POLITECNICO DI MILANO
Master in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

A DVFS-Capable Heterogeneous Network-on-Chip Architecture for Power Constrained Multi-Cores

Supervisor: Prof. William FORNACIARI
Assistant Supervisor: Dr. Davide ZONI

Master Thesis of:
Andrea MARCHESE
Student n. 823353

Academic Year 2015-2016

Abstract

In the multi-core era, the Networks-on-Chip (NoCs) emerged as the de-facto interconnect due to their scalability and flexibility. Moreover, they strongly influence the power consumption and performance of the entire platform, thus representing a key component to be optimized. In this scenario, several proposals in the literature presented Dynamic Voltage and Frequency (DVFS) capable NoCs to trade the power and performance metrics. However, the NoC traffic is highly variable due to the different phases an application crosses during its own execution. Last, the coherence protocol greatly contributes to the traffic shape. In a nutshell, the NoC traffic shape is mainly due to the coherence protocol, while the specific application only influences the traffic volume. Few seminal works discussed coherence-aware DVFS schemes for NoCs. This thesis presented a novel DVFS-capable heterogeneous NoCs. A fine grain analysis of the traffic composition enabled a further optimization of the DVFS actuation. The proposed NoC is capable to optimally balance the power and performance at low, medium and high traffic. Thus delivering an holistic solution that is roughly application independent. This is achieved thanks to a load balancer module that can route the traffic to the different implemented physical networks. The mechanism observes the traffic load and adjusts the frequency of each physical NoC to optimally match the objective function. The novel NoC has been integrated in the Gem5 full-system simulator and compared with the baseline NoC and a state of the art DVFS-capable methodology. Results are extracted considering a 64-core architecture executing the Splash2 benchmarks. Results show that the proposed NoC almost provides the same performance of the performance-aware baseline NoC with an energy reduction of 30% and one third less resources.

Acknowledgments

First of all, I would like to thank my thesis advisor Prof. William Fornaciari and my thesis supervisor Dr. Davide Zoni for their patient guidance, enthusiastic encouragement and useful critiques of this research work. They consistently allowed this paper to be my own work and conveyed to me the passion to do research.

I would also like to extend my thanks to the people of the HIPEAC Research Group for their help and their support in running this thesis.

I wish to thank my family, my mother Rosanna, my father Gianni and my brother Matteo, for their support, their continuous encouragement during all these long six years, always by my side through all obstacles. I would not have done it without them.

Finally, special thanks should be given to all my friends because they have always been with me to celebrate the success and to make me smile even when the sky was darkened.

Contents

List of Figures

List of Tables

Acronyms

BW = Buffer Write
CMP = Chip Multi-Processor
CPU = Central Processing Unit
DMA = Directory Memory Controller
DVFS = Dynamic Voltage and Frequency Scaling
EDP = Energy Delay Product
FIFO = First In First Out
GTCM = Global Traffic Congestion Manager
GPU = Graphics Processing Unit
HPC = High Performance Computing
LT = Link Traversal
LA = Link Allocation
NAVCA = Non Atomic Virtual Channel Allocation
NTC = Near Threashold Computing
NIC = Network Interface Controller
NoC = Network-on-Chip
PE = Processing Element
PLL = Phase-Locked Loop
PNET = Physical Network
RC = Route Computation
SA = Switch Allocation
SaF = Store and Forward
ST = Switch Traversal
VA = Virtual-Channel Allocation
VC = Virtual Channel
VCW = Virtual Channel Wormhole
VFI = Voltage Frequency Island
VNET = Virtual Network

VCT = Virtual Cut-Through
WH = Wormhole
WU = Wakeup

Chapter 1

Introduction

"Sometimes it is the people no one can imagine anything of who do the things no one can imagine."

Alan Turing

The multi-core architectures emerged to provide a better power performance trade-off with respect to single core processors. However, the multi-core revolution pushed to the limit the on-chip bus-based solutions, thus highlighting the *Networks-on-Chip (NoCs)* as the de-facto on-chip interconnect. The NoC provides better scalability and flexibility properties than bus architectures, while its great impact on the power, performance and area metrics imposes a cunning design to deliver a successful architecture. The adoption of the on-chip networks makes the multi-cores widespread in different market segments, ranging from the *High Performance Computing* (HPC) to the embedded systems. The rest of this chapter introduces the objectives, the research area and the background of the thesis. In particular, Section 1.2 discusses different multi-core classes to better focus on the architecture subset that are considered in this work. The goals and contributions of the thesis are detailed in Section 1.3, while Section 1.4 provides a background on the main architectural blocks discusses in the thesis, i.e. the NoC, the cache coherence protocols and the *Dynamic Voltage and Frequency Scaling (DVFS)* mechanism. Last, the structure on the thesis is devoted in Section 1.5.

1.1 The Cache-Coherent Multi-Cores: Architecture and Applications

The computer architects always tried to increase the clock frequency and the architectural complexity at each generation to match the performance level foreseen by the Moore's Law. The hit of the so called power wall at the beginning of 2000s, started the transition from single-cores to multi-cores with the final objective of increasing the performance per watt metric. This is essential to keep the pace with the Moore's Law, while constraining the power consumption. The multi-cores provide a flexible architectures where different applications can truly execute in parallel. Moreover, such architectures allow different supply voltage values and clock rates to different parts of the architecture to further increase the performance per watt ratio.

The multi-cores range from the embedded to the High Performance Computing (HPC) segments. The HPC architectures execute computationally demanding applications from a wide range of domains trying to match the capacity computing paradigm. The capacity computing paradigm aims at executing the maximum number of applications while ensuring the best performance per watt ratio. Conversely, low power requirements dominate the embedded multi-cores mainly due to their battery-powered nature. Embedded multi-cores are still general purpose architectures while the low power requirements impose a different design methodology compared to the HPC platforms. Last, the accelerators are becoming a widespread multi-core solution [40] to boost the computation of some specific tasks inside a bigger application. Usually the multi-core accelerators implement simpler in-order, GPU-like CPUs, that are coupled with a simplified non-coherent cache hierarchy. The non-coherent cache hierarchy represents the key different between general purpose multi-cores and the accelerator, task specific multi-cores. The absence of the coherence protocol greatly reduces the generated traffic, while the programmer has to manually ensure the coherence. On the other hand, cache coherent multi-cores provide as easier to program platform, that is generic with respect to the executed applications.

The applications represent an additional design dimension for multi-cores, since they drive the design of new computing platforms. Considering the multi-core scenario, different applications are expected to run concurrently on the same hardware, each of them with its specific set of requirements and constraints. Moreover, each application should believe to be executed

in isolation with respect to all the other running tasks. The requirements from different applications can be contrasting to each other and can require the same hardware resources to be fulfilled, thus highlighting the well known resource contention problem [8]. The scenario is further complicated by the end users, that claim the same *user-experience* regardless if the application is executed on a smartphone, a tablet or desktop platform. For example, current smartphones are expected to smoothly run demanding graphic applications with the same *user-experience* of a laptop. Furthermore, single- and multi-threaded applications can execute on the same multi-core and each application traverses different phases during its execution, e.g. memory-bound, CPU-bound, thus stressing the platform resources in an unbalanced and time-dependent fashion.

This thesis targets the general-purpose, cache coherent multi-cores with particular emphasis on the embedded domain, where the power requirements can further narrow the design space. Different multi-threaded tasks are evaluated with different execution phases, to strengthen the proposed architecture.

1.2 Problem Overview

Considering a general-purpose, cache-coherent multi-core, the design space is huge. Thus, it is difficult to provide a system-wide optimized architecture accounting for the computational subsystem side by side with the interconnect and the memory hierarchy. Such optimized design is further complicated by the different application classes that are executed on the platform. Each class of applications has different requirements that can possibly contrast with the requirements of other applications. Moreover, the possibility to have applications of different classes that concurrently run on the same architecture represents a possible scenario that has to be accounted during the architectural design stages. In a nutshell, the possibility to have a virtually infinite variety of applications that can be executed on the multi-core makes the application-based optimization unfeasible. The resulting architecture would be a set of accelerators, each of them specifically designed for an application or a class of applications. Thus, such a solution is not viable for general purpose multi-cores. However, the optimization of the multi-core architecture to fit a broad class of applications still represents a key objective for design architects. In particular, the cache hierarchy and the interconnect greatly impact the performance and power metric. Thus their optimization can not

be neglected during all the design stages.

The *Dynamic Voltage and Frequency Scaling (DVFS)* mechanism has been extensively exploited in the computer architecture field for decades to optimize the power-performance metric. The DVFS allows to reduce the operating voltage and frequency when less computational power is required with a net power reduction. Conversely, both voltage and frequency can be increased to offer more computational power when required by the running applications. The adoption of the DVFS mechanism coupled with a suitable policy to steer it has been explored in CPUs for decades. The multi-core revolution moved the DVFS exploitation to the emerging NoC architectures. Several proposals optimize the NoC power-performance trade-off by means of the *Dynamic Voltage and Frequency Scaling* actuators and companion policies [55, 33, 47], while others focus on the design of a more efficient NoC micro-architecture [48, 31, 32, 3].

Considering the *Splash2* benchmarks [46], Figure 1.1 shows the application performance impact due to the NoC frequency. Results are collected from an 8x8 2D-mesh topology using 2GHz out-of-order CPUs. Each benchmark has been simulated with different fixed NoC frequencies - 500MHz and 2GHz - and a strong variations in its execution time has been observed. For Example, the OCEAN application shows an execution time of 82.33ms and 27.94ms when the NoC frequency is set at 500MHz and 2GHz, respectively. The frequency reduction directly reduces the performance of the architecture. On the contrary, frequency increase improves the performance but with a non negligible impacts on the power consumption.

On the other hand, the coherence protocol represents an orthogonal design dimension to improve the power-performance trade-off in the multi-core. An optimized coherence protocol increases the data re-usability in the cache hierarchy with benefit for both the performance and the power consumption. Considering the interconnect, the coherence protocol is responsible of the imposed traffic pattern and shape. Conversely, the applications can only impact the absolute traffic volume, that is actually application dependent.

The cache coherence protocol greatly contributes to the generated traffic, thus impacting the overall system performance as well as the absolute traffic volume. Different coherence protocols have been proposed to trade the system-wide performance and the imposed traffic to the interconnect. Figure 1.2 shows the generated traffic by three different coherence protocols

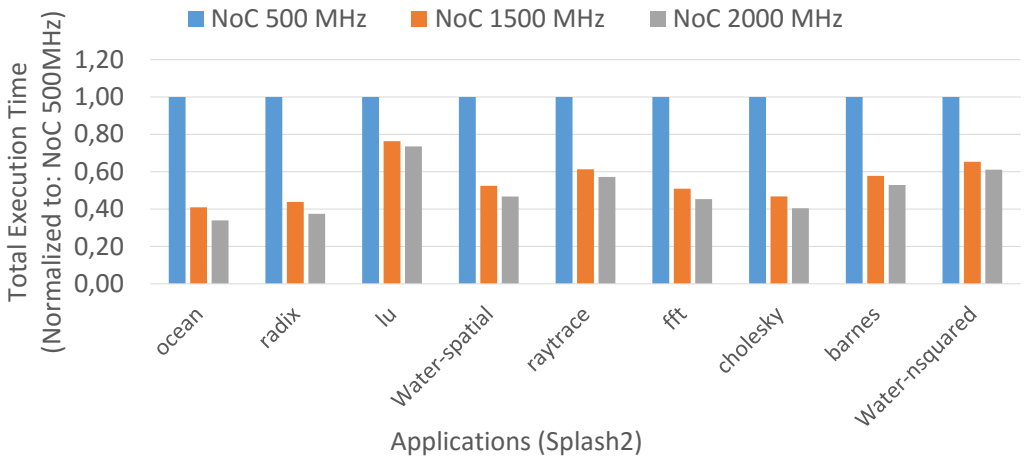


Figure 1.1: Frequency impact on performance.

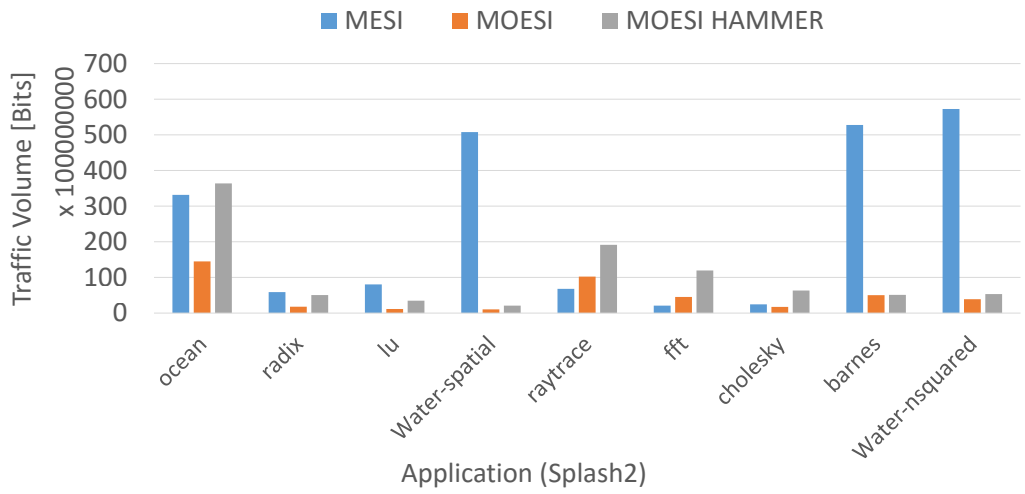


Figure 1.2: For each considered *Splash2* benchmarks, the traffic volume is reported for three different coherence protocols.

considering the *Splash2* [46] benchmarks. Conversely, Figure 1.3 presents the performance information using the same coherence protocols and benchmark set. The MESI represents the reference coherence protocol. The MOESI enhances the MESI protocol by adding the Owned (O) state. Two different MOESI-based coherence protocols have been compared. The MOESI exploits the so called directory to retrieve the coherence information, while the MOESI-hammer implements a broadcast-based mechanism to get the same information.

Results in Figure 1.2 and Figure 1.3 highlight the MESI as the less suitable coherence protocol for multi-threaded applications, mainly due to the lack of the Owned (O) state. The Owned state enhances the performance in case of data ping-ponging effects between two L1 caches. Thus, each application shows a huge imposed traffic with poor performance (with MESI) if compared to the results obtained with the other coherence protocols. However, the trade-off between the traffic volume and the performance clearly emerges when MOESI and *MOESI-hammer* protocols are compared. The MOESI represents the MESI enhancement using a directory to keep the coherence. Conversely, the *MOESI-hammer* still implements the protocol optimizations over the MESI protocol, while it exploits a broadcast mechanism to retrieve coherence information instead of a directory. To this extent, the MOESI is traffic-aware due to its ability to get the information on the most up to date copy of each required data from a specific point in the cache hierarchy. On the other hand, the *MOESI-hammer* broadcasts to anyone in the multi-core to get the same information with a net traffic increase. The *MOESI-hammer* is faster in getting such information on average, thus offering better performance, mainly due to the possibility to directly get the required information, while the implementation of a directory represents and additional point of indirection.

The coherence protocol strongly influences the traffic volumes in the interconnect. Besides the traffic volumes can be seen as an indirect power consumption metric for the interconnect. This thesis primarily focuses on the embedded multi-cores, thus the MOESI directory-based protocol is used in the rest of this work instead of the MOESI-hammer, mainly due to the power consumption constraints that are imposed at the platform level. Finally, the thesis aims to extract valuable information from the MOESI coherence protocol to steer the DVFS mechanism.

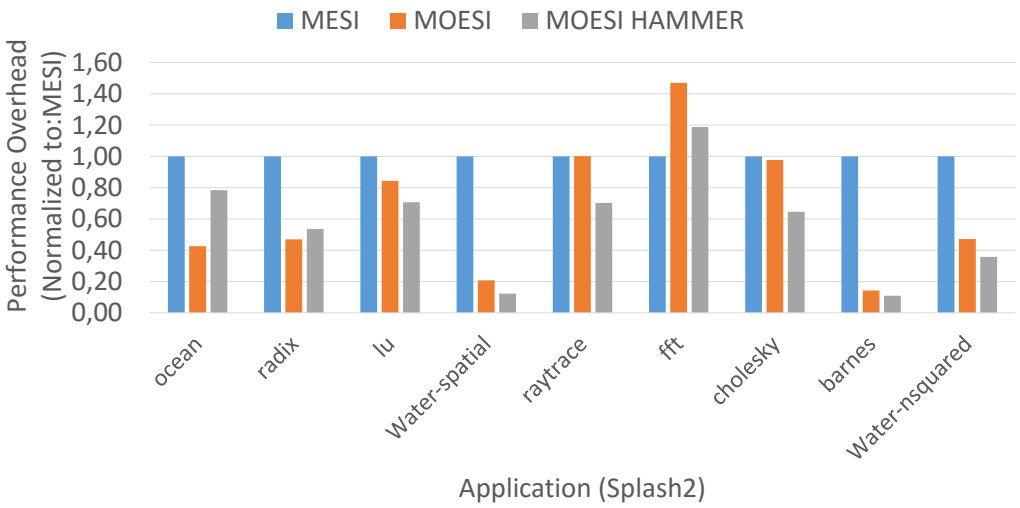


Figure 1.3: Performance overhead considering three cache coherence protocols. Results are normalized to the MESI.

1.3 Goals and Contributions

This thesis presents a novel heterogeneous NoC architecture for cache-coherent multi-cores. The traffic burstiness characteristics as well as the traffic information are exploited to design the NoC as well as to control the DVFS mechanism implemented at NoC level. In particular, the thesis encompasses the three main contributions:

- *Relationship between the interconnect load and the coherence protocol generated traffic* - The intricate relationship between the data required by the applications, the implemented coherence protocol and the impact that these two aspects have on the actual traffic on the interconnect have been investigated. Moreover, the insights of such an analysis are still valid to complement and extend the proposed interconnect.

- *Heterogeneous NoC design* - Two different, heterogeneous NoCs are used to route different traffic types. Such NoC design emerges from the traffic information analysis as well as the latency- and bandwidth-sensitive phases that the application traverses during its execution.

- *Application-aware DVFS mechanism and policy* - Using the knowledge about the application phases, its traffic distribution and the relation

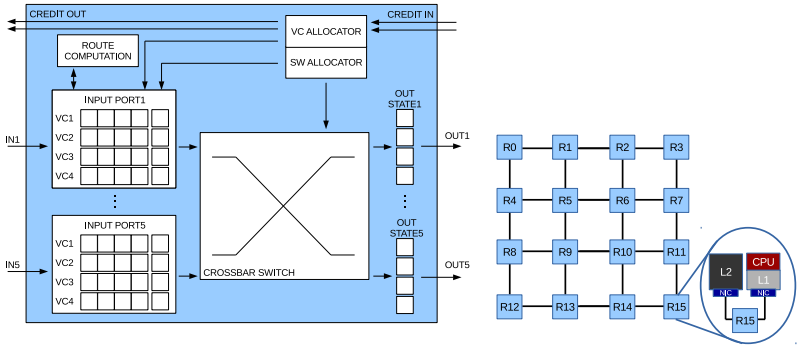


Figure 1.4: The baseline router architecture. It is presented a virtual-channel, wormhole, credit-based router highlighting its main components and how core and caches are connected to the relative router.

between the NoC frequency and the system-wide performance, the proposed NoC architecture can decide the best physical channel where the traffic has to be routed to get an optimal power performance trade-off.

1.4 Background

The Network-on-Chip is a scalable and reliable interconnect that allows nodes to exchange data. A node can be both a CPU or a part of the memory subsystem. The Noc is made of routers, links and Network Interface Controllers (NICs). Routers manage routing of data and coherence packets into the network. The NIC is the component that allow the communication between a node and a router. Two routers are connected using simple links or eventually using a router and a NIC.

Figure 1.4 depicts a Chip Multi-Processor (CMP) which relies on a 4x4 2D-mesh NoC as interconnect. Routers take care of the communication between the various CPUs and memories. Each router is connected to some other through a link. For example R_{15} is connected to router R_{11} and R_{14} through two different links. R_{15} is connected also to a L1 and a L2 cache memories; L1 is also connected to the CPU. Communication between them is supported by the NICs.

From the communication view point the various nodes exchange messages, which typically are requests and responses generated by the coherence protocol. Traditionally, the NoC splits each message in multiple packets.

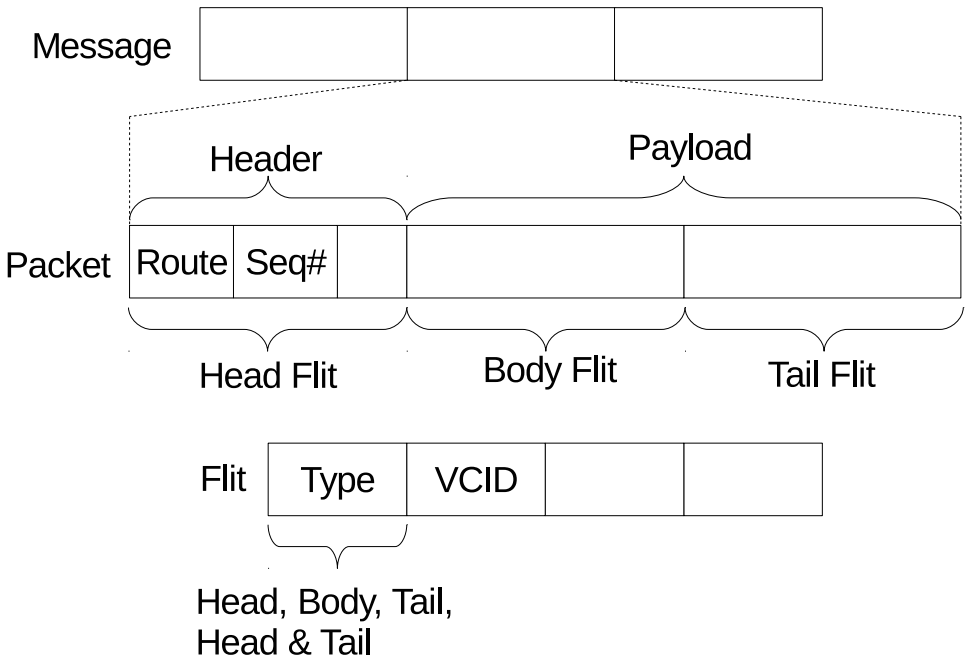


Figure 1.5: Message structure in NoC. There are presented the message, its division in packets and, furthermore, in flits highlighting the headers essential information.

Then each packet is eventually split in multiple flits to better utilize the NoC resources. These different levels of granularity are shown in Figure 1.5. Data and coherence packets have different dimension: Data packet are usually composed by 9 flits, instead coherence control packets are usually single flit packets.

The NoC itself is characterized by some features: the layout topology, the routing scheme, the switching mechanism, the flow control mechanism and the router architecture. The NoC topology defines the way routers are interconnected to each others and how memory and CPU blocks are attached to the NoC. The most common topologies used in NoC are mesh [14], concentrated mesh [3], hybrid bus based [15, 45] and high radix [22]. In Figure 1.4 it is shown a 2D-mesh network where the routers are connected in a matrix form.

Given a specific topology the routing algorithm defines each source-destination path inside the NoC. Routing algorithms can be deterministic [3] or adaptive [20, 18, 19], based on their capacity to alter the path taken for each packet.

The most used deterministic routing scheme in 2D-meshes is XY routing where a packet first goes in the X direction and then in the Y one.

The switching mechanism is in charge of the transmission of the information between the input and output ports inside a router. Some switching mechanisms are Store-and-Forward (SaF) [29], Virtual cut-through (VCT) [27], Wormhole switching (WH) [34] and Virtual-Channel Wormhole switching (VCW) [13]. The most commonly used switching technique is VCW, which associates several virtual channels with a single physical channel, overcoming blocking problems of WH.

The flow control is a mechanism responsible of managing the advance of the flits between the routers. All the switching techniques that use buffers need a mechanism to communicate the availability of buffers at the downstream router. Three mechanism are used: Credit-based, Stop & Go and Ack/Nack [14]. The most commonly used in NoCs is the Credit-based one and with this mechanism every output port knows the exact number of free buffers and the number of slots available for every buffer in the downstream router.

A message provides the node-to-node communication, but when it trespass the NIC it is divided into NoC's basic data structure: the packet. A packet is considered split in multiple atomic transmission units called flits. The first flit of each packet is the head flit. A body flit represents an intermediate flit of the original packet while the tail flit is unique for each packet and closes the packet. There is another particular case of packets called single flit packets and they are composed by a single head/tail flit.

After this general explanation of the NoC it is now presented in Figure 1.4 presents the baseline architecture of a wormhole NoC's router that is considered in the thesis.

A wormhole router supporting VCs and VNETs is considered. It is a standard 4-stage pipeline, i.e. Buffer Write/Route Computation (BW/RC), Virtual Channel Allocation (VA), Switch Allocation (SA), and Switch Traversal (ST). A single cycle for the Link Traversal (LT) is assumed as shown in Figure 1.6.

The considered NoC implements the VNET mechanism to support the coherence protocol, preventing the traffic from a VNET to be routed on a different one. When a head flit arrives to the input port of the router it has to pass through the 4 pipeline stages before traversing the link. First, it is stored in the VC buffer (BW) which has been reserved by the upstream

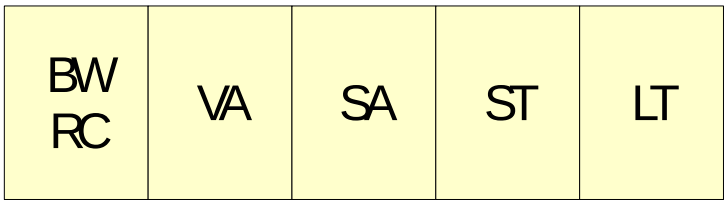


Figure 1.6: The baseline router pipeline. The figure shows the baseline NoC router 4-stages pipeline with the addition of the link traversal.

router, and the output port is computed (RC). Then, it competes in the VA stage to reserve an idle virtual channel in the selected output port. Notice that assigned VCs belong to the set of VCs associated to the VNET of the packet. Once the VA succeeds, head, body and tail, competes in packet order for a crossbar switch path (SA). Finally, each winning flit in the SA has to pass the ST and LT before reaching the next router. Note that tail and body flits pass through fewer stages since they reuse resources and information reserved by the head flit (i.e., VC and RC). n The NIC is another NoC component that provides communication between the nodes and the network. The NIC wraps up the requests from the cores as messages suitable for the NoC and rebuilds them at destination. When a message is taken from the NIC queue it is split into packets and each one of them is divided into flits. Then the whole packet divided in flits is allocated in a VC. Here all the flits will compete for the link allocation and then they will be sent to the next router.

The entity that creates the traffic in chip-multiprocessor (CMP) systems is the cache coherence protocol, thus influences the NoC behavior. CMPs typically implement a cache coherence protocol on top of the network (shared-memory CMP system). The protocol guarantees coherency and consistency when sharing memory blocks between multiple cores (processors). Indeed, the protocol keeps the coherency among different copies of the same memory block replicated through different L1 and/or L2 cache memories. The cache coherence protocol (in a shared-memory CMP system) is the main driver for the network. NoCs can be differentiate between systems with shared cache structures and systems with private cache structures. While the L1 cache is usually private for each processor, the L2 can be private or shared. In a system with private L2 caches each processor has its own L2 cache block and in case of a L2 miss, a request is sent to the memory controller. Later, the

request will be forwarded to another L2 cache or to memory. In a system with shared L2 caches the four L2 blocks are shared by all processors, and each cache block is mapped on a particular L2 bank. In case of a L1 miss, a request is sent to that bank, that can be located in the same tile or in one of the other tiles. This way the four banks of the L2 cache are part of a single L2 cache physically distributed among the tiles.

Two are the main different cache coherence protocols developed: Directory and Hammer, each one generating a different kind of network traffic.

In the Directory protocol directory information is associated with each cache block, including information about which core, if any, has a valid copy of the block, whats the current state of the block, and whether or not the block is dirty. In case of a miss in the private cache a request is sent to the home node, which forwards it to the owner core or sends the block to the requester depending on the blocks state. If L2 caches are private, the home node is located in the memory controller; if L2 caches are shared, each L2 bank is the home node to a given subset of cache blocks. Since the directory has to be stored in the home node, space overhead is the main drawback of this protocol. Cache misses generate low network traffic: in case of a read miss, two or three messages are sent: the request message to the home node, eventually a forwarded request to the owner core, and finally a message with the block; in case of a write miss, more messages may be sent since the home node has to invalidate the cores that share the block, if any.

In the Hammer protocol no additional information associated with the cache block is needed. When a cache miss occurs, a request is sent to the home node (which is located in the memory controller if L2 are private or in an L2 bank if L2 are shared, as in the Directory protocol). The home node broadcasts the request to all other cores and sends an additional off-chip request to the main memory. The cores answer by sending the block or, in case they do not have a valid copy, an acknowledgment message; so, if the system has N cores, the requesting core has to wait for N messages: N-1 data or acknowledgment messages from the other cores and one data message from the memory. If at least one core sends a data message, the memorys data is considered stale and ignored. The Hammer protocol generates even more traffic than the Token protocol, since broadcast operations are performed at every cache miss and all cores have to answer. All protocols use MOESI states: extension of MESI protocol, restrict write-back of data from cache to main memory.

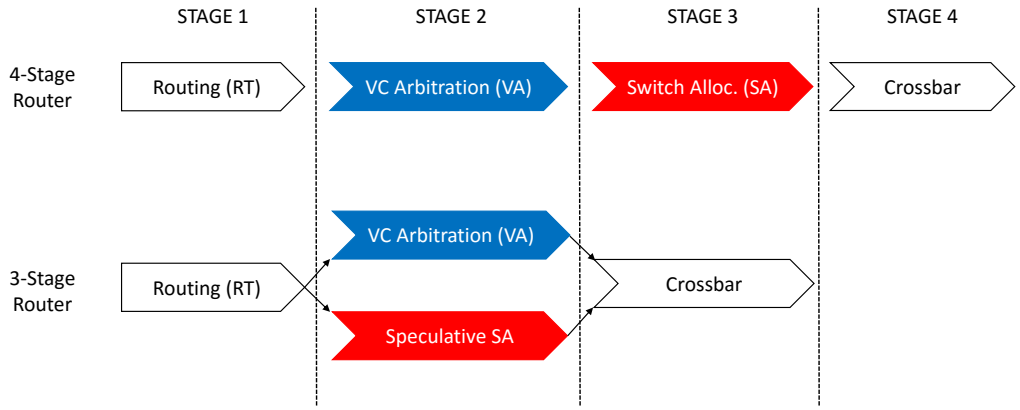


Figure 1.7: Comparison between baseline and speculative router pipeline.

Non atomic virtual channel allocation (NAVCA) - Usually VNET-based NoCs with Virtual Channels offer only atomic buffer allocation schemes, i.e. at most one packet can store flits in a buffer at the same time. However, a scenario with a mix of long and short packets, imposes a minimum buffer depth to face performance penalties due to the credit round trip time. In this scenario, short packets are usually single flit packets, since the link width is designed to transmit them quickly. As a consequence, short packets coupled with the atomic VC allocation impose a waste of buffering space, i.e. all the slots except the first are never used. The possibility to allocate a packet to a non empty-buffer using the Non Atomic Virtual Channel Allocation design improves both performance and buffer utilization.

Speculative Pipeline Optimization - We have used an optimized pipeline with only three stages as shown in Figure 1.7. This optimized pipeline has been also used in literature [37]. A speculative virtual-channel router arbitrates between output VC and switch bandwidth in parallel, this speculating that a VC will be allocated for the packet. If the speculation turns out to be incorrect, the SA speculation is wasted. As the switch allocator prioritizes non-speculative requests over speculative ones, there is no adverse impact on throughput.

1.4.1 DVFS: Dynamic Voltage and Frequency Scaling

Power consumption is one of the major design issue in multi-core processor development. Moreover, the CPU can no longer be considered the unique

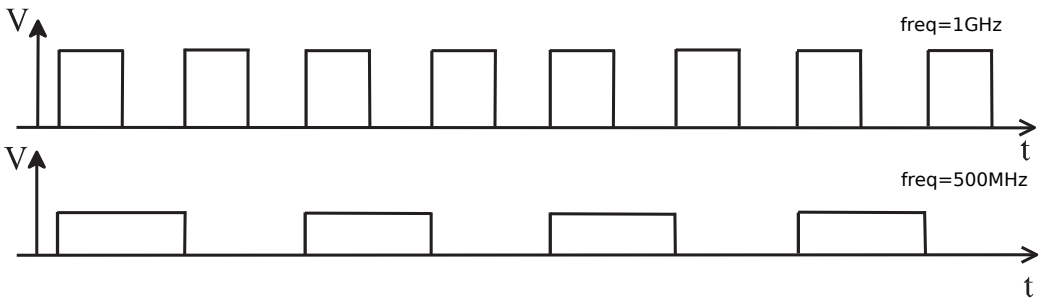


Figure 1.8: The impact of the frequency to the delay.

contributor due to the core, since the interconnect contribution to the overall energy budget is relevant. The DVFS has been exploited for power-performance optimization in cores [43] memories [10] and interconnect [24]. It provides a flexible and scalable way to jointly optimize power and performance, by addressing both static and dynamic power and dynamically adjusting the voltage and frequency values.

Dynamic Voltage and Frequency Scaling is an effective mean to reduce power consumption in CMOS chips. The power dissipation in digital Complementary Metal Oxide Semi-conductor (CMOS) circuits occurs due two main sources. Static power arising from bias and leakage current, it's dependent to the process and design technologies. Dynamic power is the other form and it get down from charging and discharging of voltage saved in node capacitances in the circuit. The dynamic power P depends on voltage V and frequency f. Formula (1.1) shows how dynamic power P depends on voltage V and frequency f.

$$P \propto fV^2 \tag{1.1}$$

The voltage value reduction implies the increase of the gate delay, thus reducing the operating frequency. Consequence of this relation cause a delay in transmission as shown in Figure 1.8.

The dynamic frequency change of different communication blocks requires a resynchronization support to avoid metastability issue and guarantee signal integrity. The *globally asynchronous locally synchronous* (GALS) design scheme allows to partition the design into different so-called *voltage and frequency island* (VFIs) considering signal resynchronization at VFI boundaries. Each VFI can work at its own speed, while the communication across different voltage islands is guaranteed through the use of FIFO resynchronizer.

This provides the flexibility to scale the frequency and voltage of various VFIs in order to minimize energy consumption. Design NoCs with multiple VFIs involves a number of critical steps: granularity, i.e. the number of different VFIs, and chip partitioning into VFIs needs to be determined since design time; also VFI partitioning need to be performed together with assigning of the supply and threshold voltages and the corresponding clock speeds for each island. Moreover, create islands of nodes running at different frequencies has some disadvantages: locally replicated DVFS controller and DC-DC converters cause area and power overhead, moreover resynchronization at the crossing between multiple frequency domains involves major performance losses. To allow reliable data transfers between components operating at different frequencies, a new component called FIFO resynchronizer must be insert between each Network Interface and router to avoid metastability issues. It is composed of a FIFO buffer with two clocks, i.e. write and read clocks, and two other signals, i.e. *isFull* and *isEmpty*, to monitor its current status of producer and consumer. This implementation allows to safely read and write it avoiding metastable states [36]. Figure 1.11 depicts the implemented FIFO resynchronizer between a router and a Network Interface Controller (NIC) that connect the CPU to the NOC.

In particular, considering a 8x8 2D-Mesh with 64 cores running at 2 GHz and the Splash2 applications, the introduction of the FIFO resynchronization model between NIC and the router introduces a performance penalty within 12%. It means that each physical Network can be his own frequency. Moreover, inside each Network each router must be at the same frequency (*FIFO NIC DUMMY NOC* architecture). If the architecture design imposes total flexibility, each router can work at different frequency. In this case, a resynchronizer must be placed between each router and each NIC with a performance degradation within 22% with an average value of 14%. We call this architecture *FIFO4ALL*. Figure 1.9 shows *FIFO4ALL* structure with a detailed view of resynchronizer position inside each node and between each router. Figure 1.10 reports the performance overhead considering the Splash2 benchmarks on an 8x8 NoC. Two resynchronization scheme have been tested. Results are normalized against the *NO RESYNCH* NoC implementation. Based on the results, our methodology supplies FIFO Resynchronizer for DVFS implementation only at NoCs border, allowing each Physical Network to run at its own frequency but inside each network each router must be at the same frequency. The goal is minimize the resynchronizer impact on both

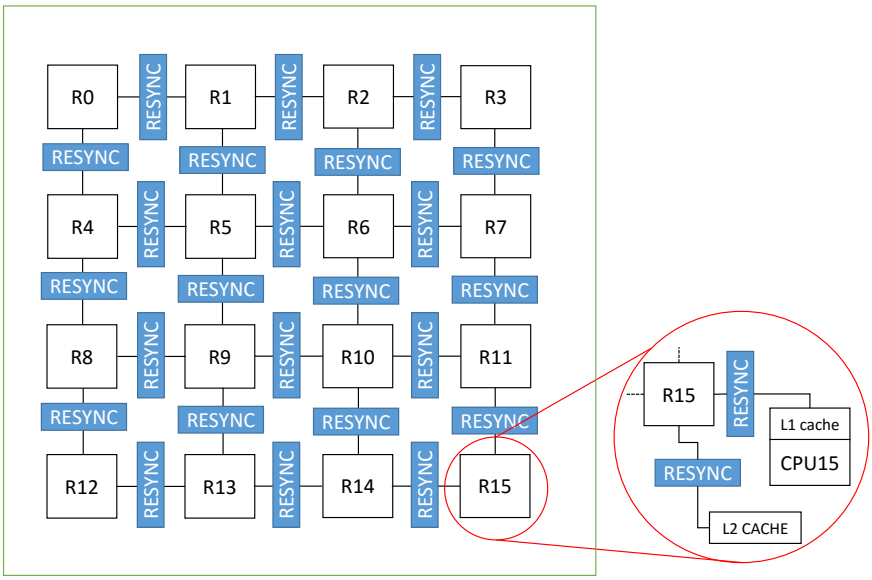


Figure 1.9: Logic view of *FIFO4ALL* architecture. The *FIFO NIC DUMMY NOC* only implements resynchronizers at the NoC border. No resynchronizers are implemented between routers.

power and performance.

In order to be able to change interconnect frequency at runtime, an actuator module must be introduced. The actuator is a PLL that allows to accurately select the router frequency. The PLL introduces timing and performance overheads. In particular, when a frequency increase is required, the voltage regulator is activated in advance to properly set the voltage in order to support the new frequency that the PLL is imposing. On the contrary, when the frequency is moved down the PLL immediately starts decreasing the frequency, while the voltage will follow it, without imposing any additional delay except the PLL response time.

It is worth noticing the resynchronization delay cannot be recovered in any way and its totally independent from the implemented DVFS policy. Due to this fact our methodology exploits a per-PNET DVFS. Independently from global or per-router implementation, metrics used to choose when change Voltage and Frequency during run time execution are essential to achieve the best power-performance trade-off.

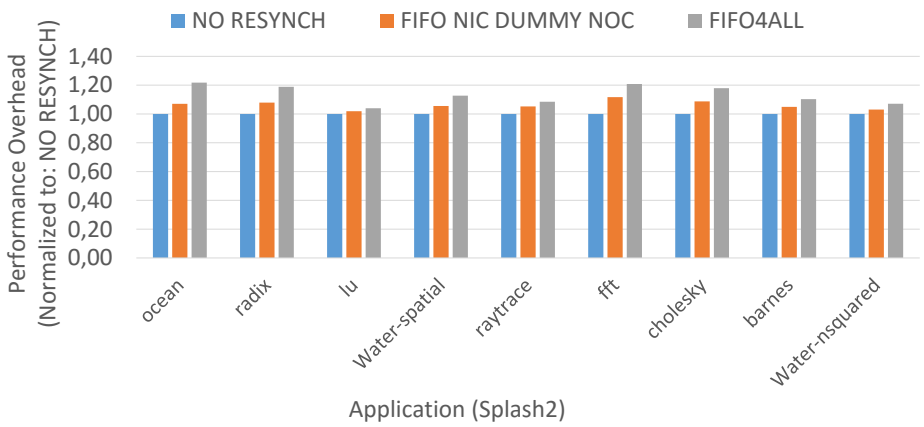


Figure 1.10: The performance impact due to different resynchronizations scheme. Results are normalized to *NO RESYNCH*.

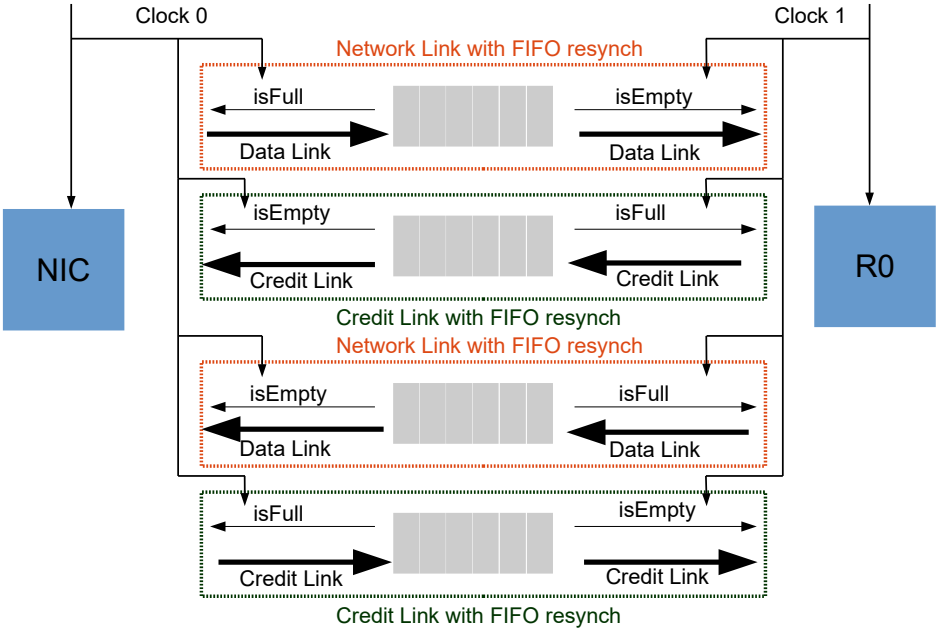


Figure 1.11: FIFO Resynchronizer between a Network Interface Controller (NIC) and a router. Four different FIFOs per each NIC-router pair are required, since unidirectional links are used for both data packets and flow control information.

1.5 Thesis Structure

The rest of the thesis is organized in five chapters. Chapter 2 describes the state of the art in DVFS, Heterogeneous Network applied to NoCs and cache coherence impacts on traffics. Chapter 3 provides a detailed description of methodology and its novel contributions. Chapter 4 details the methodology validation providing the results considering a realistic environment. Chapter 5 points out some future works on the NoC architecture.

Chapter 2

State of the Art

> *"You can't connect the dots looking forward; you can only connect them looking backwards. So you have to trust that the dots will somehow connect in your future. You have to trust in something - your gut, destiny, life, karma, whatever. This approach has never let me down, and it has made all the difference in my life."*
>
> Steve Jobs

This chapter summarizes the state of the art that is of interest in this thesis to improve the NoC power-performance trade-off. The literature is huge and different techniques and guidelines emerged to optimize the DVFS and the NoC architecture. The DVFS-aware NoC designs are discusses in Section 2.1. A review on the available, DVFS-aware simulation frameworks for NoCs is provided in Section 2.2. Section 2.3 discusses the architectural NoC optimizations to improve both power and performance. Last, Section 2.4 discusses the state of the art on the impact the coherence protocol has on the NoC due to the generated traffic.

2.1 The DVFS-aware NoC Design

Starting from single-core architectures, the DVFS emerged as a practical solution to deliver on-demand computational power to the running applications. With the multi-core revolution, the DVFS mechanism has been extensively exploited in the NoCs to dynamically adapt the platform computational power to the application needs to deliver an high-bandwidth, low

latency and power efficient interconnect. The rest of this section discusses both the mechanism and the companion policies proposed in the literature.

[33] explores different router-level policies to dynamically tune the NoC router frequency and optimize the power-performance trade-off. The per-input-port buffer utilization is the metric to decide the frequency to be applied to the whole router. More different policies have been evaluated. All routers operate at a boosted frequency. This helps in enhancing the performance at low load, with latency traffic, while slightly increasing the power consumption. When the state of the network become congested, the power consumption becomes the key challenge. Hence, the policy throttles the frequency/voltage of selected routers using the DVFS mechanism. The novel frequency tuning algorithm reduces the latency and the power consumption in the Noc by a distributed throttling and boosting of the router frequency based on the actual network load. Moreover, a distributed congestion management scheme has been implemented to provide each router with the possibility to signal its load to its neighbors. Thus, [33] provides a complete, distributed per-routed DVFS solution for NoCs.

A router-level, control-based methodology for DVFS-aware NoCs has been presented in [55]. The input buffer utilization is still exploited as the contention metric to decide the frequency to be applied to routers. An analytical model of the network traffic has been developed to better characterize the DVFS controller. The final solution exploits two different observations. First, the flow balance assumption equation expresses the connection between contention at current and next time instant. The router contention is defined as the sum of the number of flits in the input buffers and is also the quantity that has to be measured at runtime to close the control loop. A proportional controller has been implemented to tune the router frequency at run-time. It is worth noticing, the methodology eventually allows the user and/or the OS to change the high level power-performance modes, i.e. to trigger performance oriented or power saving system behaviors. Moreover, [55] presented a complete analysis on the impact the resynchronization scheme has on the overall methodology in terms of performance and power overheads. The impact of the resynchronization mechanism in DVFS capable NoCs has been further investigated in [53]. To this extent, the solution in [55] has been proved to overcome all the threshold-based DVFS policies like the one proposed in [33]. However, the used of a router-level DVFS scheme is overkilling in terms of performance and power due to the required resynchronization

scheme.

The thread voting approach proposed in [47] represents an on-chip DVFS technique for NoCs. It is able to adjust the NoC voltage and frequency (V/F) on a per region basis. Each region is composed of a subset of adjacent routers sharing the same V/F level. Considering a per thread voting-based approach, the threads can influence the DVFS mechanism by voting for the V/F level that best suits their QoSs. [47] used the thread performance level, instead of the classic network level parameters to steer the DVFS. The ingoing and outgoing traffic allows to capture the thread communication requirements. Moreover, it allows to monitor the thread data dependencies. The model uses two, thread-level parameters. One is the message generation rate that monitors the average number of outstanding L1/L2 misses. The other is the average number of data sharers for a specific data request. The first points out the per thread average network accesses, while the second reflects the contention on the specific required data. However, this work has a strong impact on the interconnect schema due to the necessity to add a novel infrastructure that allows the communication between the application threads and the policy controller.

2.2 State-of-the-Art on multi-core Simulator

The software simulation is the de-facto standard to evaluate complex multicores. Unlike hardware prototyping, it allows a relatively fast design space exploration. Besides, the obtained results greatly depend on the accuracy of the modeled components. In this scenario, the accuracy in modeling DVFS actuators and resynchronization circuits has a great impact on result reliability. The state-of-the-art provides several simulation frameworks but most of these are not accurate enough, providing DVFS and GALS implementations with power and performance behaviors that are far from reality. In particular, the resynchronizer is the hardware component that allows the communication between different VFIs. However, several reviewed simulation frameworks do not properly account its performance, power and area overheads, thus providing overoptimistic results that can partially shadow the benefit of the implemented DVFS scheme.

Table 2.1 reports different cycle accurate software simulators with particular emphasis on thier support for DVFS, GALS and NoC. *SESC* [39] develops a cycle-accurate to simulate of bus-based multicore, but without

Framework	Cycle-accurate CPU	NoC	Power	GALS	DVFS	PLL
Renau et al. (SESC) [39]	✓					
Soteriou et al. (Polaris) [42]		✓	✓			
Hsieh et al. (SST) [25]		✓	✓			
Lis et al. (HORNET) [30]		✓	✓	✓ (simple)		
Bartolini et al. [4]		✓	✓		✓	
Zoni et al. [49] (HANDS)	✓	✓	✓			
Carlson et al. (Sniper) [7]	✓	✓	✓		✓	
Prabhu (Ocin tsim) [38]		✓			✓	
Zoni et al. [53]	✓	✓	✓	✓ (accurate)	✓	✓

Table 2.1: State-of-the-Art simulation frameworks.

support the DVFS and the GALS design. *Polaris* [42] extends the concept of the NoC simulator providing tools for power and area design space exploration. Unfortunately *Polaris* does not support heterogeneous systems, and the addition of system-level reliability and variability models. *SST* [25] implemented a framework for integrated power, area and thermal simulations, but without providing support for DVFS or PLL models. *HORNET* [30] is meant to simulate large-scale architecture. The *HORNET* parallelized simulation engine can scale nearly linearly with the number of physical cores in the processor but does not provide support for any kind of power or area estimation. Bartolini [4] proposes a novel virtual platform for efficiently designing, evaluating and testing power, thermal and reliability closed-loop resource management solutions. [4] also supports the DVFS even if the PLL model is the inaccurate during frequency changes. *Sniper* [7], creates one of the most accurate distributed parallel simulator for multicores with DVFS projection and power support for CPU. Also in this work GALS support and PLL model are not takes in account into the models. *Ocin tsim* [38] offers a DVFS aware NoC simulator with support for per node power-frequency modeling to allow the fine-tuning of such optimization techniques early in the design cycle. Power management and different NoC policy are not allowed due to the limits of framework that restrict the use only for DVFS exploration purpose. *HANDS* [49, 53, 44] creates a complete and reliable cycle-accurate simulation framework with support for DVFS and GALS policy integrating accurate PLL models. The simulation flow allows to validate novel DVFS policies and NoC architectures with accurate results in terms of power and performance. Our novel heterogeneous NoC architecture has been tested with the last framework developed by Zoni [53] adding support for Physical Networks with independent DVFS interface for each channels.

2.3 Heterogeneous Network-on-Chip Design

The NoC architectural optimization represents an orthogonal design dimension to DVFS to improve the interconnect power-performance trade-off. Several proposals addressed the buffer sizing problem to optimize the NoC, since it strongly affects both power and performance [50, 35, 41]. Moreover, the NoC traffic is burst-oriented, thus it is of paramount importance to optimize the implemented NoC resources to minimize the idle time.

ViChaR [35] presents the design of a complete buffer size regulator to

dynamically resize the router buffers depending on the actual length of the stored packets. Buffer slots are assigned on a per flit basis following a daisy-chained list approach. Moreover, [11] proposed a *ViChaR*-based scheme where the dynamic buffer allocation is done on a per router basis. *ViChaR* represents the most flexible buffer management solution due to the exploitation of a per input port dynamic buffer slot allocation. However, its design complexity motivates the exploration of more lightweight techniques, since both area and power overheads greatly impact the benefit of the methodology. Besides, [35] highlights that having many VCs with shorter queues is more beneficial than having fewer VCs with a deeper queue when the traffic is low, while the opposite is true on heavy traffic conditions.

Elastistore [41] introduces a lightweight architecture that minimizes the buffering requirements without sacrificing the performance. It makes use of the buffer space in the pipelined channels to reduce the buffer cost. *Elastistore* uses just one buffer slot per VC and a large buffer that is shared between all the VCs in the same input port. Thus, different VCs can dynamically increase their buffer size where the only constraints are imposed by the total available, per-input port, shared buffer slots.

[50] exploits the coherence protocol information to dynamically assign different VCs to serve different traffic classes in the NoC. The coherence protocol imposes the minimum number of required VCs that is equal to the implemented message classes. Additional VCs can be implemented to improve the NoC performance. [50] dynamically assigns the VCs to the message classes depending on the actual NoC traffic, with the only constraint to have al least one VC per message class. The dynamic VC allocation allows to dedicate more channels to the message class with the heavier traffic volume. Moreover, the adaptivity of the mechanism can reconfigure the VC allocation depending on the traffic changes.

[32] proposes an heterogeneous NoC by implementing two different routers. Routers with deeper buffers are implemented in the central part of the 2D-mesh topology, while short buffer routers are used in the topology borders. The methodology also accounts for different combinations of small and big routers while keeping the same bisection bandwidth and the same amount of link resources compared to the baseline homogeneous NoC. *HeteroNoc* shows that placing big routers along the diagonal provides maximum benefits compared to an equivalent homogeneous network in terms of power and performance. However, [32] proposed a design that greatly exploits the XY

routing algorithm. The XY is a deterministic routing algorithm that is common in 2D-mesh NoC due to its simplicity. Besides, it imposes a severe traffic imbalance between the central and the peripheral parts of the NoC, thus its perfectly fits with the proposed methodology.

On a different but related point, [48] investigated the performance, area and power metrics comparing two different interconnect designs. One implements a single NoC that can route all the coherence traffic. The other is composed of multiple physical NoCs, where each of them can route a specific coherence traffic class. The possibility to have multiple physical NoCs can greatly reduce the contention on the shared interconnect resources, since each coherence class of traffic is routed on a physically different NoC. Moreover, it positively affect the overall system performance. Conversely, a single NoC allows a better resource utilization, since the design is globally optimized with respect to all the traffic types at once. However, the higher resource contention can negatively affect the performance.

[3] proposes a dual physical NoC to explore two different traffic distribution policies. The first policy uses a specific network to separately route read and write transactions. Moreover, it allows to equally split long packets to the two networks. A second policy splits the traffic between data (long packets) and control (short packets). However, it is not able to correctly balance the traffic between the two physical networks, thus some resources goes underutilized while others are over-utilized.

The heterogeneous NoC design in [31] exploits different traffic distribution information that have been captured during the application execution. An application traverses at least two phases during its own execution. The low traffic phase shows few outstanding cache/memory requests/responses. The high traffic phase shows high network load and contention in the NoC. Such phases are unevenly distributed during the application execution and the possibility to foreseen them can greatly improve the performance. To this extent, [31] presents an heterogeneous NoC design and policy to exploit such a feature. The interconnect is composed of two physical heterogeneous NoCs. Each injected packet is classified as bandwidth or latency sensitive at run-time, then routed to one specific physical NoC. One NoC provides low bandwidth and high frequency, while the other has a wider bandwidth but operates at lower frequency. The application-aware nature of this architecture implies a dependency between interconnect and applications.

darkNoC [6] presents a novel multi-layered NoC. Each layer is physi-

cally separated from the others and optimized to operate within a particular voltage-frequency range. The optimization of each layer to work within a specific frequency range provides a more efficient solution than DVFS in terms of performance and power consumption, since specific cell libraries can be exploited. However, the use of multiple parallel NoCs is costly in terms of area and power compared to the use of a single or dual NoC. [6] developed also an efficient network-layer switch-over mechanism to better support the application execution. All the layer in a region are managed by a hardware based *darkNoC Layer Manager* (dLM). The collaboration between router components is essential to support the correct exchange between different frequency-level of execution. For this reasons router stack is managed by another hardware component called *darkRouter Manager* (dRM). *dLM* and *dRM* autonomously coordinate each other to realize an efficient mechanism in terms of energy and time overhead, transparent to software and with aforementioned requirements. *DarkNoC* overcomes traditional DVFS solutions in terms of saved energy. However, it shows an area and performance overhead up to 30% and 40% respectively.

2.4 Cache coherence impact on traffic

The coherence traffic exploitation represents a unique source of information to further optimize the DVFS policies at NoC level. However, the concurrent execution of different applications and the traffic mix at NoC level make hard to collect valuable information to be used to optimize the NoC-based DVFS policies. A review of the literature in this direction is investigated in the rest of this section.

[2] presents *SynFull*, a synthetic traffic generator framework to explore the coherence traffic to later improve the NoC design. *SynFull* analyzes the cache coherence traffic and the applications behaviors in several directions. Cache coherence messages are a combination of requests and responses. Small control packets and large data packets coexist in the interconnect. The possibility to tight different packets to the same information flow allows to arise further information on the required interconnect resources. Furthermore, some messages are more critical than others. Thus, their fast delivery to destination greatly affects the overall system performance. Applications exhibit different phases during their execution, thus imposing a time-dependent NoC load. Exploiting the coherence information that links request and response

messages, it is possible to partially foreseen such phases in advance.

[23] presents an NoC traffic prediction scheme based on the cache-coherence communication properties. Starting from the insights in [17], the methodology investigates the correlation between the application synchronization points and coherence communication. The methodology is composed of two modules. The *Coherence Prediction Engine (CPE)* extracts the relevant data from the coherence protocol and sends them to the *Accumulation and Decision Module (ADM)*. The prediction mechanism is sync-point based, thus it splits the program into different sync-epochs. The characteristics of each sync-epoch are stored in the Prediction Table that is part of the *ADM*. Such information are used to find the correlations between different current sync-epochs and the historical information. The *ADM* keeps track of all the aggregate bandwidth requirements.

Aergia [16] develops a novel router priorization mechanisms based on the *packet slack* definition. The *Packet Slack* is the number of cycles the packet can be delayed in the network without affecting the application execution time. It is used to prioritize the critical packets - with low slack values - against the ones having a larger slack value. However, the *packet's slack* estimation is challenging, since several timing information of the packet are required at run-time. *Aergia* exploits a combination of three statistics to overcome such an issue: the L2 cache hit and miss and slack in terms of number of hops.

The coherence protocol information strongly correlate with the imposed network traffic as well as with the required network bandwidth. The ability to correctly predict the traffic shape enables a new interconnect optimization level. However, such prediction is hard due to the intricate nature between different coherence messages that are highly parallel by nature and their cardinality.

Chapter 3

Novel Heterogeneous NoC Design

> *"The number one benefit of information technology is that it empowers people to do what they want to do. It lets people be creative. It lets people be productive. It lets people learn things they didn't think they could learn before, and so in a sense it is all about potential."*
>
> Steve Ballmer

This chapter presents a novel, DVFS-capable, heterogeneous NoC architecture. The design is complemented with a policy, that exploits the DVFS mechanism and the traffic information to optimize the NoC power-performance trade-off. The proposed solution can thus adapt the NoC channel utilization depending on the run-time traffic conditions. In a nutshell, the proposed design methodology sits on three different pillars:

- exploiting the cache coherence information to optimally route the traffic within the heterogeneous NoC;
- a DVFS mechanism to dynamically adapt the offered interconnect bandwidth depending on the traffic imposed by the application;
- an interconnect made of multiple, heterogeneous physical NoCs to flexibly adapt to different traffic conditions.

The rest of the chapter is organized in five parts. The architecture pillars are provided in Section 3.1. Section 3.2 presents the methodology key concepts.

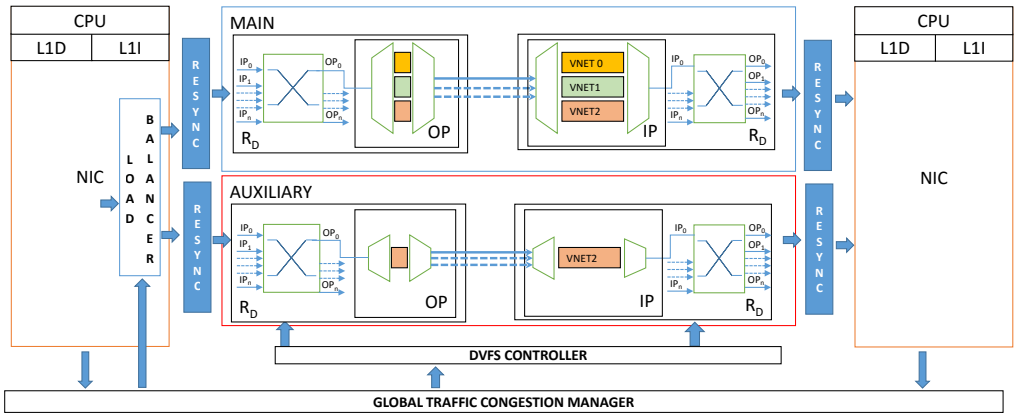


Figure 3.1: Overview of the proposed NoC architecture.

Section 3.3 discusses the exploitation of such key concepts in the novel interconnect. Section 3.4 presents the policy and the DVFS mechanism. Last, Section 3.5 overviews several use case scenario.

3.1 Architecture pillars

Figure 3.1 shows the architectural implementation of the novel NoC architecture. Our architecture focuses on data intensive contention management providing an heterogeneous network with two different Multiple Physical Networks (PNET). Each Physical Network is designed to work with specific traffic scenarios. The *main* network is specialized for single flit packet and light data traffic. In this network there are three different virtual networks to avoid deadlock as imposed by the coherence protocol. Each Virtual Network manages a different coherence message subset, as detailed in Table 3.1. The *auxiliary* network implements one single Virtual Networks and manage the long data packets flow during high contention phases. The DVFS controller provides an efficient frequency and voltage scaling on the auxiliary network. The GTCM collects traffic statistics from all the interconnect both main and auxiliary NoCs. The GTCM elaborates the received data and manages the behavior of the DVFS controller and the Load Balancer. When the traffic is low and the resources are underutilized, the GTCM reduces the frequency of the auxiliary network at 500MHz. At the same time, the Load Balance does not use the auxiliary network and routes all the traffic to the main

network. On the contrary, when the traffic is high the GTCM allows the Load Balancer to route the long packets to both the main and the auxiliary networks in a round-robin fashion. The GTCM senses the NoC contention level to decide the number of active NoCs. The contention is defined as the pressure on the shared NoC resources due to the injected traffic. The Load Balancer balances the traffic between the main and the auxiliary network at runtime. This helps our scheme to capture within-application variation in latency and bandwidth phases. The DVFS controller imposes the same frequency for all the routers in the NoC in the same network. For this reason, the resynchronizer are places between the Network Interface and the Physical Network but not between each router. The main network also needs a resynchronizer to operate at a different frequency with respect to the CPU. The continuous cooperation between the GTCM, the Load Balancer and the DVFS controller is the key of the proposed architecture.

Controller	Message Type	Direction	VNET
DMA	ResponseFromDir	From	2
DMA	reqTodir	To	1
DMA	respToDir	To	2
L2CACHE	L1RequestFromL2Cache	To	0
L2CACHE	GlobalRequestFromL2Cache	To	1
L2CACHE	ResponseFromL2Cache	To	2
L2CACHE	L1RequestToL2Cache	From	0
L2CACHE	GlobalRequestToL2Cache	From	1
L2CACHE	responseToL2cache	From	2
L1CACHE	requestFromL1Cache	To	0
L1CACHE	responseFromL1Cache	To	2
L1CACHE	requestToL1Cache	From	0
L1CACHE	responseToL1Cache	From	2
Directory	requestToDir	From	1
Directory	responseToDir	From	2
Directory	forwardFromDir	To	1
Directory	ResponseFromDir	To	2

Table 3.1: The allocation of the cache coherence messages to the Virtual Network classes

3.2 Design key points

This section describes the main contributions of the methodology, from an abstract viewpoint.

Unbalanced traffics - The cache coherence protocol regulates the packet distribution in the interconnect. In particular, the cache coherence protocol is made of a set of coherence controllers that generate coherence messages to ensure the coherence invariants. Simply put coherence messages falls into two categories: control and data messages. Such messages directly translates into packets that are injected in the interconnect. Furthermore, the Network Interface splits each packet into small pieces called *flits*. Data packets are composed by 9 flits, instead control packets are single flit. The networks carries both data and control packets. Each protocol imposes a specific type of data packets distribution. The reaction of the protocol to cache miss or a coherence request is based on the adopted mechanism. In this perspective, the number of long and short packets is distributed in different way between every network. Directory based protocols try to reduce the number of cache coherence messages. Conversely broadcast-based protocols produce an higher number of messages due to the lack of the directory. Obviously different behaviors have impact on the NoC traffic shape. Considering a Virtual Channel-based NoC, it is extremely difficult to extract per message type information, since all the traffic is mixed into the same physical interconnect. The total number of flits injected in each specific channel class is different for each cache coherence protocol with the same application as shown in Figure 3.2. The *MOESI* protocol improves the *MESI*, since it generates lower traffic while ensuring better performance. However, the *MOESI* is not able to reach the performance of the *MOESI HAMMER*. The *MOESI HAMMER* is a broadcast-based cache coherence protocol. Broadcast-based protocols generate a high amount of traffic to improve performance with a not negligible impact on the power consumption. Considering *MOESI*, *MESI*, and *MOESI HAMMER* cache coherence protocols, Figure 3.3 shows that short packets are distributed between all the available channels. On the other hand, long packets always flow throw a single Physical Network. This network greatly impacts the total power consumption, since all of its packets are made of multiple flits. By changing the cache coherence protocol a deep impact on traffic shapes is observed. See the difference between standard directory-based *MOESI* and broadcast-based *MOESI HAMMER* protocol in

Figure 3.3;

The exploitation of the cache coherence information is essential to allocate the right number of buffers and virtual/physical channels to improve the power-performance trade-off. The proposed architecture sits on the MOESI directory-based protocol. It implements a smart interconnect architecture taking advance of the traffics shape information.

Applications phases - The application behavior is another key factor in the design of DVFS capable interconnect. Each application traverses several phases during its execution. This directly impacts the NoC traffic. Each application shows more alternative different phases as highlighted in Figure 3.4. The memory bound phases are *bandwidth-sensitive* and require a high NoC bandwidth to process the load/store memory requests from the CPUs. During these phases, the CPU is likely stalled waiting for L2 and memory requests to be served. The application needs a significant bandwidth from the network to make progress during the memory-bound phases because of a large number of requests that are sent out into the NoC. Thus, the progress is less sensitive to the network latency. On the contrary, the high computational phases are *latency-sensitive* and need less bandwidth but are more sensible to the network latency. During these phases, the application only has a small number of outstanding memory requests. The CPU has all the data necessary to progress its execution and the computational throughput is high. During intensive CPU bound periods, resources are underutilized because the coherence traffic is low. Instead, the network load increases during the application memory-bound phases. Each application has a different combination of CPU-intensive and memory-intensive phases. Latency and bandwidth sensitive phases require different operating frequencies. Latency sensitive phases benefit from a high frequency, despite their lower NoC resource requirements, because a low frequency critically affects the performance of the few ongoing coherence messages. On the other hand, the performance penalty is directly proportional to the network contention during the bandwidth phases. In this phase, the frequency changes based on the bandwidth requirements thus allows for power reduction, still ensuring a reasonable performance drop. These information can provide an implicit application clustering. The application phases are crucial for an efficient DVFS-based NoC design. However, a specific architectural support, to better manage the application phases is essential to boost the performance, while reducing the power consumption.

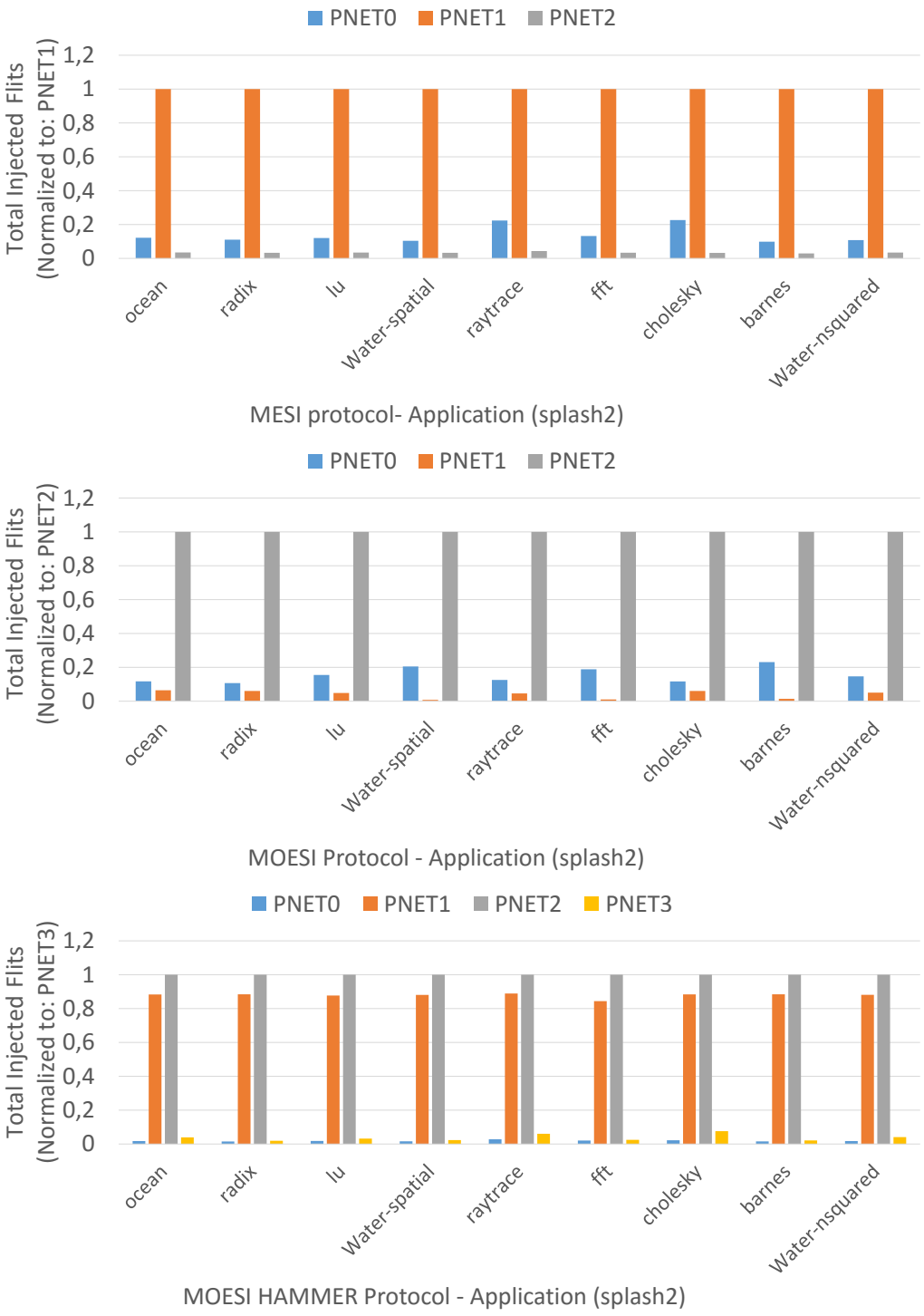
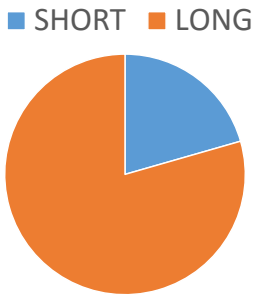
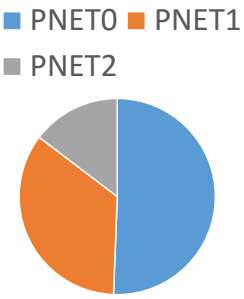


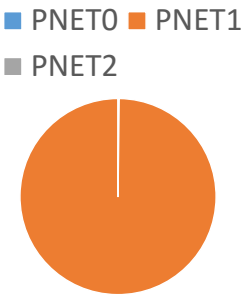
Figure 3.2: A comparison of the total injected flit for each cache coherence protocol. The different number of injected flits is reported for each Physical Network.



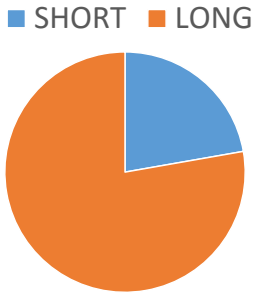
(a) The ratio between long and short packets express in flits with MESI



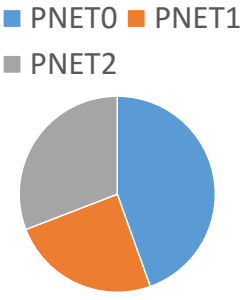
(b) Detailed distribution of single flit packet with MESI



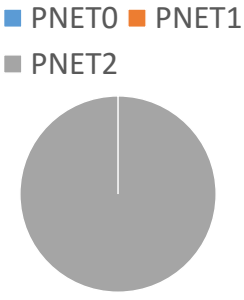
(c) Detailed distribution of long packets with MESI



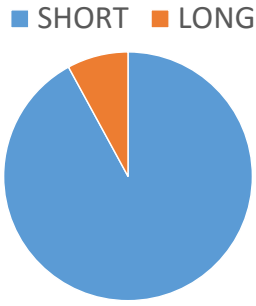
(d) The ratio between long and short packets express in flits with MOESI



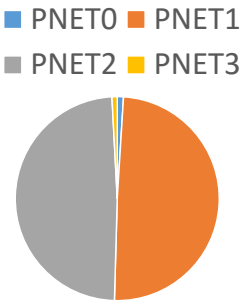
(e) Detailed distribution of single flit packet with MOESI



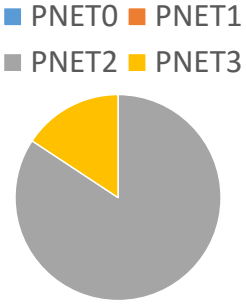
(f) Detailed distribution of long packets with MESI



(g) The ratio between long and short packets express in flits with MOESI-HAMMER

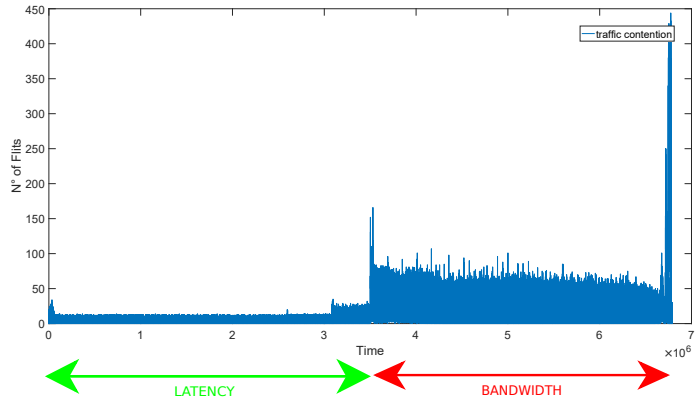


(h) Detailed distribution of single flit packet with MOESI-HAMMER

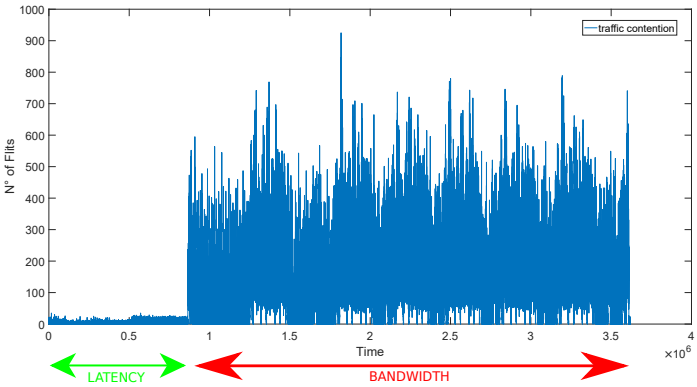


(i) Detailed distribution of long packets with MOESI-HAMMER

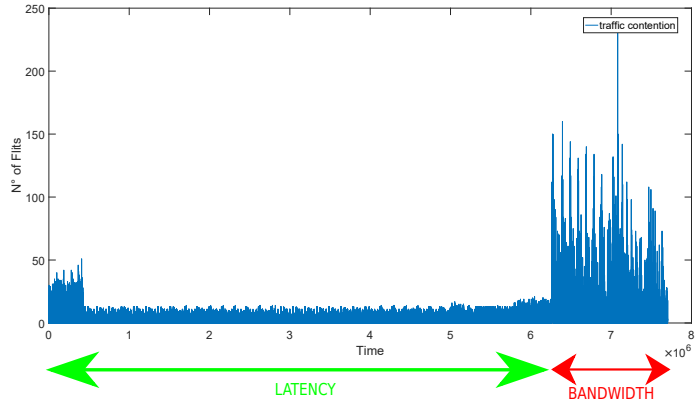
Figure 3.3: Flits distribution considering long and short packets as well as the three different protocols.



(a) *FFT*



(b) *OCEAN*



(c) *LU*

Figure 3.4: Different execution phases are reported considering three different applications.

3.3 The Heterogeneous NoC Architecture

This section describes the novel policy and NoC architecture developed as part of this work. The virtual network implementation improves the resource utilization. However, different traffic types are mixed together, thus competing for the same resources. The use of Physical Networks design reduces the resource contention, while providing an unbalanced resource utilization. From long data packets point of view with bandwidth-intensive demand, few but long buffers are better. On the other hand, single flit packets have opposite requirements, they need more buffers with small queues. However, the dynamic behavior of the applications impose a strong variability in the memory access patterns thus making generic homogeneous network design almost useless. Reducing the buffer size to claim both power and area is not a viable solution due to the intricate relation between buffer size and performance.

3.3.1 Exploit the heterogeneous network architecture

The preliminary results on a valuable set of multi-threaded applications, highlights the variability in the message distribution during the benchmarks execution. In some cases, data packets are dominant and unevenly distributed during the observed periods. The exploitation of such information is a pillar on which our architecture is based. The baseline architecture implements homogeneous networks with homogeneous buffers. The introduction of dedicated physical networks represents a solution to manage specific traffic patterns. MPs heterogeneous design exploiting simple networks components and operating independently with less contention. Long packets generate a lot of contention and taking up many resources during bandwidth sensitive phases. Furthermore, the combinations of high traffics and contention leads more power consumption and worse performance. To achieve power saving and ensuring performance, our novel architecture provides a dedicated auxiliary physical data network that is used during bandwidth-sensitive phases. This improvements increase the available bandwidth for data packets flows only during this phase. Furthermore, the auxiliary network is DVFS-capable and the DVFS mechanism can be applied to each MP in isolation. Our NoC architecture uses only the main network during low bandwidth requirement phase and switch on the auxiliary network during contention phase. More-

over, the superior adaptability exploits DVFS-capable auxiliary network to provides more flexibility in order to cover all the possible requirements in terms of bandwidth and frequency offers by the resources available. However, a negative impact on the power metric is expected during the low traffic phases, i.e. latency sensitive, CPU-bound periods. For this reason, the main network is fixed at 1.5GHz during low contention phases to support latency-sensitive traffic.

3.3.2 The Load Balancer

The policy controller continuously analyzes the traffic for all the NoC physical networks. As we have seen in section 1.2, application phases can be divided in two classes:

- Latency sensitive - This phase is characterized by a low data packet traffic and limited use of the NoC resources. In this phase the interconnect is not stressed and the auxiliary network is not in use.
- Bandwidth sensitive - Intensive Memory phases are characterized by high traffic in the NoC. In this phase, the NoC is not negligible and can reduce the overall system performance. Thus, the additional data network is exploited to relieve the stress on the main NoC.

Our methodology provides a *load balancer* to better manage the use of two physical networks where the data packets can flow. During the latency phases, the load balancer does not use the auxiliary Physical Network and redirects all the data flow to the main PNET. The Load Balancer exploits the auxiliary network to reduce the data traffic on the main NoC. However, more power and resources are requested during bandwidth-sensitive phases. The controller provides a smart use of the available resources to balance the traffic load between the two Physical Networks. The controller checks the traffic load to trigger the use of the additional network. If the contention parameter is under a specific threshold, the Load Balancer uses the main network only to accommodate all the incoming packets. On the contrary, if the contention is high, long and single flit packets are dispatched in a different fashion. Single flit packets will be always sent to the main NoC. The Load Balancer routes the long packets to the main network and the auxiliary network following a round robin approach.

The Load Balancer allocation policy chooses the correct Physical Network where to route the incoming long packets. Algorithm 1 explains the controller logic.

Algorithm 1 Load Balancer allocation policy

$LB_{k,t}$:= Load Balancer in Network Interface k at time t
GS_t := Global contention State GS at time t
S_p := Size of packet p

```
if GS_t == LOW then
    return PNET_MAIN
else
    if S_p == 1 then
        return PNET_MAIN
    else
        if LB_{k,t} == 0 then
            LB_{k,t+1} ← 1
            return PNET_MAIN
        else
            LB_{k,t+1} ← 0
            return PNET_AUXILIARY
        end if
    end if
end if
```

The Network interface implements the round robin Load Balancer policy. The policy checks at each activation the global contention state. The global contention state shows if the system is in the bandwidth or latency sensitive phase. Single flit packets always flows throw the main network. A LOW contention state means a latency sensitive phase. In this phase, also the data packets are routed in the main Physical Network. Otherwise, the Load balanced manages long data flits packets balancing the destination routes using the round robin mechanism during the $HIGH$ contention state.

3.4 The Adaptive Policy

The *Global Traffic Contention Manager* (GTCM) is the controller and command dispatcher of our architecture. It collects the on the fly flits in all the network interfaces. Every single network interface senses the traffics using a register to store the number of flits injected into the physical network and received from the network during a period. It makes samples to estimate the contention condition and decides if the application is in bandwidth-sensitive phase or latency-sensitive phase.

All the values collected by Network interfaces are sum together in the GTCM.

The variance is the metric used to discriminate between bandwidth-sensitive and latency-sensitive phases.

The GTCM works by analyzing the total traffic contention in all the nodes using last 10 sampled values. $FINJ_{n,p,t}$ defines the number of flits injected by the Network interface n on the PNET p at time t. $FINJ_{n,p,t}$ defines the number of flits received by the Network interface n on the PNET p at time t. TC_{t+1} represents the contention of the network, defined as the sum of the number of the on the fly flits. Algorithm 2 describes how GTCM collects all data used to calculate the variance.

Algorithm 2 Calculate the total contention in all networks

$FINJ_{n,p,t}$:= Number of the flits injected by the Network interface n on the PNET p at time t
$FRIC_{n,p,t}$:= Number of the flits received by the Network interface n on the PNET p at time t
TC_t := Total contention at time t
for all NIC in NoC **do**
  for all $PNET$ in NIC **do**
    TC_{t+1} += ($FINJ_{NIC,PNET,t}$ - $FRIC_{NIC,PNET,t}$)
  end for
end for
return TC_{t+1}

Algorithm 3 shows how the controller extract the variance from the samples.

The variance represents the runtime fluctuation of the traffic during the application execution. The variance express the capacity of the networks

Algorithm 3 Variance calculation

$MEAN_t$:= Mean at time t
$VARIANCE_t$:= Variance at time t
TC_t := Total contention at time t
W_{size} := Windows size, number or sample
for all TC_t in $Windows$ **do**
    $MEAN_t$ += TCt
end for
$MEAN_t$=$MEAN_t$ / W_{size}
for all TC_t in $Windows$ **do**
    $VARIANCE_{t+1}$ += $(TC_t - MEAN_t)^2$
end for
$VARIANCE_{t+1}$ += $VARIANCE_t$ / W_{size}
return $VARIANCE_{t+1}$

to dispose packets in time without allowing the queues to settle. An high variance means that the traffic fluctuating at high rate, thus an ad-hoc tracking mechanism is required to optimally trade power and performance. The global decision policy is the mechanism to check variance evolution. It eventually requests to switch the additional physical networks on. Algorithm 4 highlights this mechanism.

Algorithm 4 Global Congestion Setting

GS_t := Global Contention State GS at time t
$VARIANCE_t$:= Variance at time t
if $VARIANCE_t >= QoS_{Threshold}$ **then**
    $GS_{t+1} \leftarrow HIGH$
else
    $GS_{t+1} \leftarrow LOW$
end if

The variance triggers the change in Global contention state. When the variance exceeds the threshold imposed by QoS management the contention state is set to $HIGH$. Instead, when the variance is under the threshold, application traverses latency sensitive phase and the contention state is set to LOW.

3.4.1 Quality of service related issues

The correlation between the performance, the available resources, the frequency and the power makes difficult to choose the architecture parameters at design time. As we have seen in Section 1.2, embedded multi-cores can be used different scenarios where different requirements must be satisfied at the same time. Despite the hardware implementation should remain transparent providing a more efficient solution with respect to the software and the operating system, some parameters can be customized in order to improve the final user experience. Our NoC architecture provides a parameter to select the desired power-performance trade-off level. It takes steps from current Operating System where the user can decide the power-performance trade-off. Section 3.4 presented the *Variance* metric that is used to choose between high performance and bandwidth-sensitive operation modes. Our policy based its decision using a particular threshold to switch on or off the auxiliary data network. This parameter can be chosen by OS or the user, in order to change completely the policy behavior. In particular, three different variance thresholds have been evaluated:

- Variance Threshold at 0 (HIGH PERFORMANCE)- Using this parameter, GTCM set the contention state to high. Thus, the auxiliary network is always enable. The Load balancer always provides a balance of the data injection in the two PNET. Using this particular configuration the DVFS is disabled and the auxiliary Physical Network always works at the same fixed frequency of the main network.

- Variance Threshold at ∞ (HIGH POWER SAVING) - The orthogonality between power and performance metrics, implies that to get the optimal power saving, we impact performance. Using an high threshold value the auxiliary network is always off regardless the application phases. The results is a performance drop with a power saving.

- Variance Threshold at 16 (BALANCED) - A balanced setup tries to reduce the power consumption when the resources are not required. On the other hand, it consumes more power to reduce the contention, thus providing a network performance speedup during memory bound application phases. Preliminary results on the *Splash2* benchmark suggested 16 as a reasonable variance threshold value. It is worth noticing

16 represent a fixed parameter given a specific 8x8 2D-Mesh architecture. A new training phase is required if the architecture is changed. When this threshold is under 16 the network is not congested.

3.4.2 The DVFS smart switching policy

Our methodology takes into account the resynchronization impact between the NoC and the rest of the system, since they operate at different frequency/voltage values. In particular, the auxiliary NoC is DVFS capable while the rest of the system and the main NoC work at fixed but different frequency/voltage values.

Equation (3.1) describes the proposed metric focused on contention reduction.

$$C_{VAR}(t) = \frac{\sum_{i=0}^{n}\sum_{p=0}^{m}\sum_{v=0}^{k} C_{n,p,v}(t) - \sum_{i=0}^{n}\sum_{p=0}^{m}\sum_{v=0}^{k} C_{n,p,v}(t-1)}{\sum_{i=0}^{n}\sum_{p=0}^{m}\sum_{v=0}^{k} C_{n,p,v}(t-1)} \tag{3.1}$$

The DVFS policy regulates the auxiliary NoC based on the whole network traffic. The frequency is a controllable input, i.e. the controller knob. Moreover, the ingoing ($InFlits_{i,p}(t)$) and outgoing ($OutFlits_{i,p}(t)$) flits from the network interfaces's control volume i on each PNET p are not controllable inputs, since they depend on the NoC activity. The contention $C_{i,p,v}(t)$ for network interface i on Physical Network p and Virtual Network v at time t represents the state of a single NIC, defined as the sum of the number of the on the fly flits. $C_{VAR}(t)$ is the core of our DVFS schema explain how contention changes over time. n is the number of network interface, m is the number of Physical Networks, and k is the number of Virtual Networks.

It is worth noticing that the needs to measure contention at runtime requires communication between network interfaces to exchange the informations. Decentralized control scheme achieves this purpose. Furthermore, the DVFS controller calculates the contention using all the Physical Networks and all the respective Virtual Networks. A global estimation of all the traffic contention is the best way to manage the global traffic condition.

Algorithm 5 explains the operations managed by DVFS Controller.

The policy increases the frequency every time that C_{VAR} is higher than a particular thresholds in order to increase the available bandwidth. The DVFS controller can change frequency of the auxiliary network from 500MHz up to

Algorithm 5 DVFS Controller Policy

```
Thresold_HIGH := Threshold to increase frequency t
Thresold_LOW := Threshold to decrease frequency t
if C_VAR(t) >= Thresold_HIGH then
    if FREQ_p(t)! = FREQ_MAX then
        FREQ_p(t + 1) ← FREQ_p(t) + 250MHz
    end if
else
    if −C_VAR(t) >= Thresold_LOW then
        if FREQ_p(t)! = FREQ_MIN then
            FREQ_p(t + 1) ← FREQ_p(t) − 250MHz
        end if
    end if
end if
```

2GHz with steps of 250MHz. The controller can only increase the frequency by one step of 250MHz at once regardless of the contention metric. The minimum time period accepted by our technology model is near ∼50ns as reported in [28]. Over this period the DVFS manager realigns the frequency to the new value. The power gating represents an additional actuator to further reduce the power consumption of the auxiliary NoC when it is not in use. Traditionally, different power gating schemes have been proposed to face the aging mechanisms [51, 52], i.e. BTI, or to deliver aggressively power-aware architectures [1]. However, the power gating introduces a non negligible performance overhead due to the need to wake up the gated component before using it. Moreover, an additional power overhead is introduced when the gated component is active due to the more complex power delivery network that has to steer the power gating mechanism. Thus, the Near Threashold Computing (NTC) research area highlights the possibility to put the idle component in a so-called retention state to save power [9]. The retention state, also called Near Threashold State, allows to keep the correct data value and eventually provide a modest computing power to the component while the power consumption is dramatically reduced. The proposed architecture does not explicitly exploit these opportunities since the focus of the work is on the architectural design issues during the non-idle time periods.

3.5 Simulation Framework Overview

This section discusses some implementation and feasibility details of the presented novel heterogeneous architecture. The new NoC architecture has been integrated in an event-based cycle accurate simulator. The event-based-simulators model the execution of a determined component using events that can be directly scheduled on the component itself. This means that at each clock cycle each component in the simulated architecture, eventually processes its own events. It can generate a new event on a subsequent component to be processed after one or several cycles in the future. Our policy exploits this particular feature to check at each sample the status of the network in order to set the proper behavior of the policy. Policy Controller provide a reschedule of the policy every 10ns ∼100MHz. The presented methodology does not impact the critical path of the router, since it is logically in parallel to each pipeline stage. From the implementation viewpoint, *GEM5* [5] is the exploited event-driven simulator for multi-core architectures with NoC-Based communication. *Orion2.0* is used as the power model for the NoC, while additional components has been added to support accurate *GALS* and DVFS models. Figure 3.5 reports the information flow between the various components to support the DVFS architecture. The execution is driven by both the architectural events as well as the events generated by the policy modules that directly interact with *GEM5*. The *policy module* represents the controller of our methodology in the runtime box. It reads power and performance data and returns the frequency and voltage values to be applied to the auxiliary NoC.

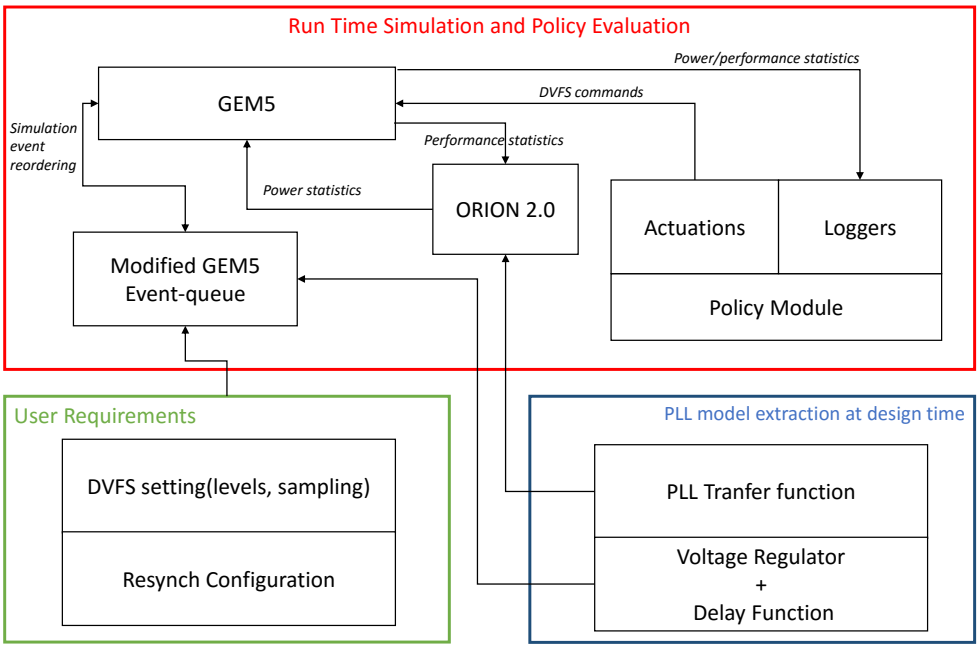


Figure 3.5: Simulation framework overview.

Chapter 4

Results

"It's fine to celebrate success but it is more important to heed the lessons of failure."

Bill Gates

This chapter discusses the assessment of the proposed heterogeneous NoC design from the performance and power consumption view point. Moreover, the energy delay product is reported as a power-performance aggregate metric.

The rest of the chapter is organized as follows. Section 4.1 describes the simulation environment setup, the target architecture and the used benchmarks . Section 4.2 details the performance results exploiting the *Splash2* applications [46]. Section 4.3 presents the energy consumption results. Section 4.4 shows the Energy Delay Product (EDP) analysis. Last, the exploration of different values for the tunable parameter of the proposed NoC is discussed in Section 4.5.

4.1 Simulation Setup

The novel architecture has been integrated in the enhanced version [54, 53, 12] of the *GEM5* cycle accurate simulator [5]. Table 4.1 reports the architectural parameters used for the simulations. A 64-core architecture at $45\text{nm}/V_{dd} = 1.1V$ technology is considered. The auxiliary physical network supports the DVFS that runs between $500MHz$ and $2GHz$ with steps of $250MHz$. 3 different levels are used with the following frequency/voltage relationship:

- $f >= 1500MHzV_{dd} = 1.1V$
- $100MHz <= f < 1500MHzV_{dd} = 1.0V$
- $f < 1000MhzV_{dd} = 0.9V$

Different voltage levels are necessary to support the frequency changes at runtime in accord with the PLL model [54]. The modeled DVFS takes into account the timing overhead due to frequency changes. Moreover, the voltage regulator imposes [28] a $5ns$ delay to increase the voltage. On the contrary, if the policy imposes a frequency reduction the new applied V_{dd} value does not cause a delay in the new frequency setup since the frequency/voltage feasibility relationship is preserved. The policy is periodically executed every $10ns$ [28]. *Orion 2.0* power model [26] has been used to extract power data of the simulated NoC architecture. We assume a MOESI-based coherence protocol that enforces 3 Virtual Network to avoid protocol-level deadlock.

The *Splash2* suite provides multi-thread applications. 9 applications are showed in Table 4.2 among the whole suite. The cache configuration is a 32KB L1 caches and a 64KB L2 cache per bank. The NoC topology is a 8x8 2D-mesh with 64 cores and NoC routers with a different number of VC based on the used architecture. Our solution implements 1 VC per VNET, while the baseline NoC and *RAFT-like* has 2 VCs per VNET. All architectures provides a buffer depth of 4 flits with a *NAVCA* degree of 4 for the control buffers and 7 flits for the data buffer.

4.1.1 Policy comparison

The proposed architecture has been tested against three different competitors. A baseline *energy-aware* architecture with the NoC limited at 500MHz and a single VC per VNET is used as the reference lower-bound for the power. The same baseline architecture with 2 VCs per VNET and the NoC fixed at 1.5GHz is used as the *performance-aware* reference. Last, the *RAFT-like* [33] solution represents the state of the art DVFS-capable NoC proposal. In the rest of the chapter we reference to the *RAFT-like* implementation as *RAFT*. Table 4.3 summarizes the architecture configurations. The proposed heterogeneous NoC implements 2 PNETs. The *Main* NoC has three Virtual Networks, while the auxiliary NoC has a single data network. All the Virtual Networks in our architecture implement single VC. *RAFT* exploits

Processor Core	2GHz, Out-of-Order Alpha Core
L1I Cache	32kB 2-way Set Associative
L1D Cache	64kB 2-way Set Associative
L2 Cache	64Kb per bamk, 8-way Set-Associative
Coherence Prot.	MOESI /at least 3 VNET protocol)
Router	Speculative 3-stage Wormhole Virtual Channelled 32bit Link Width
Technology	45nm at 1.1V - 1.0V - 0.9V
Real Traffic	Subset of the Splash2 Benchmarks.
Policy period	10ns
PLL transient	10ns
Voltage regulator transient	50ns

Table 4.1: Experimental setup: processor and router micro-architectures and technology parameters

Splash2 Applications	Domain	Problem Size
Barnes	High-Performance Computing	65,536 particles
Cholesky	High-Performance Computing	tk14.O
FFT	Signal Processing	16,304 data points
LU	High-Performance Computing	1024x1024 matrix, 64x64 blocks
Ocean	High-Performance Computing	514x514 grid
Radix	General	524.288 integers
Raytrace	Graphics	car
Water-Spatial	High-Performance Computing	512 molecules
Water-Nsquared	High-Performance Computing	512 molecules

Table 4.2: The *Splash2* subset used to evaluate the methodology. The applications domain and the inputs are also provided.

Policy	PNET available	VNET available for each PNET	VC available for each VNET	Total channels available	Frequency Range
Power-aware Baseline	3	1	1	3	1,5 GHz no DVFS
Performace-aware Baseline	3	1	2	6	1,5GHz no DVFS
RAFT	1	3	2	6	From 1,2GHz to 2GHz
Our Policy (BALANCED)	2	3 for PNET1 and 1 for PNET2	1	4	From 500MHz to 2GHz

Table 4.3: The simulated architectures.

the DVFS to optimize the power consumption without compromise performance. During high contention phases the congested routers warn the DVFS controller that reacts changing router frequency based on 4 different levels. When the total buffer utilization for all the ports into the router is under 40% of the available buffer space, the frequency is reduced to 80% of the baseline frequency. The increasing of the buffer occupation within 50% imposes a frequency increase to the baseline level. Another increase is forced when the contention takes the buffer occupation between 50% and 60% leading the frequency to the 85% of boost frequency, i.e. at 1,7GHz. When the buffer occupation is above 60%, the frequency is set at boost frequency, i.e. 2GHz. Again, when the contention of the congested router drops under 40%, a low contention signal resets the frequency to the normal operating level.

4.2 Performance Analysis

This section provides the performance results of our NoC heterogeneous architecture compared to the power-aware baseline, *RAFT* and the performance-aware baseline.

Results of the simulation are reported in Figure 4.1. 9 *Splash2* benchmarks are reported on the x axis. For each benchmark we have 4 different columns each of these represents an architecture. The simulation time normalized to the power-aware baseline architecture is reported in the y axis. The metric used to compare the performance of the architecture is the total execution time to complete the application. Our methodology outperforms the *Power-Aware baseline* architecture by 44% on average with a peak of 77% with the *Ocean* benchmark. It is due to the fact that *Power-Aware baseline* architecture is not able to scale the frequency, that is fixed at 500 MHz. On the other hand, our solution has the main NoC fixed at 1.5GHz and the auxiliary network provides an additional channel for data packets that can reach

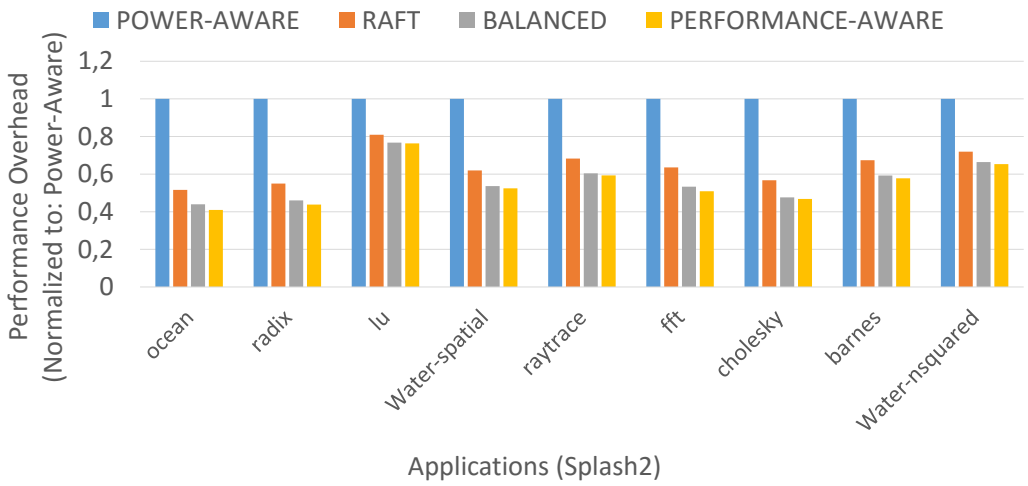


Figure 4.1: Performance results. Results are normalized to the *Power-Aware baseline* architecture.

up to 2GHz. However the possibility to increase the frequency is not the only advance that we have since, if we compare our solution with the *RAFT* and *Performance-Aware baseline*, we can still outperform both of them. In this case the main advantage of our architecture is the possibility of increase frequency of each physical network based on the different load of the traffics. The adaptivity of our solution combined with a reduced number of Voltage Frequency Island (VFI) compared to *RAFT* make us to very closer to the performance upper bound. Conversely we do have a performance boost of 13% an average with a peak of 16% in *Radix* compared with *RAFT*. Performance results shows how our policy gets very close to the best results obtained by the *Performance-Aware baseline* with an average performance penalty of about 3% using only 4 Virtual Networks instead of 6. The Load Balancer steers all the traffic to the main network during latency sensitive phases because the frequency is high and fixed, and the performance are proportional to the applied frequency. The performance improvement during these phases is primarily due to the network delay reduction. Moreover, the reduction of the bandwidth do not cause a reduction of the performance. Thus, our architecture has the same performance of the *Performance-Aware baseline* architecture during these phases using less resources, i.e. the auxiliary network is set at its lowest frequency and it is not used. On the contrary, less available resources are the main cause of the performance loss during the bandwidth sensitive phase. On the other hand, the DVFS controller boosts

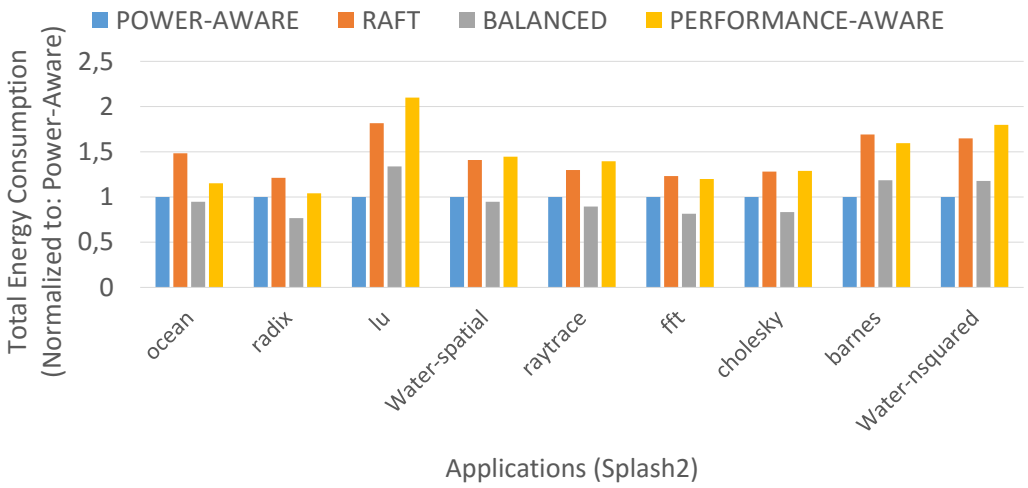


Figure 4.2: Energy results. Results are normalized to the *Power-Aware baseline* architecture.

Splash2 application	Ocean	Radix	Lu	Water-Spatial	Raytrace	FFT	Cholesky	Barnes	Water-Nsquared
Auxiliary Network ON (%)	63,30	35,89	5,19	13,60	1,10	31,67	5,12	59,71	11,69
Auxiliary Network OFF (%)	36,70	64,11	94,81	86,40	98,9	68,33	94,88	40,29	88,31
Mostly (ON > 30%)	Bandwidth	Bandwidth	Latency	Latency	Latency	Bandwidth	Latency	Bandwidth	Latency

Table 4.4: Percentage of time that the auxiliary network is on for the applications evaluated.

the frequency of the auxiliary network up to 2GHz when the contention is high.

4.3 Energy Analysis

Energy consumption results are shown in Figure 4.2.

The four different columns for each benchmark on the X axis represent the four considered architectures. The energy consumption is on the ordinate normalized using the power-aware baseline architecture. We adopted the *Energy Per Instruction*(EPI) metric to compare alternative architectures. The *EPI* is the energy per instruction dissipated by the system during the execution of an application. The energy consumption is extracted using

Orion2 [26]. The total energy is obtained as the sum of routers dynamic power, routers clock power and routers static power multiplied for the total number of instructions executed. The results show that our architecture is always near the power-aware baseline as energy consumption on average. In the energy metric the power consumption is considered equally of performance. For this reason the energy results shows a difference of only 2% on average between our architecture and the power-aware baseline. On the other hand, the limited number of resynchronizer combines with an effective DVFS policy and load balancer module allow to stay closer to the performance-aware architecture in terms of performance consuming much less energy. Respect to the performance-aware solution we save 31% of energy on average with a peak of 36% in *Lu*. *RAFT* is more power hungry due to the additional resynchronizer and the additional implemented buffers that are not properly exploited. For this reasons, *RAFT* is not able to overcome all the other architecture from the performance viewpoint. Respect to *RAFT* we consume 33% less energy on average with a peak of 37% in *Radix*. The most important aspect of our architecture is the smart use of the resources. Our architecture uses the auxiliary NoC to boost performance only when required. Table 4.4 the auxiliary NoC active time in percentage with respect to the whole benchmark execution. *Bandwidth* definition implies an intensive use of the auxiliary network that can improve the performance but not necessarily. In 5 benchmarks *(Lu, Water spatial, Raytrace, Cholesky, Water-Nsqared)* the auxiliary NoC is *ON* for less than 15% of the time. In 2 Benchmarks *(Radix, FFT)* the auxiliary network is *ON* between 30% and 40%. Considering *Ocean* and *Barnes*, the auxiliary network is *ON* for more than 50%. Thus 5 out of 9 *Splash2* benchmarks are prevalent latency sensitive, while 2 are mainly bandwidth-sensitive. In this benchmarks the action of our policy is more strong due to the fact that adding a second NoC during bandwidth sensitive phases can speed up the performance by reducing the contention. The action of the Load Balancer is to deviate the traffic by splitting the packets between the two available networks. At the same time, the DVFS can save more power by switching the frequency of the auxiliary network between 500MHz and 2GHz values to better match the actual traffic load.

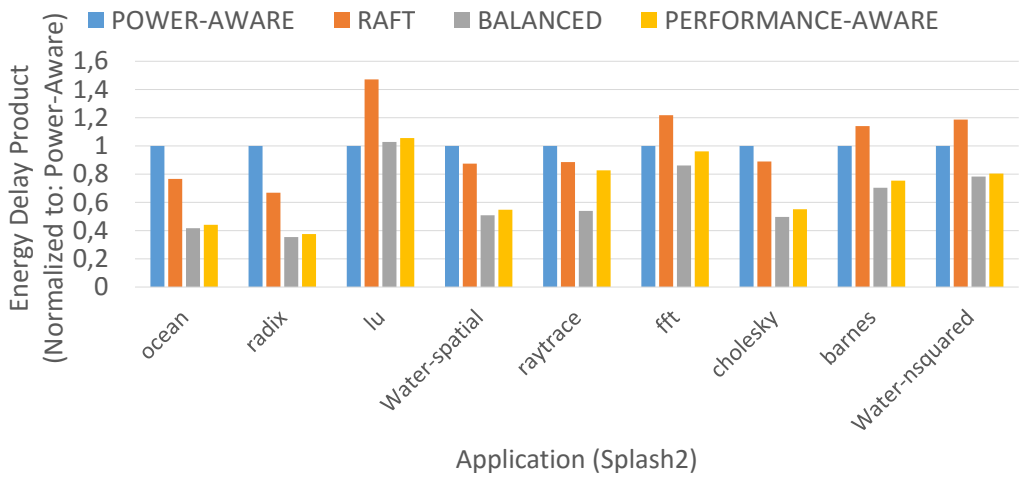


Figure 4.3: Energy Delay Product Results. Results are normalized to the *Power-Aware baseline* architecture.

4.4 Energy Delay Product Analysis

In this section we discuss the *Energy Delay Product* (EDP) results provides in Figure 4.3.

The X axis displays the *Splash2* benchmarks used for the evaluation. Four columns are reported for each benchmark, each column represents an architecture exposed in Section 4.1.1. The Y axis shows the metric used for comparison. We adopted the *Energy Delay Product* metric to compare alternative system architectures. Here, both performance and power are considered equally relevant and should be captured by a general and flexible goal function. The EDP product for an architecture a and a benchmark k is defined as:

$$EDP(a,k) = EPI(a,k) \times CPI(a,k) \tag{4.1}$$

Where the $EPI(a,k)$ is the energy per instruction dissipated by the system in the architecture a during the execution of application k and $CPI(a,k)$ is the average number of clock cycles necessary to the architecture a to execute an instruction of the benchmark k. The EDP is normalized respect to the *power-aware baseline*. The results shows that the EDP of our methodology is always better than others on average. Our EDP is 36% better respect to the power-aware baseline on average with a peak of 65% on *Radix*. However, with respect to *RAFT* and *Power-Aware baseline* our EDP is better

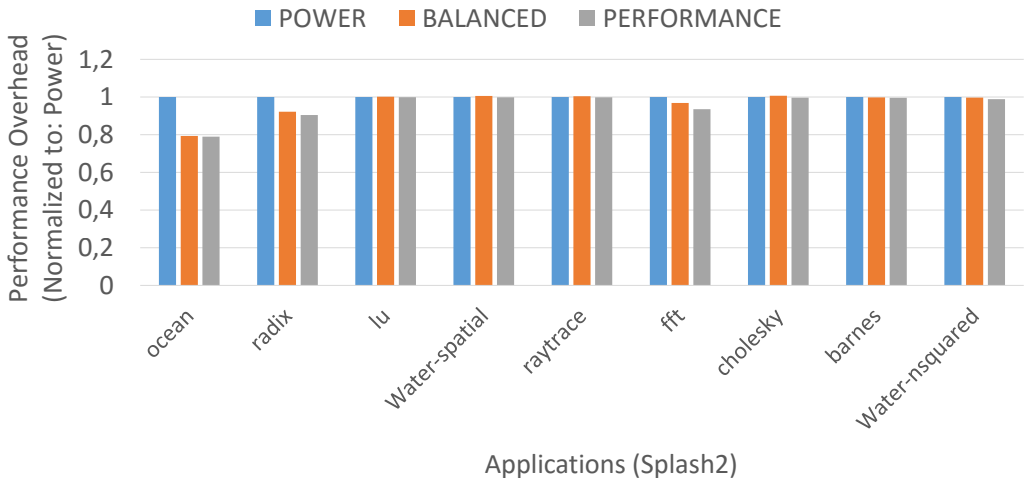


Figure 4.4: Performance results of the proposed architecture considering three different variance values.

respectively than 38% and 9% on average. Both power-aware and *RAFT* do have a high *EDP*. The power-aware baseline imposes a performance drop of 84% on average. Thus negatively impacts on *EDP* results. Instead, *RAFT* is an over-dimensioned architecture that cannot efficiently exploit the available resources. *RAFT* is negatively affected by the resynchronizer and the local frequency scaling scheme that assign the frequency/voltage values an a per router basis thus losing global optimization opportunities. Furthermore, *RAFT* is the worst result in many cases. This shows that the necessity of adding resynchronizers have a direct and negative impact on both performance and power. The possibility of increasing and decreasing frequency based on the actual traffic is not able to cover the losses caused by the resynchronizer.

4.5 Sensitivity Analysis

This section focuses on the benefits introduced by the auxiliary network on both performance and power metrics. In particular performance generally increases as the portion of the time the auxiliary NoC is kept active. Of course the power consumption follows the same relationship, while the objective is to keep it limited. Last, the increase in the auxiliary NoC active time does not always guarantee a performance improvement. The performance im-

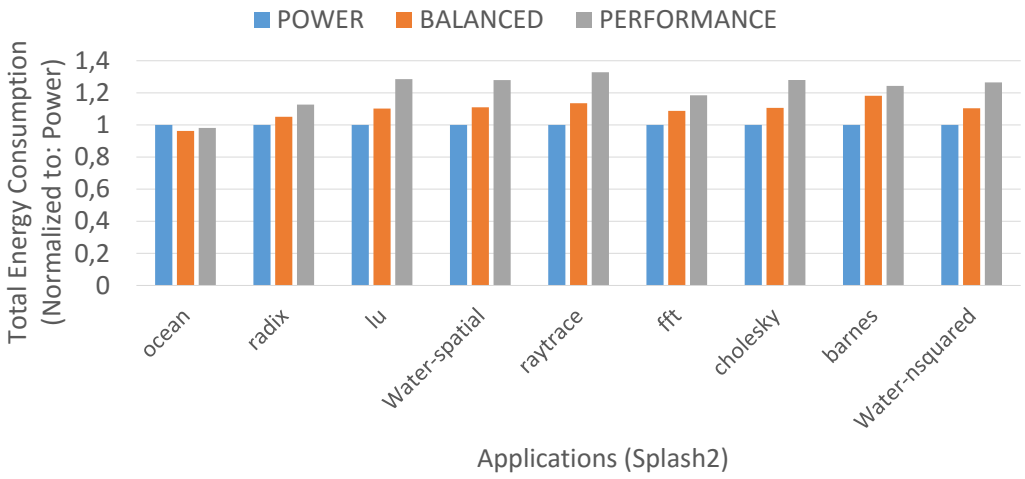


Figure 4.5: Energy results of the proposed architecture considering three different variance values.

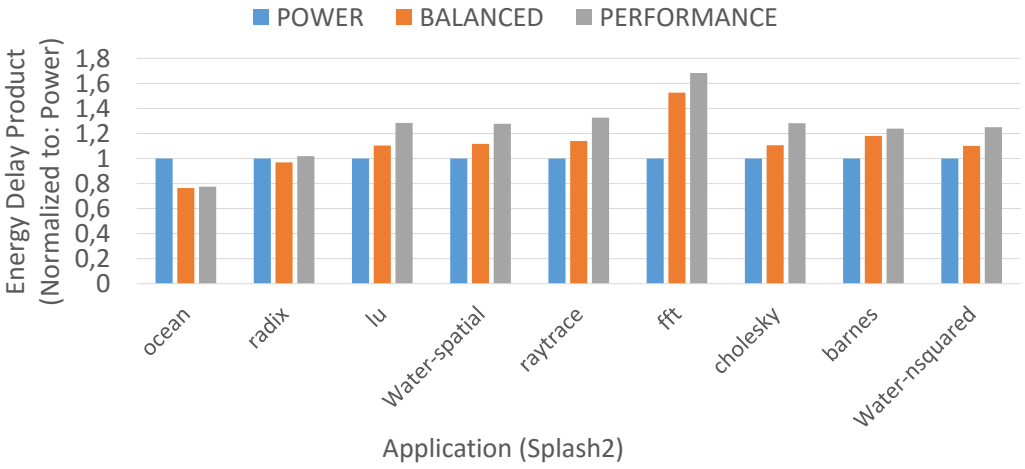


Figure 4.6: Energy Delay Product results of the proposed architecture considering three different variance values.

provement is function of the offered NoC bandwidth and the actual injected traffic. Thus keeping the auxiliary NoC always ON providing no performance boost if the application does not inject several data packets. Table 4.4 shows which are the benchmarks with higher ON time. On the other hand, excessive use of the auxiliary network have a negative impact on the power consumption. Our architecture provide a tuning parameter to change the ON/OFF ration for the auxiliary network. The implemented architecture fixes the threshold parameters at 16. However, this parameter can be sets to two extreme values to obtain a more performance aware or power aware architecture. A variance threshold set at ∞ provides a single NoC architecture where the energy saving is the main goal. On the contrary, a variance threshold set at 0 always uses the auxiliary NoC to provide more bandwidth. To avoid performance throttling during *latency-sensitive* phases the variance threshold set to 0 disables DVFS and fixes the frequency of the auxiliary network at 1.5GHz. Table 4.5 shows the architecture behavior with the three different values. Results shows how our architecture can dynamically adapt to exploit possible performance speed up using traffic load information. Table 4.4 shows that *OCEAN* is the most bandwidth sensitive benchmarks in *Splash2* suite. The auxiliary network is ON for more than 60% during this benchmark. Also *RADIX*, *BARNES* and *FFT* keep the auxiliary network ON for more than 30% of the execution time but with different results. Figure 4.4 shows the difference in terms of performance with the different setups. While *OCEAN*, *RADIX* and *FFT* improve their performance using more bandwidth, both with the balanced and performance setups, *BARNES* cannot speed the execution up with more available bandwidth. The sensitivity analysis can accurately tracks the actual load to always provide the best frequency/voltage values. Figure 4.5 reports a 10% energy saving on average for the balanced setup compared to the other two architectures. Thus, when the auxiliary network is on, the DVFS can reduce the energy consumption with an impact on performance. Furthermore, Figure 4.6 highlights that the use of the auxiliary network can provide a low performance increase respect to the consumed energy.

Name	Variance Parameter	Details
POWER	∞	The policy never switch on the second Physical Network. This configuration is used to save much power as possible but with the maximum penalty in terms of performance.
BALANCED	100	Balanced configuration is the best power-performance tradeoff parameters that we have found to detect contention phases during execution time.
PERFORMANCE	0	Variance threshold equal to 0 always keep on the second network. Disable the DVFS and fixed frequency to 1.5GHz.

Table 4.5: Different variance values for the proposed architecture.

Chapter 5

Conclusions and Future Works

In NoC based multi-cores the applications stress the interconnect in a variable and time dependent fashion. In particular, each application traverses different phases during its execution. The *Bandwidth-sensitive phase* is characterized by high traffic flows and intensive memory activities. The interconnect has to offer enough bandwidth to timely serve the large number of incoming requests. On the contrary, the *Latency-sensitive phase* can eventually show a modest traffic load. However, a small delay in the injected packet delivery can severely impact the performance. The unbalanced resource utilization represents an additional design issue. In particular, the data network dominates the number of injected flits. Thus, the data packets are responsible of increasing the NoC contention during the *Bandwidth-sensitive phases*. Starting from these considerations we presented an heterogeneous NoC to optimally balance the power and performance metrics. The proposed architecture is made of two, physically split NoCs. The *main* network provides three virtual networks to support the coherence protocol and its frequency is fixed at 1.5GHz. The *auxiliary* network is a DVFS-capable network where its frequency can be set within 500MHz and 2GHz. It has to support the application execution during *Bandwidth-sensitive phases*. Our policy can route the available network packet based on the actual network load. Results have been extracted considering an 8x8 architecture running the *Splash2* applications. We compared the proposed solution against a state-of-the-art reference methodology that exploits a per-Router DVFS scheme. Moreover, the baseline architecture has been tuned to offer the power-aware and the performance-aware solution to limit the design space and to have a reference model from both the power and the performance view point. Compared to

the performance-aware baseline, the proposed architecture shows a 3% performance overhead on average. A 79% power overhead on average emerged comparing our solution with the power-aware baseline. However the *Energy Delay Product* shows a net improvement of the proposed architecture with respect to the baseline solution as well as the state of the art reference design.

The key advantage of the proposed NoC arch is the possibility to finely tune the performance between the main and the auxiliary NoC thanks to the DVFS implementation and the coupled policy. The advantage to dynamically routes the traffic to each network and the DVFS-capable of the auxiliary network are the keys of our architecture. These features allows our architecture to work better than the others with all the traffic load: bandwidth-sensitive, latency-sensitive and a mix of the phases. The possibility to tune the auxiliary network frequency allow the architecture to cover all the resource requirements: the very low traffic load of the high computational phase, the memory intensive phase with the high number of packets flow through the network and, finally, a mixed scenario.

5.1 Future Works

The presented work is based on the traffic distribution between different virtual networks. However, a stronger connection between the coherence messages and the network load is under investigation to provide a novel power-performance policy. In particular, the coherence traffic patterns will be evaluated to design a policy that can proactively tune the voltage and frequency of the NoC. Part of the work will be carried out within the MANGO project, that is focused on the design of the next generation multi-cores for HPC systems [21].

Bibliography

[1] M. Arora, S. Manne, I. Paul, N. Jayasena, and D. M. Tullsen. Understanding idle behavior and power gating mechanisms in the context of modern benchmarks on cpu-gpu integrated systems. In *2015 IEEE 21st International Symposium on High Performance Computer Architecture (HPCA)*, pages 366–377, Feb 2015.

[2] M. Badr and N.E. Jerger. Synfull: Synthetic traffic models capturing cache coherent behaviour. In *Computer Architecture (ISCA), 2014 ACM/IEEE 41st International Symposium on*, pages 109–120, June 2014.

[3] James Balfour and William J. Dally. Design tradeoffs for tiled cmp on-chip networks. In *Proceedings of the 20th Annual International Conference on Supercomputing*, ICS '06, pages 187–198, New York, NY, USA, 2006. ACM.

[4] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, Luca Benini, and Matthias Gries. A virtual platform environment for exploring power, thermal and reliability management control strategies in high-performance multicores. In *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, GLSVLSI '10, pages 311–316, New York, NY, USA, 2010. ACM.

[5] N. Binkert, B. Beckmann, G. Black, S.K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D.R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M.D. Hill, and D.A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[6] H. Bokhari, H. Javaid, M. Shafique, J. Henkel, and S. Parameswaran. darknoc: Designing energy-efficient network-on-chip with multi-vt cells

for dark silicon. In *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, pages 1–6, June 2014.

[7] T. E. Carlson, W. Heirmant, and L. Eeckhout. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–12, Nov 2011.

[8] Dhruba Chandra, Fei Guo, Seongbeom Kim, and Yan Solihin. Predicting inter-thread cache contention on a chip multi-processor architecture. In *11th International Symposium on High-Performance Computer Architecture*, pages 340–351, Feb 2005.

[9] L. Chang and W. Haensch. Near-threshold operation for power-efficient computing? it depends... In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 1155–1159, June 2012.

[10] P. Choudhary and D. Marculescu. Power management of voltage/frequency island-based systems using hardware-based methods. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 17(3):427–438, March 2009.

[11] C. Concatto, A. Kologeski, L. Carro, F. Kastensmidt, G. Palermo, and C. Silvano. Two-levels of adaptive buffer for virtual channel router in nocs. In *VLSI and System-on-Chip (VLSI-SoC), 2011 IEEE/IFIP 19th International Conference on*, pages 302–307, Oct 2011.

[12] S. Corbetta, D. Zoni, and W. Fornaciari. A temperature and reliability oriented simulation framework for multi-core architectures. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 51–56, Aug 2012.

[13] W.J. Dally. Virtual-channel flow control. *Parallel and Distributed Systems, IEEE Transactions on*, 3(2):194–205, Mar 1992.

[14] W.J. Dally and B.P. Towles. *Principles and practices of interconnection networks*. Elsevier, 2004.

[15] R. Das, S. Eachempati, A.K. Mishra, V. Narayanan, and C.R. Das. Design and evaluation of a hierarchical on-chip interconnect for next-generation cmps. In *High Performance Computer Architecture, 2009.*

HPCA 2009. IEEE 15th International Symposium on, pages 175–186, Feb 2009.

[16] R. Das, O. Mutlu, T. Moscibroda, and C. Das. A aergia: A network-on-chip exploiting packet latency slack. *IEEE Micro*, 31(1):29–41, Jan 2011.

[17] S. Demetriades and S. Cho. Predicting coherence communication by tracking synchronization points at run time. In *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 351–362, Dec 2012.

[18] J. Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *Parallel and Distributed Systems, IEEE Transactions on*, 4(12):1320–1331, Dec 1993.

[19] J. Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *Parallel and Distributed Systems, IEEE Transactions on*, 6(10):1055–1067, Oct 1995.

[20] J. Duato and T.M. Pinkston. A general theory for deadlock-free adaptive routing using a mixed set of resources. *Parallel and Distributed Systems, IEEE Transactions on*, 12(12):1219–1235, Dec 2001.

[21] J. Flich, G. Agosta, P. Ampletzer, D. A. Alonso, A. Cilardo, W. Fornaciari, M. Kovac, F. Roudet, and D. Zoni. The mango fet-hpc project: An overview. In *Computational Science and Engineering (CSE), 2015 IEEE 18th International Conference on*, pages 351–354, Oct 2015.

[22] S.M. Hassan and S. Yalamanchili. Centralized buffer router: A low latency, low power router for high radix nocs. In *Networks on Chip (NoCS), 2013 Seventh IEEE/ACM International Symposium on*, pages 1–8, April 2013.

[23] Robert Hesse and Natalie Enright Jerger. Improving DVFS in NoCs with coherence prediction. In *Proceedings of the International Symposium on Networks on Chip*, 2015.

[24] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. Van Der Wijngaart. A 48-core

ia-32 processor in 45 nm cmos using on-die message-passing and dvfs for performance and power scaling. *IEEE Journal of Solid-State Circuits*, 46(1):173–183, Jan 2011.

[25] Ming-yu Hsieh, Arun Rodrigues, Rolf Riesen, Kevin Thompson, and William Song. A framework for architecture-level power, area, and thermal simulation and its application to network-on-chip design exploration. *SIGMETRICS Perform. Eval. Rev.*, 38(4):63–68, March 2011.

[26] Andrew B Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: a fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 423–428. European Design and Automation Association, 2009.

[27] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks (1976)*, 3(4):267–286, 1979.

[28] Wonyoung Kim, M. S. Gupta, G. Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *2008 IEEE 14th International Symposium on High Performance Computer Architecture*, pages 123–134, Feb 2008.

[29] A. Leroy, J. Picalausa, and D. Milojevic. Quantitative comparison of switching strategies for networks on chip. In *Programmable Logic, 2007. SPL '07. 2007 3rd Southern Conference on*, pages 57–62, Feb 2007.

[30] Mieszko Lis, Pengju Ren, Myong Hyon Cho, Keun Sup Shim, Christopher W. Fletcher, Omer Khan, and Srinivas Devadas. Scalable, accurate multicore simulation in the 1000-core era. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, ISPASS '11, pages 175–185, Washington, DC, USA, 2011. IEEE Computer Society.

[31] A. K. Mishra, O. Mutlu, and C. R. Das. A heterogeneous multiple network-on-chip design: An application-aware approach. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–10, May 2013.

[32] Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. A case for heterogeneous on-chip interconnects for cmps. *SIGARCH Comput. Archit. News*, 39(3):389–400, June 2011.

[33] Asit K. Mishra, Aditya Yanamandra, Reetuparna Das, and Noumya Eachempati. Raft: A router architecture with frequency tuning for on-chip networks. In *J. Parallel Distrib. Comput. (2010)*, pages 3–14, Sept 2010.

[34] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *Computer*, 26(2):62–76, Feb 1993.

[35] C. A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, and C. R. Das. Vichar: A dynamic virtual channel regulator for network-on-chip routers. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 333–346, Dec 2006.

[36] I. Miro Panades and A. Greiner. Bi-synchronous fifo for synchronous circuit communication well suited for network-on-chip in gals architectures. In *First International Symposium on Networks-on-Chip (NOCS'07)*, pages 83–94, May 2007.

[37] Li-Shiuan Peh and William J. Dally. A delay model and speculative architecture for pipelined routers. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture*, HPCA '01, pages 255–, Washington, DC, USA, 2001. IEEE Computer Society.

[38] Subodh Prabhu, Boris Grot, Paul V Gratz, and Jiang Hu. Ocin tsim-dvfs aware simulator for nocs. In *""*. Citeseer, "".

[39] Jose Renau, Basilio Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Smruti Sarangi, Paul Sack, Karin Strauss, and Pablo Montesinos. Sesc simulator, 2005.

[40] James A. Ross, David A. Richie, Song J. Park, and Dale R. Shires. Parallel programming model for the epiphany many-core coprocessor using threaded mpi. In *Proceedings of the 3rd International Workshop on Many-core Embedded Systems*, MES '15, pages 41–47, New York, NY, USA, 2015. ACM.

[41] I. Seitanidis, A. Psarras, K. Chrysanthou, C. Nicopoulos, and G. Dimitrakopoulos. Elastistore: Flexible elastic buffering for virtual-channel-based networks on chip. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(12):3015–3028, Dec 2015.

[42] V. Soteriou, N. Eisley, H. Wang, B. Li, and L. S. Peh. Polaris: A system-level roadmap for on-chip interconnection networks. In *2006 International Conference on Computer Design*, pages 134–141, Oct 2006.

[43] Vasileios Spiliopoulos, Georgios Keramidas, Stefanos Kaxiras, and Konstantinos Efstathiou. Power-performance adaptation in intel core i7. In *Proc. 2nd Workshop on Computer Architecture and Operating System co-design*, pages 10–. Computer Science and Artificial Intelligence Laboratory, MIT, 2011.

[44] F. Terraneo, D. Zoni, and W. Fornaciari. A cycle accurate simulation framework for asynchronous noc design. In *System on Chip (SoC), 2013 International Symposium on*, pages 1–8, Oct 2013.

[45] A.N. Udipi, N. Muralimanohar, and R. Balasubramonian. Towards scalable, energy-efficient, bus-based on-chip networks. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–12, Jan 2010.

[46] S.C. Woo, M. Ohara, E. Torrie, J.P. Singh, and A. Gupta. The splash-2 programs: characterization and methodological considerations. In *Computer Architecture, 1995. Proceedings., 22nd Annual International Symposium on*, pages 24–36, June 1995.

[47] Y. Yao and Z. Lu. Dvfs for nocs in cmps: A thread voting approach. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 309–320, March 2016.

[48] Y. J. Yoon, N. Concer, M. Petracca, and L. P. Carloni. Virtual channels and multiple physical networks: Two alternatives to improve noc performance. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(12):1906–1919, Dec 2013.

[49] D. Zoni, S. Corbetta, and W. Fornaciari. Hands: Heterogeneous architectures and networks-on-chip design and simulation. In *Proceedings of*

the 2012 ACM/IEEE International Symposium on Low Power Electronics and Design, ISLPED '12, pages 261–266, New York, NY, USA, 2012. ACM.

[50] D. Zoni, J. Flich, and W. Fornaciari. Cutbuf: Buffer management and router design for traffic mixing in vnet-based nocs. *IEEE Transactions on Parallel and Distributed Systems*, 27(6):1603–1616, June 2016.

[51] D. Zoni and W. Fornaciari. A sensor-less nbti mitigation methodology for noc architectures. In *SOC Conference (SOCC), 2012 IEEE International*, pages 340–345, Sept 2012.

[52] D. Zoni and W. Fornaciari. Sensor-wise methodology to face nbti stress of noc buffers. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2013*, pages 1038–1043, March 2013.

[53] D. Zoni and W. Fornaciari. Modeling dvfs and power gating actuators for cycle accurate noc-based simulators. *Journal of Emerging Technologies in Computing Systems*, pages 1–15, 2015.

[54] D. Zoni, F. Terraneo, and W. Fornaciari. A dvfs cycle accurate simulation framework with asynchronous noc design for power-performance optimizations. *Journal of Signal Processing Systems*, pages 1–15, 2015.

[55] Davide Zoni, Federico Terraneo, and William Fornaciari. A control-based methodology for power-performance optimization in nocs exploiting dvfs. *Journal of Systems Architecture*, pages 1–15, 2015.