CMP CACHE ARCHITECTURES - A SURVEY

Shirshendu Das[1]

[1]*Department of Computer Science and Engineering, Indian Institute of Technology Guwahati*

Abstract

As the number of cores on Chip Multi-Processor (CMP) increases, the need for effective utilization (management) of the cache increases. Cache Management plays an important role in improving the performance and miss latency by reducing the number of misses. In most of the cases, CMP with shared Last Level Cache (LLC) is a winner over the private LLC. Non-Uniform Cache Access (NUCA) represent two emerging trends in computer architecture. In NUCA the LLC is divided into multiple banks which lead to different banks being accessed with different latencies. Hence the heavily used blocks can be mapped or migrated towards the closer bank of the requesting core. Though NUCA is the best architecture for single core systems, implementing NUCA in CMP has many challenges. Researchers proposed many innovative ideas to implement NUCA in CMP but still there exists lot more complexities. Thus CMP cache architecture is a widely open research area. In this paper we did a survey on different CMP cache architectures based on NUCA. We have only given a basic overview and there are lot more advanced innovations which are not been covered. The performance evaluation of CMP architecture is a challenging task and must have to do for proving the correctness of any proposed architecture. Therefore, we also discussed about how the performance of CMP cache architectures can be evaluated

Keywords: *Chip-Multiprocessor, NUCA, Last-Level-Cache, Formal Verification, Full-System simulator*

----------***----------

1. INTRODUCTION

It is expected that Chip Multiprocessors (CMPs) that contain multiple CPUs on the same die will be the main components for building future computer systems [1]. In fact, several CMP based architectures [2], [3], [4], [5] have already found their way into commercial market. In the long run, it is expected that the number of cores in CMPs will increase [6], [7]. Also the CMP's gradually accommodating large on-chip Last Level Caches (LLC).

CMP cache architectures are mainly of two types i) CMP with private LLC and ii) CMP with shared LLC. In both types of architecture each core has their own L1 caches for data/instructions. They differ in the physical placement of LLC. Considering two level cache hierarchies each core in the first type has its own private L2 cache while in second type each core shares a common large L2 cache. Both types have pros and cons. Private L2 caches are relatively small and physically placed very near to the core, hence the cache access time is very less. But it has the capacity problem; since the cache size is small it causes several capacity misses. Multiple copies of same data may present in separate L2 caches, hence an L2 level coherence is compulsory. On the other hand shared L2 is comparatively very large and only a single copy of each data can store in it; the entire requesting core will share the same data block. Another advantage of shared L2 is that the cache storage can be dynamically allocated to a core depending on its workload, which is not possible for private L2. Majority of researchers found shared LLC (L2 in this case) as the best choice for CMP cache architecture. But shared LLC also has a disadvantage: due to its large size, cache access time is several times longer than private LLC.

Researchers proposed many innovative ideas to minimize the cache access time of shared LLC. Initially most of the CMP cache structure is designed to have uniform cache access time regardless of the block being accessed. For such Uniform Cache Access (UCA) LLC, access time becomes a significant bottleneck as the cache become larger. An alternative solution is to divide the large cache into multiple banks such that each bank can be accessed with different access time. This kind of design is called Non Uniform Cache Access (NUCA) and is the most promising CMP cache design in recent years.

Since different banks can be accessed at different access time a core can access its closer bank much faster than the farther banks. Hence heavily requested data blocks can be migrated towards the closer banks to reduce the hit time. NUCA with migration was first proposed by Huh et al. called D-NUCA. In D-NUCA a data can be allocate to any bank in a set of banks. But it requires searching the entire set to get the data. Hence, searching time creates a problem in D-NUCA. Though a solution for this problem (partial tag storage) exists for single core processors there is no prominent solution for CMPs.

While D-NUCA is gradually being acknowledged as too complex to implement, Chishti et al. considered an alternative called NuRAPID that was higher-performing and potentially less complex in a single-core setting. The main idea in this design is to separate the tag and data arrays in the cache. For a single core processor the tag array will be stored in the nearest bank of the cache controller. The cache access begins with a tag look-up and the request is then directly sent to the NUCA

bank that has the data. Such a design eliminates the need to search for a block by progressively looking up multiple banks. A block is now allowed to reside in any row in any cache bank and the tag storage carries a pointer to the data blocks exact location.

Shared LLC can be divided into two parts: Centralized and Distributed. Centralized shared LLC means the LLC is placed in a contiguous area on the chip and the cores are placed at the two/four sides of the LLC. Both D-NUCA and NuRAPID are based on centralized shared LLC. While distributed shared LLC is almost like private LLC; each core has its own L1 and L2 cache (together called Tile) and all the cores (Tiles) are connected by some on-chip network. The difference of distributed shared LLC and private LLC is that in distributed shared LLC, each LLC (here L2) is shared by all the cores and only a single copy of same data exists in the entire LLC.

In this paper, we have done a survey on the last level cache (LLC) of different CMP architectures. The performance improvement of LLC in CMP has several issues, e.g., migration, searching and coherence maintenance etc. In case of single core architecture, such issues can be easily solved but not for CMPs. We discussed about several CMP based cache architectures proposed to solve the major CMP LLC issues. Next section gives an overview of different types of CMP cache architectures. Section 3 describes NUCA architecture in details. Here we have explained the different types of NUCA and the data management policy of each type of NUCA. Implementation of CMP based NUCA architecture is discussed in section 4. This section describes some notable contributions in the area of CMP-NUCA. In section 5, we have discussed about how to evaluate the performance of CMP cache architectures. Section 6 concludes the paper.

2. AN OVERVIEW OF CMP CACHE DESIGN

Most modern high-performance processors have multiple cache levels within a single chip. In a multi-core processor, each core typically has its own private L1 cache (data and instruction). Every processor must have to access L1 cache in almost every cycle, sharing L1 cache with multiple cores is not a good choice. Every miss in L1 cache is served by L2 cache. If data is available in L2 then it is a hit and it immediately sends a copy of the data to the requesting L1. Otherwise L2 has to bring the data from the lower level memory first. For most of the discussion in this paper, we will assume that the L2 is the LLC.

2.1 Shared Last Level Cache

Shared LLC means a single large cache shared by multiple cores on the chip. Each core in the CMP is connected to the shared LLC through an interconnect. The interconnect may be either a bus, switch based interconnect or a hybrid type of interconnect. We are not going to discuss about the on-chip interconnects in this paper. Interested reader can see [8], [9] for details. One example organization for CMP with shared LLC is shown in Figure 1(a). In shared LLC there is no duplication of cache blocks but the same cache block can reside in multiple L1 cache simultaneously. Coherence must be maintained among the L1s and the L2.

The main advantage of using a shared cache is the available storage space that can be dynamically allocated among multiple cores, leading to better utilization of the overall cache space. Also it requires maintaining only a single copy of shared data. The primary disadvantages of a shared cache is that the working sets of different cores may interfere with each other and impact each other's miss rates, possibly leading to poorer quality-of-service. This may impose overheads if the cores are mostly dealing with data that is not shared by multiple cores. Also, a core may experience many contention cycles when attempting to access a resource shared by multiple cores. However, we will show in this paper that both of these disadvantages can be easily alleviated.

2.2 Private Last Level Cache

A popular alternative to the single shared LLC is a collection of private LLC. Assuming a two-level hierarchy, a core is now associated with private L1 instruction and data caches and a private unified L2 cache (see Figure 1(b)). A miss in L1 triggers a look-up of the cores private L2 cache. Each private L2 cache is relatively small, allowing smaller access times on average for L2 hits. The private L2 cache can be accessed without navigating the coherence interface and without competition for a shared resource, leading to performance benefits for threads that primarily deal with non-shared data.

A primary disadvantage of private L2 caches is that a data block shared by multiple threads will be replicated in each threads private L2 cache. This replication of data blocks leads to a problem called capacity problem. Another disadvantage of a private L2 cache organization is the static allocation of L2 cache space among cores. By employing private L2 caches, the coherence interface is pushed down to a lower level of the cache hierarchy. But due to the large size of LLC, maintaining coherence in this level is a complex job.

2.3 Shared vs. Private LLC

Both Shared LLC and Private LLC have some advantages and some disadvantages. The advantages of one are the disadvantages of another. Hence, recently researchers are proposing hybrid alternatives for combining the advantages of both shared and private LLC [10], [11], [12], [13].

2.4 Inclusive Cache Behavior

For much of the discussions in this paper, we will assume inclusive cache hierarchies. However, many research evaluations and commercial processors employ non-inclusive hierarchies as well. If the L1-L2 hierarchy is inclusive, it

means that every block in L1 has a back-up copy in L2. The following policy ensures inclusion: when a block is evicted from L2, its copy in the L1 cache is also evicted. If a single L2 cache is shared by multiple L1 caches, the copies in all L1s are evicted. This is an operation very similar to L1 block invalidations in a cache coherence protocol.

2.5 CMP with Shared LLC

CMP having shared LLC can be categorized into two different ways:

- Centralized Shared LLC.
- Distributed Shared LLC.

In most of the shared caches, LLC is considered as centralized entity and is placed in a contiguous area into the chip. All the cores are placed on the two/four sides of the LLC. One example of such centralized shared L2 cache is shown in figure 2(a). Even though the cache is partitioned into multiple banks and the cache controller is distributed over all the banks we still call it as centralized because it occupies contiguous space into the chip.

This type of centralized architecture is good enough for CMP's with small number of cores but when there are many number of cores, this type of architectures degrade performance [8], [14]. Researchers have proposed an alternative with distributed shared LLC [8]. In this model even though the cache is logically shared it may be physically distributed over the chip, such that one bank of the L2 may be placed in close proximity to each core. The core, its L1 caches, and one bank of the LLC together constitute one Tile. A switch based mashed network is used (most of the cases) to connect all the Tiles. Other types of networks can also be used as an interconnect (e.g., bus or hybrid). One example of such a physical layout is shown in Figure 2(b). The primary disadvantage of this organization is the higher cost in moving data/requests between L2 cache banks and the next level of the memory hierarchy.

3. NON-UNIFORM CACHE ACCESS (NUCA)

In past most of the CMPs cache structure is designed to have uniform cache access time regardless of the block being accessed. Such Uniform Cache Access (UCA) architectures certainly simplify the cache access policies. However, as cache become larger and also partitioned into multiple banks, maintaining uniform accesses time for the entire cache is not a good choice. The banks nearer to a core can actually be accessed much faster than the furthest bank. Also wire delay plays an increasingly significant role in cache design [11].

Increasing wire delay makes it difficult to provide uniform access latencies to all L2 cache banks. One alternative is NUCA designs [10], which allow nearer cache banks to have lower access latencies than further banks. NUCA architecture was initially proposed for uniprocessor systems. They consider a large L2 cache that has a single cache controller feeding one processor core (see Figure 3). The author proposed three types of NUCA architecture a) SNUCA-1 (Static NUCA-1) b) SNUCA-2 (Static NUCA-2) and c) D-NUCA (Dynamic NUCA). In all the three types, large L2 cache is divided into multiple banks and all the banks are connected between them and also with the cache controller. The difference between the types is based on the interconnect topology, number of banks and the data management policy. A detail discussion on each types of NUCA is given in next section (Section 3.1).

Data management policies of NUCA must follow the following three issues:

Mapping: The possible locations for a data block. The simplest policy is to statically map each block to a particular bank. An alternative mapping policy distributes ways and sets across banks such that a block can reside in any bank from a set of banks. A search mechanism is required to locate a block that may be in one of a set of banks.

Search: the mechanisms required to locate a data block. The search of a block can happen in an

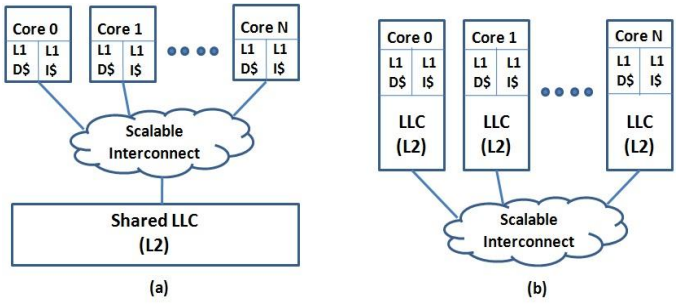

Fig 1: Example of Chip Multi Processor (a) with L2 as shared LLC (b) with L2 as private LLC.

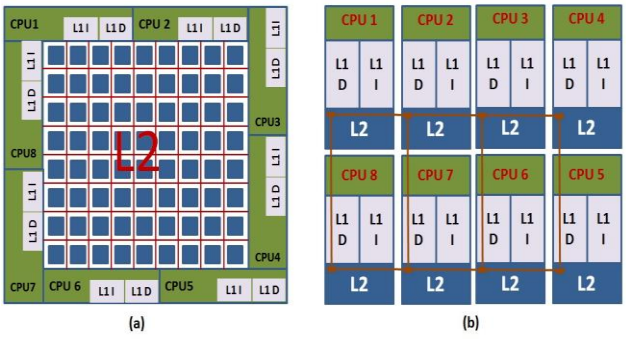

Fig 2: Example of Chip Multi Processor (a) with Centralized Shared LLC (b) with distributed shared LLC

Incremental manner, i.e., one bank after another bank. Alternatively, a multicast search operation can be carried out where the request is sent to all banks simultaneously. The second approach will yield higher performance but also higher

power. They also propose a Smart Search mechanism where a partial tag (six bits) for each block is stored at the cache controller. A look-up of this partial tag structure helps identify a small subset of banks that likely have the requested data and only those banks must now be searched.

Movement: the mechanisms required to change a block's location. The authors allowed frequently accessed blocks to be migrated from the farther banks to the closer banks.

3.1 Different of NUCA-Types

In this section we have given a brief description of all three types of NUCA implementation.

SNUCA-1: An example of such type of NUCA is given in Figure 3(a). Here each bank has a dedicated two-way transmission channel connected to the cache controller. The mapping of data into banks is predetermined, based on the block index, and thus can reside in only one bank of the cache.

SNUCA-2: Private per-bank channels used in SNUCA-1 heavily restricts the number of banks that can be implemented, since the per-bank channel wires adds significant area overhead to the cache if the number of banks is large. To overcome that limitation, the author proposed a static NUCA design that uses a two-dimensional switched network. This organization, called SNUCA-2, is shown in Figure 3(b).

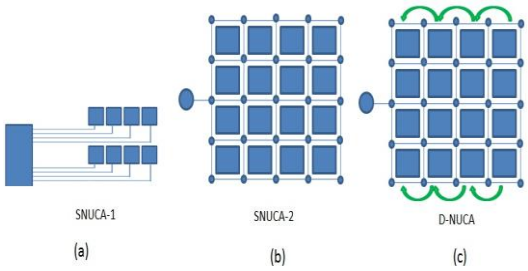

Fig 3: An example showing different types of NUCA.

D-NUCA: Even with an aggressive multi-banked design, performance may still be improved by exploiting the fact that accessing closer banks is faster than accessing farther banks. By permitting data to be mapped to many banks within the cache, and to migrate among them, a cache can be automatically managed in such a way that most requests are serviced by the fastest banks. Using the switched network, data can be gradually promoted to faster banks as they are frequently used. This promotion is enabled by spreading sets across multiple banks, where each bank forms one way of a set. Thus, cache lines in closer ways can be accessed faster than lines in farther ways. To fulfill this dynamic behavior the author proposed D-NUCA or Dynamic-NUCA. An example of D-NUCA is shown in Figure 3(c).

4. IMPLEMENTING NUCA FOR CMPs

CMPs are accommodating many mega-bytes of data in their LLC. As we already discussed, LLC can be shared by many cores and can be either physically distributed or contiguous on the chip. We next discuss about the several architectural innovations that attempt to cleverly place data blocks within LLC to optimize metrics such as miss rates, access times, quality-of-service and throughput. There are many complexity to implement NUCA for CMP LLC. In this section we will discuss the different complexities of implementing NUCA for CMPs and also several innovations to solve all these complexities.

4.1 Complexities with D-NUCA for CMP

D-NUCA outperforms the other two types of NUCA in case of single core processor [10]. There are already some notable contributions to implement D-NUCA for CMP's. However, all of this work had to suffer from the overheads of a fairly complex search mechanism, a problem that to date does not have a compelling solution.

Beckmann and Wood:

Beckmann and Wood [11] proposed the first detailed multi-core NUCA architecture. They assume a layout (see Figure 4(a)) where the shared NUCA cache resides in the middle of the chip and is surrounded by eight cores. A major contribution of this work is the classification of banks into regions and architectural policies to allow a block to migrate to a region that minimizes overall access times. There are 16 regions, out of them 8 are called local region (one for each core), 4 are called center region and remaining 4 are called inter regions. Initial placement is somewhat random (based on the block tag bits). From here, a block is allowed to gradually migrate to different regions based on the cores that access it. The basic rule for migration is as follows:

other-local => other-inter => other-center => my-center => my-inter => my-local

The authors also find out that in most of the workloads the amount of shared data is less but the frequency of accessing those datum is very high. Hence due to the above mentioned migration rule, eventually most of the shared data will saturate into the center regions, which are far away from every cores. Normal RC based wires used as D-NUCA interconnect are slow [15] and cannot access those center regions rapidly. Hence to reduce the access latency the authors used high speed transmission lines [15] to connect directly each core to the center regions (see Figure 4(b)). Note that transmission lines can communicate data at the near speed of light. The details regarding transmission lines are discussed in [9].

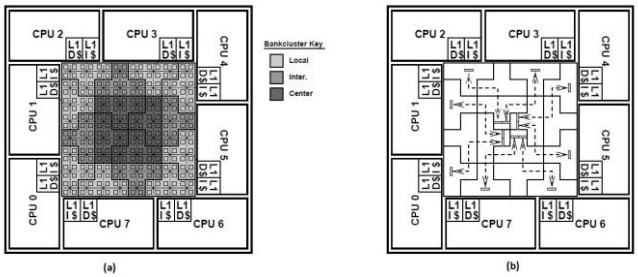

Fig 4: NUCA for CMP, proposed in [11] (image also taken from [11]).

Figure 1: An example showing different types of NUCA.
The most significant problem with the above architecture is the difficulty in locating a block. In a D-NUCA that distributes the ways across the banks, it cannot be statically determined the exactly location of a block. That means for every request it needs to search a set of banks to get the block (assume if it is a hit). Beckmann and Wood employ a multicast based search mechanism.

Huh et al.

In a paper that appeared shortly after [11], Huh et al. [16] validate many of the above observations. They used a 16-core CMP with a large banked NUCA cache in the middle of the chip. As we have mentioned in section 2.5 that such type of CMPs are called CMP with centralized shared LLC. The authors confirm that an SNUCA policy leads to longer access times on average. There is a minor performance improvement in case of D-NUCA based policy, where the ways are distributed across the banks and blocks are allowed to migrate between banks. They consider both 1-dimensional and 2-dimensional block movement where 1-D movement prevents a block from moving out of its designated column. They found that the main reason for less performance improvement in case of D-NUCA (for CMPs) is the complexity of searching the blocks. To avoid having to access numerous banks, they implement a distributed set of replicated partial tags. At the top/bottom of every column of banks, partial tags for every block in that column are stored. A look-up into this storage reveals if one or more banks in that column can possibly have the requested block. These additional look-ups of partial tags and banks (nearly 50% more than the S-NUCA case) negate half the benefit afforded by D-NUCA data proximity. They also result in increased power and bank access rates. The various tag stores will have to be updated with on-chip messages every time a block is replaced / migrated.

4.2 Some Other Implementations of NUCA

Re-NUCA: Re-NUCA [17] is a novel cache architecture that allows limited replication for shared blocks accessed by processors placed at opposite chip sides. In particular, their solution lets at most two independent copies of the same block to be stored in the same shared cache, each of them migrating towards the closest cache side, named target side.

HK-NUCA: A data search algorithm for D-NUCA, which is called HK-NUCA has been proposed in [18]. HK-NUCA means Home Knows where to find data within the NUCA cache. They considered that each block must have a home bank in the NUCA, though it can be reside or migrate to any other bank the home bank will always maintain a pointer called (HK-PTR) to for the data.

4.3 NuRAPID and CMP-NuRAPID

Due to its searching issues D-NUCA is gradually being acknowledged as "too complex to implement". To solve this problem, Christi et al. [12] proposed an alternative architecture called NuRAPID. It is less complicated and also performing better than D-NUCA for single core systems. The authors initially proposed it only for single core multi-banked cache architectures. Later they extended the concept and proposed NuRAPID for multiple cores [13] (called CMP-NuRAPID).

Two key contributions were made in the first paper:

- Instead of placing both tag and data blocks together, they propose to implement the entire tag array as a centralized structure near the processing core/cache controller. Every cache access starts with a tag look-up and the request is then directly sent to the NUCA bank that has the data. Such a design eliminates the need to search for a block by progressively looking up multiple banks. However, the movement of every block must need to inform the centralized cache controller so that the tag can be updated.
- Separating tag and data block placement. In NuRAPID a block can be stored in any row in any cache bank and the tag storage (organized in a conventional manner) carries a pointer to the data block's exact location. Similar to D-NUCA, the requesting blocks are gradually migrated towards the cache controller. Swapping places with any block that may not have been recently touched.

Since a swap can now happen between any two blocks, the block movement policy allows the closest banks to accommodate the "globally hottest" (most frequently and recently touched) blocks, and not just the hottest blocks in each set. Note that conventional D-NUCA would restrict each set to only place a small subset of ways close to the CPU, whereas NuRAPID policy allows all the ways of a hot set to be placed in a nearby bank. Such flexibility can allow NuRAPID to out-perform D-NUCA, especially if applications non-uniformly stress their sets. It can also reduce inter-bank traffic, especially if banks are sufficiently large. The overhead in providing such flexibility is that data blocks need to store reverse pointers that identify their entry in the tag array, so that the corresponding tag can be updated when a swap happens.

CMP-NuRAPID: In a follow-on paper [13], Chishti et al. extend their scheme to handle multiple cores. Just as in the NuRAPID [12] design, the CMP-NuRAPID design also decouples data and tag arrays. The data array is a shared resource; any core can place its data in any row of the data array; the data array is organized as multiple banks with non-uniform access times. Each core maintains a private tag array, with entries capable of pointing to any row of the shared data array. Keeping the tag arrays coherent is tricky: the authors assume that the tag arrays are kept coherent upon misses/movements/replacements by broadcasting changes to all tag arrays. The CMP-NuRAPID design is therefore an interesting hybrid between private and shared L2 caches. It has much of the performance potential of a shared cache, plus it allows selective replication of read-only blocks.

4.4 CMP with Distributed Shared Cache

All the above NUCA architectures are based on centralized shared cache. Since centralized cache has scalability issues, researchers also proposed many NUCA architectures for distributed shared cache. In this paper we call them as Tile Based Architecture (TLA).

Hardavellas et al. [19] put forth a novel Tile based NUCA architecture that relies on OS management of pages in a large shared L2 cache and does not require complex search mechanism. Each core is also allowed its own indexing functions, enabling each core to have a different view of the shared L2 cache this allows a private page to migrate between banks without requiring page copy in DRAM or complex hardware structures. In addition to efficiently handling both shared and private pages, it facilitates replication at various granularities.

In [20] the authors have combined on-chip networks and 3D architectures for designing large L2 cache memories for Tile based CMP. Specifically, this paper has proposed a hybrid bus/NoC (Network on Chip) fabric to efficiently exploit the fast vertical interconnects in 3D circuits, discussed processor placement and L2 data management issues, and presented an extensive experimental evaluation of the proposed architecture as well as its comparison to 2D L2 cache designs.

Some innovations also exists for implementing hybrid architectures by combining private LLC concepts and the distributed shared LLC concepts [21], [22], [23], [24], [25],[26].

4.5 Summary of this Section

The architectures discussed in this section are the basic NUCA based CMP architectures. There are lot more advanced works have already been proposed and it is not possible to cover all of them in this paper. Few of these works are based on 3D architecture, wiring technologies, data prefetching / replication / spilling, NoC (Network on Chip) routing technologies and power management etc. Also a large number of researches are still working on multiple banked CMP cache architectures.

5. SIMULATION AND VERIFICATION

A major issue to work on CMP architecture is to implement our proposed architectures. To propose or modify any architecture (may or may not be CMP cache architecture), we have to prove that our architecture is working correctly and also performing better than other existing architectures of same category. It is not always possible to do hardware implementation of CMP cache architectures because it takes lots of time and money. Therefore, hardware implementation is not preferred by most of the research institutes and organizations. There are two alternatives that the researchers can do to prove the correctness of their proposed architecture.

- Formal Verification.
- Simulation.

5.1 Formal Verification

In the context of hardware and software systems, formal verification is the act of proving or disproving the correctness of intended algorithms underlying a system with respect to a certain formal specification or property, using formal methods of mathematics. Formal verification can be helpful in proving the correctness of systems such as: combinational circuits, digital circuits with internal memory, and software expressed as source code. The detail explanation is outside the scope of this paper. In short we can say that it is a process to formally prove the correctness of our model. The concept of formal verification can be used for proving the correctness of any architectural model (e.g., NoC, SoC and CMP). In this process, we have to model our architecture in a particular process algebraic language like CCS [27], CSP [28], PROMELA [29] etc. and prove the correctness of the model by verifying all the properties of the architecture. The properties are normally written in temporal logic (e.g., LTL, CTL) [30]. Formal verification has been used to model many SoC (System on Chip) and NoC based architectures [31], [32], [33]. In [34] the author used MurΦ [35], a formal verification language, to prove the correctness of their tree based cache coherence protocol for CMPs.

5.2 Simulation

The most popular way to implement any cache architectures is to simulate the architecture with a simulator. For simulating any CMP cache we need two types of simulator (a) Cache Modeler (b) Full System simulator.

Cache Modeler: Cache modeler means a software tool that can model optimized cache architecture. A most widely used cache modeler is CACTI [36]. CACTI first invented in 1994 [37] and has been cited by more than thousand papers. CACTI takes input parameters like cache block size, cache size, associativity, number of cores and number of banks (for

NUCA) and calculate an optimized cache model. Here optimized model means cache with optimized delay, area and power. Each optimization parameters are further subdivided into many parameters (e.g., number of sub-arrays, wire delay, tag array overhead, data array overhead etc.). The complete discussion of CACTI is not possible here. Interested person can read the technical report of CACTI 6.0 [38]. CACTI tool is freely available in [39].

Full system simulator: A full-system simulator is a Computer architecture simulator that simulates an electronic system at such a level of detail that complete software stacks from real systems can run on the simulator without any modification. A full system simulator effectively provides virtual hardware that is independent of the nature of the host computer. The full-system model typically has to include processor cores, peripheral devices, memories, interconnection buses and network connections. CMP cache architecture must have to use full-system simulator for complete performance evaluation. Most of the simulators are written in C/C++ and sometimes need to change some functionality to meet our requirements.

The two most widely used full-system simulators are (a) Simics [40] and (b) Gem5 [41]. Simics is a function driven commercial simulator and initially has no support for time driven simulation. But the problem has been solved by GEMS [42] which is a time driven simulator works on top of Simics. GEMS can be used for simulating any memory architecture (cache, main memory). It can also simulate network and power consumption of a CMP architecture using two additional tools called Garnet [43] and Orion [44] respectively. GEMS is a freely distributable tool but it is dependent on Simics and cannot work independently. The combination of Simics and GEMS is the most popular full-system simulator from last one decade. The gem5 simulation infrastructure is the merger of the best aspects of the M5 [45] and GEMS [42] simulators. Gem5 is freely available [46] and independent from Simics.

Why two types of simulators: A common question may arise to any reader as why we need two types of simulators (Cache Modeler and Full-system) and what is the relation between them. In short, the answer would be; cache modeler calculates the best possible model for a cache architecture with optimized delay, area and power parameters. Full-system simulator can use those optimal parameters (especially cache access time, wire delay etc.) to make a more realistic and accurate system.

6. CONCLUSIONS

NUCA architecture is a clear winner over the UCA architectures. But implementing NUCA for CMPs has several issues. Researchers found that the migration based NUCA models of CMP are not as efficient as compared to the single core NUCA. The main problem is searching of a block within the cache banks. Many ideas have already been proposed to mitigate all these problems but till it is an open research area to find out an optimal NUCA implementation for CMPs. In this paper we have done a survey on different CMP cache architectures based on NUCA. We only gave a basic overview and there are lot more advanced innovations which are not covered. The performance evaluation of CMP architecture is a challenging task and must have to do for proving the correctness of any proposed architecture. Therefore, we also discussed about how the performance of CMP cache architectures can be evaluated.

REFERENCES

[1] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang, "The case for a single-chip multiprocessor," SIGPLAN Not., vol. 31, pp. 2–11, September 1996.

[2] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," IBM J. Res. Dev., vol. 49, pp. 589–604, July 2005.

[3] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded sparc processor," IEEE Micro, vol. 25, pp. 21–29, March 2005.

[4] Amd athlon 64 x2 dual-core processor for desktop. [Online]. Available: http://www.amd.com/usen/Processors/ProductInformation/0,,30 118 9485 13041,00.html

[5] (2008, April) Intel. Quad-core intel xeon processor 5400 series. [Online]. Available: http://download.intel.com/design/xeon/datashts/318589.pdf

[6] Intel teraflops machine. [Online]. Available: http://www.intel.com/idf/

[7] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a tera?ops processor," IEEE Micro, vol. 27, pp. 51–61, September 2007.

[8] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, Multi-Core Cache Hierarchies. Morgan Claypool Publishers, 2011.

[9] B. M. Beckmann and D. A. Wood, "TLC: Transmission line caches," in Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 36, 2003, pp. 43–.

[10] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches," SIGOPS Oper. Syst. Rev., vol. 36, pp. 211–222, October 2002.

[11] B. M. Beckmann and D. A. Wood, "Managing wire delay in large chip-multiprocessor caches," in Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 37. IEEE Computer Society, 2004, pp. 319–330.

[12] Z. Chishti, M. D. Powell, and T. N. Vijaykumar, "Distance associativity for high-performance energy-

efficient non-uniform cache architectures," in Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 36. IEEE Computer Society, 2003, pp. 55–.

[13] ——, "Optimizing replication, communication, and capacity allocation in cmps," SIGARCH Comput. Archit. News, vol. 33, pp. 357–368, May 2005.

[14] L. Hao, L. Tao, L. Guanghui, and X. Lunguo, "Private cache partitioning: A method to reduce the off-chip missrate of concurrently executing applications in chip-multiprocessors," in Computer Research and Development (ICCRD), 2011 3rd International Conference on, vol. 4, march 2011, pp. 254 –259.

[15] R. Chang, N. Talwalkar, C. Yue, and S. Wong, "Near speed-of-light signaling over on-chip electrical interconnects," Solid-State Circuits, IEEE Journal of, vol. 38, no. 5, pp. 834 – 838, may 2003.

[16] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A nuca substrate for flexible cmp cache sharing," in Proceedings of the 19th annual international conference on Supercomputing, ser. ICS '05, 2005, pp. 31–40.

[17] P. Foglia, C. A. Prete, M. Solinas, and G. Monni, "Re-nuca: Boosting cmp performance through block replication," in Proceedings of the 2010 13th Euromicro Conference on Digital System Design: Architectures, Methods and Tools, ser. DSD '10, 2010, pp. 199–206.

[18] J. Lira, C. Molina, and A. Gonzalez, "HK-nuca: Boosting data searches in dynamic non-uniform cache architectures for chip multiprocessors," in Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International, may 2011, pp. 419 –430.

[19] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Reactive nuca: near-optimal block placement and replication in distributed caches," SIGARCH Comput. Archit. News, vol. 37, pp. 184–195, June 2009.

[20] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, "Design and management of 3d chip multiprocessors using network-in-memory," SIGARCH Comput. Archit. News, vol. 34, pp. 130–141, May 2006.

[21] B. M. Beckmann, M. R. Marty, and D. A. Wood, "Asr: Adaptive selective replication for cmp caches," in Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 39. IEEE Computer Society, 2006, pp. 443–454.

[22] J. Chang and G. S. Sohi, "Cooperative caching for chip multiprocessors," in Proceedings of the 33rd annual international symposium on Computer Architecture, ser. ISCA '06, 2006, pp. 264–276.

[23] M. K. Qureshi, "Adaptive spill-receive for robust high-performance caching in cmps," in Proceedings of HPCA, 2009.

[24] H. Lee, S. Cho, and B. Childers, "Cloudcache: Expanding and shrinking private caches," in Proceedings of HPCA, 2011.

[25] H. K. Kapoor, L. Chatterjee, and R. Yarlagadda, "Clustered caching for improving performance and energy requirements in noc based multiprocessors," in Proceedings of the International Conference on Computer Design (CDES), 2011.

[26] R. Yarlagadda, S. R. Kuppannagari, and H. K. Kapoor, "Performance improvement by n-chance clustered caching in noc based chip multi-processors," in Proceedings of the International Conference on Computer Design (CDES), 2011.

[27] R. Milner, Communication and concurrency. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1989.

[28] C. A. R. Hoare, "Communicating sequential processes," Commun. ACM, vol. 21, no. 8, pp. 666–677, 1978.

[29] Promela manual. [Online]. Available: http://spinroot.com/spin/Man/promela.html

[30] M. Huth and M. Ryan, Logic in Computer Science modelling and reasoning about systems. Cambridge University Press, 2004.

[31] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Theory of Latency-Insensitive Design," IEEE Transactions on Computer Aided Design of Intregrated Circuits and Systems, vol. 20, no. 9, pp. 1059–1076, Sep 2001.

[32] H. K. Kapoor, "A Process Algebra View of Latency Insensitive System," IEEE Transactions on Computers, vol. 58, no. 7, pp. 931–944, July 2009.

[33] S. Das, P. S. Duggirala, and H. K. Kapoor, "A formal framework for interfacing mixed-timing systems," Integration, the VLSI Journal, no. 0, pp. –, 2012. [Online]. Available:

[34] http://www.sciencedirect.com/science/article/pii/S0167926012000363

[35] H. Kapoor, P. Kanakala, M. Verma, and S. Das, "Design and formal verification of a hierarchical cache coherence protocol for noc based multiprocessors," The Journal of Supercomputing, pp. 1–26, 2013. [Online]. Available: http://dx.doi.org/10.1007/s11227-012-0865-8

[36] U. Stern and D. L. Dill, "Automatic verification of the SCI cache coherence protocol," Correct Hardware Design and Verification Methods, LNCS, vol. 987, pp. 21–34, 1995.

[37] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing nuca organizations and wiring alternatives for large caches with cacti 6.0," in Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture, ser. MICRO 40, 2007, pp. 3–14.

[38] S. Wilton and N. Jouppi, "Cacti: an enhanced cache access and cycle time model," Solid-State Circuits, IEEE Journal of, vol. 31, no. 5, pp. 677 –688, may 1996.

[39] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi, "CACTI 6.0: A Tool to Model Large Caches," HP Laboratories, Tech. Rep., April 2009. [Online].

Available: http://www.hpl.hp.com/techreports/2009/HPL-2009-85.html

[40] Cacti. [Online]. Available: http://www.hpl.hp.com/research/cacti/

[41] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," Computer, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[42] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," SIGARCH Comput. Archit. News, vol. 39, no. 2, pp. 1–7, Aug. 2011.

[43] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," SIGARCH Comput. Archit. News, vol. 33, no. 4, pp.92–99, Nov. 2005.

[44] N. Agarwal, T. Krishna, L.-S. Peh, and N. Jha, "Garnet: A detailed on-chip network model inside a full-system simulator," in Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on, april 2009, pp. 33 –42.

[45] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in Microarchitecture, 2002. (MICRO-35). Proceedings. 35th Annual IEEE/ACM International Symposium on, 2002, pp. 294 – 305.

[46] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt, "The m5 simulator: Modeling networked systems," Micro, IEEE, vol. 26, no. 4, pp. 52 –60, july-aug. 2006.

[47] The gem5 simulator system. [Online]. Available: http://www.m5sim.org