Roca Pérez, A.; Hernández Luz, C.; Lodde, M.; Flich Cardo, J. (2015). Area-efficient snoopy-aware NoC design for high-performance chip multiprocessor systems. Computers and Electrical Engineering. 45:374-385. doi:10.1016/j.compeleceng.2015.04.020.

Area-Efficient Snoopy NoC Design for High-Performance Chip Multiprocessor Systems

Antoni Roca[a], Carles Hernandez[b], Mario Lodde[c], José Flich[c]

[a]*Department of Computer Science, Universitat Politècnica de Catalunya, Barcelona, Spain*
[b]*Barcelona Supercomputing Center, Barcelona, Spain*
[c]*Department of Computer Engineering, Universitat Politècnica de València, València, Spain*

Abstract

Manycore CMP systems are expected to grow to tens or even hundreds of cores. In this paper we show that the effective co-design of both, the network-on-chip and the coherence protocol, improves performance and power meanwhile total area resources remain bounded. We propose sNoC, a snoopy-aware network-on-chip topology made of two mesh-of-tree topologies. Reducing the complexity of the coherence protocol – and hence its resources –, and moving this complexity to the network, leads to a global decrease in power consumption meanwhile area is barely affected. Benefits of our proposal are due to the high-throughput and low delay of the network, but also due to the simplicity of the coherence protocol. The proposed network and protocol minimizes communication amongst cores when compared to traditional solutions based either on 2D-mesh topologies or in directory-based protocols.

Keywords: Chip multiprocessor, Network-on-Chip, network archirtecture, coherence protocol.

1. Introduction

Manycore systems are expected to grow to tens or even hundreds of cores in the same chip. A Network-on-Chip (NoC) is implemented to connect all of them efficiently [1]. NoCs are to replace conventional bus-based systems where throughput and latency is compromised as the number of cores increases. NoCs have been adopted in two system design approaches: multiprocessor system-on-chips (MPSoCs) and chip multiprocessors (CMPs). In MPSoCs, applications are usually known in advance and the chip is customized to the applications, including the NoC. In CMPs, applications are not known in advance and thus, the chip is

built with little or no information about its future use. In this paper we focus on NoCs designed for CMP systems.

Since its conception in 2001, NoC research has focused mainly in adopting the best strategies usually found in high-performance interconnects, covering aspects like topology, routing, switching, and arbitration. The main challenge found in NoC research has been the suitability of known research and solutions to the highly-constrained new domain (inside the chip). Indeed, many of the proposals have focused on providing very power- and area- efficient solutions, thus minimizing the power consumption and the area footprint of the NoC. With these constraints as a reference, the 2D mesh topology has been adopted as the baseline for NoC design, meanwhile conceptually simpler but better topologies as crossbars are discarded as they show higher area and power overheads. Real examples of simple NoCs are the ones implemented in the Polaris chip prototype by Intel [2], the Single Chip Cloud Computer by Intel [3], and the products offered by Tilera [4].

Orthogonal to the design of NoCs for CMP systems, the memory hierarchy, and its implementation, plays a key role in the final product. A shared variable programming approach is appealing from the point of view of the programmer, instead of the message passing programming approach. The inherent simplicity when programming, however, requires a coherence protocol implementation that ensures coherency and consistency along all the memory hierarchy levels. It is typical to find approaches where the processors in CMP systems have a first level (L1) of private caches and a bank of L2 caches on each tile forming a global shared but distributed L2 cache. The third level of the memory hierarchy is main memory. Figure 1 shows the CMP configuration we focus in this paper.

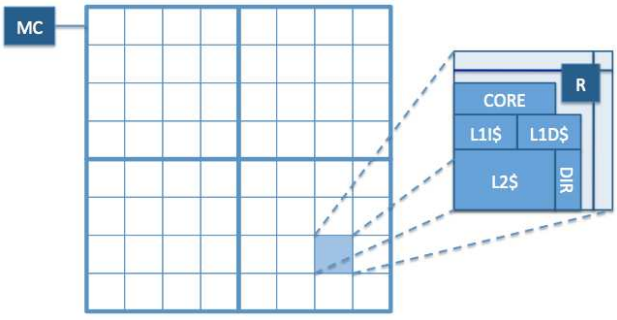


Figure 1: Tiled-CMP schematic.

The coherence protocol implemented in the system can significantly vary from

an implementation point of view. For example, snoopy protocols rely on a shared medium connecting all the processors. Typically, snoopy protocols are very simple to design and test [5]. On the contrary, the mostly-assumed directory-based protocols require a directory structure to keep the coherence information, that makes them much more complex to design and, most importantly, to test and validate, since they rely on a point-to-point network with no global visibility. This leads to race conditions of the protocol when multiple cores access the same block at the same time. The number of states of the protocol increases to an extend that prevents its validation in an affordable amount of time. Additionally, directory structures require dedicated resources at cache memories increasing area and power consumption.

When the two components (NoC and coherence protocol) are put on the same perspective we can identify an interesting conflict. Discarded topology structures like buses or crossbars offer the opportunity to implement simple coherence protocols, like the snoop-based protocol. Other preferred topologies, like the 2D mesh, do not allow snoop-based protocols thus need for more complex protocols, e.g. directory-based protocols.[1]

What we pursue in this paper is the effective co-design of both, the NoC and the coherence protocol, in order to improve performance and power meanwhile area resources remain bounded. If we analyze a typical CMP system, L1 caches and L2 banks resources and power clearly overcome resources and power consumed by the network. Thus, reducing the complexity of the coherence protocol – and hence its resources –, and moving this complexity to the network, will lead to a global decrease in power consumption meanwhile area is barely affected. In this paper, we pursue the following properties to the final designed system:

- A simple coherence protocol that can be easily tested and validated. In particular, a snoop-based protocol.
- A customized NoC scalable enough for a relative large amount of nodes, compatible with the snoop-based protocol. In particular, reaching 128 processors on the same chip.

The solution we propose is called sNoC, referring to a snoopy NoC. sNoC is built with two network components, each implemented as a mesh-of-trees (MoT) [6].

[1] One exception is the case of using a broadcast-based protocol in a 2D mesh network. Each snoopy action is converted into a broadcast. In this case, the network is flooded with many messages.

The first MoT is used to broadcast requests whereas the second MoT is used to send unicast data messages. sNoC increases network resources but helps reducing the directory structure, containing overall power consumption. sNoC includes also a customized coherence protocol for the network.

System level evaluation, in terms of performance, area overheads, and power consumption, show the viability and the higher efficiency of sNoC when compared to the typical designs of 2D meshes with directory-based protocols. The benefits of our proposal are due to the better performance of the MoT network, and the ability of the snoopy protocol to take the best of these high performance networks. In this sense, we show how a directory-based protocol underutilizes the MoT making this solution not attractive even being feasible in terms of area.

Results show that for a 64-node network our sNoC reduces execution time and power consumption up to 20% and 35% with respect to the 2D mesh, respectively. Benefits are because the snoopy protocol reduces communication between nodes, in terms of number of messages in addition to the sNoC high-throughput architecture. Introducing a high-throughput MoT with a conventional directory-based invalidation protocol reduces execution time up to 13% due to the high-throughput low-latency network, but power consumption is increased up to 15% due to network inefficiency. Additionally, introducing a simple snoopy protocol improves scalability, as the sNoC just increases area by 2%, meanwhile the same network without the snoopy protocol increases area up to 10%.

The rest of the paper is organized as follows. In Section 2, the related work is presented. In Section 3, the snoopy protocol is presented. In Section 4, we introduce sNoC architecture. In Section 5, we evaluate our design and show that it overcomes previous state-of-art solutions meanwhile scalability is guaranteed. Finally, the main conclusions are presented at the end of the paper.

2. Related Work

Snoopy protocols were the preferred solution for ensuring cache coherence in the first multicore designs. For example, the Pentium 4 chip used a source-synchronous protocol to handle a bus amongst 4 cores [7]. Snoop-based protocols are very simple in theory. However, to be practical, an efficient broadcast medium is required to handle ordered transactions. As the number of processor and memories to interconnect in the chip has increased in the past years, many network-on-chip architectures have been designed and proposed. Amongst the different possible topologies, the 2D-Mesh has been the most adopted approach due to its low implementation cost and its physical scalability [1]. In this regard, since the

adoption of the network-on-chip interconnection paradigm, snoopy protocols have been discarded due to the increased broadcasting complexity. However, simple solutions as 2D-meshes do not fulfill high-throughput low latency requirements. In this sense, some solutions as packet switching [8, 9] or – the more disruptive – introducing optical networks have been proposed. In this paper, we propose an orthogonal solution that reuses multiprocessor system snoopy concepts.

Several different protocols have been proposed to overcome the scalability problems of snoopy protocols associated to the interconnection architecture. The Hammer AMD's protocol [10] avoids keeping coherence information at the cost of broadcasting requests to all cores. In [11] a snoopy-based coherence mechanism for non-ordered networks is proposed. Directory-based protocols [12] reduce network traffic as directory structure allows to store information about the private caches state. However, these traditional cache coherence protocols introduce indirection in the critical path of cache misses [13]. To solve the problem of indirection, Token-CMP [14] and DiCo-CMP [15] protocols have been proposed. These indirection-aware protocols avoid the access to the home node allowing latency of cache misses to be reduced. All these protocols (Hammer, Directory-based, Dico-CMP, and Token-CMP) can be implemented over non-ordered interconnects [13], and therefore, avoiding the use of costly interconnection architectures. However, the inherent costs of these proposals are not negligible. The Hammer protocol [10] though efficient in terms of area requirements generates a prohibitive amount of network traffic. The remaining protocols present to a greater or lesser extent two main drawbacks. First, they introduce significant hardware costs to keep cache state information. Second, the verification process of these protocols is hard or even unfeasible [5].

Recent proposals have explored the scalability of crossbar and bus designs. Concretely, in [16], authors showed that from an implementation point of view crossbars are feasible even for a hundred of cores. In this regard, when the two components (NoC and coherence protocol) are put on the same perspective the costs of implementing a snoop-based protocol can be afforded.

Finally, some authors have pointed out the importance of adapting network-architectures to the memory architecture. In [17] a low-latency low power bus connected to a conventional 2D mesh is proposed. Authors in [17] used the bus to efficiently handle broadcast and multicast messages reducing network latency. In [18], authors introduce crossbars in NoC environment and used them to implement an efficient invalidation MOESI protocol.

In this paper, we go one step beyond and perform an efficient co-design of both the NoC and the memory coherence protocol. Concretely, we describe the

snoop-based protocol implementation [5]. The snoopy protocol efficiently makes use of a high performance network design. Our network proposal resembles a bus subnetwork plus a crossbar subnetwork. To efficiently implement a Mesh-of-Tree [6, 19], we have followed a distributed floorplan [20] approach to minimize network consumption.

3. Snoopy Protocol

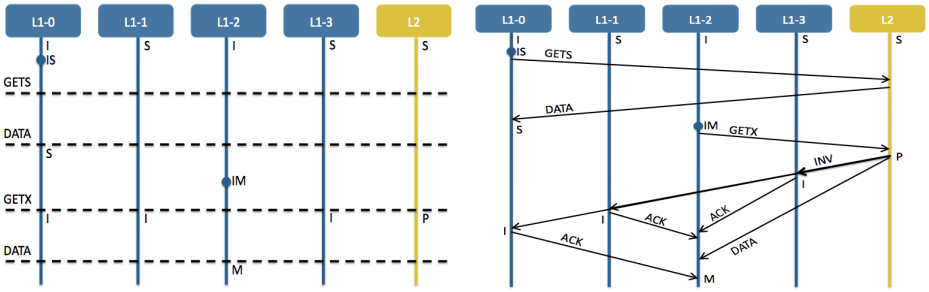
Cache coherence protocols can be classified in two classes depending on whether the request issued after a L1 cache miss is broadcast to all L1 and L2 caches in the system or send to a specific node: in the first case the coherence protocol belongs to the family of snoopy protocols, while in the second case belongs to directory protocols.

Snoopy protocols usually rely on a shared communication medium (typically a bus) with a total ordering of the messages. In case of a L1 cache miss, a request is broadcast to all the caches in the system. Each cache controller's finite state machine evolves depending on the current cache line state and on the request type, and the protocol is designed to let all the caches independently evolve to a global correct state which guarantees the single-writer, multiple-readers invariant. Messages in the interconnection network must be totally ordered to let all the caches see the same order of issued requests. The main drawback of snoopy protocols is due to the shared interconnect which limits their scalability.

In directory protocols the request in case of a L1 miss is sent to a single node, called the home node, which is in charge of managing all the requests issued by L1 caches and thus acts as the synchronization point: the requests are managed following the order of their reception at the home node; the home node is typically associated to the lower shared level of the memory hierarchy. The home node uses a data structure, called the directory, to keep track of which cores have a copy of each block in their private caches. In its typical implementation, the directory consists of a bit vector associated to each cache line, with the size of the vector equal to the number of cores in the system. This limits the scalability of directory protocols since the directory introduces an area overhead which grows with the system size.

Figure 2 shows an example of how the two protocols manage the same sequence of requests generated by L1 caches in a system with 4 cores and one bank of L2 cache. At the beginning, the block we consider is shared by the L1 caches of cores 1 and 3 (L1-1 and L1-3). Then, a read request is issued by L1-0 and a write request by L1-2. If a snoopy protocol is used, L1-0 broadcasts its request;

when the L2 cache receives the request, it sends the data block to L1-0; the block is now shared by L1-0, L1-1 and L1-3. After a write miss in L1-2, a write request is broadcast; when the request is received by the nodes which share a copy of the block, they invalidate their copy, while the L2 cache sends the requested data to L1-2, which will hold the only valid copy of the block.



(a) Snoopy protocol. (b) Directory protocol.

Figure 2: Directory and snoopy protocol behavior.

On the contrary, in a directory-based coherence protocol system, the L1-0 sends a unicast read request to the home node (the L2 cache bank), which adds L1-0 to the sharers list of that block and provides the requested data block. L1-2 also sends a unicast write request to the L2 bank, which in turn sends an invalidation message to each sharer and the requested data to L1-2; a field of the invalidation message includes the ID of the L1 which issued the request, while the data message indicates the number of sharers that are being invalidated. When a sharer receives the invalidation message, invalidates its copy of the block and sends an acknowledgment message to L1-2; L1-0 can use the block only after the reception of the data and all the acknowledgments, meaning that all the sharers have received the invalidation message and invalidated their copy of the block.

4. Network Architecture

4.1. Generic Mesh-of-Tree

In this section, we describe a generic mesh-of-tree (MoT) [6] which is the baseline of the network implemented throughout this paper. Figure 3 shows a 4-node MoT network. A MoT is a tree-based structure where packets are sent from network interface (NI) injectors to NI ejectors in two parts: fan-out tree and fan-in tree. The fan-out tree is made of 1-to-2 switches. Those switches

behave as demultiplexers, thus no arbitration logic is implemented. The fan-out tree connects the NI injectors with the fan-in trees, one per NI ejector. Fan-in trees are made of 2-to-1 switches. As there exists one fan-in tree per NI ejector packets with different destination addresses do not compete among them. MoT presents a low implementation cost as it just provides a single path between source and destination. This renders in faster implementations of the interconnect [6, 19, 20]. Additionally, these networks provide high throughput and low latency.

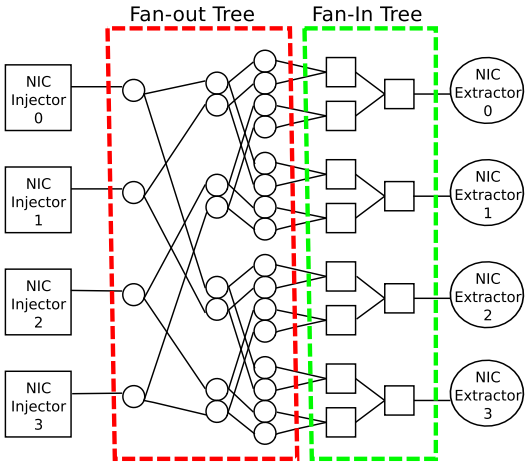


Figure 3: 4x4 Mesh-of-Tree Baseline schematic.

4.2. Network Implementation

The sNoC network is made of two subnetworks, as can be seen in Figure 4. Both subnetworks are implemented using the baseline MoT topology. One subnetwork forwards broadcast request messages meanwhile the second subnetwork forwards unicast response messages – data messages.

In a MoT, all routing complexity is moved into the fan-out tree that must compute the target fan-in tree – target node. The routing algorithm delivers incoming flits to the proper output analyzing the routing bits inserted in the flit. On the contrary, each fan-in tree switch require arbitration capabilities as can be seen in Figure 5. Note that, each 2-to-1 switch is made of a simple multiplexer with arbitration and flow control capabilities. An incoming flit requires one cycle to cross the network similarly to [19]. This MoT architecture achieves N flits/node/cycle throughput. In this sense, it resembles a registered crossbar which achieves maximum throughput [21, 20] at the expense of increasing resources. The main difference is that a registered crossbar is typically managed by a central control unit

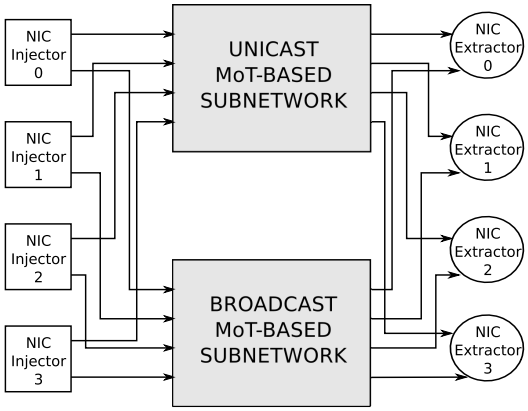


Figure 4: sNoC schematic.

which decides which input is connected to each output at a time. In our MoT subnetwork, arbitration decision is distributed through the fan-in tree switches. Distributing the arbitration policy reduces MoT complexity and increses its operating frequency. In this paper a simple round-robin arbitration policy is implemented at each fan-in tree switch.

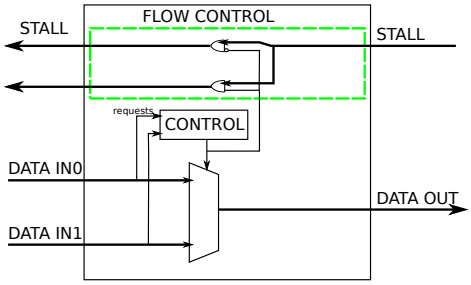


Figure 5: Fan-in tree switch schematic.

The MoT is placed at the center of the die (see Figure 6). Thus, each core is connected to the MoT by using a bidirectional link. In order to minimize link delay and power consumption, those links are pipelined. The link piepeline depth depends on core distance to the network and the core width. In this paper, we assume that all cores have the same pipeline depth set to $\sqrt{N}$ where N is the network size. Equalizing pipeline depth is important to balance network traffic. Distance between two nodes will be identical independently of core distance. Notice that, in sNoC zero-load latency is equal to $2 * \sqrt{N} + 1$, N cycle from the core to the network, one cycle to cross the MoT, and then, n to reach the destination

core.

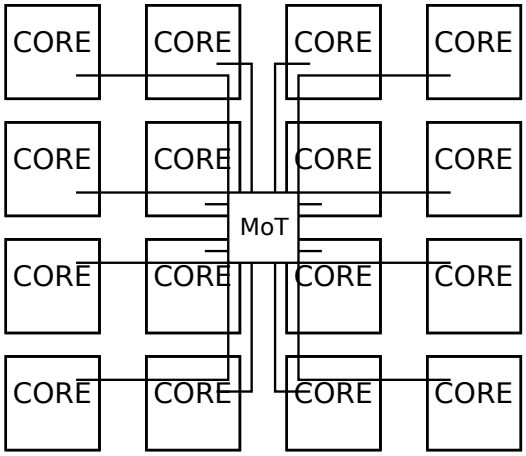
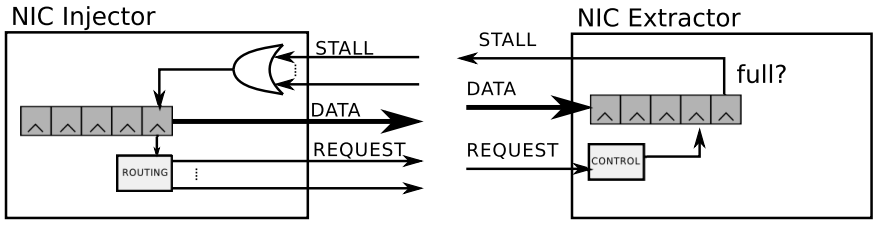


Figure 6: Network placement in the die.

The MoT subnetwork requires a flit by flit communication flow control at the switch level. That is, each switch communicates with its attached switches – one downstream switch and two upstream switches. Thus, some extra physical signals travel together with data flits. From the upstream switches, incoming flits travels with a *request* signal which validate data. From the downstream switch one stall signal is received. If active, the switch propagates a flit. If not, the switch stalls communication flow, storing incoming flits into its buffer. When the buffer gets full, it propagates the stall signal to its upstream switches, stalling the whole communication flow. When communication flow restarts, stall signals propagate cycle by cycle from downstream switch to upstream switches. Notice that a switch sends different stall signals to its upstream switches, in fact, only the upstream switch that grants the buffer could have its stall signal active. This communication scheme is called elasticity [22].

Figure 7 shows the communication flow control between the NI injector/ejector and the subnetwork is attached. If it is a unicast message only one request signal will be active at each cycle. For broadcast communications all request signals will have the same value. As it can be seen in Figure 7(b), NI ejector is simpler as it is attached to a single switch. It just stalls communication flow when its buffer resources are full.



(a) NI injector. (b) NI ejector.

Figure 7: NI injector and ejector schematic.

4.3. Request Messages Subnetwork

The subnetwork which handles broadcast request messages must be adapted to support broadcast communication. Figure 8 shows the schematic of a 4x4 broadcast subnetwork. As messages are broadcasted, routing computation is removed. As it can be seen, request messages subnetwork is a single fan-in tree connected to a fan-out tree. One important aspect of a snoopy protocol is that all accesses must be seen by all the nodes in the same order. Also, as messages are broadcast, every message must arrive to all ejector NIs. To achieve that, the fan-in tree orders incoming messages meanwhile the fan-out tree delivers messages to all ejectors NIs. In this sense, stall signal generation is modified. As fan-in tree is connected to any end node, when one end node is not enable to receive more messages it stops the whole communication flow. That means, that when a NI ejector receives a message, all NI ejectors have received the same message.The fact that a node perfectly knows that the rest of nodes are receiving the same information highly simplifies the coherence protocol.

With this implementation we obtain a totally ordered network. That is, packets introduced in cycle T get out the network before packets introduced in time $T+t$, for any $t > 0$. For that purpose, the control unit of the switches must be modified (see Figure 3). Thus, the policy to access switch resources is not a simple round-robin algorithm but older packets have priority over newer ones. When packet age is identical a simple round-robin policy is assumed.

The physical distance between NI ejectors and the broadcast subnetwork output could be large. That means that, the delay between the stall signal generation at each NI ejector and its reception at the broadcast subnetwork could be large. To decouple the overall network critical path and the stall signal generation critical path, NI ejector buffer must be designed to fulfill round-trip-time constraint. That is, it must be able to consume the flying flits that exist meanwhile the stall signal is generated and the broadcast subnetwork stops. The number of flying flits that

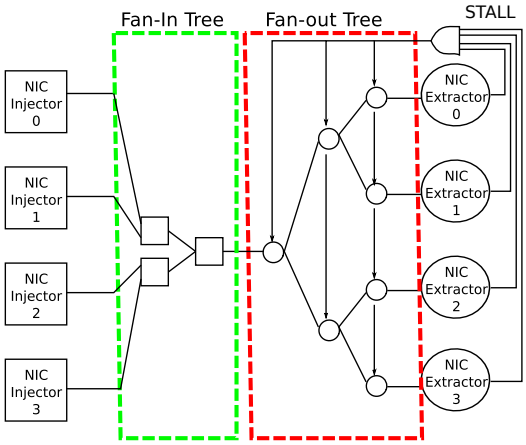


Figure 8: 4x4 broadcast subnetwork schematic.

the NI ejector buffer could consume determine how many times the stall signal delay could be larger than the network critical path.

The broadcast subnetwork resembles a bus. That is, an interconnection structure where only one flit is delivered at each cycle. However, allowing broadcast messages permits to implement a snoopy protocol. However, the MoT broadcast subnetwork is a low latency network as its operating frequency is identical to the rest of the network, removing the main disadvantage of conventional buses which are high latency.

5. Experimental Results

To analyze the benefits of introducing the snoopy protocol, we implement and compare different architectures for a 64-node network. First, we have implemented a two subnetwork MoTs as described in Section 4 with two cache coherence protocols: a snoopy protocol as described in Section 3 (sNoC) and a conventional directory-based invalidation MESI protocol (MoT_dir). Additionally, we have implemented a conventional NoC solution made of a 2D mesh with the directory-based protocol (2D_dir). In this case, three virtual networks are used in order to remove message protocol-level deadlock and, no broadcast messages are allowed. Finally, we have implemented a two virtual network 2D mesh with a directory-based protocol, where a MoT broadcast subnetwork is added which handles broadcast request messages (2D_bus). This last architecture resembles the one presented in [17] and helps to better understand the benefits of using the snoopy protocol in addition with a high throughput architecture.

The 2D meshes have been implemented by using a modular switch design as shown in Figure 9. As it can be seen, 2D switches are made of fan-in switches identical to those used to build the MoT described in Section 4. Thus, a flit requires two cycles to cross a switch when no contention exist. By implementing switches as a mesh-of-tree, throughput is increased and latency is decreased when compared with a canonical switch design with identical functionality [23]. However, the main reason to implement switches as small MoT is fairness, as 2D mesh operating frequency will be similar to the operating frequency of the MoT tree. Network performance can be measured with FPGA-based prototyping [24] or with cycle accurate network simulators. In this paper we have used a cycle accurate simulator as it provides more flexibility to model each network configuration. In particular, we have used the gNocsim simulator [25]. In the network link width is set to 64 which is equal to the short message size. Long messages are split in 9 flits. Fan-in switch buffering depth is set to two in order to minimize network area.

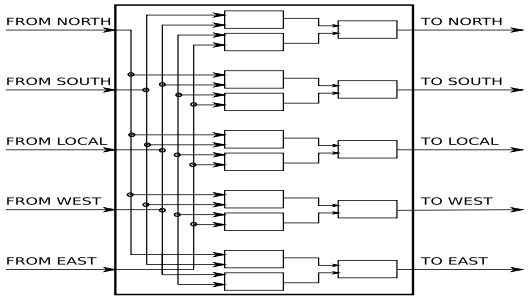


Figure 9: 2D switch schematic built with fan-in switches.

All architectures have been synthesized using the Design Compiler tool by Synopsys [26]. The placement and routing has been performed using the ICC tool by Synopsys [26]. Finally, power consumption parameters have been extracted by using the Power Compiler tool by Synopsys. The 40nm TSMC technology library have been used to implement the network [27]. After Place&Route, network area and power consumption have been computed. We have implemented all designs using a nominal voltage of 0.9V. The MoT network is the slowest network, where the critical path is 1.2ns. Thus, the NoC operating frequency has been set to 0.83GHz. Intercore distance has been set to 1.2mm.

Cache memories have been modeled with the CACTI [28] tool by HP labs to obtain their area and power consumption. 32k L1 and 256k L2 caches are used. Line size is set to 64 bytes as network link width. In both cases, one read and

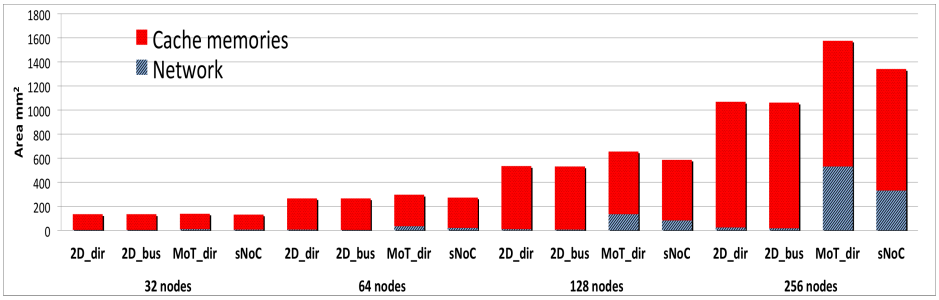


Figure 10: Area in mm^2 comparison for different network sizes.

one write port have been considered. Finally, two different L2 tag configurations have been considered to adapt L2 cache memories to the different coherence protocols. For the snoopy coherence protocol, only 23 bits are required to build the tag structure. For the directory-based coherence protocol, extra bits are needed to implement the directory structure. Thus, the L2 has been built with 87 bits per tag. Notice that memories used for the snoopy coherence protocol are smaller than the ones used for a directory-based protocol, as the tag structure is simplified.

5.1. Snoop system scalability

In this section, we analyze the MoT snoopy system scalability when compared with the other architectures. In order to explore the scalability of the different approaches we measure two parameters. First, we analyze the network and memory caches area for different number of nodes. Then, we analyze the broadcast subnetwork performance which is the bottleneck of the sNoC architecture.

Figure 10 shows the area in mm^2 for different architectures and different network sizes: 32, 64, 128, and 256 nodes. Due to computing limitations, we cannot implement the 128-node and 256-node networks. In those cases, area has been extrapolated from smaller networks results. Basically, we have estimated how many resources are required – in terms of fan-in and fan-out switches, and wiring – from smaller network cases. Finally, network area is computed as a sum of each resource area estimation. As it can be seen, network area influence over the global system area is negligible for systems with less than 32 nodes (less than 5%). As the number of nodes increases, MoT-based network area grows faster than cache memories area. For a 256-node network MoT_dir network area becomes the 33% of the whole system area. For 2D-based architectures network area remains negligible.

As it can be seen in Figure 10, 2D-based architectures have the lowest area

for large systems. On the contrary, MoT_dir has the highest area due to network unscalability. MoT_dir increment with respect to the 2D_dir is up to 11%, 23% and 48% for 64, 128, and 256 nodes, respectively. Notice that, as the snoopy coherence protocol removes directories, sNoC minimizes MoT unscalability. For a 64-node network sNoC is just a 2% larger than the 2D-mesh. For larger networks, differences increase up to 10% and 22% for 128, and 256 node networks, respectively.

In the sNoC architecture, the broadcast subnetwork is the bottleneck of the system as it only extracts one flit per cycle (identically to a bus). However, for current benchmarks this limitation does not impact the whole design. Figure 11 shows throughput demanded by applications on the broadcast subnetwork measured in flits/cycle. As it can be seen, application traffic demand is much lower than broadcast subnetwork capability, that is, one flit per cycle. Canneal has the highest bus demand but it is only 0.55 flits per cycle. On average, this demand is lower than 0.1 flits per cycle.

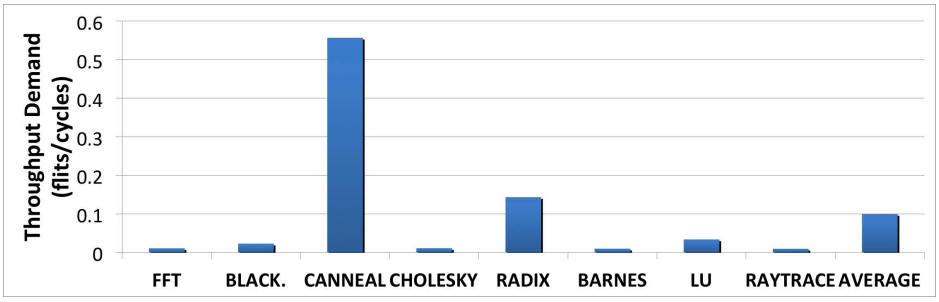


Figure 11: Broadcast subnetwork throughput demand.

5.2. Execution Time

In this section, we analyze the execution time of the different architectures presented above. Execution time is measured in cycles and real application traffic from SPLASH and PARSEC benchmarks [29, 30] have been used. Figure 12 shows the decrease in the execution time of the 2D_bus, MoT_dir, and sNoC architectures, when compared to the 2D_dir. Figure 12 shows how MoT-based solutions clearly outperforms 2D-mesh based solutions, as the MoT architectures have higher throughput and lower network latency when congestion exists. Introducing a MoT-subnetwork as 2D_bus does, improves performance but this difference is not as big as those obtained with MoT architectures. On average, 2D_bus architecture reduces execution time by 5%.

More interesting conclusion is how the coherence protocol impact on high performance networks. Notice that, the directory-based protocol underutilized MoT resources, as the MoT_dir execution time decrease is lower than the decrease obtained by sNoC. On the contrary, the simpler snoop protocol efficiently utilizes the MoT throughput. On average, MoT_dir reduces execution time by 13%, meanwhile sNoC reduces the execution time by 20%. Remember that, for a 64-node network, sNoC is just 2% larger than the 2D_dir.

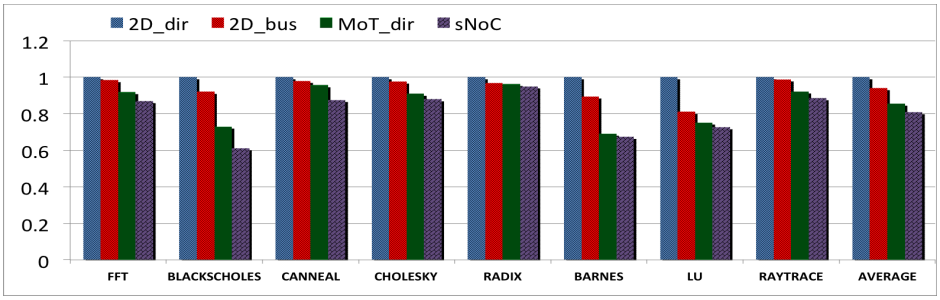


Figure 12: Normalized execution time with respect to 2D_dir.

5.3. Protocol Injected Messages

Figure 13 shows the number of messages injected for different applications when using a directory-based protocol (Directory) and the snoopy protocol (Snoopy) presented in Section 3. As it can be seen in Figure 13, snoopy protocol injects less messages in the network due to its simplicity. On average, the number of messages injected into the network is reduced by 35%. This reduction in number of messages is strongly related with the application execution time shown above, as communication between nodes requires less messages and hence, application runs faster. Additionally, reducing network messages reduces whole system power consumption, as it will be shown later.

5.4. Power Consumption

In this subsection we show the power consumption of the different architectures previously exposed. Power consumption is decomposed as the sum of the power of the L1 and L2 cache memories, and the network power consumption.

Table 1 shows the static power consumption of a single L1 and L2 cache memories. Two kinds of L2 banks have been considered, one L2 suitable for the directory-based protocol (L2_dir) which requires more TAG bits as stated in previous section, and a smaller L2 sufficient for the snoop protocol (L2_snoop).

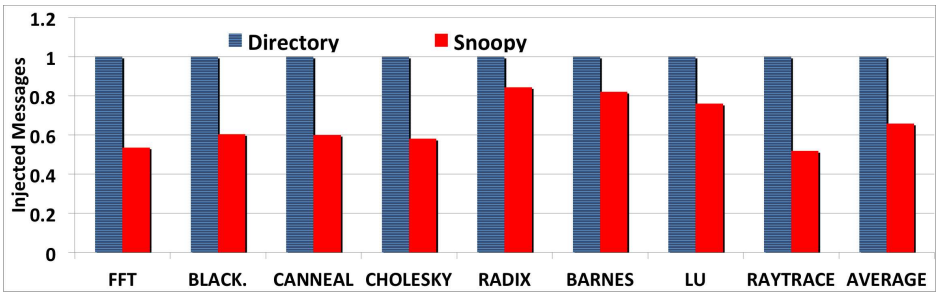


Figure 13: Network injected messages.

Table 1 also shows the power required to perform a read/write access at its maximum operating frequency.

Power (mW)	**Static power**	**Dynamic Power**
L1	0.03	1.55
L2_dir	1.01	4.32
L2_snoop	0.9	3.55

Table 1: Power consumption in mW for a single L1, L2_dir, and a L2_snoop.

Cache memory power consumption can be calculated by:

$$P_{Li} = E_{read} * A/ET + P_{static} \quad (1)$$

where A is the number of read/write accesses to a cache memory, E_{read} is the energy required for a read access, and ET is the execution time. As stated in previous section, one read/write access per cycle is allowed.

Table 2 shows the static and dynamic power consumption the different network implementations. Static power reflects the power consumption when no flits crosses the network for each configuration. Meanwhile the dynamic power consumption reflects the power consumption when flits cross the network. For the MoT designs, dynamic power consumption is measured for a unicast flit and a broadcast flit. In MoT cases, links between cores and the network are included. Notice that, for the 2D case flit power consumption depends on how many hops (H) the flits realizes. Notice that, sNoC power consumption is larger than a conventional 2D mesh. This is specially critical in case of broadcast messages. Such a high power is because any broadcast message must arrive to any end node. However, snoop protocol generates less messages than a 2D mesh. Reducing the number of injected messages minimizes the impact of the high power consumption of

a single sNoC broadcast message. However, the reduction of messages injected into the network minimizes that difference.

Power (mW)		**Static power**	**Dynamic Power**
sNoC	unicast	31.96	2.152
	broadcast	25.96	41.44
MoT_dir		31.96	2.152
2D_dir		20.41	0.236*H

Table 2: Power consumption in mW for the different network implemented.

Thus, network power consumption can be computed as:

$$P_{network} = N_{sub} * P_{Nstatic} + E_{Bflit} * IF * B/ET + E_{Uflit} * IF * (1-B)/ET \quad (2)$$

where N_{sub} is the number of subnetwork each network have, $P_{Nstatic}$ is the static power consumption of a single subnetwork, ET is the execution time of the application, E_{Bflit} is the energy required to broadcast a single flit, E_{Uflit} is the energy required for transmitting a single unicast flit, IF are the number of injected flits in the network, and B is the proportion of broadcast flits.

Figure 14 shows the power consumption (in mW) when executing different real applications over the different architectures described. As it can be seen, power consumption benefits of sNoC are strongly related with messages generated. Then, meanwhile a power consumption reduction of almost 40% in Lu and Radix applications is achieved, sNoC consumes more power in other applications with more generated messages, a 21% more in Barnes, and a 12% more in Canneal. In mean, sNoC presents a 15% reduction in power consumption. Two aspects should be considered. First, despite sNoC requires more power to transmit messages – specially broadcast messages – it reduces overall number of messages. Additionally, sNoC memories are simplified and present a lower power consumption than a directory-based memory. Power consumption saved at memories compensate the increment in power consumption that occurs at the network. On the contrary, MoT_dir presents the highest power consumption as expected, as it increases resources, meanwhile injected flits and cache memories are not optimized. On average, MoT_dir increases power consumption by 52%. 2D_bus reduces power consumption by 10%, due to reduction in resources when introducing the MoT broadcast subnetwork and its capability of injecting broadcast messages. However, this reduction is not as large as the power consumption reduction achieved by sNoC.

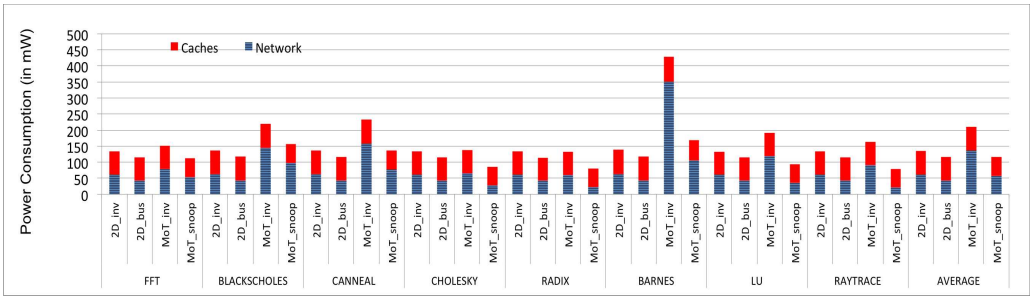


Figure 14: Power consumption (in mW) for the different architectures.

6. Conclusions

We present sNoC, a high-throughput low-latency mesh-of-tree based network which handles a snoop-based protocol. The conjunction of a high-throughput low-latency and a simple and fast coherence protocol, allows to reduce execution time and power consumption, meanwhile area overhead is bounded, when compared with a 2D-mesh network with a directory-based protocol.

It is important to understand that the benefits of sNoC rely on the synergy that exists between the snoopy protocol and our high-throughput network. First, execution time is reduced because the snoopy protocol reduces communication messages between nodes, but also, because the high-throughput network can consume these messages without congestion. Second, power consumption is reduced and area overhead is limited, because the snoopy protocol reduces cache memories power consumption and area when compared to a directory-based directory-based protocol, compensating the increment in power consumption and area of the MoT based network. In this sense, a high-throughput network with a directory-based protocol only reduces application execution time but power and area increase. Similarly, a snoopy protocol cannot be implemented in a simple 2D-mesh network as broadcast messages flood the network, increasing application execution time and power consumption.

Results show that for a 64-node network sNoC achieves a 20% in execution time, meanwhile power consumption can be achieved if the snoopy protocol reduces the number of injected messages significantly. On the contrary, a high-throughput low-latency MoT network with a directory-based protocol only reduces execution time up to 13%, but power consumption is increased by 52%. Additionally, sNoC improves MoT networks scalability. Our sNoC only increases by 2% system resources, meanwhile a simple MoT with a directory-based proto-

col increases the area by 11%.

References

[1] J. Flich and D. Bertozzi, *Designing Network On-Chip Architectures in the Nanoscale Era*. CRC Press, 2010.

[2] Y. Hoskote, S. R. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-ghz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[3] J. Rattner. Single-chip cloud computer: An experimental many-core processor from intel labs. [Online]. Available: http://download.intel.com/pressroom/pdf/rockcreek/SCCAnnouncement.

[4] Tile-gx processors family. [Online]. Available: http://www.tilera.com/products/TILE-Gx.php.

[5] D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*, ser. Synthesis Lectures on Computer Architecture, 2011.

[6] A. O. Balkan, G. Qu, and U. Vishkin, "A mesh-of-trees interconnection network for single-chip parallel processing," in *ASAP*, 2006, pp. 73–80.

[7] G. Hinton, D. Sager, M. Upton, D. Boggs, D. P. Group, and I. Corp, "The microarchitecture of the pentium; 4 processor," *Intel Technology Journal*, vol. 1, p. 2001, 2001.

[8] J. Heisswolf, R. König, M. Kupper, and J. Becker, "Providing multiple hard latency and throughput guarantees for packet switching networks on chip," *Comput. Electr. Eng.*, vol. 39, no. 8, pp. 2603–2622, Nov. 2013.

[9] H. Li, H. Gu, Y. Yang, and X. Yu, "A hybrid packet-circuit switched router for optical network on chip," *Comput. Electr. Eng.*, vol. 39, no. 7, pp. 2197–2206, Oct. 2013.

[10] *AMD Opteron Shared Memory MP Systems*, Aug. 2002.

[11] N. Agarwal, L.-S. Peh, and N. K. Jha, "In-network snoop ordering (inso): Snoopy coherence on unordered interconnects," in *HPCA*, 2009, pp. 67–78.

[12] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A scalable architecture based on single-chip multiprocessing," 2000.

[13] A. Ros, M. E. Acacio, and J. M. García, *Parallel and Distributing Computing*. IN-TECH, Jan. 2010, ch. Cache Coherence Protocols for Many-Core CMPs, pp. 93–118.

[14] M. Martin, M. Hill, and D. Wood, "Token coherence: decoupling performance and correctness," in *30th Annual Computer Architecture*, 2003, pp. 182–193.

[15] A. Ros, M. Acacio, and J. Garcia, "Dico-cmp: Efficient cache coherency in tiled cmp architectures," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 2008, pp. 1–11.

[16] G. Passas, M. Katevenis, and D. N. Pnevmatikatos, *VLSI Micro-Architectures for High-Radix Crossbar Schedulers.* NOCS, 2011.

[17] R. Manevich, I. Walter, I. Cidon, and A. Kolodny, "Best of both worlds: A bus enhanced noc (BENoC)," in *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, 2009, pp. 173–182.

[18] S. Satpathy, K. Sewell, T. Manville, Y.-P. Chen, R. G. Dreslinski, D. Sylvester, T. N. Mudge, and D. Blaauw, "A 4.5tb/s 3.4tb/s/w 6464 switch fabric with self-updating least-recently-granted priority and quality-of-service arbitration in 45nm cmos," in *ISSCC*, 2012, pp. 478–480.

[19] A. Rahimi, I. Loi, M. R. Kakoee, and L. Benini, "A fully-synthesizable single-cycle interconnection network for shared-l1 processor clusters," in *DATE*, 2011, pp. 491–496.

[20] A. Roca, C. Hernández, J. Flich, F. Silla, and J. Duato, "Enabling high-performance crossbars through a floorplan-aware design," in *ICPP*, 2012, pp. 269–278.

[21] M. Lin and N. McKeown, "The throughput of a buffered crossbar switch," *IEEE Communications Letters*, vol. 9, no. 5, pp. 465–467, 2005.

[22] J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, "Elastic circuits," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 28, no. 10, pp. 1437–1455, 2009.

[23] A. Roca, J. Flich, F. Silla, and J. Duato, "A low-latency modular switch for cmp systems," *Microprocessors and Microsystems*, vol. 35, no. 8, pp. 742–754, 2011.

[24] M. Arjomand, A. Boroumand, and H. Sarbazi-Azad, "A generic FPGA prototype for on-chip systems with network-on-chip communication infrastructure," *Computers & Electrical Engineering*, vol. 40, no. 1, pp. 158–167, 2014.

[25] U. P. de Valencia. gnocsim. a cycle-accurate wormhole-based network simulator. [Online]. Available: http://www.gap.upv.es/index.php?option=com_content&view=article&id=72&Itemid=108

[26] "Synopsys inc." Tech. Rep. [Online]. Available: http://www.synopsys.com/

[27] T. foundry. TSMC 40 nm technology. [Online]. Available: http://www.tsmc.com/english/dedicatedFoundry/technology/40nm.htm

[28] N. Muralimanohar and R. Balasubramonian, "Cacti 6.0: A tool to understand large caches."

[29] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: characterization and methodological considerations," ser. ISCA '95.

[30] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The parsec benchmark suite: characterization and architectural implications," ser. PACT '08, 2008.

Antoni Roca Perez received the M.S. degree in telecommunications and PhD in computer sciences from Universitat Politècnica de València, in 2006 and 2012, respectively. He is currently a post-doc researcher at the Universitat Politècnica de Catalunya. His area of expertise include network-on chip, IC design, and chip variability.

Carles Hernández received the M.S. degree in telecommunications and PhD in computer sciences from Universitat Politècnica de València, in 2006 and 2012, respectively. He is currently senior researcher at the Barcelona Supercomputing Center. His area of expertise include network-on chip, reliability-aware processor design, and safety-critical systems. He has participated in NaNoC, parMERASA, and PROXIMA FP7 projects, and VeTeSS ARTEMIS project.

Mario Lodde received his PhD in 2014 at Technical University of Valencia. Research topics covered during his thesis work were cache coherence protocols and their co-design with the network-on-chip. He is currently a post-doctoral fellow at Simula Research Laboratory (Oslo), where he works on fault recovery techniques for large shared memory systems.

José Flich got his PhD in 2001 in Computer Engineering. He is an Associate Professor at UPV. He published over 100 conference and journal papers. Current research activities focus routing, coherency protocols and congestion management within NoCs. He has co-invented different routing strategies, reconfiguration and congestion control mechanisms. He has been the Coordinator of the FP7 NaNoC project.