ESP-NUCA: A Low-cost Adaptive Non-Uniform Cache Architecture

Javier Merino, Valentin Puente and Jose A. Gregorio
Computer Architecture Group
University of Cantabria, Santander, Spain
merinocj@unican.es, vpuente@unican.es, monaster@unican.es

Abstract

This paper introduces a cost effective cache architecture called Enhanced Shared-Private Non-Uniform Cache Architecture (ESP-NUCA), which is suitable for high-performance Chip MultiProcessors (CMPs). This architecture enhances system stability by combining the advantages of private and shared caches. Starting from a shared NUCA, ESP-NUCA introduces a low-cost mechanism to dynamically allocate private cache blocks closer to their owner processor. In this way, average on-chip access latency is reduced and inter-core interference minimized. ESP-NUCA synergistically integrates victims and replicas thus making it possible to take advantage of multiple-readers for shared data, and to maximize cache usage under unbalanced core utilization. This architecture leads to stable behavior within the whole system across a broad spectrum of working scenarios. ESP-NUCA not only outperforms architectures with similar implementation costs such as private and shared caches by up to 20% and 40% respectively, but even outperforms much costlier architectures such as D-NUCA [13] by up to 28%, Adaptive Selective Replication [3] by up to 19%, and Cooperative Caching [5] by up to 15%. Moreover, performance variance throughout the set of benchmarks is 37% lower than with ASR, 87% lower than with D-NUCA, and 43% lower than with Cooperative Caching.

1. Introduction

The future evolution in the number of cores per chip of CMP architectures could be jeopardized by the available off-chip bandwidth. In order to minimize this effect, a large amount of intra-chip cache should be provided. Although the transistor budget is generous, multi-megabyte cache hierarchy with many-core CMP represents a challenge.

First of all, it is necessary to define the sharing criteria among the cores of the CMP, for the on-chip portion of the memory hierarchy. The most convenient sharing policy is strongly dependent on the workload. On the one hand, some applications are characterized by a significant sharing degree whereas others have little to no sharing at all. On the other hand, simultaneous threads could interfere destructively in the memory hierarchy. The usage scenarios of high-performance CMPs are very dissimilar, ranging from number crunching applications to information processing suites or desktop applications. In order to provide a truly general purpose system, the on-chip memory hierarchy should be smart enough to adapt its behavior to very different working conditions.

The Last Level Cache (LLC) may be structured as private or shared. Although this discussion is equally applicable with three on-chip cache levels, to simplify it, we will assume in the rest of the paper that only two levels are present, therefore LLC will be equivalent to L2. From an architectural point of view, private and shared caches exhibit different properties. Private caches are characterized by lower on-chip access latency, as they enable the emplacement of cache blocks closer to the owner processor. Moreover, they provide inter-thread isolation, eliminating most unnecessary inter-core interference. Shared caches are distinguished by lower off-chip miss rates than private caches because shared data is not replicated throughout different L2 locations. They can also outperform private caches when threads with unbalanced memory usage run in different cores of the chip or a reduced number of cores run active threads. Consequently, depending on the inherent characteristics of the system workload, each architectural design could outperform the other [10, 22]. Notwithstanding, the on-chip memory hierarchy of general purpose CMPs should be flexible enough to adapt its responsiveness to the requirements of the existing workload, maximizing hit rates, minimizing on-chip access latency and reducing unnecessary inter-core conflicts in order to achieve a stable performance over a large range of scenarios.

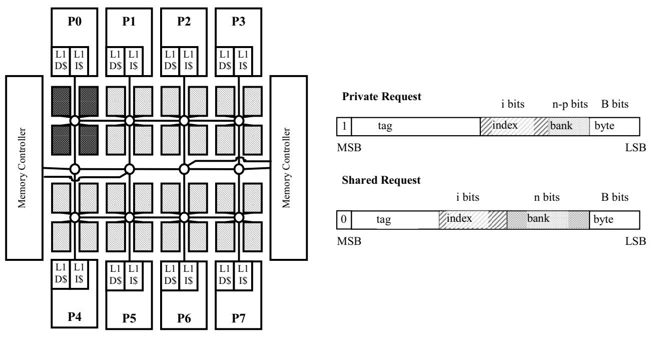
A plethora of studies have made proposals to deal with the previously identified issues. Some propose starting with a private cache and limiting the performance degradation produced by block replication [3, 5, 6, 22]. Others, using a shared scheme as the baseline, attempt to minimize on-chip latency degradation [4], the total number of misses [10] or inter-thread conflicts [8].

The rest of this paper is organized as follows: Section 2 introduces a first approach called Shared Private-NUCA (SP-NUCA). Section 3 describes the refinements of SP-NUCA to build up the final proposal: ESP-NUCA. Section 4 describes the experimental methodology employed. Section 5 justifies some architectural decisions taken in the final proposal. Section 6 presents simulation results and compares our architecture to previous proposals and, finally, Section 7 states the main conclusions of the paper.

2. SP-NUCA Architecture

2.1. Privatization of a shared S-NUCA

Using the shared S-NUCA as the initial CMP cache framework we will explain the modifications needed to arrive at the SP-NUCA cache architecture. In shared cache architectures, each block is emplaced in a fixed position depending only on its address. On the other hand, in private cache architectures each block is located according to its address and the accessing core. For a Static NUCA with 2^n L2 banks and 2^p processors, similar to the particularization shown in Figure 1a, in a shared cache each block is placed in a specific bank, depending on its address. Within the same basic NUCA substrate it is possible to define a fully private cache [10]. In this case, each L1 allocates its evicted blocks in the nearest 2^{n-p} banks, replicating blocks being accessed by multiple processors. Therefore, the behavior is not defined by workload characteristics, but statically. For instance, the highlighted banks in Figure 1a will store the private blocks for processor zero.



(a) Banks (b) Address interpretation

Figure 1. Private/Shared partition

SP-NUCA dynamically enables the adaptive utilization of L2 cache resources at block level. In this architecture we add an extra bit (the private bit) to the tag of every block in the cache to distinguish between private and shared blocks. When a block arrives from memory, its private bit is set and it is stored in the closest bank to the only processor that uses it. If another processor tries to access the same block, this bit is reset. Thus, a block is "private" if it has only been accessed by one processor and it is "shared" if it has been requested by two or more cores. This status remains with the block while it stays in the chip. The private bit is also present in the on-chip cache requests, as part of the address. To map an address to its corresponding bank we use two different functions, one for private mapping and one for shared mapping, as shown in Figure 1b. The least significant B bits are used to select the byte/word in the cache block. Then, we use the $n-p$ bits to select the private bank or the n bits to select the shared bank depending on whether it is a private or a shared request. The next i bits, the index, are used to select the corresponding set in the bank and the rest of the address is the tag. It is important to note that the address is the same; the figure shows how it should be interpreted depending on whether it is a private or a shared request. The private tag is p bits bigger than the shared one, but as they are to be stored in the same tag array, it must have the size of the private tag, slightly increasing the required area of L2 banks by p bits per line.

Note that we previously stated that a shared NUCA maps its blocks onto 2^n banks and in a private architecture each core uses its nearest 2^{n-p} banks, so using the private bit and these mapping functions we can choose the best architecture for each block. A block is considered private when it first arrives to the chip. If another core requests the same cache block, then we reset the private bit and the block is considered shared until it leaves the chip.

2.2. Dynamic allocation of shared and private ways through replacement policy

A fundamental question remains unanswered: how L2 banks are partitioned into shared and private content. As banks are w-way set-associative, different ways of the same set can hold shared or private blocks. The private bit is added to the tag comparison, so private requests can only match among private blocks and shared requests can only match among shared blocks.

Nevertheless, one of the key challenges in this proposal is deciding which ways of each set should hold private data and which ones should store shared blocks. A statically defined partition between shared and private blocks could lead to suboptimal cache utilization. In our architecture, reassigning a private way to a shared partition is done by resetting the private bit, so it is straightforward to change the number of private and shared ways dynamically during block replacement. Using this technique, we can aggressively adapt the number of private and shared ways. We choose to include this dynamic allocation of ways as a part of the replacement algorithm, because it is not in the critical path and it is a common event that enables us to make finer grain decisions. When a new block arrives at a set, the replacement algorithm should choose which way it should replace. If the data arriving is private but the private data in the set is frequently used, the replacement algorithm could choose to evict a shared block and reassign that shared way to a private one.

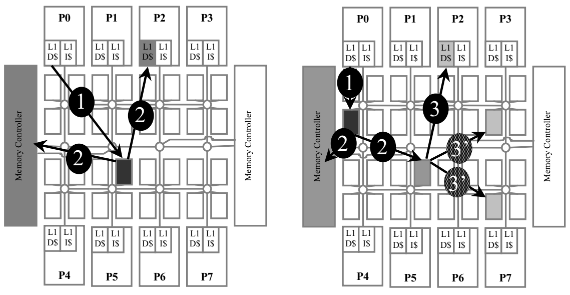
2.3. Coherence Protocol

To maintain coherence in the CMP we employ a token-based coherence protocol [17] with Token-based-directory (TokenD) [15] as the performance policy. Inclusiveness is not enforced in the memory hierarchy, so an L2 can evict a block without having to evict it from the L1 cache. Token counting assures the correctness of each operation —for instance, false misses due to migrating blocks degrade performance but do not affect the correctness of the coherence protocol. We used token coherence because it simplifies the implementation but our architecture does not depend on it.

Figure 2 exemplifies the differences between the two protocols. In S-NUCA (Figure 2a), a write from core 0 misses in L1 and is sent to the L2 NUCA bank (1). The shared bank cannot fulfill the request, so it forwards it to the L1s known to have tokens (in this example, the L1 from P2) and to memory (2). The response message(s) is/are not shown to keep the figure clear. In SP-NUCA (Figure 2b), the L1 sends the request to its private L2 bank (1). The private L2 does not have the block, so it forwards the request to the shared L2 bank and to the memory controller (2). As with the S-NUCA example, the shared bank does not have all the tokens. The L2 shared bank forwards the request to the L1s known to have tokens (3). If the shared L2 bank does not have the block, it could be possible that the block is in other private L2s, so the request will be forwarded to the other private L2 (3’). If the block is found in another private bank, its private bit is reset and it is migrated to the corresponding shared bank. Further accesses to this block by other processors will hit in the shared bank, so the extra latency when searching in other private banks is required only once for each shared block. The figure is simplified and only two requests to the other private L2 banks are depicted in the third step.

In S-NUCA this indirection through the private L2 is not required: after an L1 miss, the S-NUCA sends the request to the L2 NUCA bank. This additional step will slightly increase on-chip traffic and L2 hit latency of accesses to shared data. Additionally, when the requested block is held by a remote private bank, in the first access, supplementary steps are needed to satisfactorily finish the transaction. The L2 bank controller should be capable of forwarding or satisfying requirements from other L2 banks, increasing the number of possible actions and transactions in the controller.

An example of a request to a private block that hits in the L2 is simpler. The private block is stored in the private L2 banks of P0, so after step 1, the L2 sends a response with the data and all the tokens to P0. SP-NUCA finds the block in a nearer bank and answers it faster, while S-NUCA needs to reach the shared L2 bank, which is 3 network hops further and thus, has higher L2 latency.



(a) S-NUCA (b) SP-NUCA

Figure 2. Coherence protocol.

3. ESP-NUCA Architecture

3.1. Support of Replicas and Victims Using Flat LRU

SP-NUCA as depicted has some potential limitations. As with any private cache configurations, unbalanced private capacity from active threads in the CMP could underutilize cache. As a limit case, albeit an important scenario in a CMP, a single thread application is able to utilize only a fraction of the cache capacity. By contrast, in any shared cache a single thread is able to utilize the full extent of the cache. Another significant limitation for SP-NUCA, as with any shared cache configuration, is that multiple sharers have to access a remote bank to get a shared block, increasing on-chip latency for shared accesses. Nevertheless, many parallel applications are characterized by a large proportion of shared accesses to a relatively small proportion of cache [3]. Private cache configurations take advantage of that fact allowing the creation of local copies of the shared data in the private cache, significantly reducing the average access-time in some workloads, at the cost of potentially increasing the cache miss-rate.

As we can see, SP-NUCA has some positive properties of shared and private caches, but inherits some of their limitations. ESP-NUCA has been conceived to overcome those limitations and widen the proposal’s effectiveness to broader CMP utilization scenarios. It is feasible to tackle the previously cited weaknesses in a unified and elegant way if we support replicas and victims. A replica block is a copy of a shared block stored in the local partition of the cache. A victim block represents remote private data stored in the shared cache partition. By adding an extra group composed of replicas and victims, labeled ***helping blocks***, over private and shared blocks (***first-class blocks***), we solve the aforementioned limitations of SP-NUCA.

We can fine tune the number of allocated ways for these helping blocks by modifying the replacement policy. Our first approach would be to keep applying a procedure similar to the dynamic partition of shared and private blocks on SP-NUCA. Helping blocks are less important to keep. In fact, they should only be kept if the private and shared hit rate does not suffer. That is, an L2 cache can hold victim blocks of a remote core (and increase its on-chip hit rate), only if that does not affect its own performance. A similar effect is seen with replicated blocks: with unrestricted replication the cache hit-rate could be reduced, as shown in [3].

Although for SP-NUCA the flat LRU replacement policy is the most cost effective solution, this choice may not be the best for ESP-NUCA. Flat LRU makes no distinction between first-class blocks and helping blocks, thus it may be possible that victims and replicas could increase the cache miss rate for private and shared blocks. We need to fine tune the replacement policy to favor the creation of helping blocks it they are helpful, and minimize their presence in the cache if they are harmful for the performance of the local core.

3.2. Replica and Victim Support Through Protected LRU

In order to prevent these conflicts between first-class and helping blocks, the sets in a bank can only hold a limited number of victims and replicas. When the replacement policy has to choose a block to evict, if the number of victims and replicas in the cache is greater than or equal to this limit, the LRU block among the helping blocks is chosen. If it is not (that is, the set has less victims or replicas than it should have), then the LRU block of the whole set is chosen.

Qureshi et al [18] showed that different applications behaved very differently as the available ways per set varies. In some workloads or execution phases reducing the number of ways has little to no impact due to the reduced working set used or the low utility of the cache. In others, due to the high cache utility, the performance is very sensitive to the number of available ways. Our goal is to be able to determine on-line how the application is behaving, so we can provide room for helping blocks only when no performance degradation is observed for first-class blocks.

The hit rate is constant across the sets of a bank [18] so we can make some sets behave slightly differently to the rest and compare their performance with the "normal sets". Let n_{max} be the number of helping blocks per set. n_{max} is defined at the bank level, that is, all the sets in a given bank use the same n_{max}. We need to calculate, using local information, the potential benefit of permitting replicas and victims. In order to do this, we break down the sets in the bank according to the following three categories:

Conventional Sets Sets that accept up to n_{max} helping blocks.

Reference Sets These sets do not allow any victim or replica. As they behave like the original SP-NUCA, they provide a reference point for performance comparison. If the conventional sets have the same hit rate (for the first-class blocks) as this set, then we can conclude that the performance of those sets is not harmed by the helping blocks.

Explorer Sets These sets accept one more helping block than the current limit of the bank ($n_{max}+1$). With this information we can predict the expected performance loss of increasing n_{max}.

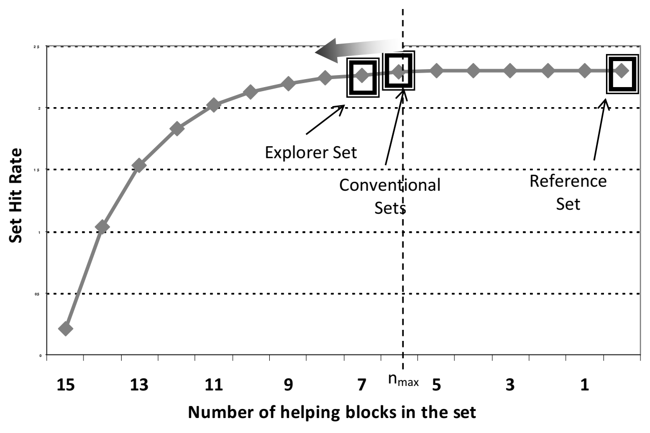
If *Conventional Sets* notice that their first-class blocks' hit-rate is lower than that of the *Reference Sets*, then they are accepting too many helping blocks, so they lower n_{max} to improve its hit rate and reduce replication and victimization. On the other hand, if the *Explorer Sets* achieve the same first-class blocks' hit-rate as the *Reference Sets*, that means that with one more victim or replica, the hit rate of the first class blocks is preserved, so they increase n_{max}.

For example, if an application is in a reduced working-set execution phase, such as Figure 3a, the *Explorer Sets* will increase n_{max} whereas in a high utility execution phase, such as Figure 3b, the *Conventional Sets* will lower n_{max}, so they will move towards the *Reference Sets* minimizing the number of helping blocks in the bank. It should be noted that the application characteristic could vary during the whole execution. Given the number of helping blocks in the *Explorer Sets* in Figure 3b it is likely that the utility of the cache has increased suddenly. The method described in this section adjusts itself as the application changes the way it is using the cache.

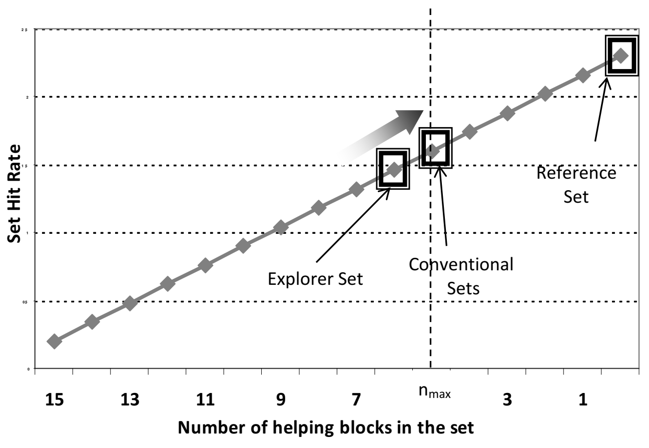
Each set has a counter (n) that indicates the number of currently stored helping blocks. At replacement, if $n < n_{max}$ the block chosen to be replaced is the LRU block of the whole set. If $n = n_{max}$ the LRU block of helping blocks in the set is replaced. According to the nature of the selected and the incoming block, n will be decremented or incremented by one or kept unchanged. For the explorer sets, the LRU block is chosen if $n \leq n_{max}$ and for the reference sets all helping blocks are refused.

3.3. First-class block Hit-Rate Estimation

In order to apply the previously described mechanism we have to determine on-line the value of n_{max}. We need some method to dynamically estimate the first-class blocks' hit-rate of these special class sets. The hit-rate is a time series data that fluctuates during the application execution. A



(a) Small working-set



(b) High utility

Figure 3. Hit-rate monitoring of first-class blocks.

technique used in statistics to extract relevant information from time series is to use moving averages. Among the different options available to estimate the moving average of the hit rate of first-class blocks we will use the Exponential Moving Average (EMA). Given a fixed N sample data[1], the EMA is defined for a data set, with value h as:

$$EMA_N = EMA_N\,(1-\alpha) + h\,\alpha \qquad (1)$$
$$\text{where} \qquad \alpha = \frac{2}{N+1}$$

In our case, h will be 1 if in the current event there is a hit for a first-class block in the set, and 0 otherwise. We will estimate independently the EMA of each of the explorer and the conventional sets. In order to make equation 1 easily implementable in hardware, α and h should be a power of 2, so let $\alpha = 2^{-a}$. We use b bits to store the EMA estimation and normalize the hit-rate between 2^b and 0, the new $h' = h\,2^b$. In fact $EMA'_N = EMA'_N - EMA'_N\,2^{-a} + h'\,2^{-a}$, and therefore[2]:

$$\begin{aligned} EMA'_N = EMA'_N - (EMA'_N >> a)\; + \\ +\;\left(2^b >> a\right) \qquad \text{on hit} \quad (2) \\ EMA'_N = EMA'_N - (EMA'_N >> a) \quad \text{on miss} \end{aligned}$$

We denote the conventional set estimated hit rate as HR_C, HR_R is the reference set estimated hit rate and HR_E the explorer set estimated hit rate. To estimate each hit-rate we use a different number of sets. Let 2^{-d} be the accepted hit rate degradation for first-class blocks. To determine the bank value for n_{max}, we establish that if the hit-rate estimated for the *Explorer Sets* is greater than $1-2^{-d}$ of the *Reference Sets*' hit-rate ($HR_R - HR_E < HR_R\,2^{-d}$), we should increase n_{max} by one. If the estimated hit-rate of the *Conventional Sets* reaches $1-2^{-d}$ value of *Reference Set*'s hit-rate $HR_R - HR_C \geq HR_R\,2^{-d}$, we will decrement n_{max} by one. These computations are performed after N references to a set. Formally:

$$n_{max} = \begin{cases} n_{max} - 1 & \text{if } HR_R - (HR_R >> d) \geq HR_C \\ n_{max} + 1 & \text{if } HR_R - (HR_R >> d) < HR_E \\ n_{max} & \text{otherwise} \end{cases}$$

As we can see, the decisions are taken from limited information. Depending on the number of sets used in each estimation, the measure will be more or less accurate. It should be noted that a large number of *Explorer Sets* or *Reference Sets* could impair performance, so it seems convenient to minimize the number of non-conventional sets in the bank. A reduced number of sets per bank will be used to estimate its behavior.

The implementation storage overhead is $log_2 w$ bits per set to store n and $log_2 w$ bits per bank to store n_{max} for a w-way set-associative cache. $3b$ bits per bank are needed to store the three estimated hit rates. Not only are the required computations easy to implement but none of the operations required are in the critical path to memory. We need to add the structures to keep track of LRU blocks in helping blocks. Note that parameters a, b and d are constants and therefore fixed by cache design. In Section 5.2, we will choose their values after a sensitivity analysis for a particular cache configuration.

Although the proposed protected-LRU motivation is

[1]Note that N, and by extension α, are constants.

[2]We denote by $x >> y$ the shift right rotation of y bits in the variable x.

similar to Dynamic Set Sampling [18], our proposal is less ambitious as we are not accurately estimating the cores' performance but inferring it via the L2 cache behavior.

4. Evaluation Methodology

4.1. Simulation Framework

The impact of the proposal on system performance will be evaluated under realistic conditions. For this purpose a full system simulator based on Simics [14] extended with the GEMS timing infrastructure [16] will be used. GEMS is an event-driven simulator that provides a complete model of the memory system, Ruby, and a detailed state-of-the-art processor model, Opal. With Ruby, the memory system simulator, we are able to obtain an accurate implementation of the memory hierarchy that we are working with. This includes interconnection network parameters, bank access time, mapping, replacement policies, etc. Using this environment a multiprocessor system can be simulated with the whole software stack, including operating system, web server, data-base management system, etc.

4.2. Workloads

The applications considered in this study are a mixture of multi-programmed and multithreaded workloads running on top of Solaris 9 OS. Therefore, any operating system activities such as process migration will be implicit in the results. With multiprogrammed workloads or transactional applications OS activity has noticeable relevance in the workload [2], which makes it essential to use full-system simulators in order to provide meaningful results.

We have selected 22 workloads composed of a mixture of four different classes with a variable number of applications per class. Two of the classes are multithread applications, being numerical and server-based applications.

Table 1. Workloads under study

Transactional Workloads: *Wisconsin Commercial Workload suite*	
Apache "1000 transactions"	JBB "10000 transactions"
OLTP "200 transactions"	Zeus "2000 transactions"
Multiprogrammed: *SPEC2000 Half Rate*	
art-4	gcc-4
gzip-4	mcf-4
twolf-4	
Multiprogrammed: *SPEC2000 Hybrid*	
art-gzip	gcc-gzip
gcc-twolf	mcf-gzip
mcf-twolf	
Scientific Applications: *NAS Parallel benchmarks*	
BT "class B"	CG "class B"
FT "class A"	IS "class B"
LU "class A"	MG "class A"
SP "class B"	UA "class B"

The first family of applications are the transactional benchmarks which correspond to the full Wisconsin Commercial Workload suite [1], released by the authors in GEMS version 2.1. All applications are running with the default input sets and configurations.

The second family is multi-programmed workloads composed of some of the SPEC2000CPU [20] applications. The Half Rate workloads have the application running in four cores of the CMP. One of the four idle cores is running system services. In all benchmarks the system is in multi-user level. The Hybrid workloads represent a multi-programmed scenario where 4 instances of the first program are run in half of the cores and 4 instances of the second program are run in the other half. Although not always considered in this kind of studies, the utilization scenario modeled is frequent in a desktop environment where only sequential programs are used.

Finally, the last family of applications considered is all the applications of NAS Parallel Benchmarks (OpenMP implementation [12] version 3.2.1).

For each data point a variable number of runs are performed with pseudo-random perturbation in order to estimate workload variability and obtain statistically meaningful results [1]. All the results provided have a 95% confidence interval.

4.3. System Configuration

The simulated system is an 8-processor CMP with the layout depicted in Figure 1a. The main configuration parameters are shown in Table 2. The cache bank access time has been obtained using CACTI 5.0 [21] with a power-efficient sequential access cache for 45nm technology.

Table 2. Main simulation parameters

Number of Cores	8
Core	Out of order, window size: 64, 16 outstanding memory requests, 4 issue width
L1 I/D cache	Private, 32K, 4-way, 64Bytes block, 3 cycle, 1 cycle tag
L2 cache	8MB, NUCA, 8x4 banks, 4 per router, 16-way, 5-cycles, 64Bytes block, sequential access, 2 cycles tag
Network Topology	Mesh with DOR routing, 128 bits wide links
Network Hop Latency	5 cycles (3 cycles router + 2 cycle link latency)

Each processor has its own private L1 Instruction and L1 Data cache and is connected to a node of the network. The L2 NUCA is distributed in 32 banks connected four by four to each switch that each CPU is connected to. The routers of the central row are connected to memory controllers. The four L2 banks nearest to each processor by themselves constitute an S-NUCA of the "private portion" of the L2 cache, as explained in Section 2.1.

5. Dynamic Cache Partitioning and Design Parameters

5.1. SP-NUCA

It must be clarified whether the flat-LRU replacement policy is the most cost effective method in SP-NUCA to dynamically partition shared and private ways. It will be compared against a much more accurate but also more costly method, such as Shadow Tags [19, 8]. The dynamic adaptation of the cache is done at set level having 8 shadow tags per set. The evaluation and enforcement of the policy is integrated within the replacement policy.

Additionally a static partition, similar to the one employed by [23], is included in the comparison. In this case, 12 ways are defined as private and the other 4 are reserved for shared blocks. As we can see in Figure 4, flat-LRU performance degradation is minimal compared with shadow tag, whereas the statically defined partition provides poor performance.

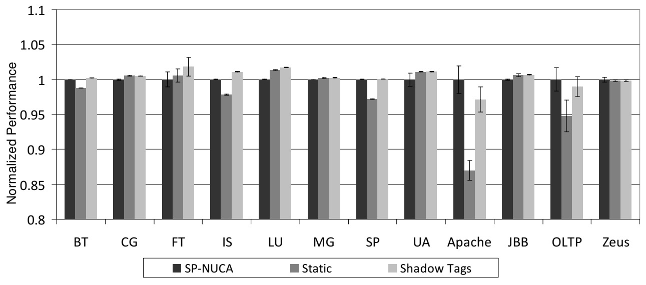


Figure 4. Dynamic partitioning in SP-NUCA. Normalized Performance of SP-NUCA LRU compared to Shadow Tags and a Static partition.

5.2. ESP-NUCA

The addition of helping blocks increases the potential for interference between threads. This subsection analyzes whether the proposed solution to limit these effects (Section 3.2) is significant. Figure 5 details the system performance normalized against SP-NUCA when different criteria are applied at replacement time when a helping block arrives at the bank. In ESP-NUCA with flat LRU, no restriction is applied when the least-recently used block in the set is a first-class block and the new block is a replica or victim. On the contrary, when protected-LRU is used, according to the implementation in Section 3.3, the performance advantage is greater. As we can see, ESP-NUCA improves performance where cache utilization is unbalanced or the replicas are beneficial. Although flat LRU improves SP-NUCA substantially, protected LRU achieves better performance stability, especially for very relevant applications like transactional workloads such as Apache and OLTP. Taking into account the limited implementation cost of protected LRU it seems reasonable to choose this replacement policy. Given the clear benefits of ESP-NUCA over SP-NUCA, in order to provide a clearer proposal, no more SP-NUCA performance analysis will be provided.

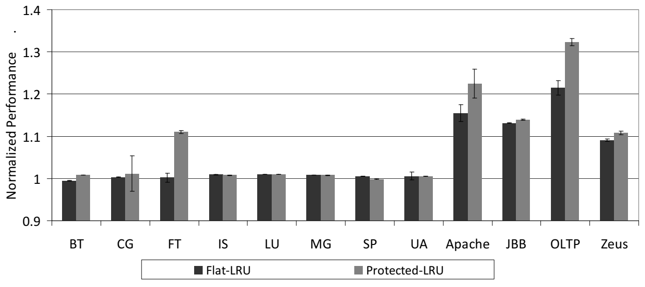


Figure 5. ESP-NUCA replacement policies normalized with SP-NUCA.

For these results the $HR_{C,R,E}$ have 8 bits (thus $b = 8$). The number of samples used in the hit-rate estimation N is 3, thus $\alpha = 0.5$ (from equation 1)) and $a = 1$ (equation 2). Two *Conventional Sets* are used for HR_C estimation, one *Reference Set* and one *Explorer Set* for HR_E and HR_R estimation. The maximum degradation for hit-rate of first-class blocks in the explorer set was found to be 88% ($d = 3$ eq (3)). This configuration to estimate different hit-rates has been determined after sweeping all parameters. Potentially, the dynamically defined d parameter provides the opportunity to add some Quality of Service Policy [11] on top of ESP-NUCA. However, we left this for future work.

This will be the configuration used for our proposal for the remainder of the paper. Note that the storage overhead is 4 bits per set to store the number of helping blocks in the set, 48 bits per L2 bank to store hit-rate estimation and 4 bits per bank to store the maximum number of helping blocks. For the configuration used, the aggregate storage overhead is approximately 9KB. Additional hardware is required to perform hit-rate estimations and the LRU implementation should be modified in order to establish the helping LRU block. There is no negative impact on hit-time given that the management of helping blocks is carried out outside the critical path to memory.

6. Performance Evaluation

6.1. Alternative Cache Designs

We have selected five counterpart architectures to show the efficacy of ESP-NUCA. In order to simplify the comparison, all architectures are implemented using a token-based coherence protocol. The first two architectures are: Static-NUCA, denoted as *Shared*, and Tiled architecture, denoted

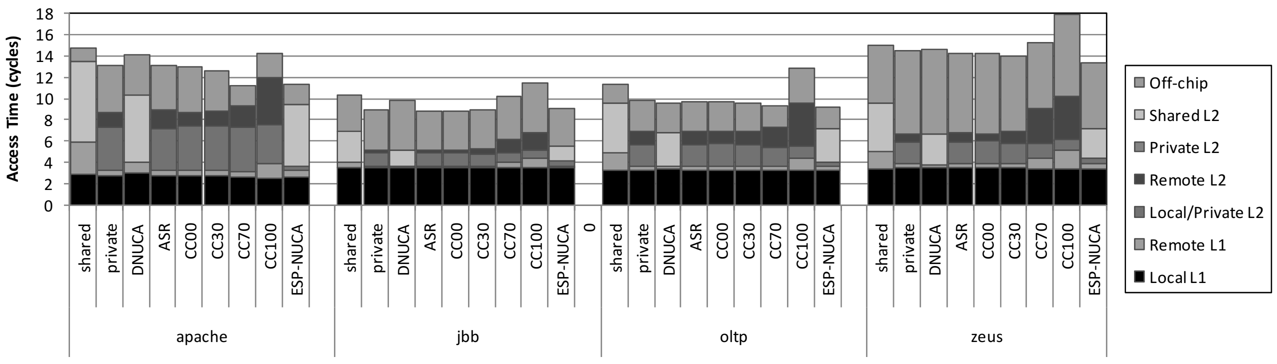


Figure 6. Average access time decomposition.

as *Private*. The S-NUCA uses all banks as a shared L2 cache (Figure 1a) and maps each cache block to a specific cache bank, depending on the address. The Tiled architecture uses the same layout (Figure 1a), associating each group of 4 L2 banks to its closest processor as a fully private cache portion. In order to make a fair comparison, all the rest of the parameters depicted in Table 2 are kept unchanged. The Tiled architecture uses unrestricted replication, so each L1 write-back is stored in its private L2 cache.

To clarify the performance of ESP-NUCA, three evolved architectures of *Shared* and *Private* alternatives have been included in the study. First, as an improved version of *Shared* architecture, the dynamically allocated NUCA (D-NUCA) [13] was selected. The D-NUCA implementation used is the same one used by [4], which assumes an idealized perfect-search and uses replication. Second, we include Adaptive Selective Replication (ASR) [3] as another possible improvement for private caches. ASR is able to self adapt the replication level to the workload characteristics. Finally, as an improved private architecture Cooperative Caching (CC) [5] was chosen. Different statically defined cooperation probabilities will be evaluated (0%, 30%, 70% and 100%). The performance results will show the average performance of all configurations, having the worst and best performer embedded in the variability bars.

Victim Replication [22] was not considered for the evaluation because it has been outperformed by both ASR and Cooperative Caching. CMP-NuRapid [6] has also been outperformed by ASR and requires a non-scalable interconnection network. Reactive-NUCA [9] is similar to our proposal, but it makes coarser-grain decisions (paged-based) and requires modifications to the OS. In any case, when the performance variability [1] is taken into account, R-NUCA seems to perform similarly to a shared NUCA, only winning in one benchmark. No software approaches, such as [7], have been included because they are complementary to hardware solutions like ours.

6.2. Transactional Workloads

Figure 6 shows the contribution to the average access time for each element in the memory hierarchy for these applications. As we can see, although the shared architecture has a low off-chip contribution, low on-chip locality impairs the final result. D-NUCA alleviates the on-chip locality problem but increases L2 miss rate, which limits its benefits. ESP-NUCA not only obtains an on-chip access time very close to D-NUCA architecture, which demonstrates the effectiveness of the mechanism improving on-chip locality, but has much better L2 hit rate than D-NUCA. In fact, the off-chip contribution is very similar to a shared cache. As we can appreciate in Figure 7, ESP-NUCA does a great job balancing on-chip hit latency reduction with less penalty in off-chip traffic. In contrast, ASR exhibits a very similar behavior to plain private cache. This is coherent with the results obtained in [9] with different workloads and evaluation infrastructure. Cooperative Caching in some cases is very similar to ESP-NUCA but in others performs poorly, being the best case changing from application to application.

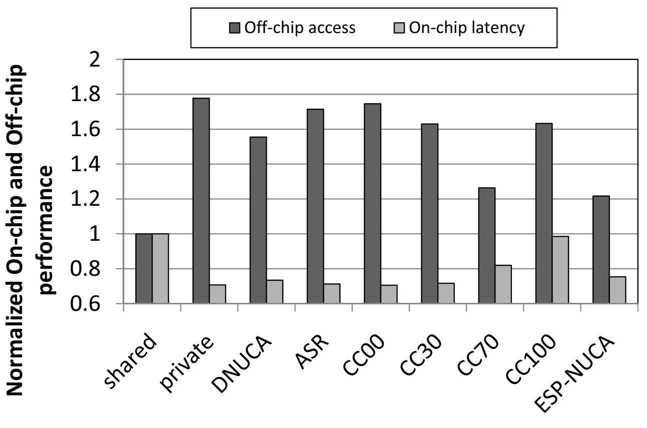


Figure 7. Average off-chip and on-chip behavior for Transactional Workloads.

As we can see in Figure 8, for these workloads ESP-NUCA's average performance improves on a conventional shared cache by 15%. ESP-NUCA is the best performer of all the counterparts achieving a superior number of workload transactions per unit of time. CC-Best in some cases outperforms ESP-NUCA, but it should be noted that the cooperation probability changes from application to application and the best CC has to be chosen for each application among the four configurations evaluated. In any case, for this class of benchmarks CC is showing an extremely variable behavior. In contrast, ESP-NUCA's behavior is fully adaptive.

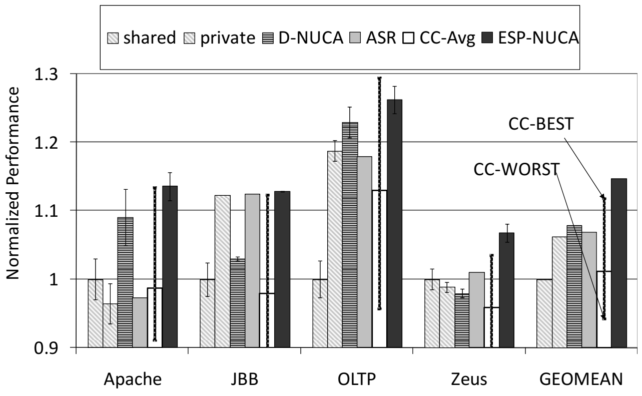


Figure 8. Shared Cache Normalized Performance for Transactional Workloads.

With the high level of shared data of these workloads [4] D-NUCA is the second best performer. Nevertheless, note the high cost of the D-NUCA implementation and the limited benefits. Something similar could be said about ASR. In contrast, ESP-NUCA with a much more limited cost is able not just to outperform other architectures but obtain better performance stability in these benchmarks too. In particular, ESP-NUCA achieves a performance variance 83% less than CC, 38% less than ASR and 74% less than D-NUCA for transactional workloads.

6.3. Multiprogrammed Workloads Results

Figure 9 shows the performance for these workloads. For single threaded applications, shared caches have a significant advantage over private caches, especially in low utility benchmarks with large data sets, such as *art* or *mcf*. The performance degrades for architectures without a cache balancing mechanism, such as *private* and ASR. They perform up to 40% worse than shared caches because in those architectures only half of the cache is available for the running thread. The CC mechanism diminishes this problem and has a quite stable performance. In *gcc* and *gzip*, the working set is small enough to fit in the private caches, and consequently the on-chip latency reduction of private architectures leads to a performance benefit. The combination of different effects inflicts a remarkable variation across the five applications chosen, except for ESP-NUCA which is the architecture with the most consistent behavior.

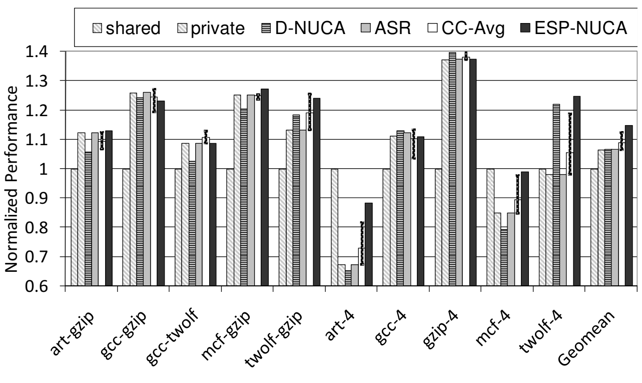


Figure 9. Shared Cache Normalized Performance for Throughput Applications.

In the hybrid benchmarks, on average ESP-NUCA is at the same level as CC-Best. In some cases, such as *gcc-gzip* and *gcc-twolf*, the inter-thread isolation achieved by CC with cooperation probability of 0% (equivalent to a private cache) is slightly better than ESP-NUCA. The *Shared* performance is clearly the worst alternative due to the large amount of unnecessary interference among threads. The average performance observed for each thread for this architecture, and to a lesser extent in Cooperative Caching, shows a high variability. ASR has a 100% higher variance in average IPC[3] than ESP-NUCA. Cooperative Caching has a 10% higher IPC variance and 110% in D-NUCA.

6.4. NAS Parallel Benchmark Results

Figure 10 shows the performance obtained for each application considered in this evaluation. The sharing degree of these applications is relatively limited, with large numbers of references and large percentages of cache capacity devoted to private data. In this context, private derived architectures have a clear advantage over shared derived ones because both the average hit-time and the inter-core interference are attenuated. Although *Shared* and D-NUCA have a poor performance, ESP-NUCA is able to tackle the situation: it is the only shared architecture derivative that is able to achieve a performance similar to private architecture derivatives.

The relatively large memory footprint of most applications should be noted. The working set for the problem sizes chosen is greater than 200MB. On average ASR complex

[3]Because there is no synchronization, we could use the average IPC of all cores as a valid performance metric.

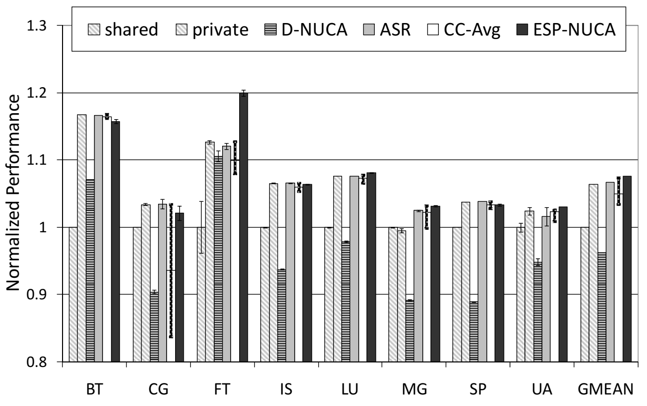


Figure 10. Shared Cache Normalized Performance for NPB Suite.

hardware implementation derives little to no benefit compared to a simple private architecture because, as the number of shared blocks is relatively small, the uncontrolled replication of shared data does not reduce hit-rate. The average performance for CC is quite variable. Note that the error bars for this counterpart include error simulation and variability observed for different cooperation probabilities. The best CC configuration is different for each application being 30% for IS, CG, MG and FT, 70% for LU and BT, and 100% for SP and UA.

Similarly to Transactional workloads, ESP-NUCA achieves better performance stability over the range of applications. In particular, the variance in execution time is 50% less than D-NUCA and 38% less than Cooperative Caching. ASR is 30% more stable than ESP-NUCA in these benchmarks.

7. Conclusions

This paper presents a cache architecture design with enhanced stability in several very different classes of applications. The properties observed suggest that ESP-NUCA could provide an interesting alternative in general purpose high performance CMP systems. Our proposal does not require unsustainably complex mechanisms to enable the cache hierarchy to self-adapt to workload behavior. By using decisions exclusively taken based on local information, we can improve the overall performance of the system. The best characteristics of shared and private caches are combined in this new architecture, to achieve unparalleled stability across a broad spectrum of benchmarks.

8. Acknowledgements

This work has been supported by the Spanish Ministry of Science and Innovation, under contract TIN2007-68023-C02-01 and by the HiPEAC European Network of Excellence.

References

[1] A. R. Alameldeen et al. Evaluating non-deterministic multi-threaded commercial workloads. *Workshop on Comp. Arch. Evaluation using Commercial Workloads*, 2002.

[2] L. A. Barroso et al. Memory system characterization of commercial workloads. In *ISCA-98*.

[3] B. M. Beckmann et al. ASR: Adaptive selective replication for CMP caches. In *MICRO-06*.

[4] B. M. Beckmann and D. A. Wood. Managing wire delay in large chip-multiprocessor caches. In *MICRO-04*.

[5] J. Chang and G. S. Sohi. Cooperative caching for chip multiprocessors. In *ISCA-06*.

[6] Z. Chishti et al. Optimizing replication, communication, and capacity allocation in CMPs. In *ISCA-05*.

[7] S. Cho and L. Jin. Managing distributed, shared l2 caches through os-level page allocation. In *MICRO-06*.

[8] H. Dybdahl et al. An adaptive shared/private NUCA cache partitioning scheme for chip multiprocessors. In *HPCA-07*.

[9] N. Hardavellas et al. R-NUCA: Data placement in distributed shared caches. *ISCA-09*.

[10] J. Huh et al. A NUCA substrate for flexible CMP cache sharing. In *ICS-05*.

[11] R. Iyer. CQoS: a framework for enabling QoS in shared caches of CMP platforms. In *ICS-04*.

[12] H. Jin et al. The OpenMP implementation of NAS parallel benchmarks and its performance. Technical report, 1999.

[13] C. Kim et al. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *ASPLOS-02*.

[14] P. S. Magnusson et al. Simics: A full system simulation platform. *IEEE Computer*, 35(2):50–58, 2002.

[15] M. M. K. Martin. *Token Coherence*. PhD thesis, University of Wisconsin-Madison, Dec. 2003.

[16] M. M. K. Martin et al. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Comput. Archit. News*, 33(4):92–99, 2005.

[17] M. M. K. Martin, M. D. Hill, and D. A. Wood. Token coherence: Decoupling performance and correctness. In *ISCA-03*.

[18] M. K. Qureshi and Y. N. Patt. Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *MICRO-06*.

[19] G. E. Suh, L. Rudolph, and S. Devadas. Dynamic partitioning of shared cache memory. *J. Supercomputing*, 2004.

[20] The Standard Performance Evaluation Corporation. Spec-cpu2000, http://www.spec.org/cpu2000/.

[21] S. Thoziyoor, N. Muralimanohar, and N. Jouppi. CACTI 5.0: An integrated cache timing, power, and area model. Technical report, HP Laboratories Palo Alto, 2007.

[22] M. Zhang and K. Asanović. Victim replication: Maximizing capacity while hiding wire delay in tiled chip multiprocessors. In *ISCA-05*.

[23] L. Zhao et al. Towards hybrid last level caches for chip-multiprocessors. *SIGARCH Comput. Archit. News*, 2008.