Network Interface Sharing Techniques for Area Optimized NoC Architectures

Alberto Ferrante, Simone Medardoni, Davide Bertozzi
Engineering Department
University of Ferrara
Via Saragat, 1 44100 Ferrara (Italy)
{alberto.ferrante, simone.medardoni,davide.bertozzi}@unife.it

Abstract

Although preliminary analysis frameworks point out the performance speed-ups achievable by on-chip networks with respect to state-of-the-art interconnects, the area concern remains one of the most daunting challenges to make this interconnect technology mainstream. A common approach to relieve the problem consists of sharing most of network interface resources among a number of processor cores. However, buffering resources need to be replicated and control logic reaches a complexity that limits maximum achievable frequency. This paper proposes full sharing of network interface resources, including buffers, thus trading performance for area. While area improvements are significant, a number of physical and system-level effects might mitigate performance degradation, making our technique a promising solution for area efficient network-on-chip realizations across a range of operating conditions.

1 Introduction

With the advent of multi-processor system-on-chip (MPSoC) technology, computation efficiency can be achieved by combining multiple programmable processors within a multicore system, hence only marginally impacting programmability and configurability. Relevant examples thereof come from commercial products both in the high-performance microprocessor domain [1] and in the embedded computing domain with tighter optimization constraints [2, 3].

Perhaps the most daunting challenge to make MPSoC technology mainstream is to realize the enormous bandwidth capacities and stringent latency requirements when interconnecting a large number of processing cores. This task is on burden of the global intrachip communication infrastructure. Networks-on-Chip are a promising solution for designing scalable communication architectures for MPSoCs, featuring better modularity and design predictability when compared to bus based systems.

Although preliminary analysis frameworks have pointed out the performance enhancements achievable by on-chip networks [4], the area concern for NoC implementation was raised by many works in the open literature. For instance, it was showed in [4] that a NoC architecture for a 30 core system takes one order of magnitude more cell area than that of a state-of-the-art multi-layer interconnect. In terms of floorplan area, the increase can be as large as a few tens of squared millimiters.

It should be observed that in realistic NoC architectures, network interfaces (NI) play a significant role in determining overall NoC area. In [5], a TV companion chip was redesigned with a NoC as the interconnect fabric, and 78% of increase in chip area was proved to come from the NIs. In the *xpipes* based system in [4], more than half of the NoC area is due to NIs. Finally, these network components should come with low area footprint, since the size of IP modules attached to them is relatively small.

Network interface sharing could be a viable option to reduce the area overhead of these components. The most common approach consists of attaching multiple cores to the same NI, thus sharing the same input port to the network (see for instance [6]). This approach involves replication of the buffering resources in the NI, thus leading to an increase of area which hardly justifies this design choice. In fact, trusting published reports, buffers can account for more than 30% of the NI area. Moreover, the replication of buffering resources increases the complexity of buffer control logic (e.g., scheduling), thus resulting in a reduction of the maximum operating frequency or in an increase of NI latency. The problem can be relieved as in [6] by infering custom-made hardware FIFOs at the cost of flexibility. Other approaches delve into the intricacies of NI design trying to reduce its complexity [7]. However, this solution can be successful only up to a certain extent, since the support for processing cores with advanced communication capabilities (e.g., multiple oustanding transactions, out-of-order completion, quality of service guarantees) requires complex NI architectures anyway.

This paper advocates for a more radical solution preserving the flexibility of fully synthesized architectures. We propose a network interface sharing architecture wherein the entire NI resources (including buffering) are shared among a cluster of processing cores or communication targets. We design traffic merging and splitting modules for this purpose, which interleave the different traffic flows on a unique network interface. While potentially resulting in more area efficient implementations, this solution needs to preserve compliance with end-to-end communication protocols linking network interfaces with communicating cores. This paper focuses on the Open Core Protocol (OCP) [25], a standard socket interface for industry-relevant MPSoCs, and proves the feasibility of traffic merging and splitting in this context.

Moreover, this paper shows that by devising proper optimization techniques at the merging and splitting modules, NI sharing outperforms a pure serialization of conflicting network access patterns. Moreover, system-level effects associated with the presence of global slave bottlenecks or

physical-level effects associated with a higher achievable operating frequency (due to a lower switch radix) may mask the further level of arbitration introduced by traffic mergers and splitters. Finally, this work assesses the non-trivial area-performance trade-off spanned by our architecture based on NI sharing in an application-specific as well as in a general purpose scenario.

After reviewing previous work in Section 2, design guidelines for a traffic merger and splitter are analysed in Section 3. Then, architecture details and related optimizations follow in Sections 4 and 5, respectively. Synthesis results are illustrated in Section 6. Basic performance tests follow in Section 7 while a system-level analysis is illustrated in Section 8.

2 Previous work

Most of previous work on NoC area optimization comes from the domain of application specific on-chip networks [8, 9, 10, 12, 11, 14, 15, 16],

When it comes to general purpose NoC architectures, area savings are mainly expected from optimized topologies still preserving structured wiring. An interesting node:router mapping of N:M is proposed in [17], instead of traditional 1:1 (k-ary n-cube) or N:1 (clustered topologies). The work in [18] proves the limitation of 2D-meshes as processor counts increase. It also proposes a concentrated mesh architecture with replicated subnetworks and express channels. The work in [19] trades-off the number of dimensions with the number of cores per router, but lacks of physical insights. Physical implications and feasibility of multi-dimensional and/or concentrated topologies have been investigated in [24].

A method for reducing the hardware complexity of NoCs by automatically configuring the architecture of the NoC switches to suit the application traffic characteristics was presented in [20].

Network interfaces, their architectures and their optimizations have not been addressed extensively in the open literature. Guidelines and trade-offs for network interface design have been summarized in [21].

Network interfaces with low area footprint have been proposed in [22, 23]. The NI hardware complexity reduction coming from aligned packet formats is highlighted in [7].

Perhaps the most complete description of a network interface is that reported in [6], which provides throughput or latency guarantees and supports multiple point-to-point connections. In practice, multiple initiator cores could be connected to the same network interface by exploiting these connections. Two message queues are allocated at NI instantiation time for each point-to-point connection, one for outgoing and one for incoming messages. In practice, the underlying principle consists of replicating the buffering resources of the shared network interface depending on the number of attached cores.

This paper describes original work in sharing network interfaces without any replication of buffering resources. It capitalizes on the concept of traffic merging and splitting and cuts down on the area footprint of NoC architectures. Moreover, the paper shows that through proper optimization techniques traffic merging can be far more effective than a pure serialization of conflicting network access patterns. This is confirmed via system-level analysis with clock cycle accuracy.

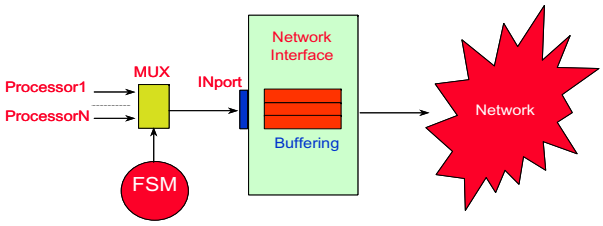

Figure 1. Concept network interface architecture

3 Design requirements

The basic idea of this paper is to allow multiple initiator and/or target cores to share the same network interface resources, as pictorially illustrated in Fig.1. Fig.1 illustrates the concept architecture of a *traffic merger*, which merges traffic flows coming from different initiator cores. A similar scheme can be applied to group a number of target cores, thus allowing them to share a unique network interface. The architecture block required for this purpose is denoted as *traffic splitter* from here on. The main requirements when designing a traffic merger (splitter) are as follows:

- **Preserving compliance to the point-to-point protocol**. Communicating cores are typically wrapped in such a way that they are connected to the network via standard point-to-point communication protocols (e.g., OCP, AXI, DTL). Mergers and splitters have to be inserted on the physical point-to-point link in a transparent way with respect to the protocol, since traffic merging and splitting are handled at a lower level of abstraction than that specified by the standard interface sockets. As an example, when an OCP master interface intends to start a new command but looses arbitration to access the shared network interface, it must not be aware of this and the blocking condition should be signaled as the unavailability of the OCP slave interface to process a new command. In practice, this can be achieved by the merger driving the SCmdAccept signal low for that master until the network interface becomes available. Without lack of generality, the OCP protocol (revision 2.1) will be hereafter used as the reference point-to-point communication protocol.

- **Providing different levels of support for multiple outstanding transactions**. In order to relieve the performance overhead induced by NI sharing, multiple outstanding transactions could be supported. Even when the processor cores do not support multiple outstanding transactions, the cluster of cores sharing the same NI appears as a single core which is capable of them. Obviously, mergers and splitters should support multiple outstanding transactions only if the network interface supports them. Unfortunately, network interfaces usually reflect processor core complexity, and if cores are blocking on read transactions, the corresponding NI will not provide the support for more advanced communication features not to overdesign the system. In this work, we focus on a network interface architecture which supports multiple outstanding write transactions but a single oustanding read transaction. As a consequence, these are also the communication capabilities supported by our mergers and splitters. As will be proved later on, a few simple optimizations are enough to break the serial dependency between conflicting transactions from different processor cores in a cluster.

- **Minimization of area and delay overhead.** Area con-

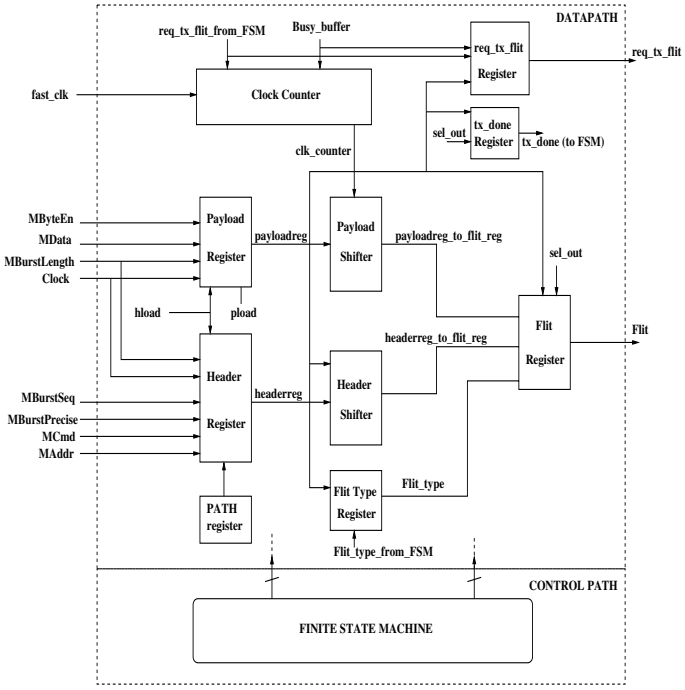

Figure 2. Architecture of the request path in the network interface initiator

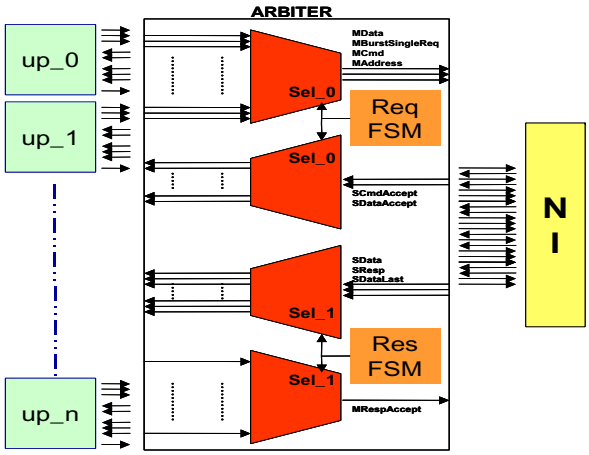

Figure 3. Detailed merger architecture

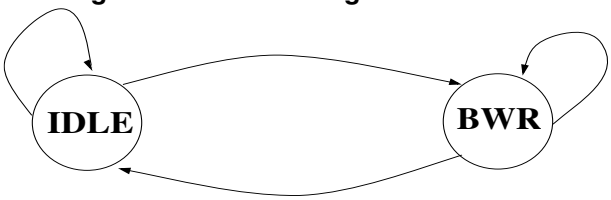

Figure 4. request FSM for the merger module

sumption depends on the level of support for multiple outstanding transactions and on the number of states in the FSM. The requirement is that the area of a traffic merger or splitter be less than that of a network interface, thus making NI sharing cost effective. As Fig.3 indicates, mergers (and splitters) might turn out to be particularly area demanding since they cannot handle the OCP signaling from a processor core as a whole, but they have to handle both the request and the response channel, and inside each channel they have to deal with handshaking signals traveling in opposite directions. Moreover, since mergers break the timing path of point-to-point links, their critical path needs to be carefully designed not to degrade the operating frequency of the processor cores and of the network components.

- *Scalability.* Merger/splitter architecture needs to support a scalable number of attachable cores that will be decided based on system performance analysis (e.g., traffic patterns, real-time requirements).

4 NI sharing architectures

Before dwelving into the architecture details of the proposed merger and splitter, an overview of the network interface architecture that will be used throughout this paper is provided. It is the interface module to the xpipes NoC fabric [27].

4.1 Network interface architecture

The network interface is designed as a bridge between an OCP interface and a NoC switching fabric. Its purposes are the synchronization between OCP and network timing, (de-)packetization, the computation of routing information (stored in a Look-Up Table, LUT) and flit buffering to improve performance. Differentiated bridges exist between communication initiators and the network (network interface initiator) and between communication targets and the network (network interface target).

Each NI is split into two sub-modules: one for the request and one for the response channel. These sub-modules are loosely coupled. Whenever a transaction requiring a response is processed by the request channel, the response channel is notified; whenever the response is received, the request channel is unblocked.

The request path of the NI is built around two registers (Fig.2): one holds the transaction header (1 refresh per OCP transaction), while the second one holds the payload (refreshed at each OCP burst beat). A set of flits encodes the header register, followed by multiple sets of flits encoding a snapshot of the payload register subsequent to a new burst beat. Header and payload content is never allowed to mix, and padding is eventually used. Routing information is attached to the header flit of a packet by checking the transaction address against a LUT. The length of this field depends on maximum switch radix and maximum number of hops in the specific network instance at hand.

The NI performs clock domain crossing, however in order to keep the architecture simple the ratio between network and core clock frequencies needs to be an integer divider.

4.2 Merger architecture

The detailed architecture of the proposed traffic merger is illustrated in Fig.3. Each OCP channel (request and response) is handled by a separate couple of multiplexer/demultiplexer blocks and by a dedicated FSM generating the control signals for the (de-)multiplexing logic.

Control logic has been carefully optimized for minimum complexity, and the resulting FSM is illustrated in Fig.4

(for the request path). The state is IDLE whenever no commands are received on the MCmd lines from the processor cores. However, the state stays the same also when single writes or single or burst reads are performed. This is because for all these transactions only one clock cycle is required by the network interface to store them. In fact, the OCP protocol allows to generate a single address even for long burst transfers, the addressing mechanism being implicit in the kind of burst transaction being executed. In contrast, whenever burst write transactions need to be performed, the merger switches to the BWR state, and the control signals are kept stable until all the write data words have been stored into the NI. Then, the merger goes back to the idle state. In essence, in the idle state an arbitration mechanism is active, discriminating between multiple requests from the processor cores. A fixed priority arbitration is currently supported, even though the extension to round robin is straightforward.

All other state machines (for the response path in the merger, for the request and the response path in the splitter) also exhibit two states and have similar characteristics, and hence are not reported here.

5 Optimizations

The network interface architecture we are targeting supports multiple oustanding write transactions but only one pending read transaction. The support for multiple outgoing reads would require a deep modification of the network interface and is outside the scope of this paper. However, since our on-chip network supports posted writes, multiple posted writes could be performed while a read transaction is pending. The merger might allow several cores with pending OCP write transactions to access the NI while the ongoing high priority core is waiting for response data of a read transaction previously stored by the NI. During this state, other cores with pending OCP read transactions would be kept out of the arbitration even though they had the highest priority. Once response data is stored by the waiting processor core, then the normal arbitration priority is restored. The support for this optimization involves the selector signals for the request and the response paths (Fig.3) to drive different values. The request path selector selects cores with pending OCP writes, which will be serialized on the network interface based on their priority. The response path selector is instead fixed at the value of the processor core with the pending read, since it has to receive incoming data.

Another optimization was implemented in our merger architecture to efficiently utilize the network interface. The original merger architecture features the behavior in Fig.5. Assume that cores 1, 2 and 3 have pending OCP commands. Following arbitration, the command of core 1 is processed. Because of network congestion, the network interface might delay the assertion of the next SCmdAccept. Meanwhile, the MCmd line between the arbiter and the NI is idle since a new arbitration round can take place only on the next SCmdAccept edge. When this latter is finally asserted, it takes a round trip delay before the new MCmd line (the one of processor core 2) can be driven to the NI. In order to avoid this delay overhead, the optimized scheme illustrated in Fig.5 was implemented. Immediately after the transaction of core 1 is stored at the NI, a new arbitration round takes place, so that the OCP signals of the next high priority core immediately drive NI inputs. As a consequence, when SCmdAccept will transition from low to high again, the new command is immediately stored without any additional penalty.

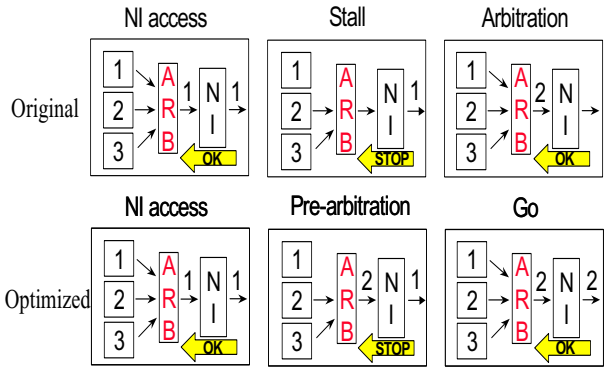

Figure 5. Comparison between original and pre-arbitrated architecture

The inconvenience of this approach is that if core 1 is the highest priority one, it cannot acquire the network interface for two consecutive transactions in case of contention, since the NI will be re-arbitrated to another waiting processor before a new command from core 1 can be driven. However, full exploitation of network resources was our primary design objective.

Similar optimizations were implemented in the FSMs of the traffic splitter. Here multiple target devices share the same network interface target. Again, only one target core will be accessed for a read transaction at a time. However, while a read transaction is taking place, all other target cores can be accessed for write transactions. These optimizations reflect the NI capability to perform one outstanding read transaction but multiple write-after-read transactions. Finally, the same pre-arbitration mechanism was implemented in the splitter, allowing prompt addressing of a new target core.

6 Synthesis results

Merger and splitter were synthesized with Synopsys Physical Compiler [26] (thus accounting for placement effects during logic synthesis) to assess their critical path and area, as well as the scalability of these parameters with an increasing number of clustered cores. A 65nm STMicroelectronics technology library was used.

When synthesizing for maximum performance, the critical path results illustrated in Fig.6-a are obtained for the merger. Timing paths in the request and in the response sections of the merger are quite close to each other, therefore the critical path is in one module or in the other one depending on the number of inputs and on the effectiveness of the synthesis heuristics. In any case, the critical path is always in the control logic and does not involve the datapath. Moreover, the critical path scales quite smoothly for the realistic cases analysed in Fig.6-a. As suggested by [24], realistic target frequencies for the OCP section of the system might be around 500 MHz, while the network could be operated at 1 GHz. By synthesizing the NI initiator under these conditions, the critical path of its OCP frontend is 1.30ns and goes to the SData output. Even assuming that in the response path of the merger the maximum delay of 0.37 ns is incurred (but it will be less than that, because this is the delay measured in the control logic and for the highest number of attached cores), the total delay would amount to 1,67ns.

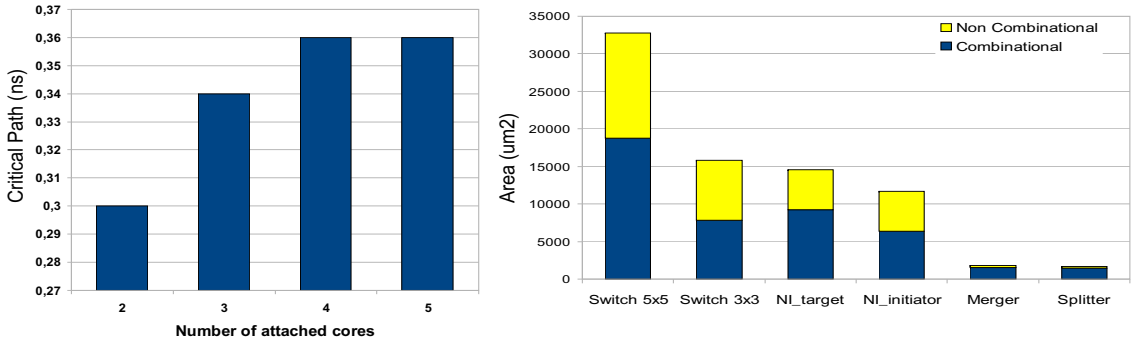

Figure 6. Synthesis results for the merger. (a) Critical path (b) Area

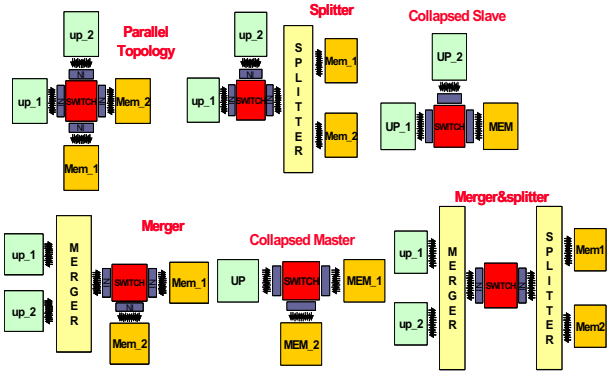

Figure 7. Simple test cases for performance characterization

If we assume that the master OCP interface does not implement combinational dependencies (hence its inputs are latched), the total delay is just increased by a library setup time (around 0.10 ns) and the worst case delay of the OCP section is well below the target 2ns delay in the worst case. As a consequence, for a realistic target operating frequency of high-performance systems, our traffic merger does not degrade the target frequency of the OCP section. Similar considerations hold for the merger.

As regards the area overhead (Fig.6-b), the merger for a cluster of 2 attached cores exhibits an area which is 1/6 that of the corresponding network interface and 1/18 that of a 5x5 switch commonly found in 2D mesh topologies. This paves the way for significant area savings at network level, as will be illustrated in section 8.

7 Basic performance tests

In order to characterize the performance degradation induced by our traffic conditioning blocks and the effectiveness of the proposed optimizations, we set up a few simple test cases which are reported in Fig.7. All test cases were modeled in SystemC with clock cycle accuracy. Microprocessor cores were emulated by means of OCP traffic generators, injecting parameterizable traffic patterns. Parameters include burst length of OCP read and/or write transactions and idle cycles between consecutive bursts. Target cores are instead SystemC models of OCP memories with tunable access latency. Each memory is assumed to be private to a correspondent processor core.

The fully parallel topology (see Fig.7) represents the ideal case: each processor core accesses its private memory without any contention. We compare this topology with the ones featuring a traffic splitter for the memories or a traffic merger for the processor cores or both of them. Finally, we also analysed a collapsed topology wherein the two processor cores sharing the same NI initiator are replaced by a single processor core generating twice the number of transactions of the original cores. This test case will unveil whether using a traffic merger is equivalent to a pure serialization of network accesses or not. Similarly, a topology wherein the two memory cores are replaced by a single memory core with doubled capacity is assessed.

We at first generated a mixed traffic pattern consisting of alternating read and write transactions. Burst length was set to 10 data words (similar results were obtained by varying this parameter), and a total of 500 read and 500 write data words were transferred between each initiator core and its corresponding private memory. Execution time results are reported in Fig.8, and are normalized to the best case. We notice that as the idleness in the system increases, execution times of the different solutions are very similar to each other, except for the collapsed master case, which takes on average two times longer than the parallel topology (as expected). With a lot of idleness in the system, network accesses of the processor cores can be interleaved without any interference, therefore the price to pay for having mergers and splitters is just the delay for the first serialization, which is negligible.

As the number of idle waits decreases, we have a neat performance differentiation. Interestingly, using a splitter does not impact performance significantly. This is due to the support for write-after-read transactions at the splitter, and hence is an effect of our optimizations. Clearly, using a traffic splitter is more convenient than collapsing the two memory cores in a single memory of increased capacity and with 2 split addressing regions.

This trend is even more apparent when we compare the topology with a merger and the one with collapsed processor cores. The former significantly outperforms the latter as the idleness in the system increases, indicating that for realistic network access traces mergers do not vanish the advantages of parallelism. The worst case for the mergers occurs when the processor cores initiate two read transactions at the same time. In fact, the lowest priority one waits for the highest priority one to complete before being processed. In contrast, with only a splitter, such contention occurs closer to the target core, thus improving performance. Finally, the

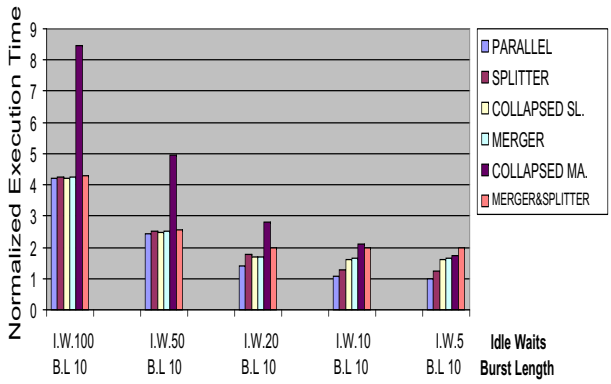

Figure 8. Execution time results for the basic topologies

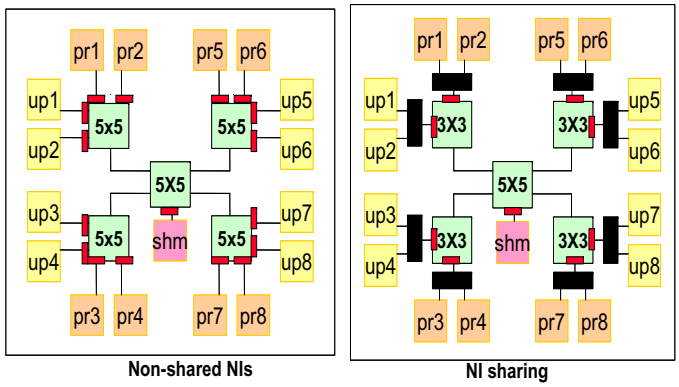

Figure 9. Application-specific topologies under test. uP denotes processor cores, pr private memories and shm the shared memory.

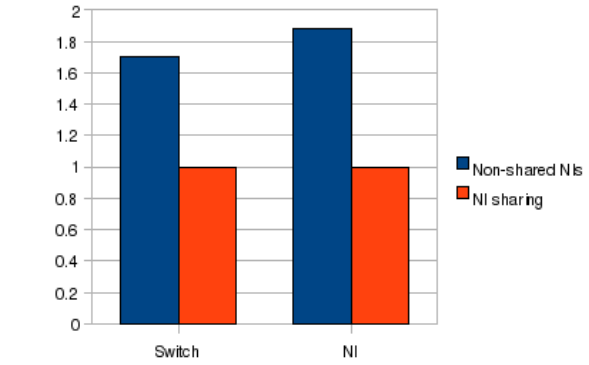

Figure 10. NI and switch area reduction in the topology with NI sharing

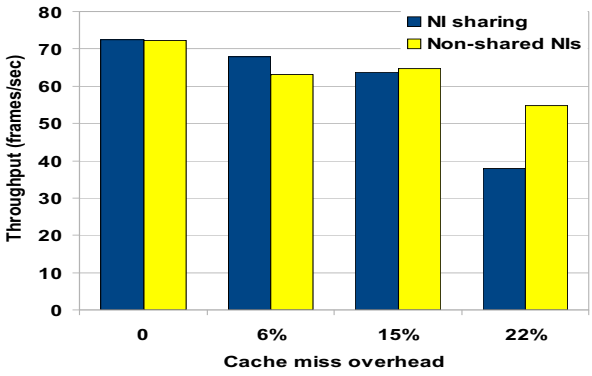

Figure 11. Throughput degradation as a function of cache miss overhead with shared and non-shared NIs

combined utilization of mergers and splitters halves the performance for short idle waits, while it keeps almost the same performance of the parallel architecture from 50 idle waits on.

With a traffic pattern made up of write transactions, we obtain similar results with high levels of idleness. When this latter decreases, all non-parallel architectures exhibit an execution time which is almost two times that of the parallel topology, without any further differentiations. This is an effect of the posted write semantics and of the intensive traffic patterns, which avoid resource under-utilization. Hence, by introducing additional arbitration rounds in the system, parallel writes get simply serialized. Finally, a traffic pattern made up of only read transactions provided execution time results similar to those in Fig.8. This behavior is related to the distance of the congestion point from the target destination.

8 System-level analysis

Let us now take system-level effects into account in determining the area-performance trade-off spanned by architectures with NI sharing. Our scheme can be applied both to application-specific and general purpose NoC topologies, as illustrated hereafter.

8.1 Application-specific topology

We consider the MPEG-4 application as a case study. In particular, we focus on the MPEG-4 decoder Profile High 4:2:2 Level 5, which supports a resolution of 1920x1088 at a frame rate of 72.3 frames/sec. The workload is assumed to be split among 8 parallel processor cores by means of an horizontal slicing technique. We consider a shared memory system, where each processor core reads encoded data from the shared memory, performs its computation (eventually accessing its private memory for cache line refills) and writes decoded data back to shared memory. Based on the specific MPEG-4 profile and resolution and on real-time constraints, we extrapolated read/write communication requirements of the processor cores from/to the shared memory. The number of cache line refills was kept as a parameter of the traffic pattern, and they will end up degrading the nominal throughput in our assumptions. A synthetic traffic pattern reflecting the above scenario was therefore generated by means of the OCP traffic generators. In practice, since we are focusing on video decoding, the amount of write data will be much higher than that of read data. We set the network clock frequency to be twice the frequency of the OCP section, which is a reasonable assumption for the xpipes architecture [24].

Since the traffic pattern is shared memory dominated, we selected the H-star topology suggested in [7] for this

application and placed the shared memory in the central switch, as illustrated in Fig.9-left. Peripheral switches connect processor cores and their associated private memories.The switch radix in all cases is 5. This topology with non-shared NIs was compared with the shared NI topology reported in Fig.9-right. Here clusters of two processor cores and those of two memory cores share the same network interface initiator and target, respectively. Because of the central switch, the maximum switch radix (determining the maximum network clock frequency) does not change. However, the amount of hardware resources changes a lot: switches have lower radix on average, the number of NIs decreases from 17 to 9, with only the addition of 4 mergers and 4 splitters.

Physical synthesis of the two topologies provided the area reports of Fig.10. NI sharing allows an impressive reduction of total NoC area by 41%. In particular, total switch area is reduced by 41.3% while total NI area by 46.7%. Mergers and splitters alltogether account for only 6% of total area of the topology with NI sharing. These results prove the effectiveness of our technique in providing area-efficient NoC realizations.

Performance results are instead illustrated in Fig.11. Nominal throughput is guaranteed for the topology with non-shared NIs in the absence of cache misses. For the same case, NI sharing does not incur any throughput degradation, since there is a strong congestion to access the shared memory, therefore an additional arbitration round at the network boundary does not result in bubbles in shared memory utilization. Moreover, the traffic pattern is dominated by posted write transactions, effectively supported by our mergers and splitters.

We express the impact of cache misses as the percentage stretching of computation time due to cache misses computed as though processor core and memory were directly connected without the network in between. Hence, it is the time needed by a processor core to directly access its private memory for cache line refills. The network latency then introduces an additional and unpredictable overhead.

With an increasing role played by cache misses, the non-shared topology exhibits a progressive throughput degradation, as expected. For 6% and 15% cache miss overheads, we can consider the variability of throughput results as a statistical variation associated with the interleaving of traffic patterns on the network. As the cache miss overhead becomes significant, the impact of NI sharing on throughput becomes apparent.

8.2 General purpose topology

The 2-D mesh is currently the most popular regular topology used for on-chip networks in regular tile-based architectures, because it perfectly matches the 2-D silicon surface. However, this topology shows poor latency scalability and is very area demanding. Concentrated topologies are a straightforward optimization of 2D mesh topologies[18, 24]. They envision the connection of more cores per switch. This way, less switches are required and the average number of hops is reduced at the cost of a lower bandwidth. The main problem with this solution is at the physical layer: a concentrated architecture makes use of switches with a higher radix, which reduces the maximum operating frequency of the network. On one hand, the total number of cycles for the execution of a given benchmark is expected to decrease with respect to a 2D mesh, while

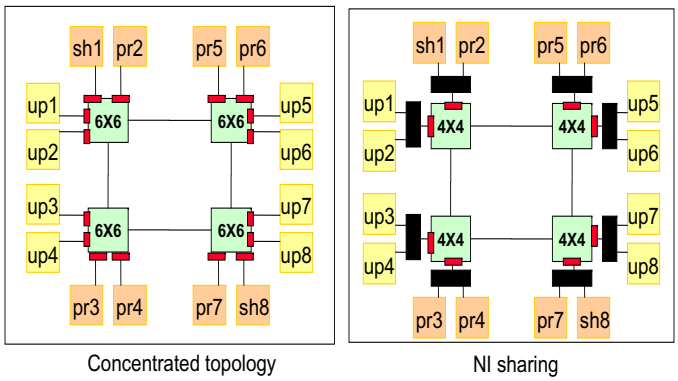

Figure 12. Concentrated topologies without and with NI sharing

on the other hand if we consider the achievable clock frequency the performance is almost the same or even worse than a 2D mesh. See [24] for further details.

The proposed traffic mergers and splitters provide an effective workaround for this problem. In fact, they allow to share the same NI, and hence to keep the switch radix unaltered. This prevents a degradation of the maximum operating frequency. As a case study, we propose the topologies illustrated in Fig.12. A concentrated topology (resulting from the optimization of a 4x4 2D mesh) is compared with a concentrated variant making use of NI sharing. This time, NI sharing reduces the maximum switch radix, and therefore raises the maximum operating frequency from 1 GHz to 1.4 GHz. Since the ratio between network frequency and OCP section frequency was kept to 2, NI sharing allows OCP cores to be operated at 700 MHz instead of 500 MHz. For what follows, we assume that OCP cores can keep up with these speeds.

Both topologies were synthesized for their target frequencies, thus coming up with realistic area numbers. The only exception concerns mergers and splitters, always synthesized for maximum performance not to degrade the frequency of the OCP section. Total area turns out to be reduced by 42%.

As regards performance, we simulated the traffic pattern of a parallel application with a producer-worker-consumer workload allocation policy. One core on one side of the chip (say, core 1) accesses an I/O device and writes computation data to shared memory $sh1$. All other processor cores (from cores 2 to 7) read their computation data from this memory, perform computation and finally write output data to the output shared memory $sh8$ on the opposite side of the chip, from where another processor core (say, core 8) moves them off-chip. This way, a longer number of hops to read input data is counterbalanced by a shorter path to write output data or viceversa. During computation, cores 2 to 7 access their private memories for cache line refills. A number of conflicts arise in the system when NI sharing is used: core 1 write accesses to $sh1$ are in conflict with read accesses of cores 2 to 7 always from $sh1$. There are also conflicts for cache line refills on the splitters serving private memories. Finally, there is a conflict between read accesses of core 8 from $sh8$ and write accesses of cores 2 to 7 always to $sh8$.

Performance results are illustrated in Fig.13. Execution cycle statistics show a negligible 3% degradation of the topology with NI sharing, proving that the merger/splitter

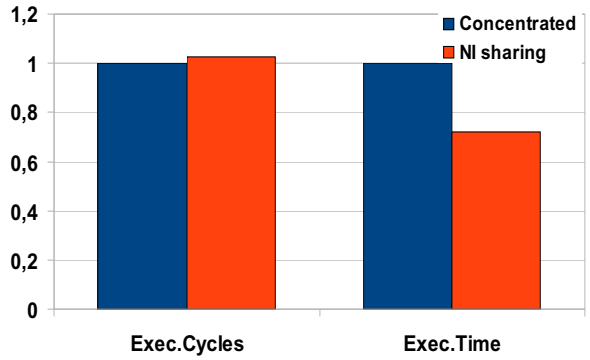

Figure 13. Impact of the lower switch radix on execution time of topologies with NI sharing

architecture optimizations have been quite effective in handling this traffic pattern. However, if we consider the operating frequency boosting that NI sharing enables, we get a 28% improvement of total execution time.

9 Conclusions

In this paper, we propose an area efficient NoC design technique relying on NI sharing. All NI resources are shared by a cluster of initiator or target cores, including buffering resources. This paves the way for significant area savings. Performance degradation of this solution cannot be reduced to a mere serialization of network access requests, in that optimizations implemented at mergers and splitters allow performance of architectures with NI sharing to more closely follow that of fully parallel architectures. Moreover, physical (improvement of maximum operating frequencies) as well as system level effects (centralized slave bottlenecks) play in favour of our solution. Our experimental results prove the effectiveness of NI sharing in reducing NoC area footprint.

Acknowledgment

This work has been partially supported by the GALAXY European Project (FP7-ICT-214364).

References

[1] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way Multithreaded Sparc Processor", IEEE Micro, pp.21-29, 2005.

[2] STn8811A12 Mobile Multimedia Application Processor, available online: http://www.st.com

[3] SPEAr Plus600 dual processor cores, available online: http://www.st.com

[4] F.Angiolini, P.Meloni, S.Carta, L.Benini, L.Raffo, Contrasting a NoC and a traditional interconnect fabric with layout awareness, in DATE, 2006, pp.124-129.

[5] F.Steenhof et al., Networks on Chips for high-end consumer-electronics TV system architectures, DATE, 2006, pp.148-153.

[6] A.Radulescu, J.Dielissen, K.Goossens, E.Rijpkema, P.Wielage, An Efficient On-Chip Network Interface Offering Guaranteed Services, Shared-Memory Abstraction, and Flexible Network Configuration, DATE 2004, pp.873-883.

[7] Kim et al., Solutions for Real Chip Implementation Issues of NoC and Their Application to Memory-Centric NoC, Int. Symp. on Networks-on-Chip, pp.30-39, 2007.

[8] A.Pinto et al., Efficient Synthesis of Networks on Chip, ICCD 2003, pp. 146-150, Oct 2003.

[9] T. Ahonen et al., Topology Optimization for Application Specific Networks on Chip, pp.53-60 Proc. SLIP 04.

[10] K. Srinivasan et al., An Automated Technique for Topology and Route Generation of Application Specific On-Chip Interconnection Networks, pp.231-237 Proc. ICCAD 2005.

[11] J. Hu, R. Marculescu, Application Specific Buffer Space Allocation for Networks on Chip Router Design, pp.354-361 Proc. ICCAD 2004.

[12] U. Y. Ogras, R. Marculescu, Application-Specific Network-on-Chip Architecture Customization via Long-Range Link Insertion, pp.246-253 Proc. ICCAD 2005.

[13] G.Palermo, C.Silvano, G.Mariani, R.Locatelli, M.Coppola, Application-Specific Topology Design Customization for STNoC, DSD 2007, pp.547-550.

[14] U. Ogras and R. Marculescu, Energy and Performance Driven NoC Communication Architecture Synthesis Using A Decomposition Approach, pp.352-357 DATE 2005.

[15] S. Murali, P. Meloni, F. Angiolini, D. Atienza, S. Carta, L. Benini, G. De Micheli, and L. Raffo, Designing Application-Specific Networks on Chips with Floorplan Information, in International Conference on Computer-Aided Design (ICCAD), pp. 355-362, 2006.

[16] K. Srinivasan, K.S. Chatha, and G. Konjevod, Linear Programming based Techniques for Synthesis of Network-on-Chip Architectures, IEEE Transactions on VLSI, 14(4):407-420, 2006.

[17] Dongkook Park Nicopoulos, C. Jongman Kim Vijaykrishnan, N. Das, C.R., "A Distributed Multi-Point Network Interface for Low-Latency, Deadlock-Free On-Chip Interconnects", International Conference on Nano-Networks and Workshops, On page(s): 1-6, 2006.

[18] Balfour, J., Dally, W.J., "Design Tradeoffs for Tiled CMP On-Chip Networks", ACM International Conference on Supercomputing, pp.187-198 2006.

[19] Gilabert, F., Gomez, M.E., Lopez, P.J., "Performance Analysis of Multidimensional Topologies for NoC", ACACES 2007, poster session with proceedings at the Hipeac Summer School.

[20] P.Meloni, S.Murali, S.Carta, M.Camplani, L.Raffo, G.De Micheli, "Routing Aware Switch Hardware Customization for Networks on Chips", Int. Conf. on Nano-Networks and Workshops, pp.1-5, 2006.

[21] Bertozzi D., "Network Interface Architecture and Design Issues", book chapter from "Networks on Chips: Technology and Tools", edited by Benini L., G.De Micheli, Morgan Kaufmann, 2006.

[22] P. Bhojwani and R. Mahapatra, "Interfacing cores with on-chip packet-switched networks", In Proc. VLSI Design, pp.382-387 2003.

[23] C. A. Zeferino, M. E. Kreutz, L. Carro, and A. A. Susin., "A study on communication issues for systems-on-chip", In Proc. SBCCI,pp.121-126 2002.

[24] F. Gilabert et al., "Exploring High-Dimensional Topologies for NoC Design Through an Integrated Analysis and Synthesis Framework", Network-on-Chip Symposium, pp.107-116, 2008.

[25] OCP protocol specification, release 2.1.

[26] Synopsys Inc., Physical Compiler, www.synopsys.org

[27] S.Stergiou et al., "Xpipes Lite: a Synthesis Oriented Design Library for Networks on Chips", DAC, pp.559-564, 2005.