Hierarchical Cluster based NoC design using Wireless Interconnects for Coherence Support

Tanya Shreedhar*, Sujay Deb†
Indraprastha Institute of Information Technology Delhi
*tanyas@iiitd.ac.in, †sdeb@iiitd.ac.in

***Abstract*—Efficient and low latency solution to cache coherence problem in shared memory multicore systems on Network-on-Chip is a crucial issue for improving system performance and scalability. The existing works utilize planar metal/dielectric interconnects to solve this problem. However, as the number of cores on chip increase, such solutions face high latency due to increase in long-range multi-hop communication between distant cores. In this paper, we propose a novel multicore hierarchical cluster based architecture that utilizes a hybrid NoC having both wired and wireless interconnects. The wireless interconnects on chip replaces the long-range multi-hop communication path to single-hop low latency connections. We also propose a utilization based clustering of cores. Finally, we present a novel cache coherence algorithm for the designed architecture. To the best of our knowledge, our work is the first to utilize wireless interconnects to solve the cache coherence problem in multicore system on NoC.**

We evaluate our system through analytical as well as system level simulations on GEM5 running SPLASH2 and PARSEC benchmarks. Through our evaluations, we find that our proposed architecture achieves an average of 94.4% reduction in overall memory overhead, 30% reduction in average L2 access latency and 32.1% reduction in L2 miss rate compared to conventional non-clustered directory architecture. Our solution also exhibits a notable decrease in overall execution time, memory requests and miss latency for all benchmarks.

***Keywords*-Cache Coherence, Chip Multiprocessor (CMP), GEM5, Multicores, Network-on-Chip (NoC), PARSEC, SPLASH2, Wireless Interconnects (WI).**

I. Introduction

Continuous increase in processor frequency has reached its limit in terms of power consumption, which has steered the path towards multicore design strategy. Recent years have witnessed the emergence of Chip Multiprocessors (CMPs) that utilize the increasing number of transistors on a chip. Multicore processors have the advantage of running on slower frequencies and yet performing much better than single-core processors running at a higher clock rates [1]. They also offer several other benefits such as reduced cost, increased reliability, increased throughput and much lower power consumption. In multicore scenario, the cohesion between inter-core communication and computation of a workload is quite complex. Network on Chip (NoC) helps to decouple communication from computation. NoC provides a low latency, high bandwidth and scalable network communication for large multi-core systems. As computation time of a core is considerably lower than shared memory read/write time, the entire execution is slowed down by memory I/O. On-Chip local cache memories are usually deployed to overcome this issue.

The overall memory structure comprises of an associated L1 cache with each core, a shared L2 cache, and a globally shared main memory. Cache coherence problem occurs when multiple cores try to update the same data simultaneously in their local L1 cache which in turn results in an inconsistent global value. A cache coherent system achieves a consistent view of memory by following a valid order of reads and writes to the memory using several protocols. Such protocols utilize two basic operations: write invalidate and write update [2]. Write invalidate ensures invalidation of copies of the shared data in every cache before the write operation. Write update maintains coherence by sending update information for all modifications to sharers. Write invalidate increases cache to cache miss whereas write update produces high traffic especially in cases where updated block is not used by the sharers until the modifying processor is finished with the updates. Hence, there can be no single cache coherence protocol that works well for varied workload applications, cache configurations, and interconnection parameters. Various hardware and software based cache coherence protocols have been proposed each having its trade-offs between cache miss latency, overheads, traffic generation and control [3] [4].

Traditionally, wired metal interconnects are used between tiles to enable them to communicate. However, they suffer from high latency and power consumption in case of multi-hop communication between distant cores. Thus, there is a need to look beyond the metal/dielectric based planar architectures and move towards new and emerging interconnects [5]. One such possibility is wireless NoC architecture that has emerged as energy efficient and high bandwidth communication backbone for multicore platform [6].

In this paper, we propose using these wireless links to solve the cache coherence problem. We propose a new hierarchical cluster based cache coherence scheme and replace multi-hop wired communication paths in a NoC

with long-range wireless links. The scheme is efficient and has the capability to support large traffic on all topologies. We extend the existing directory architecture to achieve coherence.

We present following contributions in this paper:

i. We define clustering constraints to group the cores into hierarchical clusters based on their utilization.
ii. Long-range, single-hop wireless links are used to ensure low latency communication between distant cores.
iii. We propose a novel cache coherence algorithm that ensures faster operations by using wired interconnects for intra-cluster communication and wireless interconnects for long-range inter-cluster communication.

The rest of the paper is organized as follows: We discuss the related work in Section II. Section III discusses the proposed architecture in detail. In Section IV, we evaluate our proposed system through analytical as well as system level simulations. Section V concludes the paper.

II. Related Work

Many efficient protocols have been proposed and implemented to address cache coherence in multi-core architectures. When system size is very small, bus-based snoopy protocol [7] is employed. As the number of cores on chip increases, snoopy protocol leads to high traffic and saturation. Directory coherence protocol [8], which uses a global directory to keep track of coherence performs well for large systems. However, it suffers from additional memory overheads.

Several works have been proposed to reduce the overall memory overhead of a generic directory protocol and improve its performance. Acacio et al. in [9] proposed a two-level directory structure with a fully mapped primary directory and compressed secondary directory. In [10], they further improved their methodology by adding a compressed directory within the main memory. Many authors have also designed a scalable directory coherence scheme for multicore architectures with NoC. In [11], the authors discuss a method to partition memory into local and shared segments. However, the proposed method fails to provide caching of shared data which limits its performance. Eisley et al. [12] discussed coherence protocols and directories embedded in NoC routers, which increased the complexity of the directory structure. Girao et al. [13] proposed a traditional full mapped directory but is not scalable for large systems. Zhang et al. [14] proposed a hierarchical cluster based coherence scheme. They reduced the overall memory overhead by assigning a head node in each cluster to form a two-level tree structure. Li et al. [15] introduced small cache that holds recently accessed cache lines along with L2 cache slice as fast directory to provide low hit latency. The solution suffers from increased memory overhead due to inclusion of new cache in the architecture.

In this paper, we propose to solve the cache coherence problem and also reduce the memory and latency overhead, L2 miss rate and L2 access latency. We achieve this by using a wireless NoC architecture and modifying the directory structure. Many WiNoC architectures have been proposed by several authors to improve the latency and energy performance of NoCs [6] [16] [17]. However, none of these works have tried to solve the cache coherence problem using wireless NoC architecture. We use this infrastructure and divide the system into multiple clusters each having a WI enabled core. To the best of our knowledge, our work is the first to address coherence issue using WiNOC architecture.

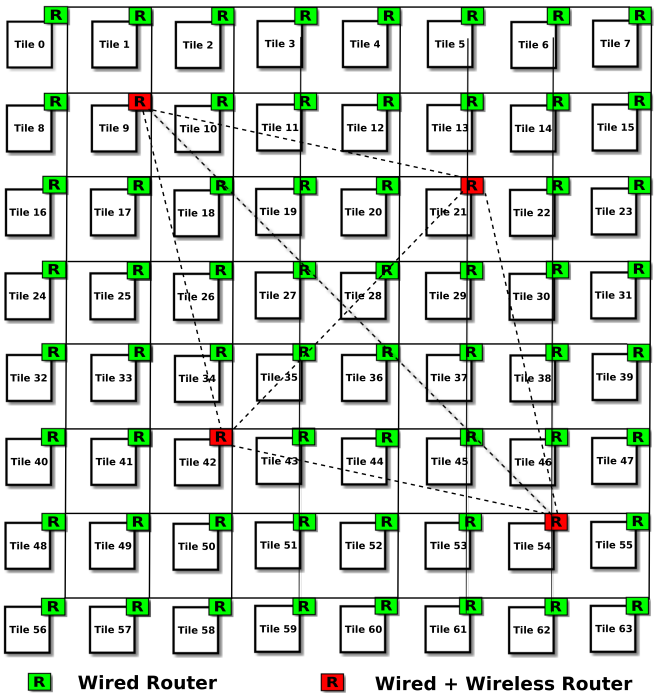

Figure 1: The proposed architecture of a 64 core system with wireless interconnects

III. Proposed Architecture

In this section, we discuss the design of system architecture and cluster formation constraints. We also provide a walkthrough example for achieving cache coherence using proposed scheme.

A. Topology of our architecture

Figure 1 shows our proposed architecture based on a 64 core system using 8x8 mesh topology. The cores are divided into 4 clusters having 16 cores each, and the NoC is made up of hierarchical wireless over wired interconnects. A single tile from each cluster is embedded with a wireless interconnect and is called the HEAD of that cluster. The HEAD is chosen such that it is at a minimum average distance from all cores of that cluster. A wired tile on the chip consists of a processor core, L1 cache and network

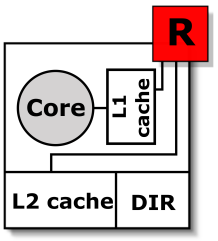

Figure 2: Architecture of a wireless tile

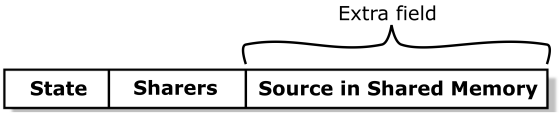

Figure 3: Extended Directory Structure

interface (NI). As shown in figure 2 the HEAD has a wireless transceiver, L2 cache and an extended directory in addition to the wired tile, which is responsible for delivery of coherence information to all the clusters. These wireless tiles are also equipped with additional transmitting and receiving buffers and are inter-connected through traditional wormhole routers. The HEADs communicate in a single-hop, broadcast fashion that significantly reduces the overall broadcast time when compared to traditional wired architecture.

The transmission power of wireless interconnects have been optimized to cover the entire chip area. To avoid interference in wireless channel, we employ fast token ring algorithm [18] in which a token is circulated amongst all the HEADs on the chip and only the HEAD in possession of the token accesses the wireless channel for transmission.

B. Cluster Formation

We cluster the cores according to their utilization as highly-utilizing cores have a higher probability of using and updating data in cache. This results in localization of data within a cluster.

Let a vector U of length N, where N is the number of cores, denote the per-core utilization. Similarly, let matrix P signify whether a core belongs to a particular cluster or not. As our architecture is divided into 4 clusters, P is of the size $N \times 4$. P contains either 0 or 1 as entries, where 0 signifies the absence of a core in a particular cluster and vice versa. We also consider a vector W of length N, which denotes whether a core is wireless-enabled or not. Similar to P, values in W indicate whether the core is wireless interconnect enabled or not. Since values of U vary for different benchmarks, the cluster creation is benchmark specific. The proposed clustering methodology follows several constraints.

I. A core can be part of only single cluster.

Algorithm 1 Proposed Cache Coherence Algorithm

Input: *Inv*: Invalidation Signal
ACK: Acknowledgement
procedure CACHECOHERENCE
  Wait for *Inv* from *RequestingNode*
  if *IncomingInv* $\xleftarrow{Wireless}$ *Inv* **then**
    if *IncomingInv(MemoryAddress)* $\xrightarrow{Directory}$ *PresenceBit*==1 **then**
      Send *IncomingInv* to sharers in cluster
      Wait for *ACK* from sharers
    end if
    Broadcast *ACK* via wireless
  else if *IncomingInv* $\xleftarrow{Wired}$ *Inv* **then**
    if *IncomingInv(MemoryAddress)* $\xrightarrow{Directory}$ *PresenceBit*==1 **then**
      Send *IncomingInv* to sharers in cluster
      Broadcast *IncomingInv* via wireless
      Wait for *ACK* from sharers
      Wait for *ACK* from *HEADs*
    end if
    Send *ACK* to *RequestingNode*
  end if
end procedure

$$\forall i : \sum_{j=0}^{3} P_{ij} == 1$$

II. Each cluster will have exactly $N/4$ cores

$$\forall j : \sum_{i=0}^{N-1} P_{ij} == N/4$$

III. Each cluster should contain only one wireless-interconnect enabled core

$$\forall j : \sum_{i=0}^{N-1} (P_{ij} \cap W_i) == 1$$

IV. P and W can only contain 0/1 values

$$P_{ij} \in \{0, 1\}$$
$$W_i \in \{0, 1\}$$

C. Coherence scheme in desired architecture

We address the cache coherence problem in our proposed architecture at two levels, intra-cluster and inter-cluster. The intra-cluster cache coherence is maintained within a cluster using the directory coherence scheme. The shared L2 cache has an associated directory that keeps the information about the states of member caches and denotes whether a copy of particular data in any member cache is valid or not.

The inter-cluster cache coherence is maintained using the wireless interconnects connected to the HEAD node of each cluster. In our proposed architecture, we ensure the consistency of shared memory address in each cluster by extending the existing directory structure as shown in figure 3 . The extended directory has an extra field which stores the source (shared memory) address of a data

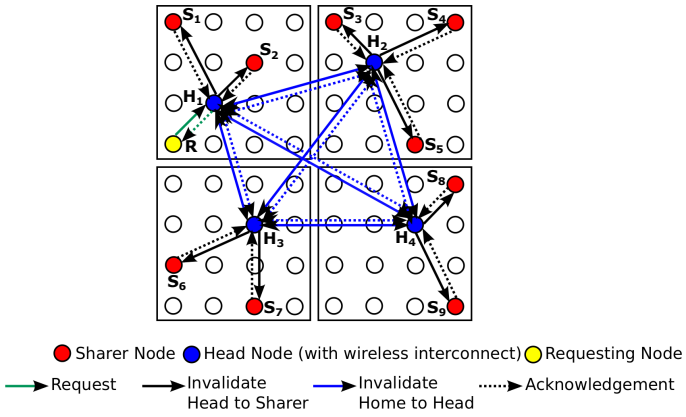

Figure 4: Hierarchical cluster coherence request with Wireless Interconnects

present in L2 cache. The inclusion of this field tracks a data entry in L2 caches of all clusters as the origination address remains the same for all. While invalidating a memory address, the cluster's HEAD broadcasts the invalidate request and the shared memory address to all clusters on the chip. Each cluster's HEAD checks their L2 cache's directory for origination addresses, and only the cores that have local data from the same source address will be invalidated. The proposed cache coherence algorithm can be found in algorithm 1.

D. Walk-through example of proposed coherence scheme

Figure 4 shows our proposed architecture where each cluster has a HEAD core namely $H_1 - H_4$. The requester R wants to update a local data and therefore needs to invalidate all existing copies of data. The copy of the requested data is shared amongst 9 cores ($S_1 - S_9$). The proposed cache invalidation is carried out in the following order:

1) The requesting core R sends an invalidate request to its cluster's HEAD node H_1.
2) The HEAD node H_1 on receiving the request sends invalidate signal to:
   a) Sharers S_1, S_2 in its own cluster via wired interconnects
   b) HEAD nodes H_2, H_3, H_4 of other clusters using wireless interconnects. The receiving HEADs further re-route the invalidation signal to the sharers $S_3 - S_9$ after matching the source address from associated directory.
3) The sharer cores in each cluster invalidates the requested data entry in their local caches and sends an acknowledgment back to their respective clusters HEAD node.
4) On receiving the acknowledgment, the cluster HEAD cores H_2, H_3 and H_4 broadcasts the acknowledgment to be received by the originating cluster HEAD H_1 via wireless interconnects. In case of no sharers in a particular cluster, HEAD of that cluster broadcasts the invalidation acknowledgment
5) On receiving the acknowledgment from all other HEADs and data sharers from its cluster, H_1 sends the acknowledgment to requester R via wired links.

The requester updates the data value only after receiving the acknowledgment from its cluster HEAD.

Parameter	Notation
Shared memory size	S_i byte
L1 cache size	C_{L1} byte
L2 cache size	C_{L2} byte
Memory block size (L1 cache line)	$line_{L1}$ byte
L2 cache line	$line_{L2}$ byte
Number of L1 caches (processors)	N_{L1}
Number of L2 caches (heads)	N_{L2}
Number of members in each cluster	$N_{cluster}$

Table I: Memory notations

IV. Performance Evaluation

We evaluate our proposed architecture using a full system simulator, GEM5 [19]. We modify the Ruby cycle accurate memory simulator of GEM5. Table II shows the system parameters used during the simulations. We have simulated 16 cores and 64 cores systems for validation of our proposed architecture.

To evaluate our proposed cache coherence protocol, we utilize Alpha cores using MOESI (Modified, Owned, Exclusive, Shared and Invalid) based extended directory scheme. We also employ a wide variety of parallel threaded applications for our evaluation. We use SPLASH2 [20] and PARSEC benchmarks with recommended input parameters.

A. Memory Overhead Analysis

Table I shows the notations we used to derive the total memory overhead in the proposed architecture. Memory overhead of a directory is the total memory utilized by the directory for its storage in bits. This can be determined by multiplying the number of blocks in a directory to the number of bits in each block. Following this methodology, we compute and compare the memory overhead in non-clustered directory and proposed clustered directory protocol with wired and wireless interconnects for 16, 32, 64, 128 and 256 core system.

Case 1: Non-clustered directory structure:

$$O_{NC} = \sum_{i=1}^{n} \frac{S_i}{line_{L1}}(N_{L1} + 1) \tag{1}$$

Case 2: Clustered with wired and wireless interconnects directory structure:

$$O_{WI} = N_{L2}\frac{C_{L2}}{line_{L2}}(N_{cluster} + \theta)bits \tag{2}$$

where θ is the length of the extra field in extended directory as shown in figure 3. The value of θ is taken such that it is able to store the address of data in shared memory.

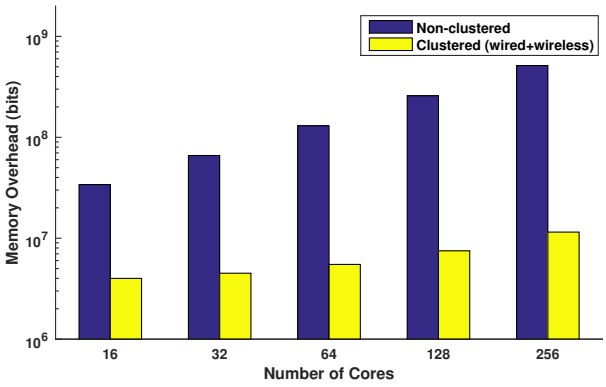

Figure 5: Memory overhead of 16, 32, 64, 128 and 256 core system

Component	Parameters
Processor	ALPHA
Number of cores	16, 64
Cache line size	64 bytes
L1 I/D cache	32KB
L2 cache bank (non-cluster)	2MB
L2 cache size (clustered)	2MB
Number of L2 caches (clustered)	4
Coherence Protocol	Extended MOESI directory
Shared Memory	128MB
Topology	8x8 Mesh with wireless interconnects

Table II: System parameters

Figure 5 shows the memory overhead of 16, 32, 64, 128 and 256 core systems. It can be seen from the results that while increasing the number of cores, the overhead of our system remains fairly constant when compared to non-clustered directory. We observe 88-97% decrease in memory overhead compared to non-clustered architecture. These results support our claim of scalable cache coherent architecture.

B. Execution Time

Figure 6 shows the comparison between normalized execution time of our proposed clustered architecture to that of non-clustered architecture for 64 core system. It is evident from the results that our proposed protocol design achieves 21% speedup compared to non-clustered directory. This increase can be accounted to clustering of cores based on their utilization which results in localization of network traffic and cache data.

C. L2 cache access latency

Figure 7a shows the simulated results of L2 access latency for different benchmarks on a 64 core system. It can be observed that our architecture achieves a 30% improvement to non-clustered architecture.

The L2 cache access latency for an architecture is directly proportional to inter-core Manhattan distance. Figure 4 places the HEAD core of a cluster at a maximum distance of 4 hops to any core of that cluster. This scenario is the worst case for a 64 core system. Therefore, the

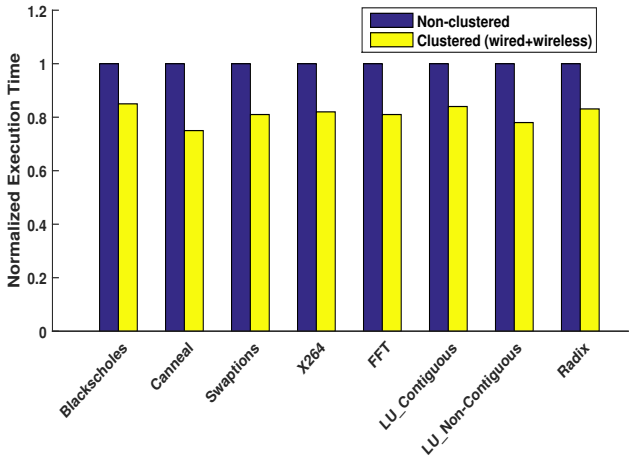

Figure 6: Normalized execution time for different benchmarks

worst case Longest Manhattan Distance (LMD) of our architecture is of 9 hops due to the inclusion of single-hop, long-range wireless links. On the other hand, a non-clustered system will have the worst case LMD of 14 hops. Calculating the decrease, we verify that our simulated results follow closely to our analysis.

D. Normalized L2 Miss Rate

Figure 7b shows the normalized L2 miss rate observed in our simulations. The results indicate that our proposed architecture achieves an average of 32.1% reduction compared to the traditional non-clustered approach. As we follow utilization based clustering of cores, the data needed by cores at execution time is at most times already present in the L2 cache which results in lesser misses at L2 cache.

E. Area Overhead

In this section, we quantify the area overhead associated with the use of on-chip wireless interconnects. The total area overhead with WI (including transceiver and antenna) is determined to be 0.72 mm^2 [16]. We limit the number of WIs to 4 in a 64 core system. Therefore, assuming a $20mm \times 20mm$ die, the wireless interconnects consume only ~1% of total silicon area overhead.

Even though the inclusion of wireless interconnects imposes an area overhead to the overall architecture, yet the advantages of WIs outweigh its drawbacks. Other than the results discussed above, we have also observed a significant improvement in several other parameters. The memory request rate decreases up to 45% and average miss latency shows a decline of 33-39%. We also observe an increase in percentage performance while increasing the number of cores in a cluster.

V. Conclusion

Scalable solution to cache coherence issue in multicore scenario is an open research problem. In this paper, we

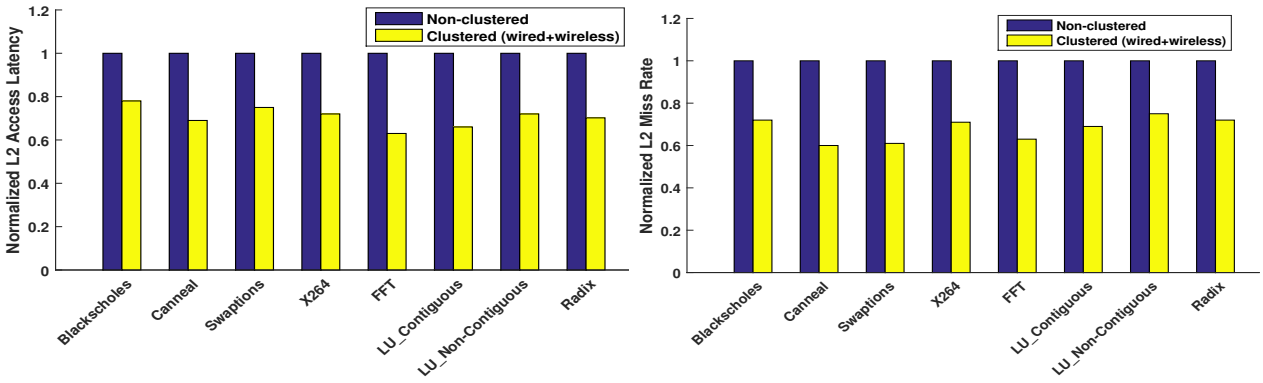

Figure 7: (a) Normalized L2 access latency (b) Normalized L2 miss rate for different benchmarks

have used wireless NoC architecture to achieve scalability. We cluster the cores based on utilization and remove the long-range, high latency inter-cluster wired metal interconnects with single-hop wireless communication links. We have used a simulated environment for verification of our proposed scheme. Our mathematical analysis and simulated results are in close agreement. Our system achieves reduced memory overhead, lower execution time, lower L2 access latency, lower L2 miss rate, lower memory requests and miss latency.

Acknowledgment

This work is partially supported by the TCS Research Scholar Program and DST INSPIRE Faculty Fellowship granted by the Dept. of Science and Technology, India.

References

[1] J. Parkhurst, J. Darringer, and B. Grundmann, "From single core to multi-core: preparing for a new exponential," in *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. ACM, 2006, pp. 67–72.

[2] P. Stenström, "A survey of cache coherence schemes for multiprocessors," *Computer*, vol. 23, no. 6, pp. 12–24, 1990.

[3] S. V. Adve, V. S. Adve, M. D. Hill, and M. K. Vernon, *Comparison of hardware and software cache coherence schemes*. ACM, 1991, vol. 19, no. 3.

[4] J. Archibald and J.-L. Baer, "Cache coherence protocols: Evaluation using a multiprocessor simulation model," *ACM Transactions on Computer Systems (TOCS)*, vol. 4, no. 4, pp. 273–298, 1986.

[5] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *Computers, IEEE Transactions on*, vol. 54, no. 8, pp. 1025–1040, 2005.

[6] S. Deb, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo, "Wireless noc as interconnection backbone for multicore chips: Promises and challenges," *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, vol. 2, no. 2, pp. 228–239, 2012.

[7] D. J. Sorin, M. Plakal, A. E. Condon, M. D. Hill, M. M. Martin, D. Wood *et al.*, "Specifying and verifying a broadcast and a multicast snooping cache coherence protocol," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 13, no. 6, pp. 556–578, 2002.

[8] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *ACM SIGARCH Computer Architecture News*, vol. 16, no. 2. IEEE Computer Society Press, 1988, pp. 280–298.

[9] M. E. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato, "A two-level directory architecture for highly scalable cc-numa multiprocessors," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 16, no. 1, pp. 67–79, 2005.

[10] M. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato, "An architecture for high-performance scalable shared-memory multiprocessors exploiting on-chip integration," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 15, no. 8, pp. 755–768, 2004.

[11] F. Petrot, A. Greiner, and P. Gomez, "On cache coherency and memory consistency issues in noc based shared memory multiprocessor soc architectures," in *Digital System Design: Architectures, Methods and Tools, 2006. DSD 2006. 9th EUROMICRO Conference on*. IEEE, 2006, pp. 53–60.

[12] N. Eisley, L.-S. Peh, and L. Shang, "In-network cache coherence," in *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*. IEEE, 2006, pp. 321–332.

[13] G. Girão, B. C. de Oliveira, R. Soares, and I. S. Silva, "Cache coherency communication cost in a noc-based mpsoc platform," in *Proceedings of the 20th annual conference on Integrated circuits and systems design*. ACM, 2007, pp. 288–293.

[14] Y. Zhang, Z. Lu, A. Jantsch, L. Li, and M. Gao, "Towards hierarchical cluster based cache coherence for large-scale network-on-chip," in *Proceedings of the 4th IEEE International Conference on Design & Technology of Integrated Systems in Nanoscale Era*, 2009.

[15] C. Li, H. Wang, Y. Xue, X. Zhang, and D. Wang, "Fast hierarchical cache directory: A scalable cache organization for large-scale cmp," in *Networking, Architecture and Storage (NAS), 2010 IEEE Fifth International Conference on*. IEEE, 2010, pp. 367–376.

[16] S. Deb, K.-P. Chang, X. Yu, S. P. Sah, M. Cosic, A. Ganguly, P. P. Pande, B. Belzer, and D. Heo, "Design of an energy-efficient cmos-compatible noc architecture with millimeter-wave wireless interconnects," *Computers, IEEE Transactions on*, vol. 62, no. 12, pp. 2382–2396, 2013.

[17] S. Deb, A. Ganguly, K. Chang, P. Pande, B. Beizer, and D. Heo, "Enhancing performance of network-on-chip architectures with millimeter-wave wireless interconnects," in *Application-specific Systems Architectures and Processors (ASAP), 2010 21st IEEE International Conference on*. IEEE, 2010, pp. 73–80.

[18] A. S. Tanenbaum and M. Van Steen, *Distributed systems*. Prentice-Hall, 2007.

[19] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[20] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *ACM SIGARCH Computer Architecture News*, vol. 23, no. 2. ACM, 1995, pp. 24–36.