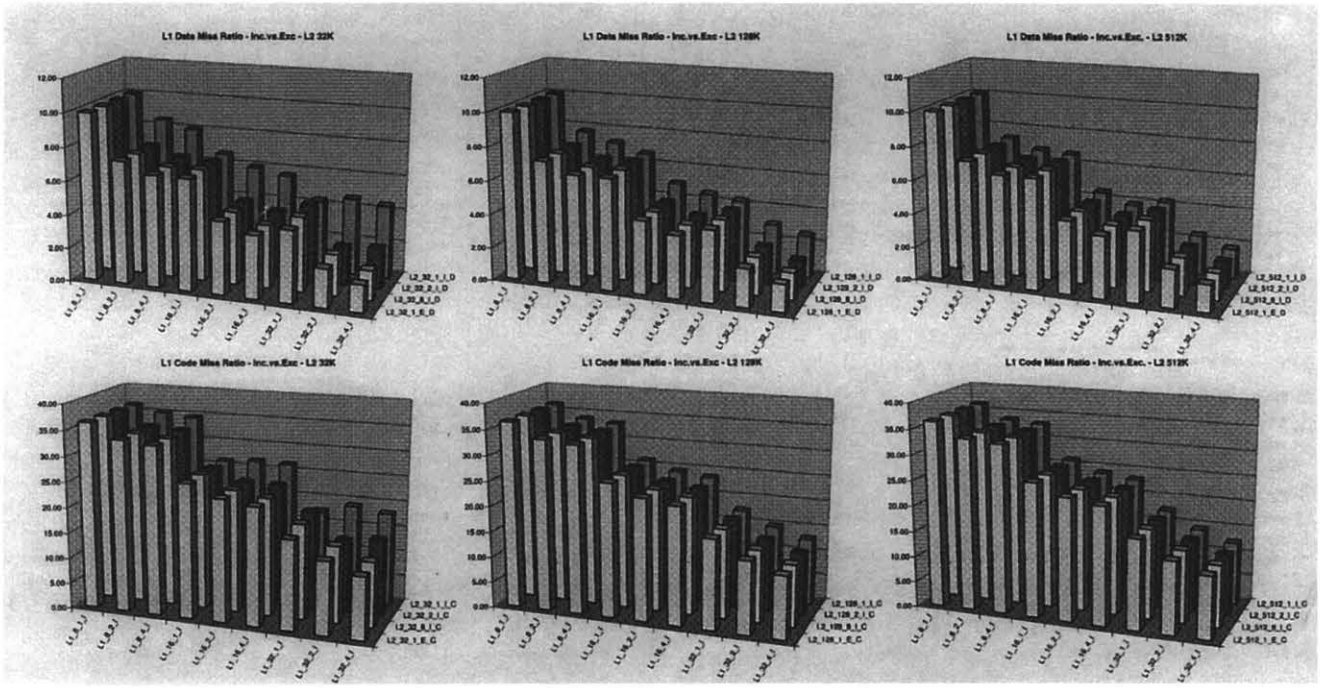

Fig. 1. L1 miss ratio results. Inclusive vs. Exclusive.

Thus, both the increase in associativity and capacity makes the exclusive caching better than the fully inclusive. However, this increase in performance could be higher if only exact swaps occur in the exclusive caching, as explained in Section III.

B. Exclusive vs. No-holes caching

Figure 3 compares the exclusive and the no-holes caching strategies to evaluate the degradation of the L2 performance due to the non-exact swaps. Note that the L1 performance is not studied since it is very low for the no-holes caching.

The Figure shows that the non-exact swaps have almost no effect on the L2 performance for the cache configurations studied in this work, we have observed that, in average, at most 12% of the non-exact swaps in the exclusive caching negatively affect the performance. This maximum percentage corresponds to the case in which both the L1 and the L2 caches are 32KB and direct-mapped. For larger L2 caches, and also for larger L2 associativity degrees, this percentage soon drops to almost zero. Moreover, the number of non-exact swaps correspond to less than 10% of all the accesses to the cache system. In any case, the miss ratio increase due to non-exact swaps is, in average, always less than 15%.

Even with this decrease in L2 performance, the exclusive caching strategy has always better L1 and L2 performances than those of the fully-inclusive caching. For typical two-level cache configurations, the performance of the exclusive caching is as much as 60% better for code fetches and as much as 75% for data accesses.

V. Conclusions

This work presents a study on the performance of the two-level exclusive caching strategy. The exclusive caching is compared against the fully-inclusive one for 6 SPEC95 programs and for several sizes and associativity degrees of level-one and level-two caches.

The exclusive caching strategy always presents better performance (in the level-one and the overall cache) than the fully-inclusive one. For typical two-level cache configurations in which the level-two cache is direct-mapped, the performance of the exclusive caching is as much as 60% better for code fetches and as much as 75% for data accesses.

References

[BW88] J-L. Baer and W-H. Wang. On the inclusion properties for multi-level cache hierarchies. *15th Symp. on Computer Architecture*, pages 73–80, June 1988.

[Gwe98] L. Gwennap. Shift to on-chip cache pays off. *Microprocessor Report*, 12(16), December 1998.

[HP95] J.L. Hennessy and D.A. Patterson. *Computer Architecture: a quantitative approach*. Morgan Kaufmann Publishers, 2nd edition edition, 1995.

[JW93] N.P. Jouppi and S.J.E. Wilton. Tradeoffs in two-level on-chip caching. Technical report, Western Research Lab, Compaq, October 1993.

[Prz90] S.A. Przybylski. *Cache and Memory Hierarchy Design*. Morgan Kaufmann Publishers, 1990.

[SL88] R.T. Short and H.M. Levy. A simulation study of two-level caches. *15th Symp. on Computer Architecture*, pages 81–88, June 1988.

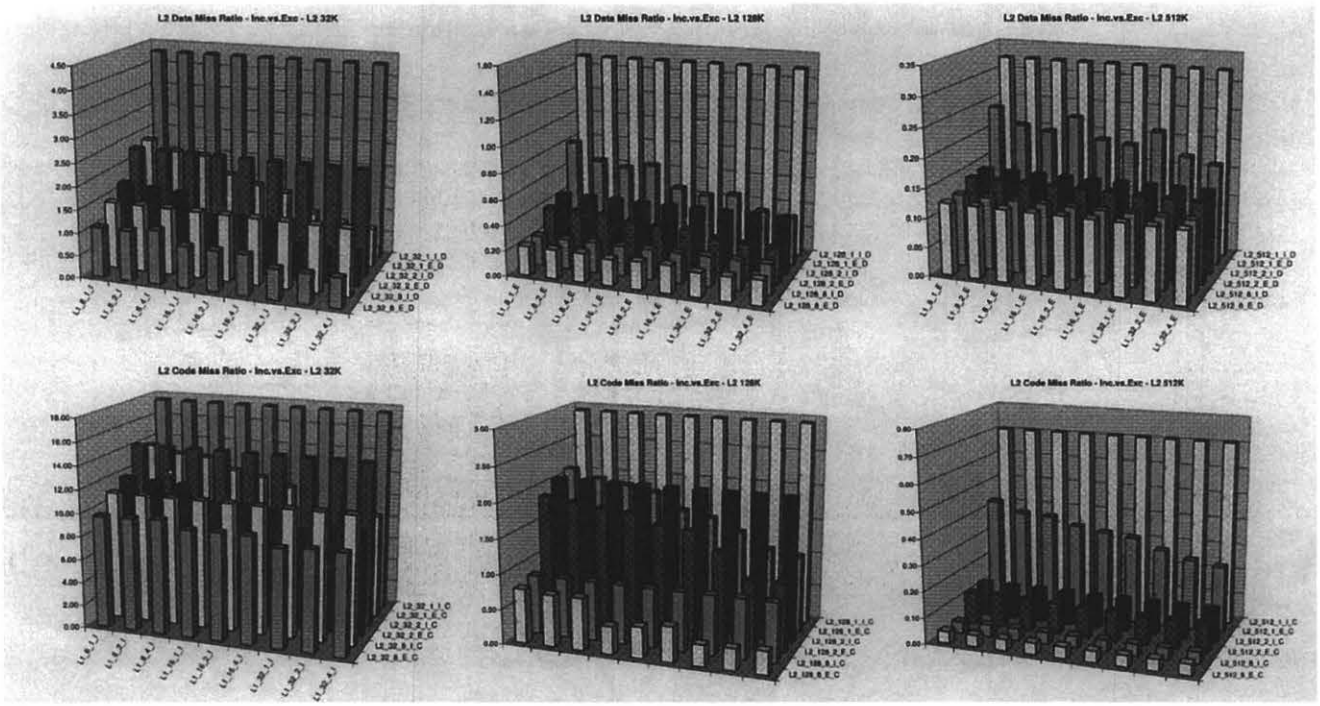

Fig. 2. L2 miss ratio results. Inclusive vs. Exclusive.

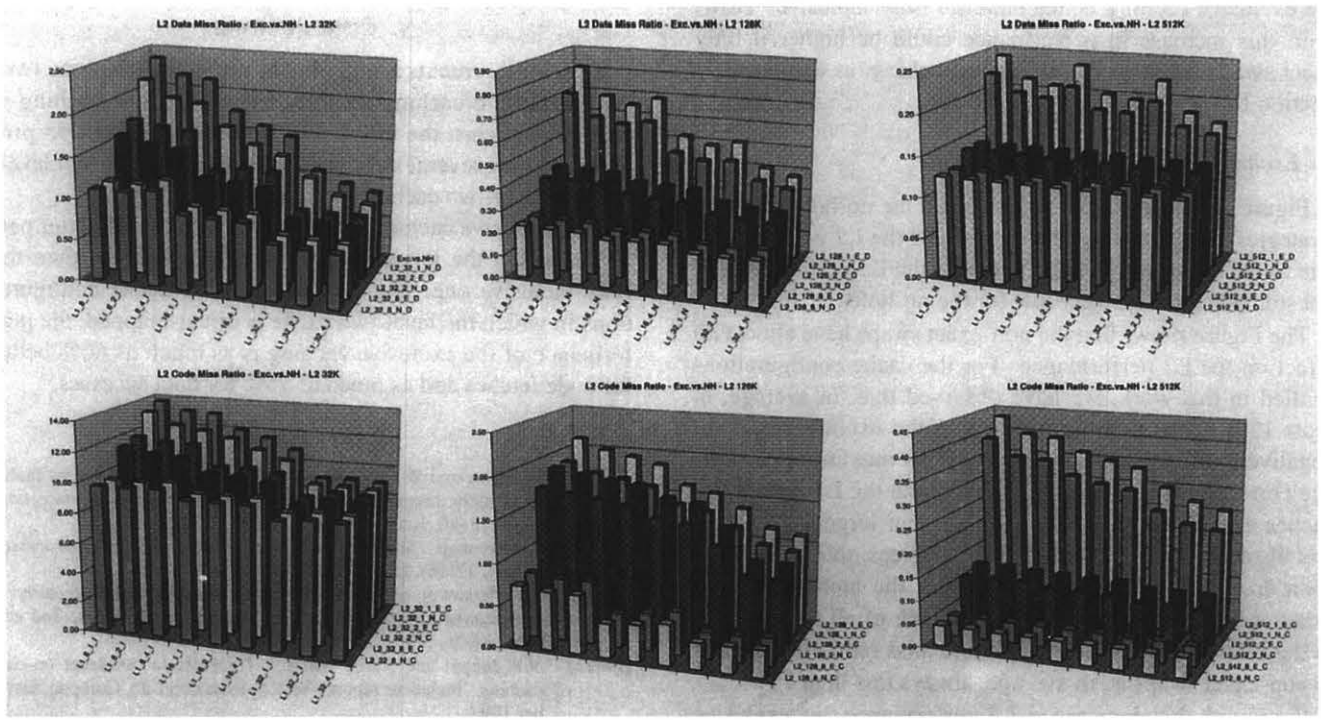

Fig. 3. L2 miss ratio results. Exclusive vs. No-Holes.