Dogo Rangsang Research Journal
ISSN : 2347-7180
UGC Care Group I Journal
Vol-08 Issue-14 No. 01 February : 2021

A Survey of the Non-Inclusion Property in Non - linear and non Caches

Dr. AMARESH SAHUDr. B.PURNA SATYANARAYANA
Dept. OF Computer Science and Engineering, NIT , BBSR
amareshsahoo@thenalanda.com*, bpurnasatyanarayana@thenalanda.com

Abstract

Memory systems are becoming the focal point of computer architecture. We are expected to witness an increase in on-chip cache memory in the coming years, barring the development of novel micro- architecture techniques that boost processor speed. Recent designs frequently use three tiers of on-chip cache memory. More advanced cache design methodologies and possibly a reevaluation of some cache concepts are required due to the increasing reliance on on-chip caching. The inclusion property, in our opinion, is a strong contender for these ideas. This characteristic, while simplifies memory coherence protocols in multiprocessor systems, duplicates data over many levels of cache, which results in an inefficient use of cache memory space on the chip. In addition, stringent enforcement of the inclusion condition implies a "ripple effect" during updates, where one update at one cache level may trigger many updates at higher levels of the hierarchy. Many non-inclusion cache techniques for upcoming cache systems are covered in this research. We offer a couple of design options for non-inclusive cache designs. We demonstrate that the primary benefit of a non-inclusive cache design is its comparatively high level 2 (L2) hit rate, which reduces the average memory system access time. L2 miss rates for specFP and INT are each reduced by 40% and 28%, respectively, via non-inclusive cache.

Key Words: Cache memory, cache access time, memory performance, inclusion property, multi-level cache.

1 Introduction

Cache memories have been used to improve the performance of computer systems by exploiting temporal and spatial locality characteristics for many years [14]. The very rapid advances in process technology means that a larger transistor budget is available to architects each year. In addition to many innovative microarchitectures to fully utilize this budget with execution logic, we witness the emergence of larger and more sophisticated on-chip caches in every consecutive generation of processors. Because larger cache size means slower cache, the trend will be toward increasing the length of cache hierarchy, that is, increasing the number of cache levels. A quick glance at die photos of recent processors should be sufficient to notice the reliance of architects on cache memory to put chip area to use.

With the advent of multiple CPU cores on a chip, more and more sophisticated on-chip caches are appearing on the scene while the sizes are growing at the same time. At the far end of the complexity spectrum, IBM's POWER4 architecture [16] has a 1.5MB L2 cache shared among its two processor cores and organized as three slices, the IBM's POWER5 has L2 cache of size 1.875MB with a 36MB off-chip L3 [13], and Intel Itanium [18] has a 3-level on-chip cache with combined capacity of 3MB. As the size and complexity of on-chip caches increase, the need to decrease miss rates gains additional significance, together with access time.

The inclusion property, which dictates that the contents of a lower level cache be a subset of those of a higher level cache, is highly desired in a multiprocessor system primarily because it facilitates memory controller and processor design by limiting the effects of cache coherence messages to higher levels in the memory hierarchy. Overall performance is improved when the lower level caches are isolated from the effects of coherence checks and invalidations by the inclusion property. However, a cache design that enforces inclusion is inherently wasteful of space and bandwidth: Every cache line in lower levels is duplicated in the higher levels, and updates in lower levels trigger many more updates in other levels, wasting bandwidth. If the current trends of larger cache lines and more sophisticated caches continue, the inclusion property should be considered a prime candidate as part of a rethinking of multi-level cache design.

The main advantages that can be gained by forcing non-inclusion are the following.

- Context switches will be faster, because fewer messages will need to be moved up the hierarchy for the write backs and invalidations.
- The effective size of the cache system increases because we are getting rid of data duplication. This is very important when two consecutive cache levels have large cache size, namely L2 and L3 caches, or when using Chip-multiprocessor and the aggregate level 1 caches are

similar in size to level 2 cache, as in Piranha [3].

- Conflict misses in the second level cache are reduced. This is due to the fact that heavily referenced blocks are moved to the first level cache, leaving room for other blocks to come in the second level cache. This implicitly increases the associativity of the second level cache assuming that the non inclusion will be applied to level one and level 2 caches.
- We can save in bandwidth, because fewer updates will need to be done when a block is dirty and is written back to memory. In which case, the block will not need to be written back to all the higher level caches in the hierarchy until reaching the memory.

In this paper, we aim to gain insight into the effects of non-inclusive multi-level caches on performance. Several different designs where the caches up and down the hierarchy are mutually exclusive presented, their performance is compared to a basic cache architecture.

We will also discuss potential solutions to cache coherence, when non-inclusive caches are used in multiprocessor systems.

The rest of the paper is organized as follows. Section 2 gives an overview of the related work on improving cache performance through caches that may violate inclusion property. The proposed model, called non-inclusive cache (NIC) is presented in Section 3. The NIC is evaluated experimentally in Section 4, followed by discussion of findings. Section 5 concludes and summarizes the paper, and outlines some of the future work we intend to do on this subject.

2 Related Work

Important issues in cache memory design and multi-level on-chip caches have been studied in great detail. A comprehensive introduction to cache concepts can be found in [14] and multi-level cache design issues are introduced in [19].

The main idea and concept behind the inclusion properties for multi-level cache hierarchies was analyzed by Baer et al. in [2]. The possibility of relaxing the inclusion property has been identified in some details in several studies. The most extensive work has certainly been done within the context of prefetching, which implies the violation of inclusion property in many cases. An extensive survey of prefetching techniques is presented in [17]. Also non-inclusion has been used in Piranha [3], because the aggregate L1 capacity of all the processing elements of the chip multiprocessor is 1MB, and maintaining inclusion with the 1MB L2 available wastes space due to the duplicate data. However, in order to maintain intra-chip coherence, duplicate copies of L1 tags and state are kept in L2. An algorithm for exclusive cache hierarchies has been studied in [20], and showed some improvement, however, at the expense of the hardware complexity.

As part of their work on tradeoffs inherent in on-chip multi-level cache design, Jouppi and Wilton [19] proposed an exclusive caching scheme similar to the swapping scheme we analyzed. They found that a non-inclusive cache strategy is fairly effective in reducing level 2 conflict misses. In addition to finding that a combination of set associativity and exclusive caching yielded performance improvements, they also suggested that maintaining the inclusion property between the sum of the first two levels of caches and a third level of off-chip caching can be a solution to the problem of simplifying the design of cache-coherent multiprocessors when exclusion is used.

McFarling [8] described a multi-level cache design called dynamic exclusion which used tags to denote in which level in the hierarchy an instruction cache block will be kept. Using a finite state machine to identify instruction cache access patterns, this approach resulted in important reductions in the miss rate of direct-mapped instruction caches.

Run-time cache bypassing methods [5, 6] propose using memory reference behavior to place the data in the cache hierarchy. This method uses sets of cache blocks called macroblocks and maintains a central table (MAT: Memory Address Table) to keep track of dynamically inferred usage patterns. This data is then used to move cache blocks in the hierarchy, violating the inclusion property at times.

Another interesting idea, is partitioning L1 cache into mutually exclusive cache in order to overcome wire delay, is presented in [9]. Based on the results of previous work done on the subject, it is evident that the success of a particular non-inclusion strategy depends heavily on the method by which cache blocks are moved up and down the cache hierarchy. This is discussed in the next section.

3 Non-Inclusive Cache (NIC) Model

In a conventional cache memory system, when there is a cache miss at any level, the block is brought from the memory into *all* cache levels. If this cache block is rarely referenced again, then it has a high chance of displacing some more useful data from the cache, which consequently degrades the overall performance of the cache. This problem, of useful data being displaced by less useful ones, can be tackled in two different ways. The first method keeps track of the reference patterns, and the references that are less likely to be referenced again are not brought into the cache, and are brought directly to the processor. An example of this method is the run-time cache bypassing methods [5, 6]. The second method is relaxing the inclusion property.

The inclusion property can be categorized into three main groups. The first one is the inclusive scheme. This is the conventional one where the lower level caches[1] are subsets of the higher level ones. This has been considered the standard due to its simplicity. Cache hierarchy in almost all the current processors is inclusive. The second category is the partially inclusive cache hierarchy. In this case, some blocks may be included in both cache levels and some are not. The third category is the mutually exclusive cache hierarchy where a block cannot be present in two cache levels at the same time. The second category needs a lot of bookkeeping, does not make use of the whole cache area, and does not simplify coherence. Hence, we will concentrate in this paper on the

[1] Throughout this paper, when we mention *lower level*, we mean the level closer to the processor.

third category, which is the mutually exclusive cache. From that point on, when we mention non-inclusion, we will be talking about mutually exclusive caches.

The duplication of data that results initially from the inclusion property can be avoided if we use a non-inclusive cache system. Non-inclusion relaxes the constraint of each cache level being a superset of the higher levels in the hierarchy. Hence, a block can exist in level 1 cache without necessarily existing in level 2 cache. However, the blind application of this concept might not yield the expected performance.

A crucial factor for the high performance of such a system is the algorithm by which blocks move up and down in the cache hierarchy. We have chosen to evaluate three different non-inclusive cache designs of varying sophistication. While we assumed that the memory hierarchy only contained two cache levels, the ideas presented in this paper can be extended to cache hierarchies with an arbitrary number of levels. The cache block sizes are assumed to be equal for all levels that violate inclusion, for simplicity but without loss of generality.

Basic: This is the most basic scheme, and is used to test the validity of the basic concept of non-inclusion. In this scheme a miss in both cache levels brings the block into level 2 from the memory, and the required data is delivered to the processor. We have decided to opt for this scheme of moving the newly imported block to level 2 instead of level 1 in order not to pollute level 1 with a block that may be referenced only once. If the block is referenced for a second time, before it is replaced from level 2, then it means it has temporal locality, and thus, gets upgraded to level 1 cache. Therefore, the main difference from a traditional inclusive cache hierarchy is that the processor can accept data from either level 1 cache or level 2 cache, but not both. If level 1 misses and level 2 hits, the block at level 2 migrates to level 1, and the displaced block from level 1 (if any), moves to level 2. Hence, level 2 cache acts as a victim cache [7] for level 1. We do not access L1 and L2 simultaneously, as this will not be power efficient.

Autonomous Prefetching (AP): This scheme is similar to the basic scheme, except that when the block migrates from level 2 to level 1, the successor of that block is prefetched into the level 2 cache. This scheme can be developed further by using well-established techniques from the prefetching literature [17], like prefetching multiple successor blocks, at the cost of higher complexity. Figure 1 gives the state transition diagram of the basic and AP methods.

It is to be noted that a hit in L1 does not trigger any actions or state transition in the basic or AP methods.

Controlled Swapping (CS): Represents a further enhancement over the basic scheme. This scheme introduces a saturating counter for each block in both cache levels. Our counter concept borrows from the run-time cache bypassing concept [5, 6] to provide a simple means of tracking reference patterns. If the block hits at level 1, its counter is incremented. If the block hits at level 2, its counter is incremented and compared to the counter of the block at level 1, that would have been displaced if the block at level 2 migrates to level 1. If the counter at level 2 is higher than the one at level 1, the swapping is done in a similar fashion to the basic scheme, and both counters are reset. However, if the counter at level 1 is higher or equal, the block that hit at level 2 remains at level 2 and the counter at level 1 is decremented. If both cache levels miss, the block is brought into level 2 and both its counter and the corresponding counter at level 1 are reset.

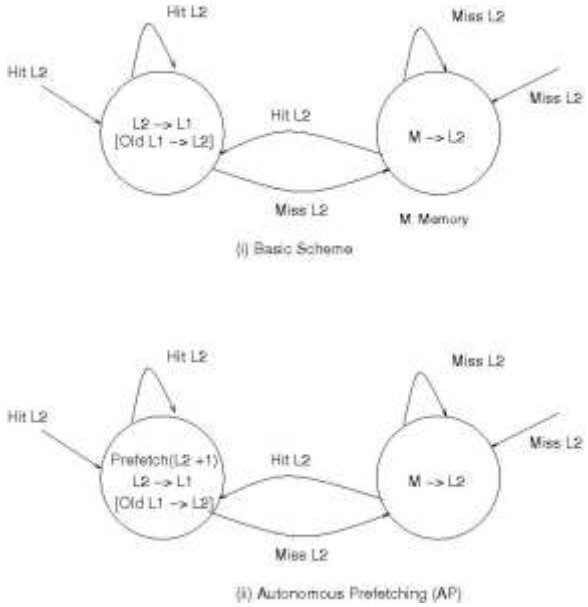

Figure 1: Non-inclusion state diagram

Figure 2 shows the main actions which are done by a non-inclusive cache. First, a block comes from the memory to the

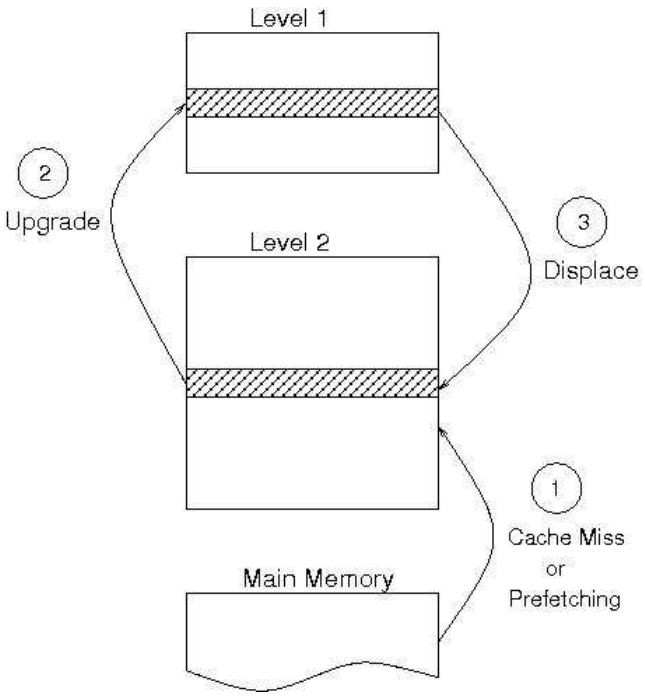

Figure 2: Main actions in non-inclusive cache system

L2 cache. This is better than bringing it directly to L1, because L1 is checked first and is the fastest, hence we need to be careful not to displace useful data from it. If we find that the block is referenced several times, then the second action is the *upgrade*, where a block moves to a level closer to the processor. This happens when a block is referenced several times (twice in the basic system and the autonomous prefetching, and depending on the counters in the controlled swapping). When a block is upgraded, it may *displace* another block, depending on the associativity of the cache in the lower level. The upgrade and displace actions form a kind of swapping. The inclusive cache does not have this swapping action.

It is to be noted that the block is also brought from memory in case of prefetching. Prefetching is more efficient in case of non-inclusive cache because it has more space for data, due to the elimination of duplicating data.

Hardware Cost

The cost of having non-inclusive cache is not high. It is similar to the hardware needed for run-time memory management for example. For the basic system, all we need are two multiplexers, and a buffer. The first one is at the vicinity of the processor, to choose data coming from L1 or L2. The second multiplexer is connecting the non-inclusive cache hierarchy, to the main memory or the higher level cache, similar to what is shown in Figure 3. In case of an L1 miss and L2 hit, the block at L2 migrates to L1, and the corresponding block at L1, if any, moves to L2. There are two moves here, from L1 to L2, and from L2 to L1. This swap operation needs a buffer in order to be accomplished. The displaced block from L1 is copied to the buffer, then the new block is brought from L2. Finally, the block at the buffer is copied to L2.

For the AP system, the extra hardware needed, in addition to what is needed for the basic system, is the prefetching hardware. The amount of hardware needed here depends on the type and technique of prefetching [17]. A comparison of the different hardware prefetching techniques and their cost is beyond the scope of this paper. However, the technique presented in this paper requires a small adder to calculate the address of the data to be prefetched.

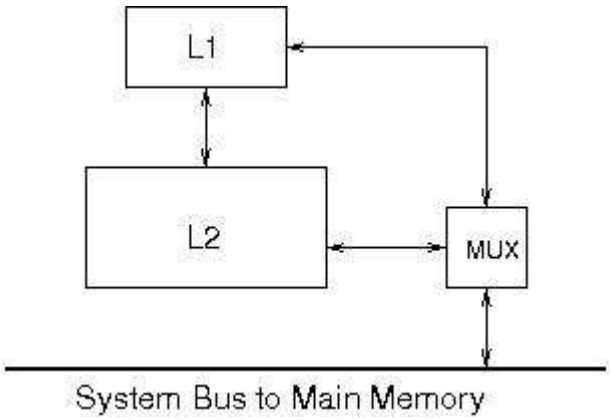

Figure 3: A suggested design of two level non-inclusive cache

Finally, for the CS system, the extra hardware required, above the basic non-inclusive system, is the addition of a counter for each cache block at both L1 and L2 cache. From our preliminary experiments, we found that a two-bit counter is enough for efficient working. Moreover, a comparator is needed to compare the two counters, and trigger block swapping, if needed.

Taking into account the large cache sizes, the hardware requirement for a non-inclusive cache is not high.

NIC in Multiprocessor Systems

The main reason the inclusive caches have gained such wide acceptance is its simplicity in handling coherence in case of multiprocessor systems. However, non-inclusive caches can also handle coherence with easiness. There are many solutions for handling coherence using non-inclusive caches.

- With systems that have three levels of caches or more, such as Intel Itanium [18], we can have non-inclusion between L1 and L2, and inclusion with L3 or the level nearest to the system bus, as shown in Figure 4.
- A technique used in Piranha [4] is to duplicate the tags of L1 in L2 cache, or the cache nearest to the bus.
- Another way of handling coherence in non-inclusive environment is to have a separate table with the tags of all the non-inclusive caches. This table is used for invalidation, using directory-based techniques instead of snoopy.

However, evaluating these techniques is out of the scope of this paper and will not be considered further.

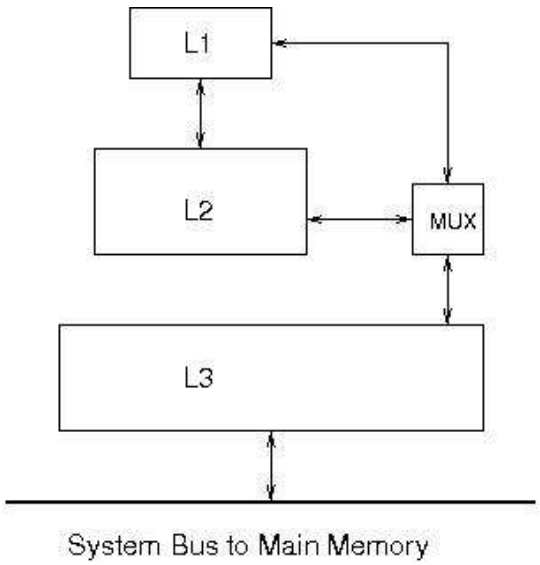

Figure 4: L1-L2 non-inclusive and L3 inclusive

4 Experimental Evaluation

In this section we present a detailed quantitative evaluation of the presented technique. Such an evaluation is important to

study the performance gain that can be obtained from the NIC model, and to see how efficient it is with standard benchmarks.

Experimental Methodology and Setup

For our microarchitectural simulations, we used a modified version of the out-of-order processor simulator of the Simplescalar 3.0 tool set [4], for the PISA (portable ISA) instruction set. Table 1 shows the parameters we used for the simulation. Although the table shows the usage of a bimodal branch predictor, we have obtained similar results by using a hybrid branch predictor of two-level and bimodal. However, due to space as well as to avoid redundancy, we have not included those results. Cache latencies have been obtained using CACTI [12] to get the access latency. We used integer and floating point benchmarks from the SPEC2000 suite with reference inputs. The benchmarks have been compiled using the Simplescalar PISA gcc cross-compiler with the optimizations specified in the makefile provided with the suite. Each benchmark was simulated for 500M instructions after skipping the startup phase as described in [10].

Table 1: Simplescalar simulator parameters

Parameter	Value
Decode Width	4
Issue width	4
Commit width	4
Instruction Fetch Queue Size	4
Branch Predictor	Bimodal with 2048 table size
Instruction Fetch Queue Size	4
Load/Store Queue Size	4
BTB Configuration	512 sets, associativity 4
Return Address Stack Size	8
L1 – Icache	32KB, 4-way set assoc., LRU, 32 byte line size, 1 cycle access lat.
L2 - Icache	256KB, 4-way set assoc., LRU 64 byte line size, 6 cycle access lat.
L1 – Dcache	64KB, 2-way set assoc., LRU, 32 byte line size, 1 cycle access lat.
L2 - Dcache	512KB, 4-way set assoc., LRU 32 byte line size, 8 cycle access lat.
Memory Latency	100 cycles for the first chunk, 2 cycles afterwards
Memory Bus Width (in bytes)	8
ALUs available	4 integer ALUs (1 integer multiply/divide) 4 floating point ALUs (1 floating pint multiply/divide)

In [10] the authors have made a profiling study about the characteristics of each benchmark (such as percentage of each type of instructions, basic blocks profiling, etc). This profiling is simulation infrastructure independent, a characteristic of the benchmark. Therefore, we used their suggested number of instructions to skip. For the controlled swapping method, we used 2-bit saturating counters for each block. Table 2 shows the total number of loads and stores committed. We present the results of data cache only.

Experiments and Discussion

In this section we present and analyze our simulation results.

Miss Rates. The first set of experiments involves the miss rate at level 1 cache. We expected a conventional cache to be slightly better than the non-inclusive cache because in case of cache misses, the blocks are brought into level 2 cache first in the non-inclusive cache, not to level 1 directly as the case would be in a conventional cache. Therefore, a future access to the same block will still incur a level 1 cache miss penalty. However, this scheme has the advantage of avoiding cache pollution due to rarely referenced blocks. The results are shown in Figure 5. The conventional cache design has slightly lower miss rate. The second best scheme is the autonomous prefetching. This because in AP the prefetching is not causing cache pollution and is offsetting the disadvantage of moving the block to L2 first. The only exception is twolf. This is due to the fact that many references are used one or two times, hence they cause cache pollution, which is removed by the basic non-inclusion scheme. The simple prefetching done in AP, does not perform well for twolf, hence the miss rate is higher. The controlled swapping scheme performs the worst, as the block movements to level 1 are delayed further due to the counters. CS is used to see

Table 2: Total number of loads and stores committed

Integer Bench.	**# of Ref.**	**FP BENCH.**	**# of Ref.**
bzip2	235864202	*ammp*	255247899
gcc	389188032	*applu*	127876490
gzip	150676316	*apsi*	187965389
mcf	282954558	*art*	212728035
parser	260534355	*equake*	161264237
perl	248595729	*mesa*	249097078
twolf	254806912	*swim*	136862642
vortex	275013769	*wupwise*	175958198
vpr	214718042		

whether we can *filter* more unwanted references.

The miss rates for the level 2 cache are shown in Figure 6. In contrast to level 1, the conventional cache has the higher miss rate in this case, and by a large margin. This is primarily due to the duplication of data: the extra data included in the level 2 cache in case of the inclusive scheme is much smaller than the extra data in the non-inclusive schemes.

Furthermore, the conflict miss is reduced in L2 cache due to the fact that the highly referenced blocks move to L1, leaving room to more blocks to come from memory. AP is performing the best in L2. Because of the fact that the larger size of L2 cache, coupled with its conflict miss reduction, results in much less pollution caused by the simple prefetching scheme used. The effect of non-inclusion is very apparent in the SpecFP results, where the conventional cache has very high L2 miss rate.

Instruction per Cycle. Figure 7 shows the instructions committed per cycle. All the schemes give comparable performance. On average, autonomous prefetching slightly outperforms the other schemes. The reason for the comparable IPC, although L2 miss rate is lower for the non-inclusive hierarchy, is due to the data that are referenced once or twice. The data that are referenced once, cause cache miss for both

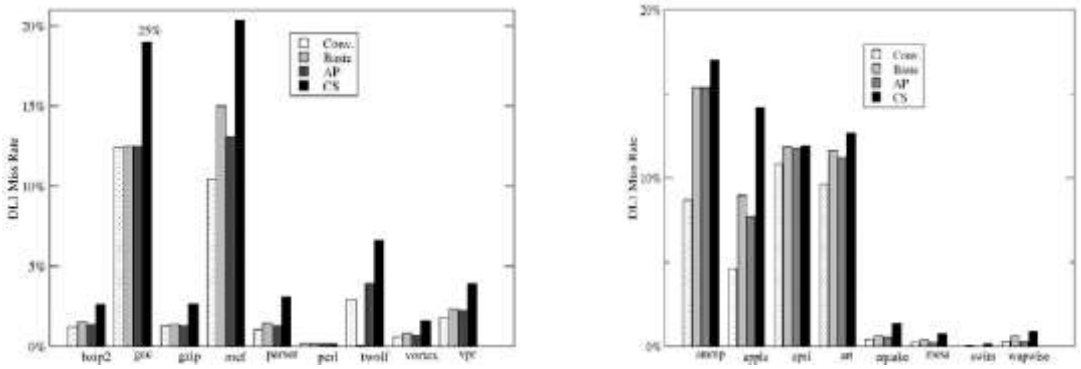

Figure 5: Level 1 data cache miss rate

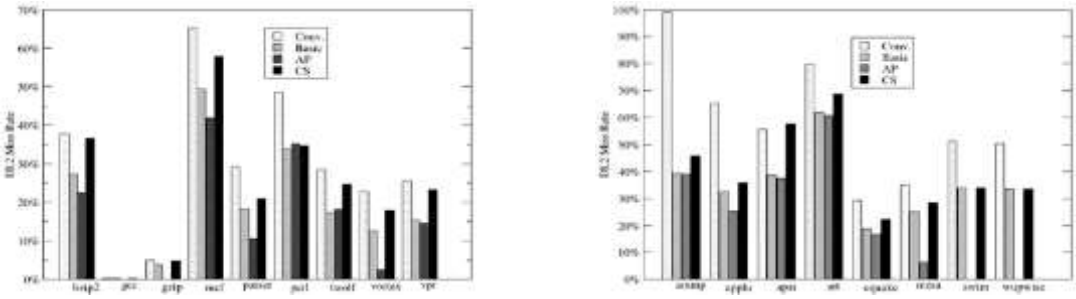

Figure 6: Level 2 data cache miss rate

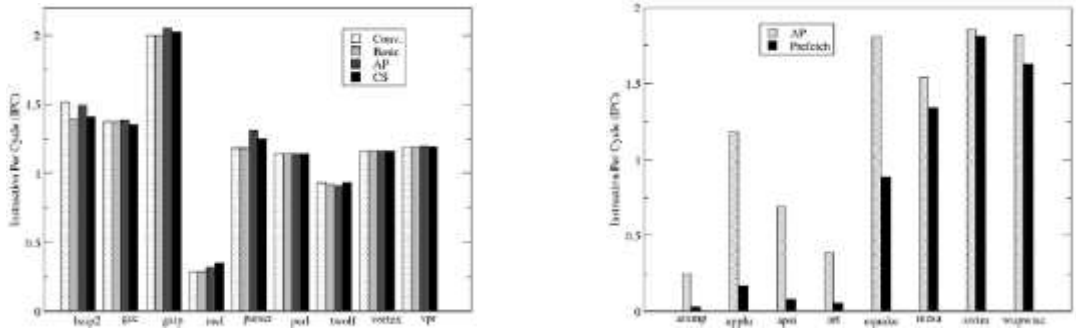

Figure 7: Instructions committed per cycle

inclusive and non-inclusive caches. The data that are referenced twice, will miss in the first access, and will hit in the second. However, the access time for the second time is lower in the inclusive cache, due to the fact that L1 responds, while in the non-inclusive cache, L2 responds, after L1 misses. Only when the data is referenced for the second time, in the non-inclusive hierarchy, it is upgraded to L1, after which the response will be higher.

We tried to enhance the autonomous prefetching even further by controlling the prefetching. The underlying idea was that some prefetched data might not be needed and it might displace useful data. The controlled prefetching attempts to prevent this by adding a reference bit to each block at level 2 cache. When a block is displaced from level 1 and inserted into level 2, it has its reference bit set. We do not prefetch the successor of the block which migrated to level 1 unless the reference bit of the block that will be displaced from level 2 cache due to this prefetch is not set. If this reference bit is set, no prefetching takes place at this time and the bit is reset. Despite these measures, we found that the performance of the controlled prefetching scheme was very close to that of the basic scheme. An explanation of this finding lies in the fact that the prefetched data is more often useful than not due to the locality of reference of most applications of the Spec2000. Therefore, we do not present results for the controlled prefetching in this paper. It is worth mentioning here that if both cache levels are accessed simultaneously, an improvement is expected in IPC for the non-inclusive cache more than the conventional cache. This is due mainly to the fact that in the non-inclusive scheme we are accessing new data by accessing both caches, not duplicated data with some new data like the conventional scheme.

However, for power issues, we decided to access them in sequence, as done in any conventional cache.

Prefetching in Inclusive vs Non-Inclusive Hierarchy. In order to assess the effect of prefetching on an inclusive versus non-inclusive cache, and also to see whether the low miss rate of AP is due to the non-inclusion or the prefetching, in this section we compare the L2 miss rate as well as the IPC of an inclusive traditional hierarchy with prefetching and the AP scheme. Both of them are using the same simple prefetching technique that we have described earlier. AP is a non-inclusive cache hierarchy, and the traditional scheme is in inclusive hierarchy.

As we can see from Figure 8, the AP has much lower L2 miss rate. This is due to the fact that a blind prefetching can easily pollute the cache. When the cache size is smaller, it becomes even more sensitive to the pollution. Since a non-inclusive cache is in reality of bigger size than the inclusive one, due to the elimination of data replication, the pollution has less effect on the non-inclusive cache. This is also reflected in the IPC, shown in Figure 9, where AP has higher IPC than a traditional prefetching scheme. Prefetching did not help some benchmarks, such as ammp, art, and mcf, in the inclusive scheme. On the contrary, prefetching has caused pollution to the caches for those benchmarks, negatively affecting the performance. As we have seen, the pollution caused by prefetching has offset the lower access time of the inclusive cache hierarchy, and made it worse than the non-inclusive one.

Bandwidth Utilization. Besides cache performance and its effect on overall system performance, it is important to study the bandwidth consumed in the memory hierarchy. This means the amount of data moving between L1, L2, and memory. The importance of bandwidth is related to the power dissipated from wires, which constitute a significant portion of the total power dissipated by the memory system. Moreover, high bandwidth exposes wire delay and can affect performance. In order to be technology independent in our measurements, we calculated the bandwidth as bytes per cycle. Figure 10 shows the bandwidth consumed by the schemes. The first thing we notice is that AP is consuming more bandwidth than the other schemes. This is expected because of the extra traffic caused by the prefetching. CS scheme is performing the best together with the conventional system. We found that most of the bandwidth consumed in the non-inclusion basic scheme as well as the CS scheme is between L1 and L2 caches, due to the upgrade of blocks from L1 to L2or vice versa. Most of the bandwidth consumed by the conventional scheme is from L2 to main memory, hence causing congestion in the system bus, harming the overall system performance. The system bus is usually used by the

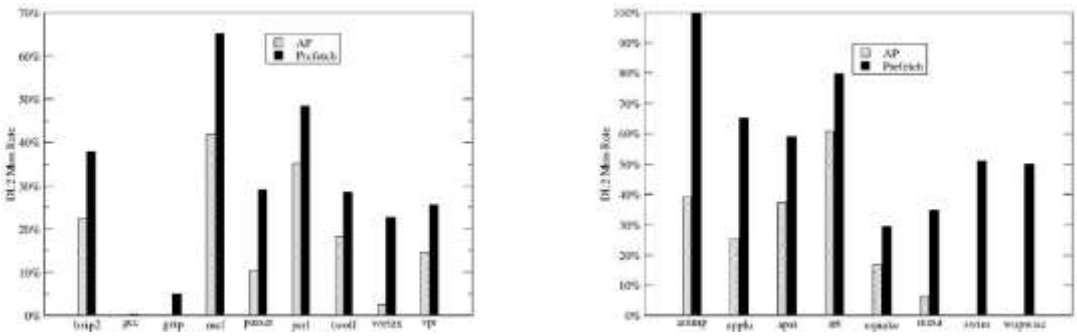

Figure 8: Level 2 data cache miss rate of AP and inclusive prefetch

processor, as well as as devices such as DMA or graphics adapter. However, since in our simulations we did not model bus congestion, the aforementioned facts about the bandwidth have not affected the IPC.

Power Implication. In this section we discuss the power implication of the proposed schemes at level 1 and level 2 data caches because, in this paper, we apply non-inclusion in the data cache hierarchy only.

Power consumption is now a pivotal factor in any system. For portable devices, it is important because of the battery life. For desktop machines, it is important because of the packaging cost. Caches usually consume a significant amount of power. Power consumption can be divided into dynamic power and static power. Dynamic power is due to the switching activity [15], and many methods have been proposed to deal with it [1].

The other component of power consumption is static power due to current leakage. This type of power consumption is becoming increasingly more significant, and the semiconductor industry association (SIA) predicts that it will reach 50 percent of the power consumption in the very near future [11], in the current sub-micron era.

For dynamic power consumption we used CACTI to get total power dissipation per cache. Then, in order to get a realistic idea about the behavior of each scheme in terms of power and energy, we calculated the number of bytes per nJ. This calculation is done using the following equation.

(effective cache hierarchy size)/ (average energy consumed).

The effective cache size is the amount of *unique* data. In cache of the conventional scheme, this size is the size of the level 2 cache, because level 1 is a subset of level 2. In case of non-inclusive cache, it is the sum of both the size of level 1 and level 2 caches. The effective size of the three non-inclusive schemes is the same. We calculated the average energy consumed as follows.

(energy consumed at L1) + (miss rate of L1[2])(energy consumed at L2).*

The logic behind the above equation is that level 1 will be accessed anyway, but level 2 will be accessed only if L1 misses. Table 3 shows the results obtained. As we can see from the table, gcc, gzip, perl, twolf, apsi, and swim are - cheaper, in terms of energy consumed, when using a non

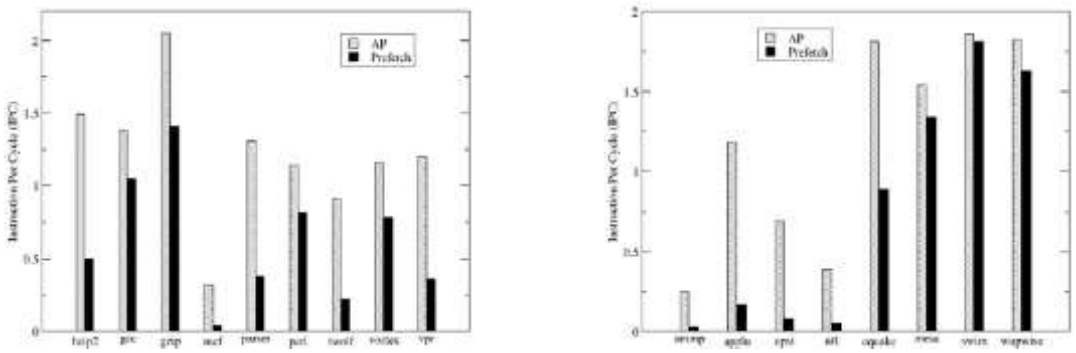

Figure 9: Instructions committed per cycle of AP and inclusive prefetch

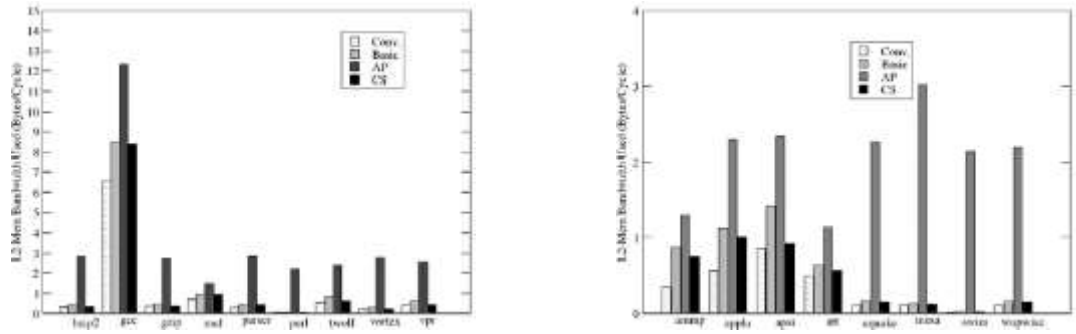

Figure 10: Bandwidth used in memory hierarchy (bytes/cycle) including prefetching

[2] Unless statead otherwise, L1 always means data cache level 1.

inclusive cache. On the other hand, some benchmarks, such as vpr, equake, and art do not show big gain or loss when using non-inclusion or inclusion.

Therefore, the performance of non-inclusive cache, in terms of energy, is application dependent. However, we can see from the above equation that as the size of caches at each level increases, which is very likely due to the advances in technology, the benefit of non-inclusive cache, in terms of energy, becomes more apparent.

Leakage current is the result of inactivity. That is, if a block in the cache is not modified for a long time, current will start leaking, consuming static power. The working of non-inclusive cache significantly reduces leakage current. This is because of the modifications done to each cache frame due to block upgrade. The way the non-inclusive cache works involves more cache block movements than the traditional cache. This is due mainly to block upgrade from L1 to L2, and the swapping. Due to this movement, cache frames receive more modifications, than traditional caches, and this reduces the leakage current.

Finally, as we have seen from Section 3.1, the extra hardware needed for the functionality of the non-inclusive cache is negligible relative to the big cache size.

Table 3: Bytes per nJ for different configurations

Bench.	Conv.	Basic	AP	CS
bzip2	178864.63	165755.4	180683.74	105018.16
gcc	20987.3	23555.27	23573.72	15612.07
gzip	174205.21	183998	195980.86	101820.19
mcf	25080.27	19700.2	22481.82	14590.21
parser	203338.5	171420.6	195980.86	89151.15
perl	614208.06	690984.07	690984.07	690984.07
twolf	82930.72	953790.43	71386.52	43704.91
vortex	312671.76	265017.97	308485.36	157102.07
vpr	127650.95	114183.06	117757.55	71556.27
ammp	29902.58	19234.31	19258.93	17449.79
applu	54864.8	32584.08	37779.2	20817.71
apsi	24108.3	24817.98	24962.08	24675.52
art	26977.32	25276.58	26178.1	23281.91
equake	379149.55	336043.76	347691.58	183998
mesa	539835.26	473526.01	583752.97	290610.96
swim	875564.46	896934.31	985010.02	707223.02
wupwise	464794.33	343720.28	522893.62	256000

5 Conclusions

In this paper we show that non-inclusive strategies can be particularly effective in reducing cache misses at level 2 in our 2-level on-chip cache scenarios. The IPC numbers are comparable to those obtained with a conventional cache system, but the access time is expected to be lower for the non-inclusive cache strategy due to the lower miss rates at level 2 and the comparable miss rates at level 1 cache. We presented several schemes for the non-inclusive cache and showed that non inclusion results in reducing L2 miss rate by 28 percent for specINT and 40 percent for specFP.

We believe that the results we obtained in this study support our belief that more performance can be obtained by using more sophisticated non-inclusive strategies. In our future work, we wish to explore methods to alleviate the adverse effect of high level 1 miss rates of non-inclusive caches so that the performance gain obtained by decreasing the miss rates at higher levels can be reflected better in overall performance. Furthermore, we intend to develop mechanisms to allow the usage of non-inclusive caches in multiprocessor systems, and study the interaction between cache coherence protocols and non-inclusive caches in detail.

Acknowledgement

This work was supported in part by a grant from The City University of New York PSC-CUNY research award program.

References

[1] A. Agarwal, H. Li, and K. Roy, “Drg Cache: A Data Retention Gated-Round Cache for Low Power,” *Proc. 39th Int’l Design Automation Conf.*, 2002.

[2] Semiconductor Industry Association, International Technology Roadmap for Semiconductors, 2001.

[3] J. L. Baer and W. H. Wang, “On the Inclusion Properties of Multi-Level Cache Hierarchies,” *Proceedings of the 15th Annual International symposium on Computer Architecture*, 22:73-80, 1988.

[4] L. Barroso K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, “Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing,” Int’l Symposium on Computer Architecture (ISCA), 2000.

[5] D. Burger and T. M. Austin, *The Simplescalar Tool Set: Version 2.0*, Technical Report, Dept. of CS, Univ. of Wisconsin-Madison, June 1997.

[6] T. L. Johnson and W-M. Hwu, “Run-Time Adaptive Cache Hierarchy Management via Reference Analysis,” Int’l Symposium on Computer Architecture (ISCA), 1997.

[7] T. L. Johson, D. A. Connors, M. C. Merten, and W-M Hwu. “Run-Time Cache Bypassing,” *IEEE Transactions on Computers*, 48(12):1338-1354, 1999.

[8] N. P. Jouppi, “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffer,” *Proc. 17th Int’l Symposium on Computer Architecture*, pp. 364-373, May 1990.

[9] S. McFarling. “Cache Replacement with Dynamic Exclusion,” *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pp. 191-200, 1992.

[10] S. J. E. Wilton and N. P. Jouppi, “Tradeoffs in Two-Level On-Chip Caching,” *Proceedings of the 21st Annual International Symposium on Computer Architecture*, pp. 34-45, 1994.

[11] P. Racunas and Y. N. Patt, “Partitioned First-Level Cache Design for Clustered Microarchitectures,” *Proceedings of the 17th International Conference on Supercomputing*, pp. 22-31, 2003.

[12] S. Sair and M. Charney, *Memory Behavior of the Spec2000 Benchmark Suite*, Technical Report RC-21852, IBM T. J. Watson Research Center, October 2000.
[13] P. Shivakumar and Norman P. Jouppi, *An Integrated Cache Timing, Power, and Area Model*, Technical Report, Western Research Laboratory, 2001.
[14] B. Sinharoy, R. N. Kalla, J. M. Tendler, R. J. Eickemeyer, and J. B. Joyner, "POWER5 System Microarchitecture," *IBM Journal of Research and Development*, 49(4/5):505-521, 2005.
[15] A. J. Smith, "Cache Memories," *ACM Computing Surveys*, 14(3):473-530, 1982.
[16] C-L Su and A. M. Despain, "Cache Designs for Energy Efficiency," *Proc. of the 28th Annual Hawaii Inter'l Conf. on System Sciences*, 1995.
[17] J. M. Tendler, J. S. Dodson, Jr, J. S. Fields, H. Le, and B.Sinharoy, "POWER4 System
[18] S. P. Vanderwiel and D. J. Lilja, "Data Prefetch Mechanisms," *ACM Computing Surveys*, 32(2):174-199, 2000.
[19] D. Weiss, J Wuu, and V. Chin. "The On-Chip 3-MB Subarray-Based Third-Level Cache on an Itanium Microprocessor," *IEEE Journal of Solid-State Circuits*, 37(11):1523-1529, 2002.
[20] Y. Zheng, B. T. Davis, and M. Jordan, "Performance Evaluation of Exclusive Cache Hierarchies," IEEE Int'l Symposium on Performance Analysis of Systems and Software (ISPASS), 2004.