UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

HoL-blocking avoidance in on-chip networks

Master thesis in Computing Engineering
Year 2011/2012

Departamento de Informática de Sistemas y Computadores

José Vicente Escamilla López

Director:
José Flich Cardo

September, 2012

Contents

Chapter 1

Introduction and Motivation

The high-performance computing domain is taking advantage of the inclusion of multicore solutions in the form of Chip Multiprocessor (CMP) [NO97] and System-on-Chip (MPSoC) [WJM08] systems. As the integration scale goes further, more cores, nodes, or processing units are included in the same chip. Examples of the many-core integrated CMPs are the products developed by Intel inside its Tera-scale Computing Research Program such as two prototype chips: The Teraflops Research Chip [Corb] with 80 cores, and the single chip cloud computer (SCC) [Cora]. Tilera products, like the Tile-GX [Corc] with up to 100 cores, represents a good example for a high-end MPSoC system. These systems provide support for the specific needs of the different applications to be run including multimedia, wireless networking, and cloud-computing applications.

Both design platforms, CMPs and MPSoCs, rely on an on-chip interconnection network infrastructure that provides the communication between all the processing nodes. This must be a high-bandwidth, low-latency network to avoid slowing down processors while waiting for remote data. Networks-on-chip (NoCs) suit well when a large number of processing nodes are present [FB10], as is the case of the Intel prototypes and Tilera products. In general, NoC design is challenging due to the tight constraints found in on-chip systems. Thus, an NoC must be simple in its mechanisms, exhibiting low hardware overhead, low power-demanding, and at the same time being performance-efficient, independently of the applications running on the system.

As technology advances, specialization will drive the inclusion of dedicated devices as accelerators, encoders, DMA devices, etc. This heterogeneity [KTJR05] and the increasing number of components will drive a change in the traffic present in those devices. We can expect bursts of traffic flowing from device to device, at intermitent and unpredictable frequencies. This kind of traffic may create temporary hotspots where traffic is concentrated, thereby leading to the appearance of network congestion that is likely to have a negative impact on the rest of the traffic. Efficiently dealing with the problems derived from congestion can significantly improve the overall chip performance. This master thesis focuses on the topic of congestion effects avoidance in future manycore systems.

In addition, power-saving in CMPs and MPSoCs is becoming an important design factor. Designing energy efficient devices provides several benefits such as less heat dissipation which minimizes electronics failures and increases batteries life for hand-

held devices, an essential property for devices like smartphones. As the number of interconnected devices in a CMP or MPSoCs grows the network must grow in accordance. This growth leads to more energy consumption and increases the energy proportion consumed by the network. For example, the Alpha 21364 NoC consumes about 20% [MBL+01] of the total power consumption. Several works have been carried out in order to decrease the energy consumption of the network but all techniques imply performance loss to a greater or lesser extent in the NoC. A very extended and effective technique for power-saving consists in decreasing voltage and frequency of the network (DVFS) when workload is low. This technique is very effective in reducing power consumption however, reducing voltage and frequency can lead to a performance penalty due to congestion. Even when the clock is decreased at the correct momment, if the workload is quickly increased the frequency needs to be rapidly increased due to the high demmand and this may take some time for increasing and stablilize the clock. During this time lapse, network is unusable and congestion quickly appears. For example, in [CXK+12] authors propose to use AMAT metric (Average Memory Access Time) in order to compute DVFS levels for power-saving. In such proposal the network may congest. However, in this proposal only one DVFS domain is used. In most cases DVFS is used with multiple domains (VFI, Voltage-Frequency Island). Each domain contains one or more adjacent cores and their switches. In [YLH+12] an hybrid regulator scheme is proposed in order to satisfy different needs depending on the application being executed. For applications which need quick and frequent DVFS adjustments (fine-grain) on-chip regulators are used since on-chip regulators are quick but power-limited. Accordingly, for applications which need less frequent DVFS adjustments (coarse-grain) off-chip regulators are used since, despite of such regulators are very slow, they are no power-limited so can control several VFIs. In any case, the network may suffer traffic peaks as the off-chip regulator is too slow to put the network at its maximum frequency at time, leading to congestion. In addition, with multiple-domain DVFS, there are adjacent VFI working at different clocks so, if cores whitin a VFI working at high frequency need to send data to or through another VFI domain working at a slow clock frequency, the border-switch of the slow VFI that receives this high data rate flow will get congested. There is an example in Figure 1.1 where switch 6 must adapt both speeds thus creating a congestion spot and impacting performance.

Congestion is not a problem by itself. This view of the problem is recently new [GQF+06] and opposed to the common belief where efforts were directed towards detecting and eliminating congestion spots or even avoiding the appearance of congestion spots. The real problem that impacts performance is the Head-of-line blocking effect that congestion spots cause. Head-of-line blocking (HoL blocking) occurs when a packet at the head of a buffer is blocked (because its requested output port/resource is unavailable, that is, congested) and this packet blocks other packets allocated in the same queue, even if those packets request free output ports/resources. If HoL blocking originated from congestion spots is removed, then the congestion becomes harmless and there is no need to remove the congestion spot. In this master thesis we follow the same approach and focus on head-of-line blocking effects.

To summarize, power-saving techniques are necessary mechanisms and an important design trend nowadays, but their use may cause congestion problems that need to be adressed. For that purpose, in this master thesis we present two evolved mechanisms to

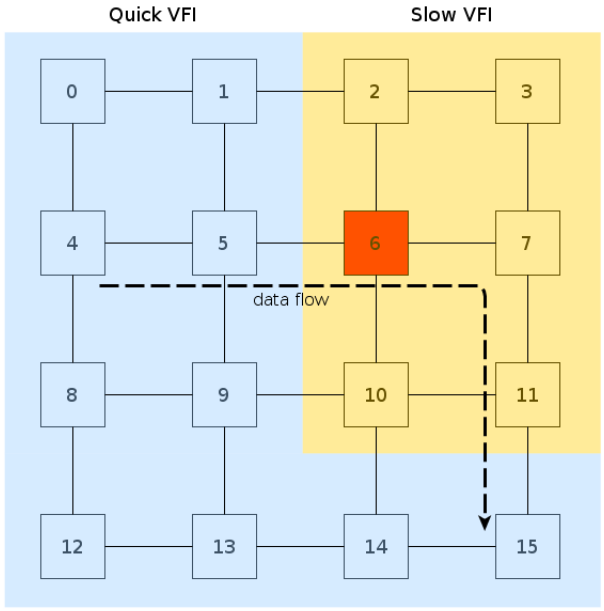


Figure 1.1: Congestion in switch 6 caused by a high data rate flow traversing an slow clock VFI

countermeasure the side effects of congestion. In particular, we propose BAHIA (Burst-Aware HoL-blocking Injection Avoidance) and ICARO (Internal Congestion Aware hol-blocking RemOval), two mechanisms that dynamically detect bursty and congested traffic respectively in the network, then isolating them and thus guaranteeing that non-harmful traffic is unaffected. All these proposals are made by taking into account the strict limitations in area, power, and delay imposed by the special environment of on-chip networks.

1.1 Contributions

In this master thesis two different mechanism are proposed in order to face harmful effects (HoL-blocking) derived from the excess of traffic in NoCs. Both mechanism are based on the same approach which consists in separating harmful traffic from the non-harmful one making use of virtual network queues. On the one hand a mechanism for avoiding harmful effects caused by bursty traffic is proposed (BAHIA). In this mechanism end-nodes receiving traffic detect incoming bursts for later isolating harmful traffic into a special virtual network keeping the no-bursty one in the regular virtual network queues. On the other hand another mechanism is proposed for facing congestion (ICARO). In this mechanism congestion is detected at network switches composing the network. Later, end nodes are notified and thus react by isolating traffic belonging to the congestion trees into the special virtual network queue.

In this master thesis we propose the basic mechanisms that will be used for dealing congestion scenarios in challenging environments, specially in those where multiple domains of voltage and frecuency islands will be common. In this document, however,

we do not evaluate the mechanisms in such scenarios. This is left for future work.

1.2 Work Structure

The rest of the master thesis is organized as follows: Chapter 2 presents the state-of-the-art. In Chapter 3 the solution for facing bursty traffic is described and evaluated. Accordingly, in Chapter 4 the ICARO method is introduced and evaluated. Finally, in Chapter 5 conclusions are presented. In addition, publications derived from this work are enumerated.

Chapter 2

Related Work

There is a number of solutions in [IZG+07], [DMMD09], and [GKM09] focused on reducing the negative effects of resource sharing through quality of service (QoS) policies and mechanisms, based on priority schemes. Although all these solutions can alleviate or delay the negative impact of network congestion by prioritizing different traffic types, their main objective is to differentiate traffic and they are not actually focused on dealing with congestion itself. As a consequence, congestion may appear within each traffic class due to unpredictable traffic patterns.

In [BM09] authors first make an analysis of the impact of resource sharing with different traffic patterns and the implication of dependences between packets of the same data flow in the efficient utilization of these resources. Finally authors propose to change the abstraction unit from mapping packets to virtual channels to mapping flows to virtual channels and follow a destination-based mapping policy.

Regarding specific congestion-control mechanisms, the authors in [NFM+11] present a mechanism for congestion control and its relation to the system scalability for buffer-less on-chip networks. As the authors describe, their experiments show congestion problems in this kind of networks (system size ranges from 16 to 4096 nodes), thus demonstrating the need for a congestion control mechanism to prevent performance degradation. However, as this work focuses on buffer-less networks, the results cannot be directly applied to the buffered NoCs we consider in our research. The main corpus of congestion-control solutions on buffered NoCs are based on the idea of congestion-awareness mechanisms implemented on the switches, either based on deterministic or adaptive routing. The solutions presented by the authors in [GGK08], [WAHS06], [MRN+05], and [LZJ06], describe mechanisms that collect congestion information from the neighbour nodes through the routing process and ingress/egress buffer monitoring, in order to offer an alternative path to route around a congested area. However, this strategy may end up producing more congested resources, as it is impossible to avoid the congested region if all the congested traffic has the same target (e.g. the memory controller).

In the case of high-end MPSoCs, all the previous congestion methods can be effectively used as well. However, these methods do not specially deal with traffic bursts. Bursty traffic has been analyzed and explored within the concept of QoS especially when addressing how QoS application requirements (latency, jitter, and bandwidth) can be met within the network. Here we are not interested in the QoS aspect of the

traffic, but on the congestion effects that uncontrolled bursts may create. Indeed, in [ODH+07] and [MOP+09] the importance of bursty traffic in the congestion control framework is pointed out. In that sense, the BAHIA method described in the next section addresses the negative effects of bursty traffic.

In [BM06] the problems derived from bursty traffic are addressed by increasing buffer size in order to get room enough to absorb bursts. This approach is relatively expensive in terms of silicon area and power, and as reported in the article, with non-bursty traffic this results in a suboptimal utilization of the resources. In addition, it is difficult to predict burst sizes, hence probably a burst may overflow buffers, then resulting in contention. In BAHIA, however, buffer size is not modified as bursty traffic is separated at the sources.

In [MJ09] a solution to reduce the latency in worst-case bursty traffic is proposed. However, this mechanism is based on temporarily ejecting packets and later re-injecting them with a priority-based approach. This achieves good results as it helps in the worst case, but at the cost of increasing latency of newer packets. Moreover, if we consider a scenario with several virtual networks, this mechanism requires three queues per virtual network at each node, thus becoming an expensive solution in an NoC context.

There is a plethora of publications for congestion management in off-chip networks, and a increasing number for on-chip networks. However, as stated previously, our approach to deal with congestion is different and relatively new. The idea is, instead of detecting congestion, notifying sources, and then removing congestion (by halting injection), we follow the approach of separating congestion by using dedicated resources. Indeed, our approach followed in ICARO is the application of the RECN mechanism that was proposed for off-chip networks [GQF+06]. However, due to the tight limitations in area and power in on-chip networks, we need to follow a different way of implementing it. This is the main property of ICARO in comparison with RECN.

Chapter 3

BAHIA

3.1 BAHIA Description

BAHIA (Burst Aware HoL-blocking Injection Avoidance) provides a method to, at runtime, isolate detected bursty traffic in a network, in order to prevent bursts from causing HoL-blocking. Detection of bursty traffic is performed at any end-node receiving a burst. All the end-nodes are then notified of this detection, so that thereafter the bursty traffic can be identified in order to be separated from non-bursty traffic, thereby avoiding the HoL-blocking that the former could produce to the latter. BAHIA makes use of virtual networks to separate traffic, hence BAHIA requires at least two virtual networks: the "default" virtual network, for non-bursty traffic, and an extra virtual network for bursty traffic. Therefore, if no bursty traffic is detected, the traffic is injected always through the default virtual network, but when a burst is detected, the bursty flow is mapped to the extra virtual network.

Note that virtual networks can be implemented with virtual channels, with no need to increment the number of switch ports. Also, both virtual networks can be used for performance reasons while no bursty traffic is detected. This is left to future work.

3.1.1 Burst Detection

As mentioned above, the detection of bursty traffic is performed at the end-node receiving the burst. For that purpose, each end-node periodically calculates its rate of received traffic. The traffic rate is calculated every "polling interval" (PI) cycles, which is a predefined parameter of BAHIA. If that rate exceeds a given high-threshold (HT) value, this end-node will notify the other end-nodes that it is receiving bursty traffic. Similarly, any end-node notifying bursty traffic must be able to detect the end of the burst. For that purpose, the traffic reception rate is compared with a given low-threshold (LT) value. Accordingly, in this case, all the end-nodes will be notified of the end of the burst. It is worth mentioning that an appropriate configuration of the two aforementioned thresholds (upper and lower; HT and LT) is important to achieve the best BAHIA performance. Indeed, in our evaluation experiments we have thoroughly tuned these values, as explained in Section 3.2.

On the other hand, an alternative burst detection mechanism could be conceived at the sender. Indeed, if a node is going to inject a burst then it can know that in

advance. However, detecting at the senders prevents the effective detection of different lightweight bursts from different sources to the same destination, thereby being not so efficient in preventing the negative effects of bursts. Hence, we opted for performing burst detection at the receiver part of the end-nodes.

3.1.2 Burst Notification

In order to notify of a traffic burst to all the end-nodes, BAHIA makes use of a simple dedicated signaling network (Burst Notification Network, BNN). Basically, the whole BNN is a set of one-bit-wide overlapped control networks, each one managed by a specific end-node. Each control network bit connects its manager end-node to the rest of end-nodes, the former being the only one able to activate/deactivate the signal (i.e. to set the bit to one/zero) of this control network, while the latter being just signal receivers. Thus, every end-node owns an exclusive one-bit-wide control network to notify bursts events to the rest of end-nodes. Figure 3.1 shows the one-bit-wide control network managed by end-node 0, but note that every end-node owns a similar, independent one-bit-wide control network to send notifications to the rest of nodes, so that in this 4x4 mesh network, there would be other 15 one-bit-wide control networks besides the one shown. The overlapping of these independent control networks allows every end-node to notify bursts without risk of collisions with notifications from other end-nodes. Therefore, the BNN can be viewed as an N-bit-link, where N is the number of end-nodes and every bit (wire) in the link corresponds to a specific end-node in the system.

Any end-node notifies the rest of end-nodes of a burst by setting to high value the signal of its BNN line. Due to the simplicity of that signal, it reaches all the end-nodes in a few cycles. The time spent in propagating and processing this signal is modeled by the "notification delay" (ND) parameter in the simulations. It is worth mentioning that the processing of this signal is negligible (and so the required hardware). Regarding the area overhead introduced by the BNN, in [GLFB11] an additional dual-network for routing data transmission is propossed. This dual-network comprises needed logic for coding and decoding data, flow controlling, handling transmission failures mechanisms, etc. Despite of its relative complexity, authors conclude that the area overhead of the hardware required to implement their proposal is 12.5% of the switch. Basically, the BNN consists in a set of wires which are set to a high or low signal value for notifying, hence no logic for processing data is needed. Therefore, relying on the hardware overhead study performed in [GLFB11], we can conclude that the hardware overhead for implementing the BNN is negligible. Notice also that the BNN network does not implement any flow control mechanism. It consists of a set of wires with possibly some repeaters. As working plan we will evaluate the impact on area on a real switch design, in order to confirm its negligible area overhead.

3.1.3 Traffic Separation

Every end-node implements a "burstiness bit-vector", each bit corresponding to a specific end-node in the network, as can be seen in Figure 3.2. When an end-node detects a high signal value in the BNN line associated to an end-node, the former

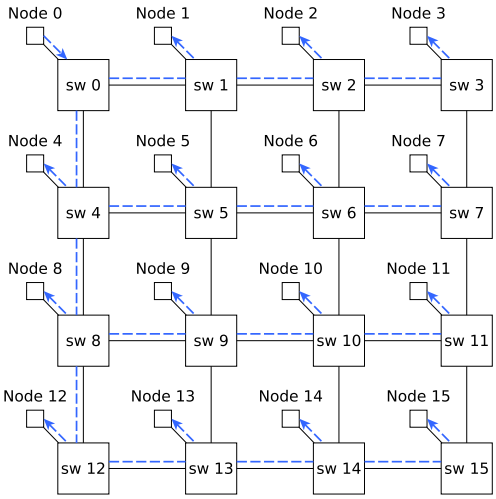


Figure 3.1: Node 0 communicates burst events through this 1-bit wire

will set to one in its burstiness bit-vector the bit corresponding to the latter. All the messages are mapped to the default virtual network once generated, but in every clock cycle the head of all virtual network queues but the extra one are checked searching for the head of a message. If a head is found, the destination of the message is checked to obtain the value of the bit in the burstiness bit-vector corresponding to the destination end-node of the message. If that bit is set to one, the whole message is transferred to the extra virtual network. The process of checking the virtual network queues and transferring to the extra virtual network is performed in parallel and can be executed while reading from the queue currently selected for injecting, hence a message stored in the head of a queue can be transferred to the extra virtual network queue while the next one is injected (if it must not be transferred to the extra virtual network too). Obviously, messages addressed to an end-node whose associated bit in the burstiness bit-vector is set to zero, will remain mapped to the default virtual network. It is worth mentioning that although the proposal uses the burstiness bit-vector, in a real hardware implementation this structure can be removed by simply inspecting the signals in the BNN lines.

Figure 3.2 shows the basic structure of the sender part of an end-node that has messages addressed to end-nodes 1, 5 and 6. The message at the header in that queue is addressed to end-node 5. As can be seen, in the burstiness bit-vector, the bit corresponding to end-node 5 is set to one (i.e. end-node 5 previously notified that it was receiving a burst), so this message must be mapped to the extra virtual network. Indeed, before injection, the arbiter at the sender node checks the burstiness bit-vector and transfers the message addressed to end-node 5 to the queue associated to the extra virtual network, so that this message will be later injected from that queue.

It is worth pointing out that, although messages are injected either from the default queue or from the extra one, all of them are initially mapped to the default VN. This is because, if an end-node directly maps to the extra VN the messages addressed to an end-node that has recently notified a burst, there may be messages still stored in the default queue that are addressed to the same destination, and this could introduce

out-of-order message injection (and so delivery) as queue selection policy is based on a simple round-robin algorithm. Hence, to guarantee in-order message injection and delivery, all the messages are first mapped to the default VN and the arbiter is provided with some additional intelligence to check the burstiness bit-vector, in order to evaluate whether a message should be directly injected from the default queue or, on the contrary, it should be transferred (by changing pointers) to the extra virtual network.

Once an end-node notifies that it is no longer receiving bursty traffic (by setting to low value the signal of its BNN line), the other end-nodes will reset the corresponding bit in their burstiness bit-vector. Thereafter, new messages addressed to this end-node will be injected from the queue associated to the default virtual network. However, note that this may also introduce out-of-order message injection and delivery, as messages addressed to the end-node may remain in the extra queue. The example of Figure 3.2 also shows a situation where there are messages in the extra queue addressed to an end-node (specifically, end-node 0) whose associated bit in the burstiness bit-vector has changed from one to zero.

In these cases, in-order packet delivery can be preserved if messages addressed to a specific end-node are injected from the default queue only if there are no messages addressed to the same destination in the extra queue; otherwise, the packet must be transferred from the default queue to the extra one. However, this implies that other information than that of the burstiness bit-vector is necessary, besides some additional tasks for the arbiter. Specifically, every end-node in BAHIA implements a presence vector. This vector contains an element per end-node in the network and every element is a counter indicating how many messages adressed to this end-node are stored in the extra queue. Every counter is incremented each time a message addressed to the corresponding end-node is moved from the default VN to the extra one, and decremented when messages are injected to that end-node from the extra VN. When a message reaches the head of the default queue and the bit associated with its destination in the burstiness bit-vector is set to zero, the counter associated with that destination in the presence vector is also inspected: if the value of that counter is zero, the message is injected from the default queue; otherwise, the message is moved to the extra queue.

The whole mechanism necessary to keep in-order message injection is a post-processing mechanism, in the sense that messages are mapped to their final VN once they reach the head of the default queue, and not before. As mentioned above, the arbiter should be in charge of performing this post-processing mechanism, that can be summarized in the next pseudocode:

```
for each default_vn in the_end_node {
        if(!isVNempty(default_vn)) {
                msg=getHeadMsgFromVN(default_vn);
                if(isNodeReceivingBurst(msg.destination) ||
                   numFlitsInExtraVN(msg.destination) > 0)
                        moveMessageToExtraVN(msg);
        }
}
```

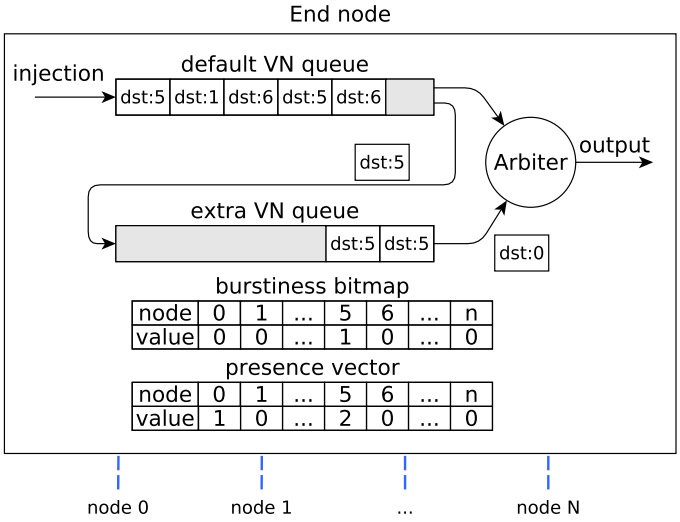


Figure 3.2: Flow followed by messages in a node

Note that postprocessing can be performed in parallel of the transmission of a message, either from the default queue or from the extra queue. Every switch in the network will also have two different queues (virtual channels) per input port, and messages injected from the default VN will be mapped on the first queue at switches, while messages injected from the extra VN queue will be mapped on the second queue at switches, thus avoiding HoL blocking effects from detected bursty traffic to non-bursty traffic. In the next section we evaluate the basic mechanism.

3.2 Evaluation

In this section we present an evaluation of BAHIA based on the results of simulation experiments performed in bursty traffic scenarios. Notice that in a non-bursty scenario there will be no impact on performance, since the BAHIA mechanism will not activate. First, we describe the MCSL [LXW+11] traffic pattern used for our evaluations. Then, we describe the simulation environment. Third, we offer a robustness analysis of the diferent parameters that define the BAHIA behavior. To carry this out we stablish a baseline configuration, then running simulations with different values of each parameter, in order to find out the parameters of the configuration that optimizes BAHIA performance. Next, we compare the results obtained with and without BAHIA in the same scenario. Finally, we perform an analysis of the influence of the number of virtual networks over BAHIA and the performance improvement regarding similar no-BAHIA scenarios.

In order to better appreciate the BAHIA effect on the traffic, results are shown in two graphs: one containing default virtual network(s) data and one containing the extra virtual network data. Notice that we do not target overall throughput increase or overall latency reduction. Our aim is to separate bursty traffic from non-bursty one, thereby keeping the latter unaffected by the HoL-blocking the former may produce.

3.2.1 MCSL Traffic Traces

One of the main problems that arises when exploring new techniques for on-chip-networks is the traffic pattern employed for simulating the network behaviour. In order to evaluate new implemented techniques in the on-chip-network is of outmost importance to employ traffic patterns that were as closer to common real traffic as possible. For simulating traffic patterns is common the use of synthetic traffic patterns, like uniform, bit-reversal and hotspot. However, such patterns may become non-realistic and may generate worst-case only simulation results since real applications in real hardware generate very different traffic patterns. A way for simulating more realistic scenarios consists in employing real traffic traces extracted from cycle-accurate processor simulators. These simulators execute intruction by instruction real binary programs, model the memory accesses and generate files containing these memory accesess. Then, the NoC simulator, with these files is able to accurately simulate the network transactions. This technique is very accurate but is heavyweight and requires a huge amount of time to perform simulations.

In [LXW+11] authors have developed an interesting benchmark generation methodology for generating realistic traffic traces. These traffic traces are much more lightweight than real traffic traces extracted from processor simulators. In addition, in [LXW+11] authors demonstrate that despite their lightweighting, these traces generate traffic patterns corresponding to real applications very closer to their real traffic. The main advantage of these traffic traces consists in, since traces are very lightweight, NoC simulations run very quickly and accurately, thus, increasing development speed in on-chip-networks research.

For evaluations in this master thesis we have decided to employ MCSL traffic traces due to its clear advantages. MCSL authors provide with traffic traces files for different NoCs architectures and applications but adapting the NoC simulator for reading these files and controling traffic in the NoC according to the data contained in such files is left for the NoC simulator developer. Since in our experiments we make use of our own NoC simulator we have to carry out all modifications in order to get it work with MCSL traffic trace files.

MCSL traffic traces are structured in tasks and edges. Every task specifies a task id, the node in which its executed, the order regarding other tasks executed in the same node and its duration. On the other hand, every edge specifies an edge id, source task id, destination task id, the size of the data, a source memory address and a destination memory address (explained later).

With this information the parser developed for our NoC simulator constructs a list of tasks for every node ordered by its execution order. Every task has associated its input and output edges in order to know which data must receive before starts executing (input edges) and which nodes has to delivery data to at the end of its execution. By doing this, we model data dependencies between processes.

Every edge represents a transmission between a source task and a destination task. By the benchmark design methodology, each edge owns two memory addresses, a local memory address that belongs to the source node and a remote memory address at the destination node. When a task starts it consumes the required data from its local memory. If this data is not available, task should wait until data is received. Accordingly, when a task finishes, it may generate data to be consumed by another

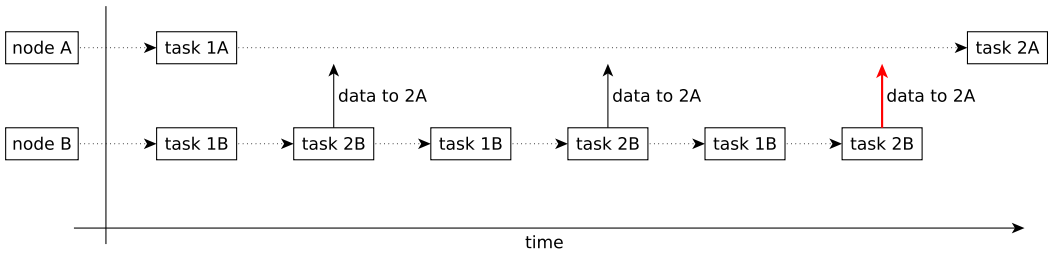


Figure 3.3: MCSL dependences transmission synchronization problem.

task. However, this approach arises a synchronization problem between tasks.

In the exmaple of Figure 3.3, we see the case for 2 nodes executing different tasks. Tasks in node A are very low tasks while in node B tasks are more lightweight, so they are executed quickly. By the application structure, in node B the same 2 tasks are executed alternatively so task 2B sends data to be consumed by task 2A three times to the same memory address before task 2A consumes any data. Since each edge has two memory addresses in order to write data to be consumed by the destination task (one in the remote memory and one more in the local memory), the first two executions of task 2B has no problems. The first execution writes its data in the remote address at node A memory space and the second time data is stored in the local memory space of the edge. However, the third time task 2B tries to write data it has no free memory space to write into, so we have to deal whith this problem. To solve the problem described we have implemented a mechanism for halting tasks that has exhausted all memory space for any of its outogoing edges and resume its execution when it is ready for sending more data.

The mechanism implemented for avoiding the described problem briefly consists in the following. When a task spends all of the memory space of any of its edges the task is halted. Then, when a destination task consumes the data contained in its local memory, it notifies to the source node of such data through a special message (Memory Slot Release, MSR) that contains the memory address released (that identifies the edge and, thus, the local memory address of the sender node associated to this edge). When source task received a MSR notification it sends the data contained in the local memory address of the edge to the remote memory address, releasing the local address for storing more data and resuming the task halted for this edge. By doing this we satisfy all dependences and achieve to finish succesfully simulations for MCSL traffic traces files obtaining the benefits of using these realistic traffic patterns derived from real applications like the traffic patterns corresponding to the H264 codec used in our evaluations.

3.2.2 Simulation Environment

An NoC simulator has been used for the experiments. Results are shown every 5000 simulated cycles. The network topology modeled in all the experiments is a 2D mesh built from 16 switches arranged in a 4x4 mesh distribution, each switch being connected

to a single end-node. Regarding the traffic pattern, traffic is generated from extrapoled MCSL recorded traffic traces of the H264 video encoder [LXW+11]. Since our goal is to isolate HoL-blocking generated by the traffic burst of the video encoder processes, in addition to this realistic traffic pattern a synthetic background traffic is injected. This synthetic traffic consists in all nodes injecting at a data rate of 0.3 flits/cycle. Table 3.2 summarizes the configuration of the scenario modeled in our experiments.

	HT (flits/cycle)	LT (flits/cycle)	PI (cycles)	ND (cycles)
baseline	0.45	0.35	1000	2
HT analysis	0.40, 0.45 0.6, 0.75, 0.9	0.35	1000	2
LT analysis	0.45	0.25, 0.35, 0.4	1000	2
PI analysis	0.45	0.35	100, 200, 400, 800, 1600	2
ND analysis	0.45	0.35	1000	1, 2, 4, 8, 16

Table 3.1: BAHIA robustness analysis configuration

Topology	4x4 2D regular mesh	
Virtual networks	no BAHIA	2VN
	BAHIA	1 default VN + 1 extra VN
Switching technique	Wormhole	
Flow control	Stop&go	
Flit size	4 bytes	
Switch queue size	16 flits	
Traffic pattern	h264 video enc. + 0.3 flits/cycle	
Synthethic messages size	10 flits	

Table 3.2: Scenario configuration for bursty traffic

3.2.3 Parameters Tuning

BAHIA behaviour is defined by four parameters: BNN notification delay (ND), high-threshold (HT), low-threshold (LT) and polling interval (PI). In order to explore the robustness of the mechanism, we carry out an analysis of every parameter by independently simulating different variations of the parameters.

The baseline configuration, and the different values of the parameters used for the robustness analysis of BAHIA are shown in Table 3.1.

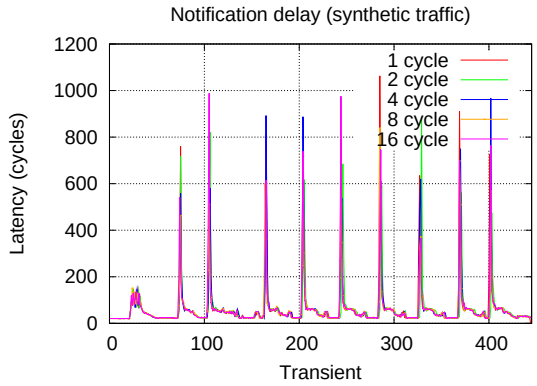


Figure 3.4: Notification delay analysis.

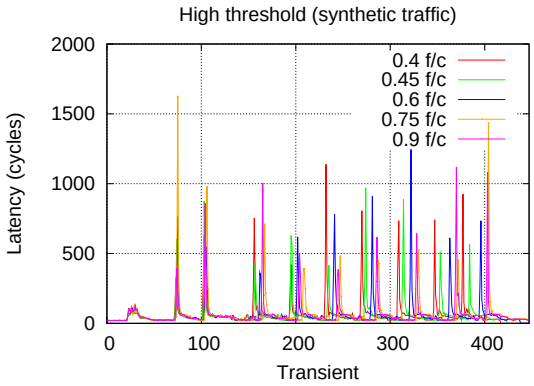


Figure 3.5: High-threshold analysis.

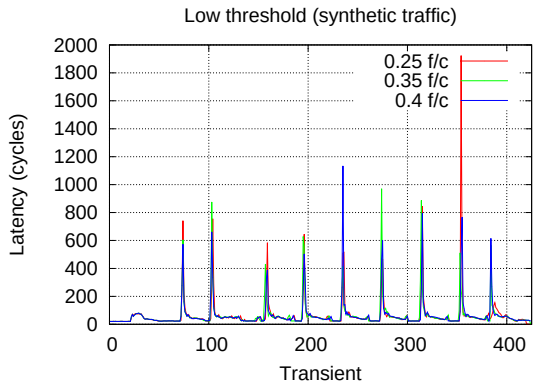


Figure 3.6: Low-threshold analysis.

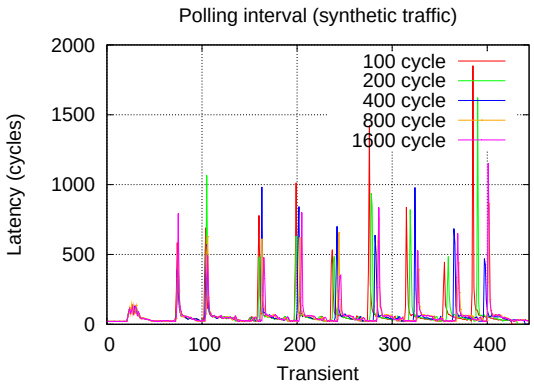


Figure 3.7: Polling interval analysis.

First, we analyze the effect when varying the notification delay of the BNN network. In Figure 3.4 we can see the results. This figure shows how notification delay has negligible effects on the BAHIA behaviour. The figure shows the latency of messages (for the synthetic traffic) when the BAHIA mechanism is running for different notification delays, from 1 cycle up to 16 cycles. As can be seen, latency of messages is unaffected and show almost the same values. This gives some reliability against unexpected jitter delays and grants flexibility for hardware implementation since BAHIA has no strict delay requirements.

Next we analyze the effect when varying the value of the detection (high threshold at the receiver side of the end-nodes). Figure 3.5 shows results for different values of the detection threshold (HT), ranging from 40% to 90% of traffic reception rate. Note that, the lower the threshold, the more aggressive the mechanism (i.e. more sensitive to traffic bursts), but it may incur in false positives (i.e. non-bursty traffic detected as bursty). By contrast, the higher the threshold, the more selective the mechanism possibily not detecting lightweight bursty traffic. As we can see in Figure 3.5, the higher the threshold the lower the accuracy in detecting bursts, as more peaks in latency occur for the synthetic traffic pattern.

Similarly, Figure 3.6 shows the results obtained when varying the value of the low threshold (used to detect the end of bursts). As can be seen this parameter does not affect BAHIA performance because, when a burst ends, all virtual networks are free

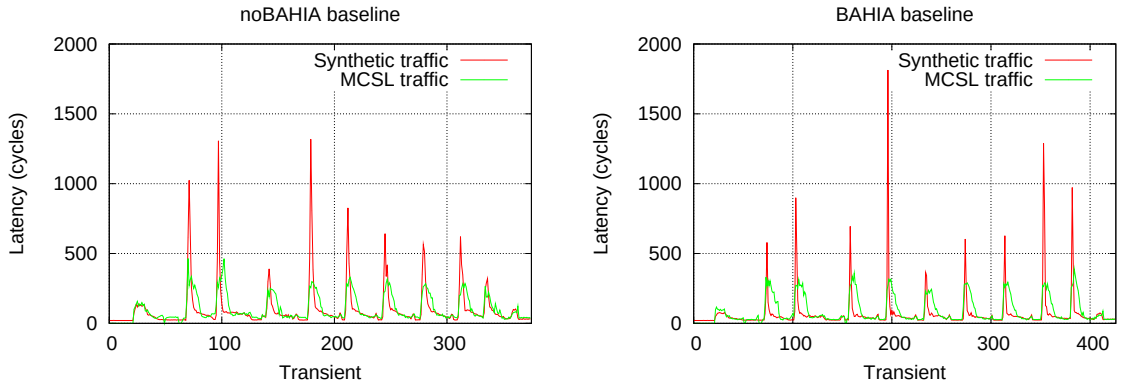


Figure 3.8: Latency for the BAHIA vs no-BAHIA analysis.

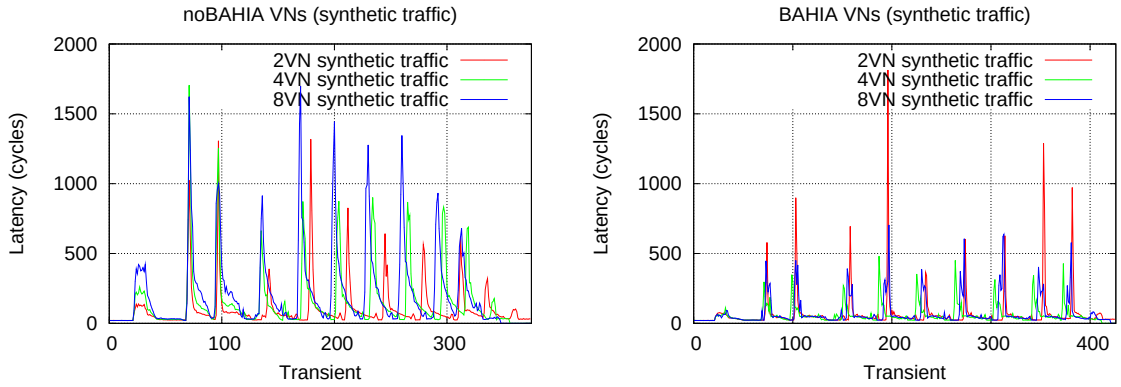


Figure 3.9: Latency for synthetic traffic in the BAHIA VNs analysis.

from HoL-blocking effects, so traffic flows smoothly through all virtual networks, thus no matter which virtual network messages are mapped to.

Finally, the value of the polling interval has been tested. As can be seen in Figure 3.7, all the values but 100 cycles seem to be a correct choice, correctly reacting to the burst so that no false positives are detected. Note that the polling interval tuning presents a close relationship with the high-threshold analysis: having a small polling interval is similar to having a lower HT value. The contrary also applies: a large polling interval could lead to the mechanism filtering short transient bursts. We conclude, then, that a small polling interval is convenient and not harmful.

3.2.4 BAHIA vs No-BAHIA Analysis

In this section we carry out a general analysis of BAHIA in comparison with similar scenarios without BAHIA in order to quantify the performance improvement of BAHIA. In the previous section we have delimited optimal values for the parameters that defines BAHIA behavior for the scenario used in the simulations, so in this analysis we have set BAHIA parameters according to these optimal values.

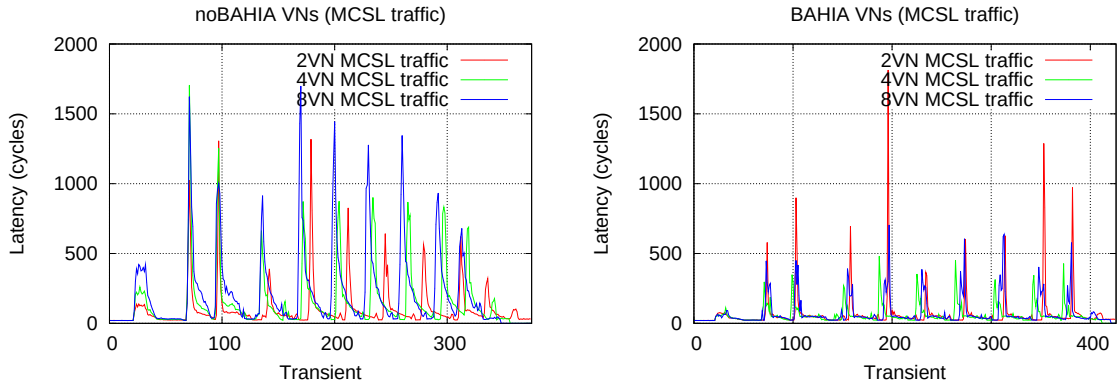


Figure 3.10: Latency for MCSL traffic in the BAHIA VNs analysis.

Latency (cycles)			
	noBAHIA	BAHIA	Improvement
2VN synthetic traffic	60.44	41.96	44.04%
2VN MCSL traffic	124.11	119.92	3.49%
4VN synthetic traffic	93.83	47.72	96.63%
4VN MCSL traffic	226.56	148.22	52.86%
8VN synthetic traffic	152.7	52.36	191.63%
8VN MCSL traffic	400.46	201.67	98.57%

Table 3.3: Average latency for BAHIA and no-BAHIA scenarios

Baseline Analysis

For this analysis, we have performed simulations for the baseline configuration of BAHIA. Figure 3.8 shows the latency achieved with and without BAHIA. In this figure we can see that, despite of the higher peaks for the synthetic traffic in BAHIA graph, these peaks of latency are shorter in time, achieving a moderated improvement of approximately 30% in the average latency regarding the same traffic in the no-BAHIA case as seen in Table 3.3.

Virtual Networks Analysis

Now, we turn our attention to the scenario when different sets of virtual networks are available. We analyze 2, 4, and 8 virtual networks. For the BAHIA case we keep a single extra virtual network and the remaining virtual networks are used as normal queues. Thus, all the bursts are mapped into the same virtual network. For the no-BAHIA case all the queues are used equally.

Figure 3.9 and Figure 3.10 show the results. As can be seen, the higher the number of virtual networks the higher the latency increase in the no-BAHIA case. In contrast, when using BAHIA, non-bursty traffic latency keeps bounded and minimized, reaching high latency reductions up to a factor of 3x. Numeric results can be seen in Table 3.3.

Chapter 4

ICARO

4.1 ICARO Description

Previously, BAHIA has been presented as a mechanism for facing bursty traffic by isolating bursty traffic from no-bursty one in order to avoid HoL-blocking. This is an effective strategy in bursty-prone environments such as heterogeneous NoCs, or NoCs interconnecting hardware accelerators. However, congestion is not always related to the receiving data rate, thus BAHIA may not be the best strategy fully avoiding HoL-blocking for all cases. Detecting congestion trees and isolate them, is a more general problem suitable for any type of NoC and it is expected to be a more effective mechanism. In this chapter a mechanism for avoiding HoL-blocking by detecting congestion and isolating it is presented. ICARO follows the same strategy as BAHIA in the sense that harmful traffic is identified and then isolated to a special virtual network queue but, instead of detecting and isolating bursty traffic, ICARO detects congested traffic and then isolates it. In the next sections we will describe ICARO mechanism, which can be divided in three stages: congestion detection, congestion notification and traffic separation.

4.1.1 Congestion Detection

In BAHIA mechanism we have seen that bursts are detected by end-nodes by computing received data rate from default VNs and then comparing with defined thresholds. This approach is not valid for detecting congestion since a high traffic rate burst at an end-node may not always involve congestion. Congestion is defined as a contention situation extended over the time and, in turn contention arises when two or more flows compete for the same resource. More precisely, in mesh NoCs contention arises when two or more flows compete for the same output port in the switches, since only one of these flows wins the requested output port at a given time, hence the other flows will must wait for the next arbitration turn, increasing latency for such flows. Also, since such flows can not be injected, these flows remain in their input queues while more flits are enqueued on these input queues, forcing flow control mechanism to work in order to avoid queue overflow by halting injection at the source node and propagating this behaviour through the flows tree. This chain reaction is congestion. Let's see an example for a 4x4 mesh network. In Figure 4.1 we can see a data flow from node 1 to

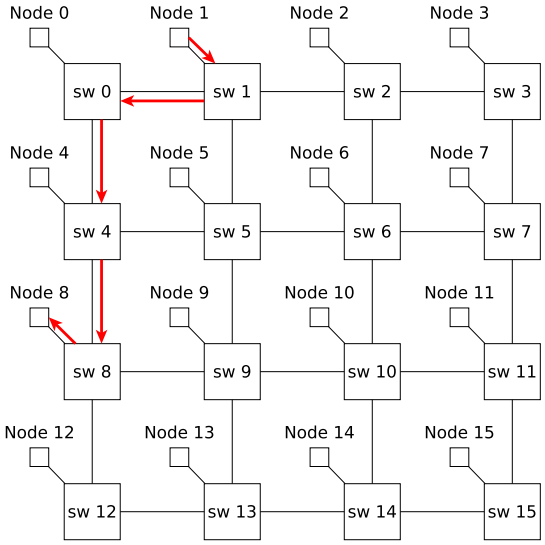


Figure 4.1: Example of high data rate flow whithout congestion

node 8 at a 1 flit/cycle rate. This flow clearly will be detected as a burst by BAHIA but, if there are no additional flows crossing the switches involved in the path followed by such flow, there is no congestion and, therefore there is no harmful effect to be removed.

Since our goal is to detect congestion we must follow a different approach. In order to do this, our proposal builds on monitoring variables related to congestion situation instead of only inspecting data rate received at NICs. Congestion begins as contention in the switches, so seems clear that a correct approach is to detect this effect at switches instead of at end-nodes as in BAHIA. Our proposal can be divided in two stages or parts: contention detection and congestion detection. Contention arises when two or more flows compete for the same output port. To detect this, a modified switch implementation is proposed in order to keep record of how many input ports are requesting every output port at every cycle. To perform this, a counter of $ceil(\log_2(p-1))$ bits (requests counter) where p is the number of ports of the switches in the network (including the internal port) is added to the switch. We asume a pipelined switch design with four stages: IB, RT, VASA, and X. In IB the incoming flit is allocated in the input buffer. In RT the message is routed (the output port is obtained). In VASA, the flit contends for virtual channels and for the crossbar. Finally, in X stage, the flit crosses the switch. In a ICARO switch, in addition, at the RT stage the counter associated to the output port requested is increased by one. Analogously, when this output port forwards a flit, this output port becomes free, so its associated counter is decreased. By doing this, we keep record of the number of ports requesting every output port, so when the requests counter reaches 2 or greater value means that the port associated to this counter is requested by two or more input ports so there is contention since one of the flows is ready and waiting for the port requested. Once the counter of a given port reaches the value of 2, another counter (congestion counter) associated to this port starts increasing by one at every cycle.

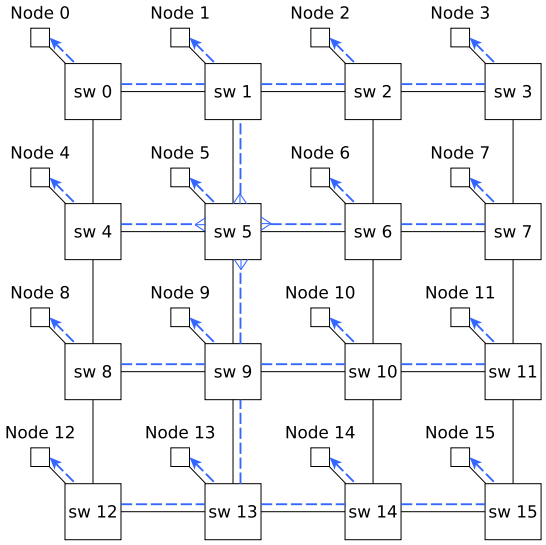


Figure 4.2: CNN line for switch 5

Periodically (at a regular frequency called *polling interval*; PI), an indepent module in the switch checks the congestion counters for every output port. When a congestion counter reaches a given threshold (*congestion time threshold*; CTT) congestion at this port is assumed so notification is triggered for this port at the current switch. Notice that the notification is sent when congestion is detected but not when contention is detected. This means notifying only important events and not transient contention situations that do not necessarily lead to congestion.

4.1.2 Congestion Notification

As well as in BAHIA, ICARO makes use of a simple parallel network in order to notificate for congestion. For BAHIA it is enough with a simple high/low signal from every NIC since BAHIA takes only into account whether the destination is receiving bursty traffic or not in order to take the decission of post-processing or not messages, however for ICARO this approach must be redefined. ICARO notifies from the switches to the end-nodes, detecting the ports suffering congestion in order to separate traffic at source end-nodes depending on whether the message just being injected will cross a congested port or not (explained in next section). Since ICARO takes into account every port at every router in the network, a more elaborated notification mechanism is required.

Our proposal for the notification network (Congestion Notification Network, or CNN) keeps the same network infrastructure (the wires) as used for BAHIA (see Figure 4.2). Every switch has its own private network (with a given delay; ND). This one-bit network must transmit only bits of data (where p is the maximum number of ports at switches). Every bit transmitted corresponds to a port of the source switch and the value of every bit informs whether the corresponding port is congested or not. A typical transmission of a notification from a given switch would be:

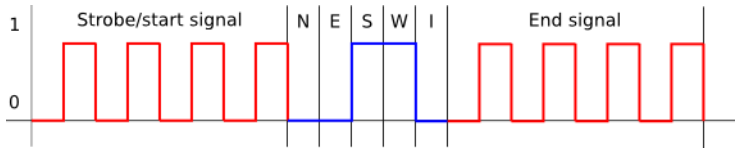


Figure 4.3: Signal format for congestion notification

1. Synchronization/start signal. A few cycles of strobe signal in order to synchronize between switches (useful for asynchronous NoCs) and to mark the beginning of a transmission.

2. p bits of data informing of the state of every port of the source switch.

3. A few cycles of strobe signal in order to mark the ending of the transmission.

In Figure 4.3 an example of signal transmitted through the CNN can be seen. Transmission starts with four cycles of strobe signal (the number of cycles of strobe signal is a fixed value) that also marks the begining of a transmission. After the strobe signal four bits are transmitted, each one corresponding to a port of the switch. In our example the notification informs that ports South and West are congested while the North, East and internal ports are not congested. Finally, after the four bits of data, another four cycles of strobe signal marks the ending of the transmission.

Following this approach each switch in the network will be able to inform to all nodes in the network which of its ports is congested.

Since we have added some complexity to the notifications mechanism it is advisible to implement a fault-tolerance mechanism to the CNN in order to deal with notifications failures. In addition, this fault-tolerance mechanism is useful in the notifications storing mechanism as explained latter. When a switch sends a congestion notification for a given output port, the switch expects that, after a reasonable time, traffic for the congested port of such switch will be received through the extra virtual network. If too much time elapses (re-sending timeout; RST) and the congested switch keeps receiving traffic to the congested port through a default VN, this switch assumes that congestion notification has not been received or stored succesfully by one or more nodes so the congested switch will send the notification again. This process is repeated until all traffic to the congested port is received through the extra VN or congestion disappears. Note that receiving duplicated notifications has no effect for the nodes that are working correctly enqueuing traffic to the congested switch port through the extra VN.

4.1.3 Traffic Separation

As in BAHIA, the network is provided with at least two virtual networks: one extra virtual network and one or more default virtual networks. At the end-nodes, all traffic allocated is enqueued always into a default virtual network and at every clock cycle the head of all queues are checked in parallel. If the flit at the head of a queue is a header flit (the first flit of a message), then, the destination of such message is checked in order to find out whether the message must be reallocated into the extra virtual network. In BAHIA, destination is checked to find out whether the destination was

receiving bursty traffic or not. However, in ICARO we follow other criterion based on the congestion points information received from the switches in order to decide whether messages must be reallocated into the extra virtual network for avoiding HoL-blocking. Thanks to the notification from the switches, the end-nodes know which ports of which switches are congested, thus the end-nodes know which points in the network are trouble points (hot-spots). When a header is found at the header of any default VN, the destination is extracted from the message, then checks the information regarding congestion points stored in the end-node in order to find out if the message will cross any congestion point throughout the route to the destination end-node. If the message will cross any congestion point, it is reallocated (post-processed) to the extra VN.

Congested Points Storage

In order to decide whether a message must be reallocated into the extra virtual network or not, every end-node must know which congested points exist in the network. If we have a 4x4 2D mesh, means that the network contains 16 swiches with 5 ports each one (assuming one end-node connected per switch). It also means that every end-node in the network must store and manage up to $16(switches)*5(ports) = 80bits$ of data in order to keep record of which points in the network are congested. In this case seems not to be very expensive in area and power overhead, but if we scale that example to a CMP like Tile-GX which contains 100 cores we obtain that for storing all values we would need up to $100(switches)*5(ports) = 500bits$ per end-node. This does not scale if we pretend to cover the whole congestion network status, so we propose to use a cache memory approach for storing data regarding the congested points existing in the network.

We assume that most of the time only a small portion of the congested points in the network may be active simultaneously so we make use of a small cache memory with a few number of entries in every end-nodes. This cache memory does not have a typical cache memory implementation in which providing a single value (like a memory address) a single stored value is returned. Instead of this, we implement a memory cache with some additional logic for storing three values in each row:

1. Switch that notified congestion
2. Congested port of that switch
3. Counter (explained latter)

For the cache store operation, the cache memory is provided with the values *switch* and *port* while the *counter* field is initialized internaly to 1 if the entry is new (*counter* field is controlled internally by the memory).

A message will be reallocated to the extra VN depending on whether this message will cross a congestion point, so the module in charge of reallocating messages when needed only knows the destination end-node of the message and only needs to know whether the message must be reallocated or not, so we decided to provide the cache memory with some additional logic in order to simplify the reallocating decission process.

According to this, for the cache read operation, the destination end-node of the message is provided to the memory. The memory internally checks all rows in order to find a point that belongs to the path that will be followed by the message starting at the current switch until the destination end-node. If the memory finds one or more points that belongs to that path, the memory output will be *TRUE* or *FALSE* otherwise. In order to find out whether a point belongs to a path from the source to the destination end-node the cache memory uses this simple algorithm where the variable *found* corresponds to the output signal of the memory, *dstNode* corresponds to the input signal (destination end-node of the message being processed) and *srcNode* corresponds to the node owner of the cache memory:

```
x_src = srcNode % routers_in_dimension_x;
y_src = srcNode / routers_in_dimension_y;

x_dst = dstNode % routers_in_dimension_x;
y_dst = dstNode / routers_in_dimension_y;

found=FALSE;
for each entry in cache {

        x_sat = entry->router % routers_in_dimension_x;
        y_sat = entry->router / routers_in_dimension_y;

        if(entry.port == North && y_src > y_dst && y_sat <=
           y_src && y_sat > y_dst && x_sat == x_dst)
                found = TRUE;

        if(entry.port == East && x_src < x_dst && x_sat >=
           x_src && x_sat < x_dst && y_sat == y_src)
                found = TRUE;

        if(entry.port == South && y_src < y_dst && y_sat >=
           y_src && y_sat < y_dst && x_sat == x_dst)
                found = TRUE;

        if(entry.port == West && x_src > x_dst && x_sat <=
           x_src && x_sat > x_dst && y_sat == y_src)
                found = TRUE;

        if(entry.port == Internal && dstRouter == entry.router)
                found = TRUE;
}
```

In the pseudocode *x_sat* and *y_sat* corresponds to the x and y coordinates of the congestion point stored in the entry, *x_src* and *y_src* corresponds to the coordinates of the current node (source) and *x_dst y_dst* corresponds to the coordinates of the destination end-node.

The algorithm assumes the use of the XY routing algorithm in the 2D mesh. By

doing this, paths are known before injecting the message and intermediate points located in the cache memory can be easily inspected and decided if they will be along the message path. For instance, if a hot-spot is in the same row of the source end-node and the destination is also in the same row, then if the intermediate point is in between both nodes then the message will cross the hot-spot. The previous algorithm covers all possible combinations.

Out of Order Delivery Avoidance

As explained for BAHIA, reallocating messages between virtual networks may cause out of order injection and so delivery. In order to avoid this effect, an effective strategy employed in BAHIA consists in keeping record of how many flits are enqueued in the extra virtual network for each destination. For that purpose, BAHIA employs a dedicated counter for every destination end-node in the network. However, in ICARO, out of order delivery may arise to all the end-nodes for message flows passing through congested points since the reallocating policy forces remapping messages from default VNs to the extra. In Figure 4.4 we can see two examples of nodes affected by congested points. In the left network there is a congested point at the east link in switch 9. As we assume XY routing source nodes affected by this congested point are nodes 8 and 9. These nodes may reallocate messages to the extra VN depending on whether the destination of the messages will cross congested point in switch 9. Accordingly, the destination end-nodes affected by such congested point are the nodes enclosed in the *end nodes* square. These nodes may receive messages through the extra VN due to the congested point in switch 9, thus these nodes may receive out of order messages. This relates a congested point with out of order delivery to the set of destination end-nodes affected by the congested point. Thats why in each entry in the cache memory at the end-nodes we introduce the *counter* field. This counter in a given entry (a given congested point) is increased when a flit that will cross the given congested point is reallocated to the extra VN and decremented when a flit that will cross the given congested point is injected.

The traffic separation mechanism with out-of-order avoidance mechanism can be briefly explained with the next pseudo-code similarly as the BAHIA case:

```
for each default_vn in the_end_node {
        if(!isVNempty(default_vn)) {
                msg=getHeadMsgFromVN(default_vn);
                for each entry in cache_memory {
                        if(pointBelongsToPath(entry, msg.
                           destination) || entry.counter > 1)
                                moveMessageToExtraVN(msg);
                }
        }
}
```

Note that entries are initialized with *counter* field at 1, so a value of 1 in this field means that zero messages belonging to this congested point are enqueued in the extra virtual network. Basically, the algorithm ensures that messages passing through

a congested point will be allocated in the extra VN. However, even if the end-node receives a notification that the port is no longer congested but there are still flits in the extra VN passing through that point, the end-node will still map flits into the extra VN, so to avoid possible out-of-order injection of flits. In the next section we describe the cache deallocation policy that ensures this in-order injection mechanism.

Cache Memory Entry Deallocation

When a congestion notification is received a new entry is created at the cache memory with the received information (switch and port) and the *counter* field is initialized to 1. From that moment, when a message is reallocated into the extra VN due to that congestion point *counter* field is increased by the number of flits of such message. Accordingly, when flits crossing that congestion point are injected through the extra VN, *counter* field is decreased. Decongestion notification may arrive at any moment but we only must remove such entry when this notification arrives and there are no messages destined to that congestion point enqueued in the extra VN since we need the *counter* field to avoid out-of-order injection. A possible solution would consists in adding a one-bit-wide field to each entry in the cache memory in order to track the status of the congestion point and remove the entry when this field and the *counter* field were zero. However, we have opted for merging both fields by initializing the *counter* to 1. Accordingly, when an end-of-congestion notification arrives, *counter* field is decremented. In this way, when *counter* reaches zero value, means that there is no longer congestion and there are zero flits belonging to this congestion point enqueued in the extra VN, so this entry can be deallocated.

Cache Memory Replacement Policy

Cache memory for storing congested points information has a limited size. This size must be enough to satisfy the most common case by minimizing the amount of entries replacement. However, there is a trade-off between the cache size and the performance achieved, and most importantly, a cache replacement policy is needed when the cache is full. Our cache memory implementation differs strongly from a typical cache memory regarding the replacement policy. While in a typical cache memory relies on the existence of a main memory whereby the whole data is stored and can be retrieved, our cache memory implementation has no means to obtain removed data from cache so removing an entry must be performed very careful. Obviously, when a new congestion notification arrives the cache memory looks first for free entries to store the notification data. In case of not finding a free entry then it looks for entries subject to be replaced. If a cache entry has the *counter* field greater than 1 means that is keeping record of the amount of flits belonging to this congestion point enqueued in the extra VN for out-of-order injection avoidance, so such entries must not be removed. However, it may exist entries with the *counter* field set to 1, which means that a congestion notification has initialized this entry but there are no messages enqueued in the extra VN to track. Since the notification mechanism is provided with fault-tolerance techniques this entry can be replaced. In case there are no entries available to be replaced the notification is ignored relying on the fault-tolerance techniques of the notifications trusting in trying to store the notification latter. It is worth mentioning

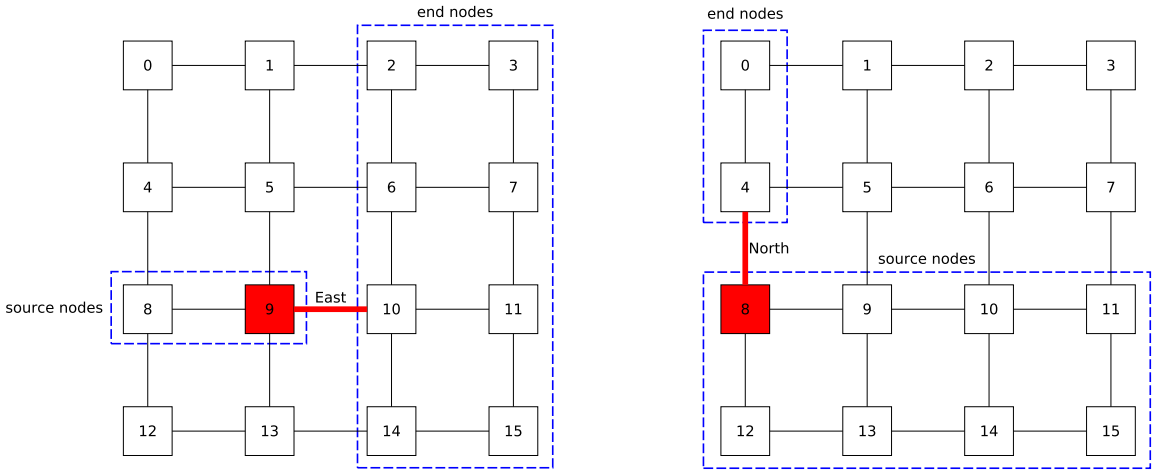


Figure 4.4: Nodes affected by congestion points

that, even in the case of not allocating a new hot-spot in the cache, this does not cause a malfunction of the mechanism or out-of-order injection. In the worst case it would decrease slightly (probably negligible) the effectiveness of the mechanism as it is expected to be very uncommon. Due to the fault-tolerance mechanism, it might be possible that duplicated notifications are received because there were nodes that didn't store any notification. When a notification is received and cache memory tries to store it, if it detects that there is already an entry for that congested point, the storing request is ignored.

Walkthrough Example

In Figure 4.5 we can see an example of the out-of-order injection avoidance mechanism. In Figure 4.5a congestion notification is received from switch 9 informing that the east port of such switch is congested. Next, in Figure 4.5b node 8 tries to inject a message to node 2. Since this message will cross the east port of switch 9 this message is transferred to the extra VN and the *counter* field of the cache is increased by one (note that *counter* field actually counts flits instead of messages but in the example is simplified for better understanding). As explained in the BAHIA section, this transfer is performed by an independent hardware module so transfering to the extra VN and injection (if possible) of the next message in the current VN can be done in parallel. Next message is destined to end-node 13 so it will not cross the congestion point, therefore is injected. Then, in Figure 4.5c node 8 sends a message to node 2 through the extra VN. Note that, despite a round-robin arbitration for VNs is used, reallocation of messages from default VNs to the extra one is performed at each cycle independently of the VNs being processed so message destined to end-node 14 from default VN is reallocated to the extra one. Note also that the *counter* is decreased by the injection of the message destined to end-node 2 but is increased again when reallocating the message from the default VN so actually the *counter* keeps the 2 value. Following, in Figure 4.5d the node 8 sends the message to the destination end-node 14 and decreases the *counter*. Finally, in Figure 4.5e node 9 sends a decongestion notification, therefore

the *counter* is decreased and, as *counter* reaches zero value, the entry in the cache is safely removed.

4.2 ICARO Evaluation

In this section we perform a preliminary evaluation of ICARO based on simulations in congestion-prone scenarios. First the simulation environment is described. Second, preliminar results comparing ICARO-based solution against the same scenario without ICARO are presented. Finally, results for simulations with several amount of virtual networks are shown.

4.2.1 Simulation Environment

For better comparing with BAHIA, simulations are performed with the same NoC simulator and scenario as for BAHIA: 16 nodes arranged in a 4x4 2D mesh network. Accordingly, realistic traffic traces extracted from the H264 codec in MCSL format are injected as congestion-prone traffic merged with an uniform traffic pattern at 0.3 flit/cycle. Since our goal is to keep uniform traffic unaffected from the H264 traffic, both traffic latencies are separated in the graphs shown. More details about the characteristics of the simulation are shown in Table 3.2.

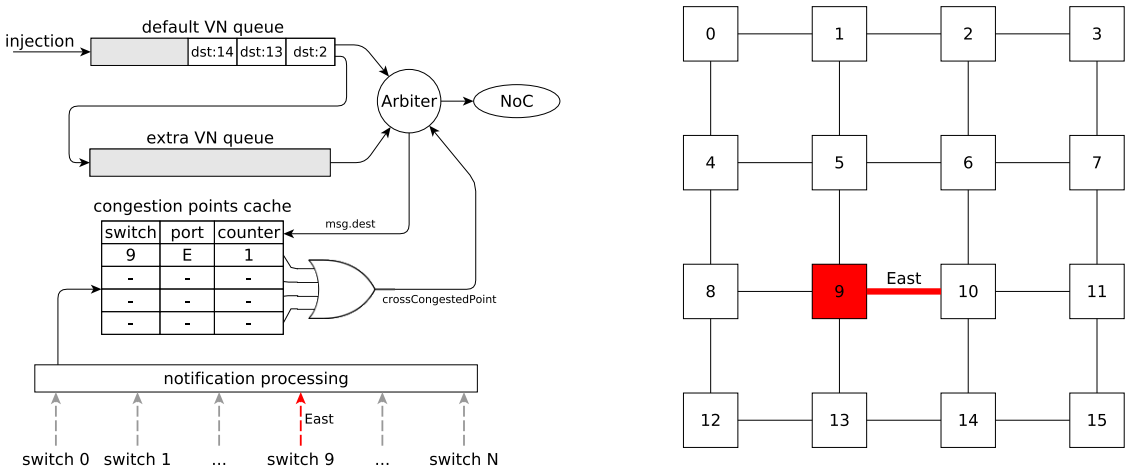
4.2.2 ICARO vs no-ICARO Analysis

In this section we perform an analysis of the improvement achieved by ICARO against the same scenario whitouth ICARO. For running ICARO simulations some parameters are required in order to get it work: polling interval (PI), congestion time threshold (CTT), notification delay (ND), re-sending timeout (RST) and cache size. All paramaters but notification delay (which is a paramter given by the network hardware) are configurable parameters which determine the ICARO behaviour and may impact in the improvement achieved. These must be subject of study but, since this is a preliminar analysis we set these parameters to reasonable values (described in Table 4.1, leaving a more deep analysis of such parameters for future work).

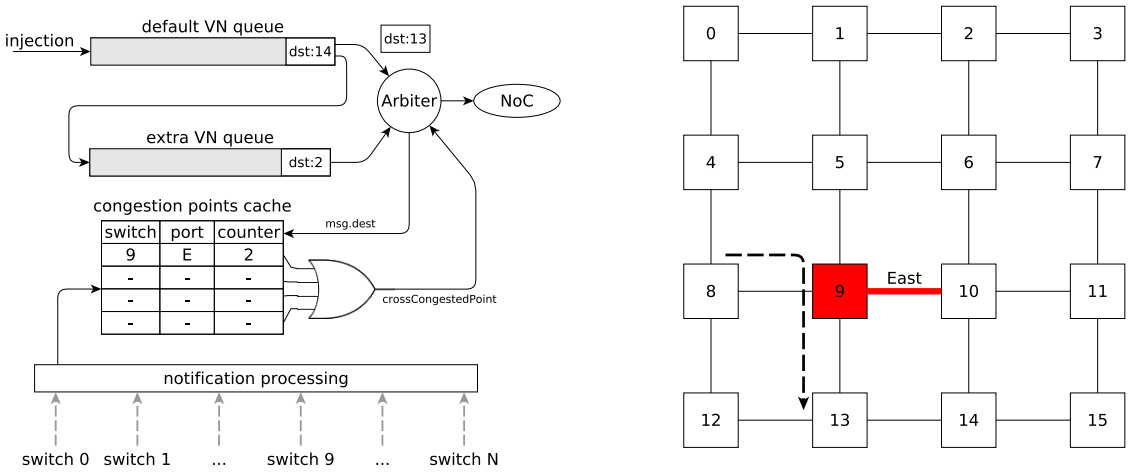
Parameter	Value
Polling interval	1000 cycles
Congestion time threshold	300 cycles
Notification delay	4 cycles
Re-sending timeout	300 cycles
Cache size	8 rows

Table 4.1: ICARO configuration parameters

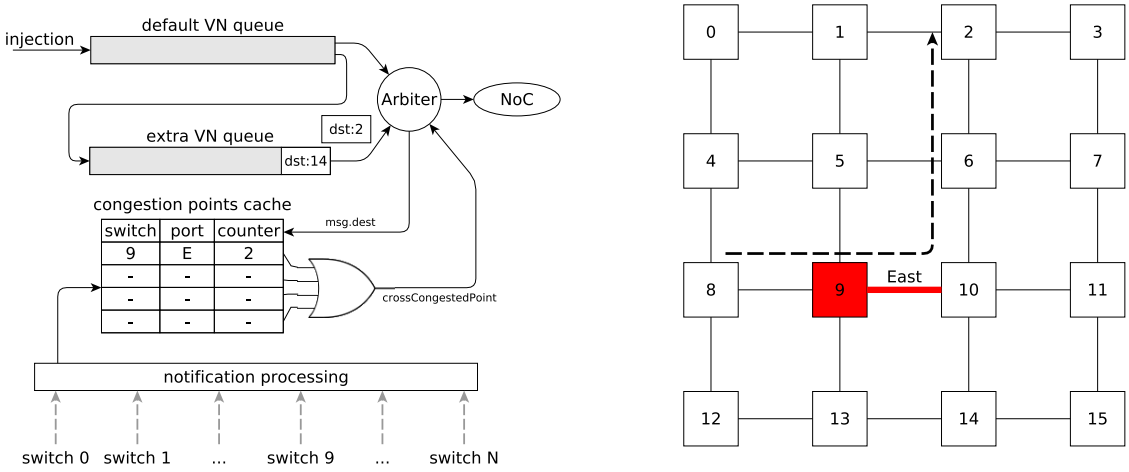
In Figure 4.6 we can see the latency for MCSL and synthetic traffic with 2 virtual networks. Clearly either MCSL and synthetic traffic latency is improved. MCSL traffic latency is improved by 63% and synthetic traffic latency by 33%.



(a) Congestion notification is sent from port East of node 9

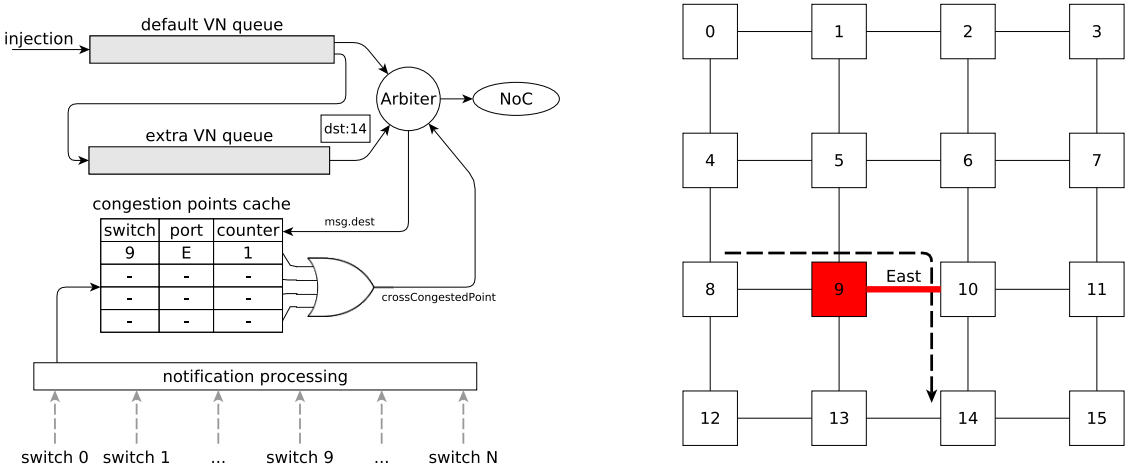


(b) First message injected through congested route

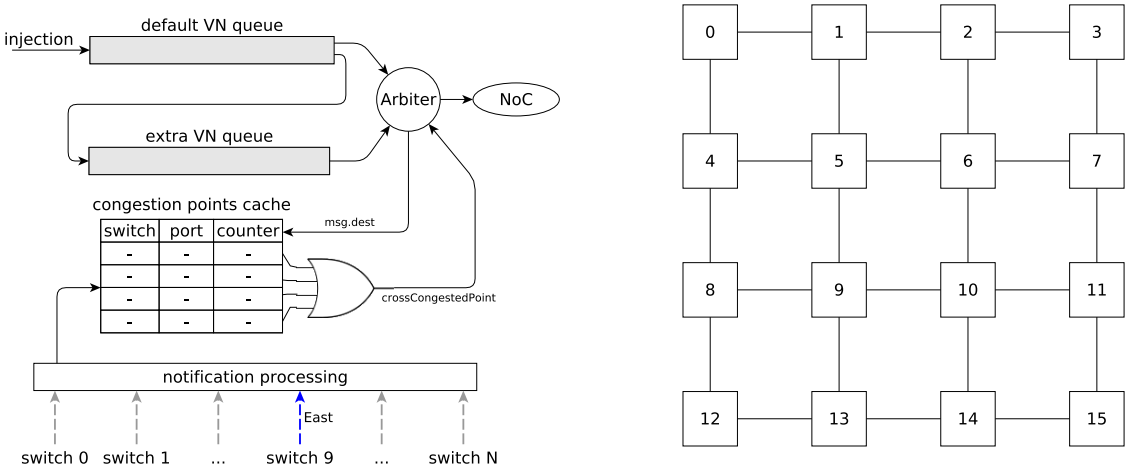


(c) Message injected through uncongested route

Figure 4.5: Test



(d) Second message injected through congested route



(e) Second message injected through congested route

Figure 4.5: Example of cache entries evolution

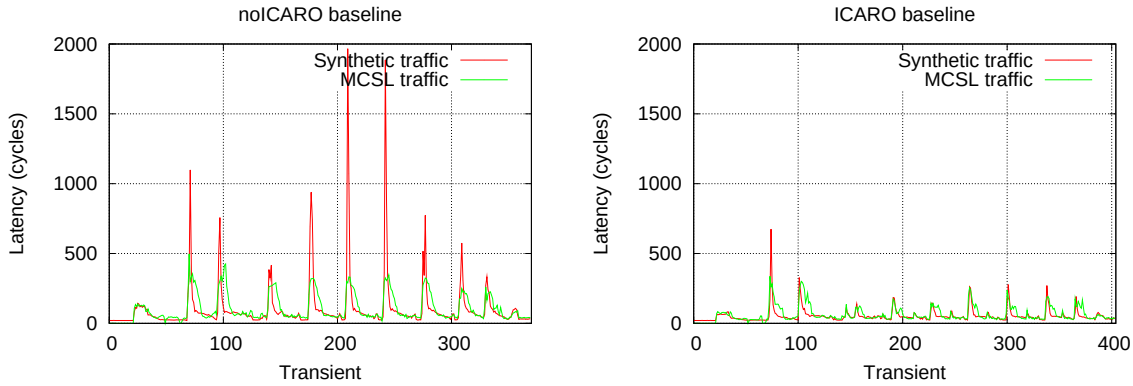


Figure 4.6: Latency for synthetic traffic in the ICARO baseline analysis.

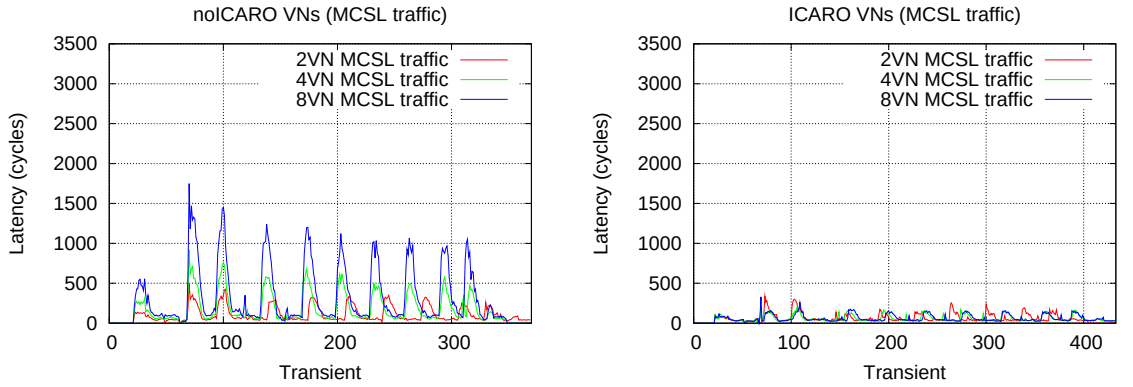


Figure 4.7: Latency for MCSL traffic in the ICARO VNs analysis.

4.2.3 Virtual Networks Analysis

In this section we shown results from the simulations of scenarios with and whitout ICARO for several amounts of virtual networks.

The left graphs of Figures 4.7 and 4.8 show the latency for both MCSL and synthetic traffic. As can be seen, HoL-blocking increases the latency with the amount of virtual networks. However, in the right graphs we see that with ICARO we achieve latency reductions up to 430% for MCSL traffic and up to 280% for synthetic traffic.

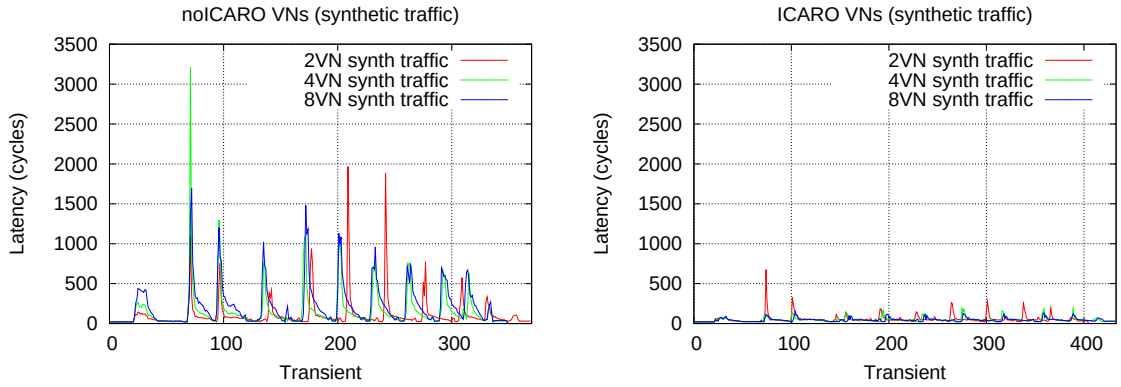


Figure 4.8: Latency for synthetic traffic in the ICARO VNs analysis.

Chapter 5

Conclusions, Future Work and Related Publications

It is well-known HoL-blocking considerably degrades overall network performance. The increasing number of devices interconnected in CMP and MPSoCs increases the network sizes and, therefore its power consumption. This makes necessary to adopt power-saving techniques in order to minimize it. However, power-saving techniques always imply network performance reduction that may result in congestion and traffic burst. In fact, this harmful effect is pronounced as more and more virtual networks are available in the system. In this master thesis we have presented separated solutions for bursty-traffic (BAHIA) and congestion (ICARO) to solve these problems by isolating bursty/congested traffic from non-bursty/non-congested one. With these mechanisms we do not address congestion directly but we avoid HoL-blocking. We have evaluated the improvement with BAHIA and ICARO compared with similar scenarios whithout any mechanism for avoiding HoL-blocking, achieving good results in keeping latency and throughput of regular traffic, obtaining up to three times better latency in the case of BAHIA and up to four times for ICARO.

As future work we plan to implement and evaluate power-saving techniques as proposed in [CXK+12] or [YLH+12] together with ICARO in order to evaluate the improvement of our proposal and perform more in-deep analysis regarding ICARO parameters. In addition, we plan to evaluate the area and power overhead introduced by ICARO in order to asses the scalability of the proposal.

The current master thesis document represents the current status of the research in the direction of congestion management strategies for on-chip networks. This is a emerging topic in such networks. Indeed, the challenge is tailoring and defining adequate mechanisms for this new restricted environment where not large resources can be used. In addition, congestion management strategies will be needed for future systems where aggressive power saving mechanisms will put the network in ultra low modes, thus potentially leading to congested scenarios. In this master thesis we have provided the first stage of the thesis in this topic, providing feasible and low cost mechanisms for such on-chip networks. The next steps are 1) evaluating and optimizing the proposals in terms of area and power, and 2) combine the mechanisms with aggresive power-saving mechanisms.

Following, publications regarding this master thesis:

- Jose Vicente Escamilla, José Flich and Pedro J. García. Burst-Aware HoL Blocking Avoidance. In International Summer School on Advanced Computer Architecture and Compilation for High-Performance and Embedded Systems, july 2012. ISBN 978 90 382 1987 5

- Jose Vicente Escamilla, José Flich and Pedro J. García. BAHIA: Burst-Aware Head-of-Line Blocking Injection Avoidance. In Jornadas de Paralelismo, September 2012

Bibliography

[BM06] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Comput. Surv.*, 38, June 2006.

[BM09] A. Banerjee and S.W. Moore. Flow-aware allocation for on-chip networks. In *Networks-on-Chip, 2009. NoCS 2009. 3rd ACM/IEEE International Symposium on*, pages 183 –192, may 2009.

[Cora] Intel Corp. The single-chip cloud computer. Available at `http://techresearch.intel.com/ResearchAreaDetails.aspx?Id=27`.

[Corb] Intel Corp. Teraflops research chip. Available at `http://techresearch.intel.com/ProjectDetails.aspx?Id=151`.

[Corc] Tilera Corp. Tilera tile multicore processors. Available at `http://www.tilera.com/products/processors/TILE-Gx_Family`.

[CXK+12] Xi Chen, Zheng Xu, Hyungjun Kim, P. Gratz, Jiang Hu, M. Kishinevsky, and U. Ogras. In-network monitoring and control policy for dvfs of cmp networks-on-chip and last level caches. In *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, pages 43 –50, may 2012.

[DMMD09] Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das. Application-aware prioritization mechanisms for on-chip networks. *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture - Micro-42*, page 280, 2009.

[FB10] José Flich and Davide Bertozzi. *Designing Network On-Chip Architectures in the Nanoscale Era*. Chapman & Hall/CRC, 2010.

[GGK08] Paul Gratz, Boris Grot, and Stephen W Keckler. Regional congestion awareness for load balance in networks-on-chip. In *HPCA*, pages 203–214. IEEE Computer Society, 2008.

[GKM09] Boris Grot, S.W. Keckler, and O. Mutlu. Preemptive virtual clock: a flexible, efficient, and cost-effective QOS scheme for networks-on-chip. In *Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 268–279. ACM, 2009.

[GLFB11] Alberto Ghiribaldi, Daniele Ludovici, Michele Favalli, and Davide Bertozzi. System-level infrastructure for boot-time testing and configuration of networks-on-chip with programmable routing logic. In *VLSI-SoC*, pages 308–313. IEEE, 2011.

[GQF+06] P.J. Garcia, F.J. Quiles, J. Flich, J. Duato, I. Johnson, and F. Naven. Efficient, scalable congestion management for interconnection networks. *Micro, IEEE*, 26(5):52 –66, sept.-oct. 2006.

[IZG+07] Ravi Iyer, Li Zhao, Fei Guo, Ramesh Illikkal, Srihari Makineni, Don Newell, Yan Solihin, Lisa Hsu, and Steve Reinhardt. QoS policies and architecture for cache/memory in CMP platforms. *ACM SIGMETRICS Performance Evaluation Review*, 35(1):25, June 2007.

[KTJR05] R. Kumar, D.M. Tullsen, N.P. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32 – 38, nov. 2005.

[LXW+11] Weichen Liu, Jiang Xu, Xiaowen Wu, Yaoyao Ye, Xuan Wang, Wei Zhang, M. Nikdast, and Zhehui Wang. A noc traffic suite based on real applications. In *VLSI (ISVLSI), 2011 IEEE Computer Society Annual Symposium on*, pages 66 –71, july 2011.

[LZJ06] Ming Li, Qing-An Zeng, and Wen-Ben Jone. Dyxy: a proximity congestion-aware deadlock-free dynamic routing method for network on chip. In *Proceedings of the 43rd annual Design Automation Conference*, DAC '06, pages 849–852, New York, NY, USA, 2006. ACM.

[MBL+01] S.S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The alpha 21364 network architecture. In *Hot Interconnects 9, 2001.*, pages 113 –117, 2001.

[MJ09] M. Millberg and A. Jantsch. Priority based forced requeue to reduce worst-case latencies for bursty traffic. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1070 –1075, april 2009.

[MOP+09] R. Marculescu, U.Y. Ogras, Li-Shiuan Peh, N.E. Jerger, and Y. Hoskote. Outstanding research problems in noc design: System, microarchitecture, and circuit perspectives. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 28(1):3 –21, jan. 2009.

[MRN+05] T Marescaux, A. Rangevall, V Nollet, A Bartic, and H Corporaal. Distributed congestion control for packet switched networks on chip. In *Parallel Computing: Current Future Issues of High-End Computing, Proceedings of the International Conference ParCo*, volume 33, pages 761–768. Citeseer, 2005.

[NFM+11] George Nychis, Chris Fallin, Thomas Moscibroda, S. Seshan, and O. Mutlu. Congestion control for scalability in bufferless on-chip networks. Technical report, 2011.

[NO97] B.A. Nayfeh and K. Olukotun. A single-chip multiprocessor. *Computer*, 30(9):79 –85, sep 1997.

[ODH+07] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler, and Li-Shiuan Peh. Research challenges for on-chip interconnection networks. *Micro, IEEE*, 27(5):96 –108, sept.-oct. 2007.

[WAHS06] Dong Wu, Bashir M. Al-Hashimi, and Marcus T. Schmitz. Improving routing efficiency for network-on-chip through contention-aware input selection. In *Proceedings of the 2006 Asia and South Pacific Design Automation Conference*, ASP-DAC '06, pages 36–41, Piscataway, NJ, USA, 2006. IEEE Press.

[WJM08] W. Wolf, A.A. Jerraya, and G. Martin. Multiprocessor system-on-chip (mpsoc) technology. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 27(10):1701 –1713, oct. 2008.

[YLH+12] Guihai Yan, Yingmin Li, Yinhe Han, Xiaowei Li, Minyi Guo, and Xiaoyao Liang. Agileregulator: A hybrid voltage regulator scheme redeeming dark silicon for power efficiency in a multicore architecture. In *High Performance Computer Architecture (HPCA), 2012 IEEE 18th International Symposium on*, pages 1 –12, feb. 2012.