N

NUMA Caches

Alessandro Bardine, Pierfrancesco Foglia, Cosimo Antonio Prete, Marco Solinas
Università di Pisa, Pisa, Italy

Definition

NUMA is the acronym for Non-Uniform Memory Access. A NUMA cache is a cache memory in which the access time is not uniform but depends on the position of the involved block inside the cache. Among NUMA caches, it possible to distinguish: (1) the NUCA (Non-Uniform Cache Access) architectures, in which the memory space is deeply sub-banked, and the access latency depends on which sub-bank is accessed; and (2) the shared and distributed cache of a tiled Chip Multiprocessor (CMP), in which the latency depends on which cache slice has to be accessed.

Discussion

Introduction

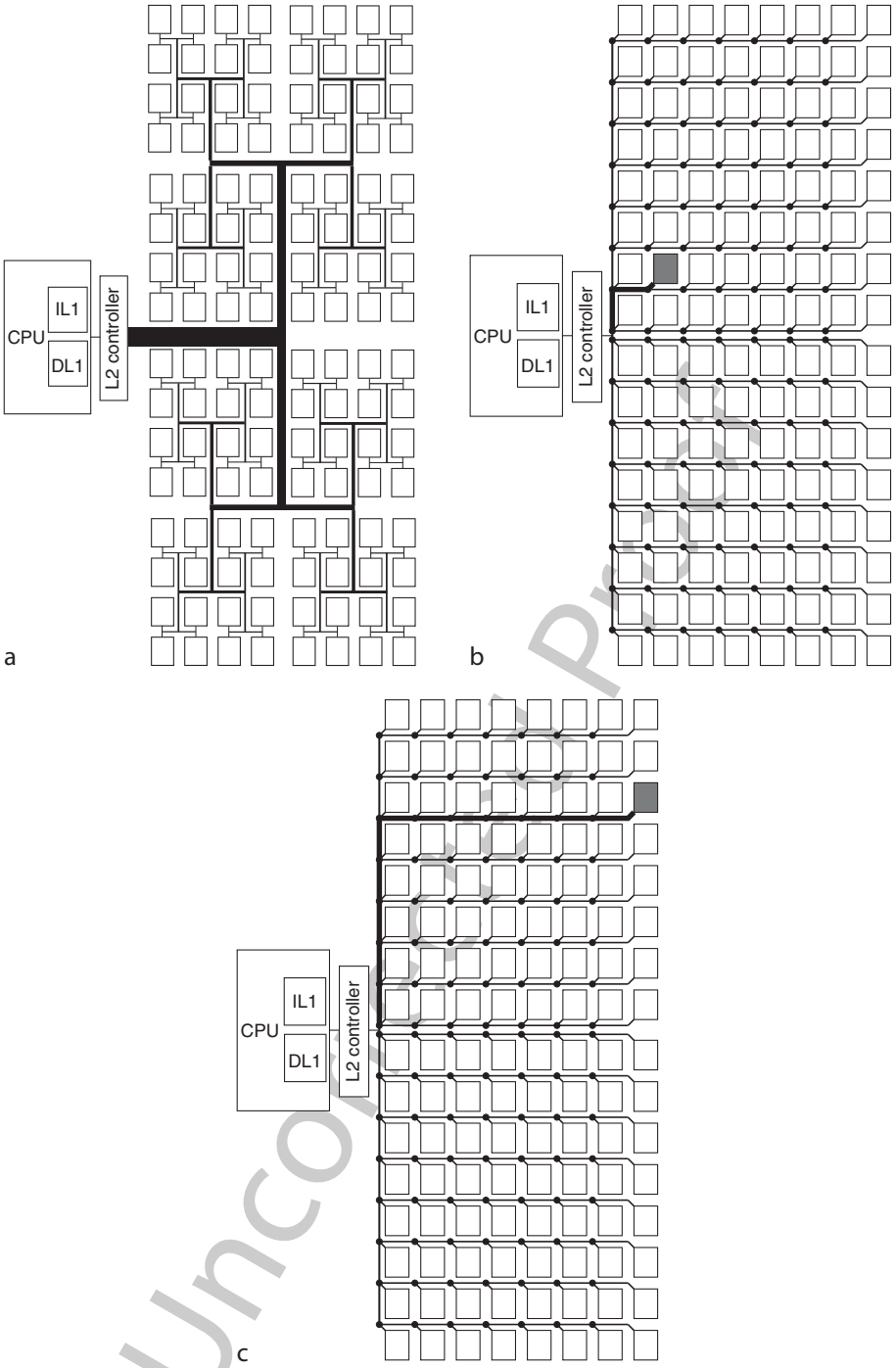
For the past decades, microprocessors' overall performance has been improved thanks to the continuous reduction of transistor size obtained in silicon fabrication technology. This scaling contributed in both (1) allowing designers to put on a chip more and more transistors, and thus to implement on the same die more and more complex microarchitectures, up to arrive to single Chip Multiprocessor (CMP) systems, and (2) increasing processors' clock frequencies.

As the memory bandwidth requirements of cores have increased as well, the increased number of on-chip transistors has also been used to integrate on the same die deeper and larger memory hierarchies. However, reducing feature sizes has also introduced the *wire-delay* problem: sizes of on-chip wires have been also reduced, resulting in larger delay for signals propagation [1]. In particular, considering a huge traditional on-chip cache, the wire delay and the increase of clock frequency lead to an increase of the latency experienced on each access. The bulk of access time involves signals' routing to and from cache banks and not the bank access themselves. Exploiting a non-uniform access time for on-chip caches is a viable solution to the wire-delay problem.

By allowing non-uniform access time also to cache memories, it is possible to design a cache based on independently accessible banks, connected via a scalable connection fabrics, typically a Network-on-Chip (NoC) [2, 4, 5], in which the access latency is proportional to the physical distance between the requesting unit and the accessed bank. This organization is called NUCA (Fig. 1b and c): banks close to the requesting unit can be accessed faster than a traditional monolithic cache. If the cache management policies succeed in keeping the most performance impacting blocks in such banks, the overall performance will be boosted. NUCA architectures have been proposed for both single-core [2, 3, 6] and multi-core [4, 5, 7–13] environments.

Needs for scalability in CMP designs has lead to the design of CMP tiled systems that are composed by many identical nodes, called tiles. Each tile is composed by one cpu, its private cache levels, and the LLC. Nodes are typically connected through a NoC. In these systems, the LLCs can be used as a globally shared cache, the LLC inside each node being a slice of the whole cache. Hence, the access time depends on respective position of the requesting node and of the memory slice involved in the access, and this results in an non uniform access time. Those systems results to be implicitly NUMA caches as a consequence of their tiled architecture, differently from NUCA architectures whose design explicitly focuses on the non-uniformity in access time in order to tolerate the wire-delay effects.

David Padua (ed.), *Encyclopedia of Parallel Computing*, DOI 10.1007/978-0-387-09766-4,



NUMA Caches. Fig. 1 Examples of systems adopting a traditional sub-banked UCA cache **(a)** and a NUCA cache **(b** and **c)**. In all the designs the memory space is partitioned in 128 banks (white squares in the picture). UCA adopts the H-Tree connection model, while NUCAs use a network on chip made up of routers (black circles in the picture) and links. In this way, each bank is independently accessible from the others and the access latency of a block that is in a close to the controller bank **(b)** is lower than access latency that is in a far bank **(c)**

NUMA Caches in the Single-Core Environment: NUCA

In a traditionally organized cache, each bank is connected to the controller via a fixed length path, usually organized according to the H-Tree model (Fig. 1a). Thus, in a wire delay–dominated context, the access latency of each bank is dominated by the constant length of the path followed by control and data signals, and it is independent from the physical position of the bank. This organization is referred as UCA (Uniform Cache Access) as its access time is *uniform*.

The basic idea of NUCA, first proposed by Kim et al. [2], is to design the cache so that banks are connected among them and with the controller via a connection fabric, typically a NoC. Figure 1b and c show a system equipped with a Level 2 NUCA, partitioned in 128 independent banks, connected with a partial 2-D mesh NoC.

The adoption of a NoC allows to access each bank independently from the others. In particular, in a NUCA, there is a different access latency for each bank depending on its physical distance from the controller. Being in a wire delay–dominated environment, the signal propagation delay dominates overall access latency. Hence banks that are closer to the controller (Fig. 1b) can be accessed faster than the others that are farther (Fig. 1c). As a consequence, the access time is *non-uniform* as it is proportional to the physical distance to be traversed by signals. This architectural design is complemented by data management policies that maintain critical blocks in faster banks. This allows better performance with respect to UCA architectures by obtaining a lower average cache access latency, thus hiding wire-delay effects.

Data Management Policies: S-NUCA and D-NUCA

When designing a NUCA, one of the main choices is related to the *mapping rule* that dictates which bank can hold each block. The simplest mapping rule is the *static mapping*: a block can exclusively reside in a single predetermined bank, basing on its memory address. A NUCA cache adopting the static mapping is called Static NUCA (S-NUCA). An alternative rule is the *dynamic mapping*: a block can be hosted in a set of banks, called *bankset*, and can be moved inside the bankset. A NUCA cache adopting the dynamic mapping is called Dynamic NUCA (D-NUCA) [2].

S-NUCA

In an S-NUCA made up of N banks, the bank to be accessed is select basing on the value of *log(N)* bits of the address. Proposed policies for static mapping are *sequential mapping*, which uses the most significant bits of the index field of the physical address, and the *interleaved mapping*, which uses the low-order bits of the same field in order to distribute among different banks blocks that are memory contiguous, thus reducing bank contentions.

When searching for a block, the controller injects into the NoC, a request that will be delivered to the pertaining bank. If the block is present, i.e., the cache access results in a *hit*, the bank sends it to the controller. If the block is not present, i.e., there is a *miss*, the block is retrieved from the main memory, stored in the cache bank, and then delivered to satisfy the request.

Despite the simple design and management policies, an S-NUCA usually exhibits lower-average access latencies with respect to traditional UCA caches. However, as the data mapping doesn't take into account the data frequency usage, it may happen that more frequently used data are mapped in banks that are far from the controller, thus resulting in higher access latencies and poor performances.

D-NUCA

In a D-NUCA, banks are grouped into banksets, and each block can be stored in any of the banks that belong to the pertaining bankset: for example, in the systems of Fig. 1b and c, a bankset could be made up of all banks belonging to the same row. The bankset is chosen basing on the address of the block itself, using *log(number of bankset)* bits of the index field, adopting either the sequential or the interleaved mapping. Blocks can move from one bank to another of the same bankset thanks to the *block migration* mechanism. Migrating most frequently used data from farthest banks to closest banks allows to reduce the average access latency, and thus to achieve better performance with respect to S-NUCA.

The three main topics in designing a D-NUCA are:

- *Bank Mapping*: how to map bankset to physical banks?

- *Data Search*: how the possible locations are searched to find a block?
- *Movement and replacement*: how and when the data should migrate? In which bank of the bankset a new block should be placed?

Bank Mapping

The bank mapping policy of a D-NUCA dictates how to group the available physical banks into banksets. Many different policies can be adopted, each offering a different trade-off in terms of implementation complexity and fairness among the various banksets of the latency distribution. Proposed mapping policies are [2]: (1) *simple mapping*, (2) *fair mapping*, and (3) *shared mapping*.

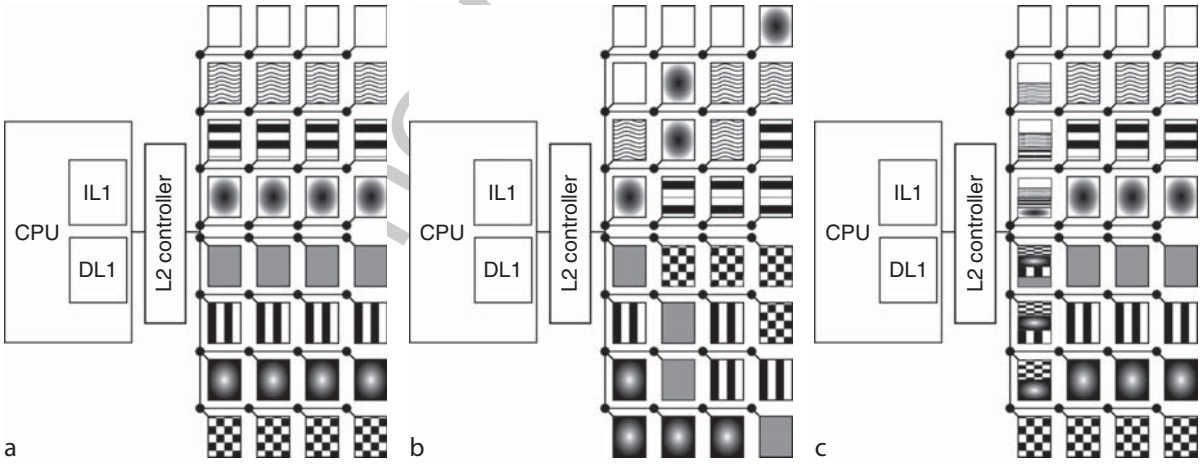
The simple mapping (Fig. 2a) maps a bankset to all the banks belonging to the same row. Such policy is simple to be implemented, but there are very different access latencies between banksets that are mapped to the central rows of the cache and those that are mapped in the other rows; to access them, a longer network path along the vertical direction must be traversed. In the fair mapping (Fig. 2b), this is avoided at the cost of additional complexity: the banks are allocated to banksets so that the average access times to each bankset are equalized. In the shared mapping (Fig. 2c), the banksets share the banks closest to the controller so that a fast bank-access is provided to all the sets.

Data Search

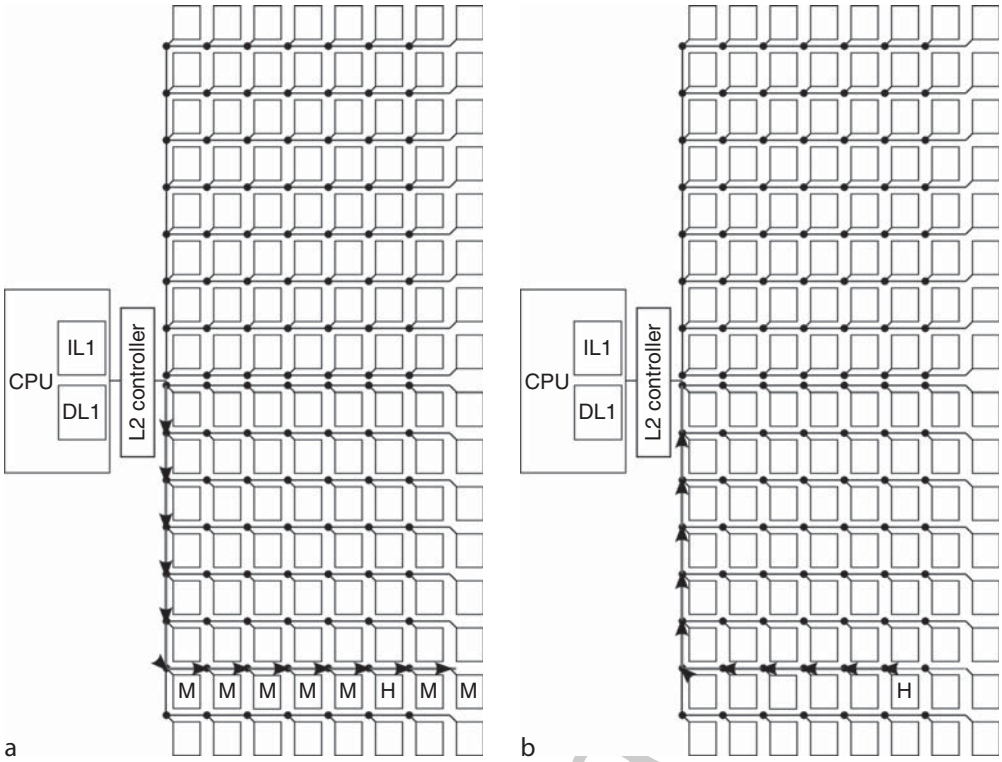
When the search is performed, all the banks of the bankset that can contain a referred block must be searched. The controller injects in the network a request that will be delivered to all the pertaining banks. Each bank performs a search inside itself. If all the banks miss the searched block, then there is a *cache miss*, and the block must be retrieved by the next level in the hierarchy memory. Alternatively, one bank contains the searched block, there is a *cache hit*, and the block is sent to the controller.

Examples of proposed search policies for D-NUCA caches are *multicast search* and *incremental search* [2]. In the multicast search, the request is delivered in parallel to all the possible banks, and each of them starts the search as soon as possible. Figure 3 shows an example of memory operation in a D-NUCA adopting the simple mapping scheme and the multicast search. As opposite, in the incremental search, the banks of a set are sequentially searched from the closest to the farthest. The delivery of the request to each bank is performed only if the previous bank has missed the block. In such a way, the number of bank accesses and the associated energy consumption are reduced at the cost of an increase of the operation total latency.

Other solutions have been proposed [5] mixing the two techniques: the banks of one bankset are divided in two or more groups; inside each group the multicast search is adopted, while the various groups are searched incrementally.



NUMA Caches. Fig. 2 Three examples of bank mapping: simple mapping **(a)**, fair mapping **(b)** and shared mapping **(c)**



NUMA Caches. Fig. 3 The multicast search operation in a D-NUCA adopting the simple mapping: the request first traverses the vertical links to the pertaining bankset, then traverses the horizontal links and routers where is replicated to reach all the banks of the bankset **(a)**. The bank containing the block sends a reply to the controller **(b)** that follows the inverse path of the request message

Data Movement and Replacement

Basing on the locality principle, chances there are that, when a block is referred, it will be referred again in a short time. So when a block is accessed, it makes sense to move it in another bank of the bankset that is closer to the controller. This movement is called *data promotion* [2].

An ideally desirable policy would be to use LRU ordering to order the blocks in the banksets, with the closest bank holding the MRU block, second closest holding second most-recently used, etc. Maintaining this ordering would require a heavy block redistribution among banks on each access, with consequent impact on network traffic and energy consumption.

For this reason, more practical low-impact policies are employed usually based on *generational promotions* [2] of blocks basing on their access pattern. For example, in the D-NUCA cache of Fig. 1c on every hit, the accessed block is moved in the left direction by one bank. In such a way, the next access to the same block will incur a lower latency. This migration policy is called *per-hit promotion*.

More generally, designing a generational data promotion, in the case of a D-NUCA, requires the definition of three elements: the *promotion trigger* (number of hits after which a promotion must be triggered), the *promotion distance* (number of banks that the promoting block must be advanced), and the *initial placement* of a new block after a cache miss.

Another important design decision involves what to do with the block that, in the new bank, is eventually occupying the line in which a promoting block must be inserted. Basic choices for this design issue are the *demotion* of the block in the bank previously hosting the promoted block or its eviction from the cache. Demotion requires more bank accesses and network traffic

than eviction; however, the latter affects performance as it evicts blocks without taking into account their actual usefulness.

Different choices result in different power, network traffic, and performance trade-off; however, the majority of the proposed D-NUCA designs [4–6, 8–10] adopt the per-hit promotion (i.e., one hit as promotion trigger and one bank as promotion distance) and insert new block in the farther from the controller bank. Increasing the value of promotion trigger has been shown to affect performances too much as it limits the promotion of newly loaded data that, basing on the temporal locality principle, are likely to be requested at the next accesses. At the same time, increasing the promotion distance involves the rapid demotion of data that, because of the spatial locality principle, may be needed again in the next future. Using different insertion point for newly loaded data means evicting from the cache blocks that are still useful as they occupy banks that are close to the controller.

NUMA Caches in the CMP Environment

The increase in the number of available on-chip transistors and the demand for increasing performance have driven the design of CMP systems that include more elaborating cores and a multilevel cache memory hierarchy in a single die. The adoption of a large on-chip multilevel cache allows to guarantee fast memory access to each core.

CMP caches are usually organized in two or more levels in which the Last-Level-Cache (LLC) can be shared among all on-chip cores, while all the other levels are private for each core. As the LLC is usually quite large, it is likely to be affected by the wire-delay problem. For this reason, a CMP system can adopt a shared NUMA cache as its LLC [4, 5, 8–10].

System Topologies

Taking into account the respective position of cache banks and cpus, CMP systems can be classified in two main topology families: *dancehall* and *tiled*.

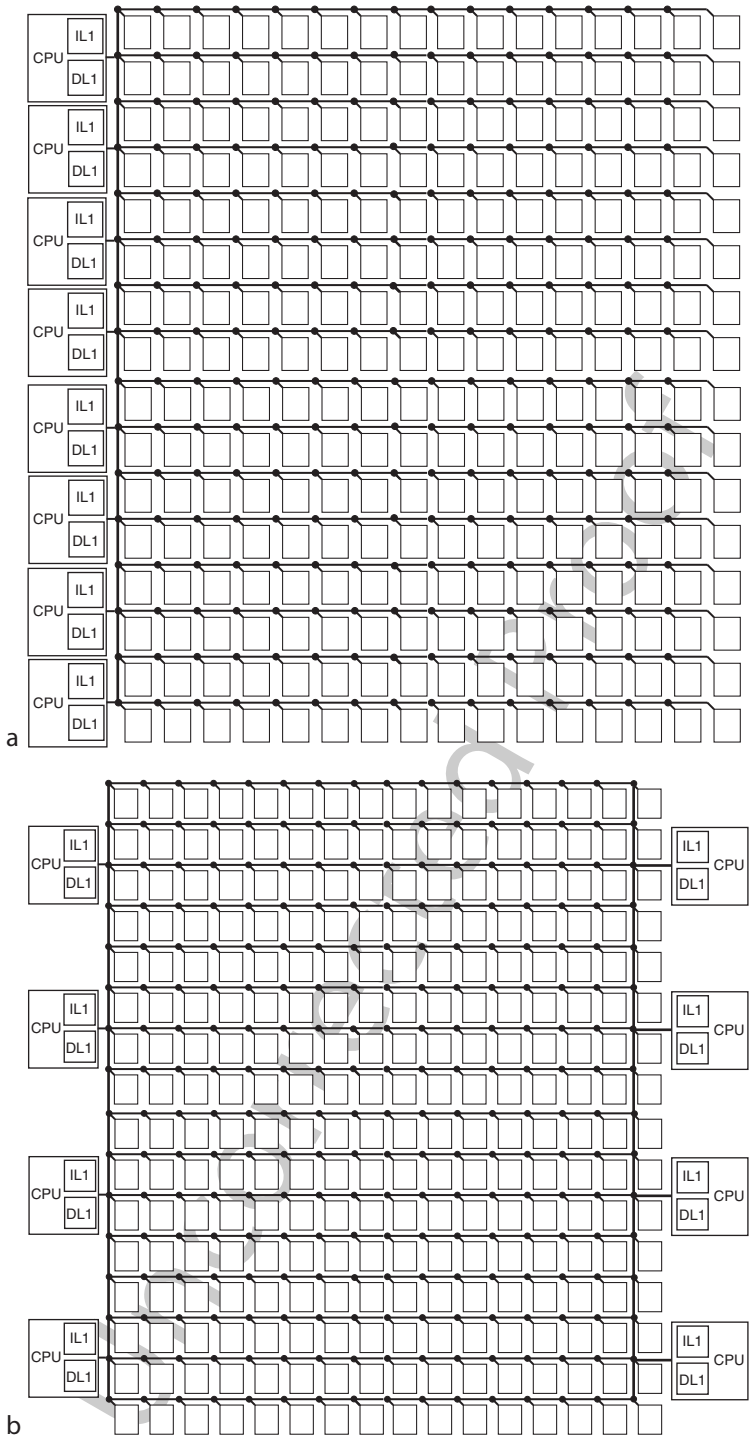
In the dancehall, the shared LLC cache is concentrated in an area of the chip, while cpus stay at one or more sides of the cache. Taking into account a NUCA LLC CMP [4, 5, 7–10, 12], cpus and cache banks are all connected via a NoC (or any other connection fabric), resulting in a NUCA design similarly to the single core case. Figure 4a shows an example of dancehall CMP in which cpus are plugged to the same side of the NUCA. Variations of this simple configuration have half of the cpus plugged at one side of the shared NUCA, and the others plugged to the opposite side (Fig. 4b), or all the cores distributed along all the sides of the banks matrix [5].

The tiled topology, as shown in Fig. 5, is composed by many identical nodes, called tiles. Each tile is composed by one cpu, its private cache levels, and a LLC that can be used as a private cache or as a local slice of a globally shared LLC [11–13]. In the last case, the access time to the shared LLC depends on the position of both the requesting node and the looked-up LLC slice. In particular, for a core, an access to a remote slice will result in greater latency, with respect to an access to the local slice, due to network paths that vary with the physical distance from the local to the remote tile.

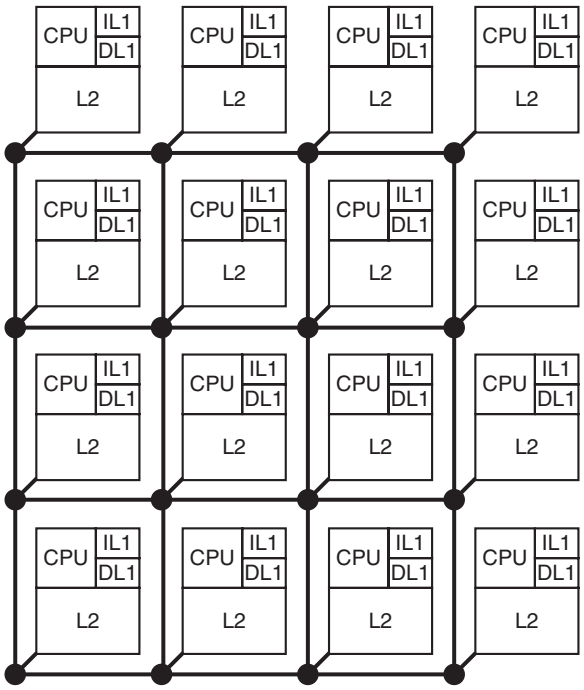
When designing a CMP system, choosing between such two families of topology represents a trade-off between performance and scalability. The three main components of the NUMA access time are: (1) the number of NoC hops to be traversed to reach the hosting bank, (2) the latency of each hop, and (3) the access time to the banks that stores the block. Hence, if the bank to be accessed is small and close to the requesting node, the access latency will be limited. If the interested bank is huge and far from the requesting node, the cost of the cache access will increase proportionally to the number of hops to be traversed.

Adopting the dancehall topology results in a large number of small banks, each characterized by small response latencies. Moreover, as banks are small, NoC links are short because they have to surround cache banks. Thus they are characterized by a limited link traversal delay. Instead, with the tiled organization, there is a small number of greater and slower banks. Links are longer since they surround the entire tile, and consequently, their traversal delay is higher. Consequently, a request to any nonlocal slice traverses a small number of higher cost network hops.

Adopting a tiled topology will increase scalability at the cost of high cache access due to huge cache slice and high network hop traversal cost. Instead, adopting a dancehall topology minimizes the latency of each



NUMA Caches. Fig. 4 NUCA based CMP systems adopting the *dancehall* topology, with 8 cpus with private L1 caches, and a shared L2 NUCA cache made of 16 × 16 banks. All the cpus, together with their private caches, can be plugged at the same side of the NUCA **(a)**, or at two opposite sides **(b)**



NUMA Caches. Fig. 5 A NUCA based CMP system adopting the *tiled* topology, with 16 cpus with private L1 cache, and a shared NUCA cache composed by 16 slices, one for each tile

hop, as well as the access time to each single bank, but presents a smaller degree of scalability than the tiled case.

Coherency in CMP Systems Adopting NUMA Caches

In a CMP system, private cache levels must be kept coherent. Given that NUMA LLC adopt a communication fabric which does not guarantee sequencing of coherence traffic (e.g., the NoC), a directory-based coherence protocol, similarly to the ones proposed for classical DSM systems, must be adopted. The *directory* is a node of the system that tracks which of the private caches hold a copy of each block. In CMP systems with NUMA LLC, the directory can be *centralized* or *distributed*.

A centralized directory is a monolithic node of the system separated from the remaining nodes. This solution presents some scalability issues because all the nodes must access it to guarantee the correctness of memory operations. When the number of such nodes increases, then directory contentions increase as well, resulting in higher response time and thus affecting performance.

The distributed directory is typically implemented inside each NUMA cache bank. Directory information is stored near the TAG field of each cached block, using some specific status bits. As a consequence, directory accesses are distributed among all banks and, when a *miss* in the last private level cache occurs, an unique access is used to retrieve both the block and the directory information.

Mapping Policies

Also for the NUMA cache in CMP systems, the two possible mapping policies are static and dynamic. Due to the presence of many traffic sources, in both cases the NoC traffic increases proportionally to the increase in the number of cores per chip. At the same time, the contention of shared blocks introduces new phenomena that are strictly tied to both the mapping policy and the topology.

Static Mapping

Static mapping has been adopted both in the *tiled* [11] and in the *dancehall* [8, 9, 12] designs. The static mapping associates each memory address to a single cache bank, which is able to satisfy both read/write and coherence requests coming from any of the private next-level cache.

The path that request and coherence messages have to traverse to reach the interested bank depends on the respective position of requestor and bank. If the bank is far from the requestor, the message has to traverse many network hops before reaching the bank. If the bank is close to the requestor, the network path will be short. As the bank to be accessed is chosen basing on the block address, the banks that a processor accesses depend on how the *working set* of the running process/thread is mapped on memory, i.e., which are the physical addresses that are accessed. So, if the running process/thread's *working set* is mapped to remote banks, then the cache access latency will be high; otherwise it will be small. Consequently, both topology aspects and memory mapping rules contribute in affecting or enhancing overall performance.

Dynamic Mapping

Block migration can be used to improve performance also in CMP systems with a NUMA LLC. *Dancehall* [4, 5, 7, 10, 12] and *tiled* [11–13] designs have been proposed in which cache blocks are able to move among banks aiming to reducing response latency.

As for the *dancehall* case, the design results in a D-NUCA similar to the single core case. Also in this context, the shared D-NUCA is usually considered as partitioned in many banksets, and cache blocks are able to migrate among banks belonging to the same bankset [4, 5, 8–10]. In the systems shown in Fig. 4a and b, banksets are represented by the rows of the NUCA bank matrix. If D-NUCA succeeds in bringing the most frequently accessed blocks near to the referring cpu(s), overall performance will be boosted. However, the performance gain that could be obtained with migration comes with some design issues, as in a CMP, and there are many traffic sources that complicate the managing of the block migration process.

Blocks that are shared by two or more threads can be accessed by different nodes if the sharers are running on different cpus. This is not necessarily a disadvantage but depends on the overall system topology. By considering the system of Fig. 4a, as all the cpus are placed at the same D-NUCA side, all the blocks migrate toward the same direction because the faster way is the same for all cpus. Instead, in the system of Fig. 4b, the fastest way for half of the cpus is the slowest for the others, and vice versa. If the sharers are running on cpus plugged at opposite D-NUCA sides, shared blocks will alternatively migrate in both directions, and none of the cpus will succeed in bringing such blocks in the respective faster way. This phenomenon is known as *conflict hit*, and if not properly managed, reduces the performance gain deriving from block migration.

Another design issue of D-NUCA CMP systems regards the *initial placement* of a new block after a cache miss (i.e., choosing which bank of each bankset will store incoming blocks). Considering the *dancehall* configuration of Fig. 4a, choosing the fastest way as the block entry point would interfere with most referred blocks, and similar considerations can be done as in the single core case. Instead, for the system shown in Fig. 4b, the farthest way for half of the cpus is the closest for the other half, so blocks loaded by half of the cpus would interfere with most frequently used blocks of the other half. Consequently, a possible trade-off could be choosing the *initial placement* in the central banks.

Alternative schemes, for both the *dancehall* and *tiled*, have been proposed [11, 12], in which blocks are initially placed near the first requestor and stays there until a different cpu requests the block. In this case, the block migrates toward a bank (or a slice) that is determined according to its memory address basing on a static mapping policy. On subsequent requests, the block is not further moved. Dually, migration schemes have been proposed so that blocks are initially placed basing on their memory address and subsequently migrated basing on the position of the referring cpu [13].

Dynamic block movement designs must face the *false miss* problem. This is a particular race condition that can occur when a subsequent access to a migrating block arrives when the block is still *on-the-fly*. A block migration involves two banks: the *source* and the *destination*. If the new request is received by the *source* after the block has been sent and by the *destination* before the migrating block has arrived, both the accesses will result in a miss; actually, the block is present in cache and must not be retrieved by the next level memory. That has two disadvantages: (1) affect performance because off-chip accesses are always more expensive than cache accesses; (2) the correctness of memory operations is affected as there would be two (or more) unmanaged copies of the same block. Techniques for resolving such problem have been proposed, for example, a directory holding all the cached block addresses can be accessed in order to recognize false misses [5], or a specific false miss avoidance algorithm can be designed [10].

Research Directions

Promising lines of studies for NUMA caches are:

1. Power-consumption saving techniques. While being able to reduce wire-delay effects, NUCA caches still suffer of another typical problem of nanometers technologies: the static power consumptions due to the increase of leakage currents. Besides, due to the additional NoC traffic and bank access, D-NUCA exhibit an increase also in the dynamic power consumption [14]. Even if NUCA caches are made up of

traditional cache banks, the direct adoption of well-known techniques like, for example, drowsy memories and decay cache lines, may not be effective particularly in D-NUCA because of the data movement that modify bank access pattern. An alternative proposed technique [15, 16] exploits different usage of banks due to block migration to power-gate farther and less used banks, and has been introduced for both the single-core case and the CMP environment in which all the cpus are connected to the same cache side. However, when cpus are distributed around more NUCA sides due to the problems that arise in D-NUCA based CMP systems, the concept of farthest banks is ambiguous and has to be redefined; thus the raw application of such technique is no longer possible.

2. Performance improvement via remapping policies. Memory remapping at compile time is traditionally used to improve cache memory performance, in particular for reducing conflict misses. In the context of NUCA caches, similar techniques can be used also for placing data block in banks (or banksets) that reduce their average access latency [17]. Typical samples of used metrics in mapping policies are the data usage frequency and their being shared or private. Remapping can be implemented also via hardware, for example, by designing specific protocols that dynamically change the hosting bank (in the case of S-NUCA, similarly to [12]) or bankset (in the case of D-NUCA).
3. Efficient block migration schemes are an open issue for CMP caches. Generational promotion schemes as in the case of single-core D-NUCA are effective in *dancehall* systems in which cores are all plugged at the same cache side. In all the other cases, such simple schemes are no longer effective due to the *conflict hit* problem that can arise on shared data. Alternative solutions are likely to take into account the position of cores that share a specific block in order to migrate it in a position that optimizes overall performance (similarly to [13]). Moreover, blocks replication could be evaluated as a viable solution to the *conflict hit* problem so that each copy can migrate toward a specific requestor. Such scheme must take care of the need of each application. In fact, while the convenience of replicating Shared-Read-Only block is obvious, there may be classes of application, in which replicating also Shared-Read-Write blocks can contribute in improving performance.

Related Entries

►Cache Coherency
►Interconnection Networks/Topology/Routing
►Memory Models
►Memory Wall
►Network On Chip
►Shared Memory Multiprocessor

Bibliographic Notes and Further Reading

The first NUCA cache architecture was proposed in [2], and demonstrates that a dynamic NUCA structure achieves an IPC 1.5 times higher than a traditional Uniform Cache Architecture (UCA) when maintaining the same size and manufacturing technology. Leveraging on alternatives data mapping policies, NuRapid [4] and Triangular D-NUCA cache [6] have been proposed as alternative designs for NUCA architectures in order to optimize the trade-off between area occupation and performance. An energy model for NUCA architecture is proposed in [14] together with an energy/performance trade-off evaluation for NUCAs and its comparisons with UCA. The Way Adaptable D-NUCA architecture is proposed in [15, 16] shows that it is possible to dynamically adapt the cache size to the application needs, thus consistently reducing NUCA static power consumption while marginally affecting the performances. The impact of NoC elements parameters on NUCA performances have been analyzed in [18, 19]. The same work proposes an alternative design for NUCA caches that relaxes the constraints imposed on network routers latency by clustering multiple banks around each router.

In the context of on-chip multiprocessors, the behavior of D-NUCA as shared L2 caches has been analyzed in [4, 5, 7]; these works evaluate the performance of different data mapping and migration policies. Topology studies for S-NUCA and D-NUCA, related to the dancehall configurations, can be found in [8–10]. These works also propose a directory coherence protocol specifically suited for CMP NUCA environment and a solution to some design issue of block migration

policies in D-NUCA-based CMP systems. The analysis in [11] is based on the tiled architecture, and studies also an intelligent block placement, while in [12] is proposed a scheme called SP-NUCA in which private blocks are dynamically recognized and then stored in S-NUCA banks that are very close to the owner. The system proposed in [13] is based on a tiled architecture and implements a migration mechanism that places blocks in slices depending on the sharers' position.

Bibliography

1. Matzke D (1997) Will physical scalability sabotage performance gains? IEEE Comput 30(9):37–39.
2. Kim C, Burger D, Keckler SW (2003) Non uniform cache architectures for wire-delay dominated on-chip caches. IEEE Micro 23(6):99–107.
3. Chisti Z, Powell MD, Vijaykumar TN (2003) Distance associativity for high-performance energy-efficient non-uniform cache architectures. Proc. 36th int. symp. on microarchitecture. San Diego, CA, pp 55–66.
4. Huh J, Kim C, Shafi H, Zhang L, Bourger D, Keckler SW (2005) A NUCA substrate for flexible CMP cache sharing. Proc. of the 19th int. conf. on supercomputing. Cambridge, MA, pp 20–22.
5. Beckmann BM, Wood DA (2003) Managing wire delay in large chip-multiprocessors caches. Proc. of 37th int. symp. on microarchitecture. San Diego, CA, pp 55–66.
6. Foglia P, Mangano D, Prete CA (2005) A cache design for high performance embedded systems. J Embedded Comput 1(4):587–598.
7. Chisti Z, Powell MD, Vijaykumar TN (2005) Optimizing replication, communication, and capacity allocation in CMPs. Proc. of the 32nd int. symp. on computer architecture. Madison.
8. Foglia P, Panicucci F, Prete CA, Solinas M (2009) Investigating design trade-off in S-NUCA based CMP systems. Proc. workshop on UNIQUE CHIPS and SYSTEMS (UCAS-5). Boston, vol. 1, pp 53–60.
9. Foglia P, Panicucci F, Prete CA, Solinas M (2009) An evaluation of behaviors of S-NUCA CMPs running scientific workload. Proc. of the 12th euromicro conference on digital system design (DSD). Patras, Greece, pp 26–33.
10. Foglia P, Panicucci F, Prete CA, Solinas M (2009) Analysis of performance dependencies in NUCA-based CMP systems. Proc. of the 21st international symposium on computer architecture and high performance computing (SBAC-PAD), Sao Paulo.
11. Hardavellas N, Ferdman M, Falsafi B, Ailamaki A (2009) Reactive NUCA: near-optimal block placement and replication in distributed caches. Proc. of the 36th international symposium on computer architecture (ISCA-09). Austin, pp 184–195.
12. Merino J, Puente V, Prieto P, Gregorio JÁ (2008) SP-NUCA: a cost effective dynamic non-uniform cache architecture. ACM SIGARCH Computer Architecture News 36(2):64–71, New York.
13. Hammoud M, Cho S, Melhem R (2009) ACM: an efficient approach for managing shared caches in chip multiprocessors. Proc. 4th int. conf. on high performance embedded architectures and compilers (HiPEAC-09). Paphos, Cyprus, pp 355–372.
14. Bardine A, Foglia P, Gabrielli G, Prete CA (2007) Analysis of static and dynamic energy consumption in NUCA caches: initial results. Proc. of the MEDEA 2007 workshop. Brasov, Romania, pp 105–112.
15. Bardine A, Comparetti M, Foglia P, Gabrielli G, Prete CA, Stenstrom P (2008) Leveraging data promotion for low power D-NUCA caches. Proc. of the DSD '08. 11th EUROMICRO conference on digital system design architectures, methods and tools. Parma, Italy, pp 307–316.
16. Bardine A, Comparetti M, Foglia P, Gabrielli G, Prete CA (2010) Way-adaptable D-Nuca caches. International Journals of High Performance Systems Architecture 2(3/4):215–228.
17. Bartolini S, Foglia P, Prete CA, Solinas M (2010) Feedback driven restructuring of multi-threaded applications for NUCA cache performance in CMPs. 22nd international symposium on computer architecture and high performance computing. Petropolis, Brazil.
18. Bardine A, Comparetti M, Foglia P, Gabrielli G, Prete CA (2009) Impact of on-chip network parameters on NUCA cache performance. IET Computers & Digital Techniques 3(5):501–512.
19. Bardine A, Comparetti M, Foglia P, Gabrielli G, Prete CA (2008) Performance sensitivity of NUCA caches to on-chip network parameters. 20th international symposium on computer architecture and high performance computing (IEEE SBAC-PAD 2008). October 29–November 1, Campo Grande, Brazil.