A Latency Simulator for Many-core Systems

Sunil Kumar
The LNM Institute of Information Technology, Jaipur, India
`sunil@lnmiit.ac.in`

Tommaso Cucinotta and Giuseppe Lipari
Scuola Superiore Sant'Anna, Pisa, Italy
{`t.cucinotta,g.lipari`}`@sssup.it`

Keywords: Simulation, Many-Core Architecture, Interconnect, Network-on-a-Chip, Latency

Abstract

In this paper we present MCoreSim, an open-source simulation framework for massively parallel and many-core computing systems based on OMNeT++. The simulator supports tile-based architectures with distributed memory and mesh-based interconnects. Its primary purpose is to allow for investigations on the impact of the heterogeneous in-chip communication latencies, as arising due to the network-on-a-chip structure of future and emerging many-core processors, on the performance of the hosted applications. We plan to use MCoreSim to study the variety of possible choices in realizing a suitable software stack for these systems, especially in terms of the choices at the kernel design level.

1. INTRODUCTION

The current era of processor manufacturing has reached petascale computing, i.e., 10^{15} operations per second, and the first approaches towards exascale (10^{18} operations per second) are already planned. To make this happen, a new paradigm of computing is emerging that pushes the pedal on parallelism at the hardware infrastructure level. Indeed, personal computing has already moved towards having multiple cores per chip. With peta/exa-scale computing architectures, the parallelism in hardware is planned to be pushed to the extreme, towards thousands of cores inside the same chip with complex interconnection logic and distributed memory elements. The design of this kind of chip architecture has to address two main aspects: computation and communication. As the core count increases, the interconnection of cores becomes a more critical and challenging task. Also, in this context classical models of computing are at serious risk of being disrupted. For example, the globally coherent cache memories that characterize nowadays multi-core systems might not be feasible in a context with thousands of interconnected cores, as it would lead to excessive and unneeded communication overheads.

These new trends in hardware designs calls for new paradigms of software design and development as well, for giving the highest consideration and attention to issues related to the scalability of the mechanisms with respect to the number of cores, as well as to the latencies involved in the various core-to-core and core-to-memory communications occurring in the chip as a result of either explicit or implicit programmers instructions. In fact, classical software stacks which rely on basic/simple services provided by a single centralized Operating System (OS) instance, may need to be replaced by distributed OS kernel designs (inside the chip) enriched with properly designed inter-thread/process communication primitives [31, 29]. However, worth to mention are also the valuable attempts of increasing scalability in traditional OSes like Linux [12, 19].

In order to properly investigate on the research issues related to designing software stacks able to make an optimum or sufficiently good usage of the computation capabilities available in the upcoming massively parallel machines, suitable models and simulations need to be made available to the community. To this purpose, in this paper we present MCoreSim, a new open-source simulation infrastructure for future many-core systems based on the OMNeT++ framework which we are building to investigate on these issues in the context of the S(o)OS European Project[1].

2. RELATED WORK

Application mapping, multi-threading, memory hierarchy, network communication are the main research issues in many-core system design, covering a broad range of topics from the Operating System down to the hardware. Due to the unavailability of these future massively parallel computing machines, we need a suitable simulation environment for the system.

Most of the existing works in many-core simulation are focused on either full system simulation or architectural simulation. Full system simulation provides hardware simulation as well as emulation of applications, like the HP Labs' COTSon simulator [4], where functional emulation is based on the AMD SimNow [13], or the M5 Simulator System [11]. On the other hand, architectural simulation targets hardware simulation (specially network-on-chip) for multi-core systems, like the PhoenixSim [14] based on OMNeT++, and the noxim[2] and NIRGAM [22] based on SystemC. Most simulators are based on a shared memory paradigm but, as the core count reaches to more than thousand cores, the message passing paradigm is more useful. On other hand, QEMU [8] is

[1]Service-oriented Operating System. More information is available at: `http://www.soos-project.eu/`.
[2]More information is available at: `http://noxim.sourceforge.net/`.

an open-source machine emulator working also as a Virtual Machine Monitor (KVM). Now developers of QEMU seem to be willing to extend the emulation to many-core systems as well.

A completely different approach can be found in [5], where a functional hardware description language (CλaSH) is defined based on the Haskell language, which can be leveraged both to synthesize the hardware in VHDL code and to simulate its behavior.

As compared to the above mentioned approaches to simulating multi/many-core systems, we focus on a high-level abstraction level of the simulation that neglects (as of now) the specific internals of how each core carries out computations. Instead, we focus mainly on an accurate simulation of the delays caused by the complex network-on-a-chip based communications occurring within the interconnects of the new chips (see Section 3. later). This will allow us to investigate on proper data and code distribution across different cores and related synchronization paradigms to be used on these future machines.

3. BACKGROUND

In this section, some background on many-core computing systems is summarised, focusing on three different levels: computing and communication infrastructure; communication paradigm; software/OS level.

3.1. Computing Infrastructure: Processor Architecture

Processor manufacturers have gone very far away from the traditional architecture of the processor, in which the chip used to contain a single Central Processing Unit (CPU). From the time when they started to integrate the cache memory elements into the chip for boosting performance, nowadays processors mimic more and more the System-on-a-Chip (SoC) paradigm typical of the embedded domain, in which a multitude of heterogeneous hardware components are all integrated in the same chip, such as multiple cores, hierarchies of cache memories, interrupt controllers, timers and other peripherals. Also, the complexity of the new chips and the growing requirements in terms of computational power led to novel paradigms of interconnection among the various hardware components within the chip (and within the system), mainly driven by the need for increasing the level of parallelism in communications. This led to an increasing trend in abandoning the traditional "interconnection bus" concept in favour of Network-on-a-Chip (NoC) interconnection solutions, in which hardware elements communicate with each other by means of packets (similar to how networking among systems used to do), whilst proper routing elements are responsible for delivering these packets to their destination. Most modern general-purpose processors (GPP) are homogeneous, symmetric multi-core processors following the "multi-core" paradigm, i.e., parallel "fat" cores. There are also architectures that follow the "many-core" paradigm (e.g., GPUs and GP-GPUs), where there is a massively high number of smaller processing units where a workload is typically distributed over many of them. Also, processors are switching more and more to heterogeneous multi-core designs.

The ways the processing elements will be connected in future tera/exa-scale chips is a critical factor for the expectable performance. This is covered in the next section.

3.2. Communication Infrastructure: Interconnect

A good interconnection mechanism determines the efficient execution of an application or set of applications, based on the required computation and communication needs. In single core processors, most components are connected by a hybrid bus topology. But, as the core count increases, the traditional bus-based interconnection suffers of major bottleneck drawbacks when trying to scale to many cores. So, more efficient physical topologies for multi-core architectures are being explored, like the widely known two-dimensional (2D) ones: mesh, ring and torus. These are suitable for small-scale systems, but three-dimensional [6, 28] and n-dimensional topologies have been proposed as well. Every mechanism has its own advantages and disadvantages. The major performance metrics are: topology regularity, network diameter, bisection bandwidth, and power consumption [23].

3.3. Communication Paradigm

The communication paradigm specifies how cores communicate with each other. Here, major challenges are flow control and routing:

Flow-control There are two possible levels at which flow control is realised: switch-to-switch and end-to-end. Flow-control at the switch-to-switch level mainly depends on switching and buffer management techniques. There are two types of switching techniques: packet switching and circuit switching. Circuit switching is suitable for communications with QoS guarantees because the required link resources are reserved for the entire lifetime of a particular communication and no arbitration is needed [9]. However, it limits the possibilities of time-sharing of the links across multiple communications.

Packet switching techniques fall into four main categories:

Store & Forward: incoming packets are stored first, then they are forwarded to the next hop;

Wormhole switching: each packet is divided into flits (small chunks), which are sent one by one to the intermediate router; the router stores and processes the header flit first, then it forwards it to the next hop and the other flits just follow the path taken by the header flit;

Virtual-Cut-Through: each packet is divided into flits and sent one by one to the intermediate router; the router stores all the flits in a buffer, then, after the routing decision, forwards them to the next hop.

Each one of the above schemes has advantages and disadvantages. Among all the packet switching techniques, Wormhole switching seems to be the most suitable for fast computing, because of its lower latency and limited buffer requirements.

The performance of a switching technique also depends on the channel buffer management between routers to avoid overflows. In the *Go-and-Stop* (or on/off) control, the sender and receiver routers synchronise with each other by means of explicit *stop* and *go* signals. In the *Credit-based* control, the receiver router communicates the available receive buffer size to the sender router.

Routing algorithm The routing problem becomes more and more relevant as the number of cores increases. The goal of a routing algorithm is to distribute traffic evenly among the paths supplied by the network topology, so as to avoid deadlock and minimise contention, thus improving network latency and throughput. Based on the application requirement, routing paths may be adaptive or deterministic, and multi-path or single-path. There are various algorithms which have been proposed, but the most commonly used ones are Dimension-Ordered-Routing (DOR) and Turn-Model routing. In DOR, by looking at the distance vector between the sender and the destination, a message is forwarded first in the lower dimension, then in the other one. DOR uniformly distributes minimal paths between all pair of nodes [2]. In Turn-Model routing, the packet has two choices in each router: proceed on the same direction or turn in the other direction, based on the availability of the path or link [16, 32].

3.4. Software/OS level

The OS and/or application has to deal with application mapping, communication and task scheduling. The goal of application mapping is to map a set of processor-cores to a specified application in an optimised manner (software-hardware mapping). In multi-core systems, the hardware topology is pre-configured, so we will have to map and schedule applications and their communications on the available hardware resources. Given some knowledge on the application components and their computing and communication requirements, we end up with a graph mapping problem, where the application-level graph needs to be properly mapped to the available physical level topology. These approaches may be based on an a-priori characterisation of the application requirements and an off-line optimisation process [15, 27]. However, this is not appropriate for dynamic real-time applications with strongly varying workloads. For this class of applications, online dynamic mapping is more appropriate, where the cores to applications mapping may be dynamically varied depending on the actually imposed workload on the various subsystems [17, 18]. For communication and task scheduling it is possible to use various scheduling schemes, based on resources availability and requirements.

4. THE SIMULATION FRAMEWORK

4.1. Background on OMNeT++

MCoreSim is a many-core simulation framework developed by using OMNeT++[3], a tool for simulation of communication networks and freely available for academic use. OMNeT++ is a component-based discrete-event simulation library and framework for simulating networked systems. It provides developers with the capability to define highly reusable *modules*, which can be instantiated and interconnected in different designs. Modules have *gates* for interaction with other modules and can be combined with each other at arbitrary nesting levels. Modules communicate through message passing via defined gates and channels (connection between modules). Modules, channels and gates can be parameterised by leveraging the Network Description (NED) Language, and used for customising the behavior according to specific application scenarios.

4.2. MCoreSim Simulator Architecture

As discussed in Section 3., due to the difficulties in maintaining a globally coherent cache/memory view in many-core systems, the message passing paradigm is the most suitable one to be used for communications. In many-core systems, the network-on-a-chip paradigm requires message passing to be implemented in hardware by the on-chip interconnection logic. MCoreSim models a tile-based networked systems. In this there are two major components: the *tile* and the *connection strategy* (topology). The actual topology simulated in MCoreSim may be customised via a set of configuration options. The most important of them are summarised in Table 1. The parameters meaning will become clear in the following description of the main simulator components.

4.2.1. Interconnection

Most of commercially available multi/many-core chips are based on mesh or torus topologies [30, 7, 1]. In OMNeT++, interconnection of modules is modeled by using the `network` keyword in the NED language. This construct is exploited in MCoreSim, allowing for the specification of a

[3]More information is available at: `http://www.omnetpp.org/`.

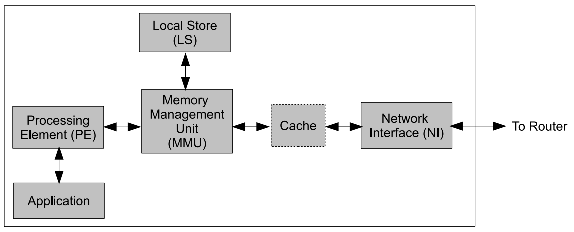

Figure 1. Computing Unit

configurable 2D mesh and torus. MCoreSim defines a *link* between each pair of tiles, modeled as a physical channel that has configurable properties like delay, data-rate, etc., that are defined by extending the `ned.DatarateChannel` channel model available in OMNeT++.

4.2.2. Tile

MCoreSim models a tile-based architecture. Each tile has two major components: the *computing unit*, responsible for computations and the *communication unit*, responsible for the communication among tiles.

Computing Unit Architecture Each Computing unit (see in Figure 1) has a Processing Element (PE), a local memory element (Local Store and/or Cache) and a Network Interface towards the local communication unit. A Memory Management Unit (MMU) is also needed for routing memory operations towards the Local Store or remote memory elements through the communication network. In MCoreSim, all these units have been implemented as OMNeT++ modules. An application is also modeled as a module that provides the set of instructions to be executed on one or more computing unit.

Logically, a Processing Element (PE) can be represented as general purpose or special purpose processing unit. Its main function is to fetch the instructions from the application program and simulate the time needed to carry out the involved computations interacting with the necessary local or remote memory elements and/or other cores.

The Network Interface (NI) enables the transfer of data (or instructions) to/from other tiles and/or I/O peripherals. First, it stores the packet to be transmitted in a local buffer then it involves the communication unit for realizing the actual transfer to the destination processing element or memory or I/O controller.

Communication Unit Architecture Communication unit or router provide the inter-connectivity of tiles. A typical router architecture [26, 25] consists of a buffer for input packet storage and a computing unit that performs buffer management, routing, packet scheduling, port arbitration and a switch to output port. In MCoreSim each of this functionality is defined as a module, such as the Route Computation (RC), Virtual channel Allocation (VA), Switch Arbitration (SA), and Switch (ST) modules. Based on the topology details, the number of buffers can be parametrised (see Figure 2).

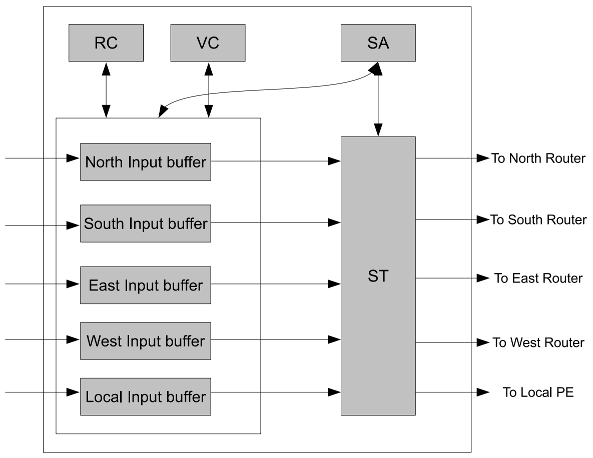

Figure 2. Router Architecture

Packet Format In OMNeT++ messages and packets provide interaction between modules. These are implemented as the `cMessage` and `cPacket` classes respectively. Packets are used for network level communication and Messages for everything else.

In MCoreSim, an Application is specified as a sequence of instructions, resulting in a sequence of Instruction packets to be fetched from some memory element, and executed by the Processing Element. Furthermore, Interrupt packets are used for controlling the transfers needed during an instruction execution, and realizing the necessary flow control mechanisms at the network level. Furthermore, flit packets implement router-to-router communications and Router Control messages for interaction inside the router element. Instruction and Flit packets are implemented sub-classing `cPacket`, and Interrupt and Router Control messages by sub-classing the `cMessage` class.

4.3. MCoreSim Protocol Library

The MCoreSim protocol library provides the protocols for simulating communications and computations. Up to now the library implements protocols related to a basic communication and computational model.

4.3.1. Protocols for Communications Plane

The protocols planned to be implemented in MCoreSim Communication Plane are the network-on-chip protocols as mentioned in Section 3.2.. These are related to buffer management, switching, arbitration, packet scheduling and routing. As of now, we implemented the elements detailed below.

Routing MCoreSim supports mesh and torus regular topologies, and the popular deterministic routing Dimension-Order Routing (DOR) [2]. In this routing, no storage element like register file or routing table is required.

Switching As discussed in Section 3.3., there are various switching protocols available but, as the number of cores increases, the on-chip transmission bandwidth available to each core decreases. Due to this constraint, we implemented in MCoreSim wormhole switching [24]. In this approach, a Network Interface splits a processor Instruction packet into smaller size data units called Flit, then it transfers them towards destination through the network. At the destination tile, the Network Interface re-assembles the processor Instruction packet and sends it to the PE for further execution.

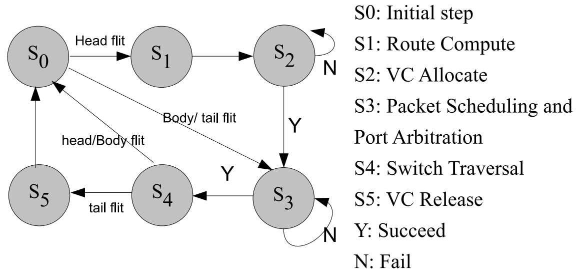

Figure 3. Sequence Chart of Router

Between the source and destination NIs, routers handle Flit packets in the following manner (see in Figure 3). The Route Computation (RC) module computes route according to the specified routing algorithm, the Virtual channel allocation (VC) module allocates a virtual channel for the next router input buffer, as specified by the RC module, and according to the Virtual channel allocation algorithm. Then, the Switch Arbitration (SA) module, based on packet scheduling and port arbitration algorithms, sets the path up from the input buffer to the output port in the Switch Traversal (ST) module.

Flow Control As discussed in Section 3.3., there are various flow control protocols available for switch-to-switch or end-to-end transmissions. In MCoreSim, a credit-based Link-level Flow Control protocol is implemented.

This credit flow control protocol [21, 10] is employed in between network interface (NI) and router, and among routers. The credit-based flow control mechanism indicates to the output port the buffer space availability in the adjacent input ports. Once an output port runs out of credits, it will stop sending flits to the adjacent input port (see Figure 4 and 5).

Buffer Management The MCoreSim Router element has a buffer for each input channel port. For buffer management, the Router implements a virtual channel based protocol [20, 3]. In this each physical channel is divided into a number of virtual channels and each virtual channel has its own buffer space. After the route computation, a request is made to the virtual channel allocator (VC) to allocate an unused virtual channel to the new route. After the allocation, the switch arbiter (SA) schedules the packets that flow through the output port of the Router.

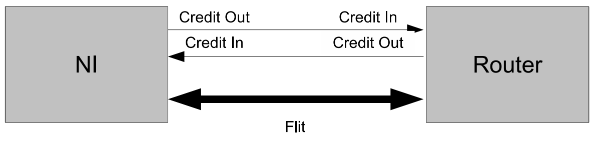

Figure 4. Flow Control- Between NI and Router

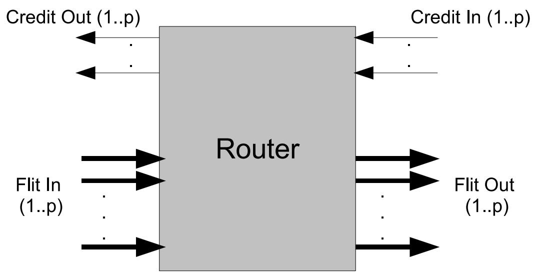

Figure 5. Flow Control- Between Routers

Among the variety of arbitration algorithms available, in MCoreSim only FIFO based port arbitration is implemented as of now.

4.3.2. Protocols for Computation Plane

The Computation Plane has three major components: processing plane, memory and I/O plane, application modelling.

Processing Plane In MCoreSim, the Processing Plane consists of several Processing Elements that constitute an entire chip. A Processing Element may be general-purpose, simulating a typical "CPU" behaviour, or specialised, like GPUs. Each general-purpose PE works by following the traditional fetch-execute cycle. It processes mainly three types of operations: memory access operations, computing operations and message-passing operations. In a computing operation, the PE executes instructions by using its ALU/FPU unit, however only the delay needed for this execution is simulated in MCoreSim. In a memory operation, the PE routes the memory access towards a local or remote memory element with the help of the memory management unit (MMU). If the access is local, then it is routed to the Local Store, otherwise the PE generates a message-passing Instruction towards a remote destination. Message-passing operations are routed to their destination tile via the Communication Plane through the local Network Interface.

Parameter	Value	Description
topology	mesh	Topology type (mesh or torus)
datarate	50Gbps	datarate of a channel
delay	13ps	delay in channel
flitSize	1500 B	size of flit packet
numX	10	number of nodes in X direction
numY	8	number of nodes in Y direction
bufferSize	1024 B	buffer space available in router
NIbufferSize	1024 B	buffer space available in NI
router_type	DOR	routing algorithm for topology
clockRate	5e9	clock rate of processing element
m_clockRate	1e9	clock rate of memory element
numVC	1	number of virtual channels per input port

Table 1. Configuration parameters for MCoreSim.

Memory and I/O Plane There are various memory hierarchy types available for many-core systems. MCoreSim implements the Memory Plane as a globally shared memory. In this global memory every PE has its own private memory space called Local Store. Each PE can directly access the Local store but for remote access uses the Communication Plane. The MMU provides the mapping between the addresses generated by the PEs and the either local or remote memory elements.

Application Modelling As of now, an MCoreSim application supports three types of instruction: memory access, computing and message-passing instructions. Furthermore, the applications are statically configured to run on specific cores, for now. In the future, the Application Plane will support a richer set of instructions (but the focus of our simulation will anyway keep the number of supported instructions at the bare minimum), and it will provide dynamic application mapping and scheduling across the available computing infrastructure, so as to better emulate the software stack and OS services as described in Section 3.4.

4.4. Performance Analysis

In OMNeT++ there is a concept of *signal*. This is used for collecting statistical information out of a simulation run. With the help of signals, MCoreSim can seamlessly collect statistics for various events that occur during the simulation.

In the Communication Plane, useful performance metrics are the average and/or maximum packet latency, the network bandwidth and the network throughput. For the Processing Plane, useful performance metrics are the average/maximum execution time for a given Application, which may be varying depending on the latencies experienced due to the underlying topology.

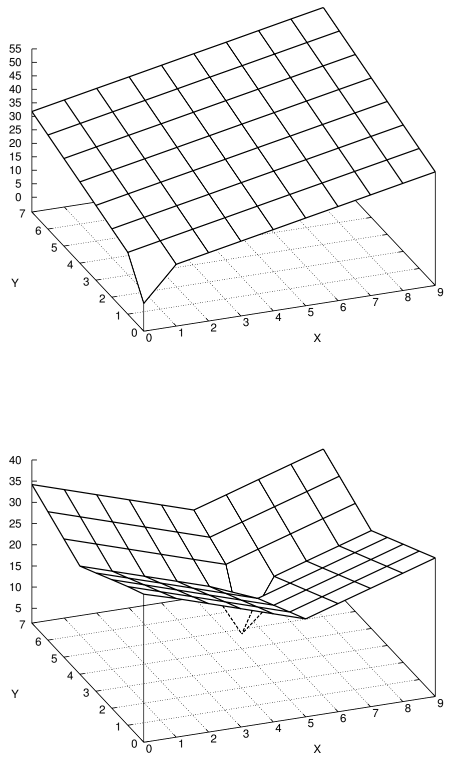

Figure 6. Round-trip times for communications among the tile [0,0] (top) or [4, 5] (bottom) and each other tile.

5. SIMULATION EXAMPLES

In this section, we show the results obtained by running the simulator in a sample scenario. These are useful to let the reader understand what kind of result is achievable with the simulator.

The scenario we run is very simple: each core, in turn, pings each other core, and measures what is the round-trip time. This is useful for understanding how the latency for communicating with cores inside other tiles (and accessing the memory elements within) change with the coordinates of the destination tile.

In the example, we have the system completely idle, running solely the ping application. The round-trip latencies are shown in the 3D plots of Figure 6, where on the X and Y axes we report the coordinates of the destination tile, and on the Z axis the latency time (in nanoseconds) as measured during the simulation. Under a completely unloaded network, the latency is perfectly linear in the number of hops necessary to reach the destination, except for the local communications, which complete in much lower time as they do not need to involve the router at all. This is highlighted in the plots by the downwards peaks in correspondence of the source tile coordinates.

6. FUTURE WORK AND CONCLUSIONS

In this paper we presented MCoreSim, an OMNeT++ based framework for simulating in-chip communication latencies in many-core architectures. We described the motivations at the basis of our simulation design choices, and provided an overview of the simulation model. In the future, we plan to exploit MCoreSim so as to investigate on the effectiveness of particularly designed data structures and synchronisation protocols among kernel components running within an OS for many-core systems. In fact, certain choices at the kernel level in terms of what data is shared and/or replicated across multiple cores, and how access to that is synchronised, turn out to be critical for the performance of the user-space applications. The proposed simulation framework will help in investigating into these and other related tough research issues, in the context of future and emerging massively parallel many-core machines which are not available yet today, playing with the possible topologies and in-chip protocol options and measuring the corresponding achievable (simulated) performance.

7. ACKNOWLEDGEMENTS

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7 under grant agreement n. 248465 "S(o)OS – Service-oriented Operating Systems."

REFERENCES

[1] *An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS*, 2007.

[2] Jose Miguel Montañana Aliaga, Michihiro Koibuchi, Hiroki Matsutani, and Hideharu Amano. Balanced dimension-order routing for k-ary n-cubes. In Leonard Barolli and Wu chun Feng, editors, *ICPP Workshops*, pages 499–506. IEEE Computer Society, 2009.

[3] N. Alzeidi, A. Khonsari, M. Ould-Khaoua, and L. Mackenzie. A new approach to model virtual channels in interconnection networks. *J. Comput. Syst. Sci.*, 73:1121–1130, December 2007.

[4] Eduardo Argollo, Ayose Falcón, Paolo Faraboschi, Matteo Monchiero, and Daniel Ortega. Cotson: infrastructure for full system simulation. *SIGOPS Oper. Syst. Rev.*, 43(1):52–61, 2009.

[5] Christiaan Baaij, Matthijs Kooijman, Jan Kuper, Arjan Boeijink, and Marco Gerards. CλaSH: Structural descriptions of synchronous hardware using haskell. In *Proceedings of the 13th EUROMICRO Conference on Digital System Design: Architectures, Methods and Tools*, pages 714–721, USA, September 2010. IEEE Computer Society.

[6] Kaustav Banerjee, Shukri J. Souri, Pawan Kapur, and Krishna C. Saraswat. 3-d ics: A novel chip design for improving deep-submicrometer interconnect performance and systems-on-chip integration. In *Proceedings of the IEEE*, pages 602–633, 2001.

[7] Max Baron. The single-chip cloud computer: Intel networks 48 pentiums on a chip. *The Insider Guide to Microprocessor Hardware*, April 2010.

[8] Daniel Bartholomew. Qemu: a multihost, multitarget emulator. *Linux J.*, 2006:3–, May 2006.

[9] Marcelo Daniel Berejuck and Cesar Albenes Zeferino. Adding mechanisms for qos to a network-on-chip. In *Proceedings of the 22nd Annual Symposium on Integrated Circuits and System Design: Chip on the Dunes*, SBCCI '09, pages 25:1–25:6, New York, NY, USA, 2009. ACM.

[10] Jonathan Billington and Smit Saboo. An investigation of credit-based flow control protocols. In *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, Simutools '08, pages 34:1–34:10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[11] Nathan L. Binkert, Ronald G. Dreslinski, Lisa R. Hsu, Kevin T. Lim, Ali G. Saidi, and Steven K. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26:52–60, July 2006.

[12] Silas Boyd-Wickizer, Austin T. Clements, Yandong Mao, Aleksey Pesterev, M. Frans Kaashoek, Robert Morris, and Nickolai Zeldovich. An Analysis of Linux Scalability to Many Cores. In *OSDI 2010: Proceedings of the 9th USENIX conference on Operating Systems Design and Implementation*, 2010.

[13] AMD Developer Central. AMD SimNow(TM) Simulator. `http://developer.amd.com/cpu/simnow/Pages/default.aspx`.

[14] Johnnie Chan, Gilbert Hendry, Aleksandr Biberman, Keren Bergman, and Luca P. Carloni. Phoenixsim: a simulator for physical-layer analysis of chip-scale photonic interconnection networks. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '10, pages 691–696, 2010.

[15] W-K. Chen and E. F. Gehringer. A graph-oriented mapping strategy for a hypercube. In *Proceedings of the third conference on Hypercube concurrent computers*

and applications: Architecture, software, computer systems, and general issues - Volume 1, C3P, pages 200–209, New York, NY, USA, 1988. ACM.

[16] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *IEEE Trans. Parallel Distrib. Syst.*, 11:729–738, July 2000.

[17] Chen-Ling Chou and Radu Marculescu. Incremental run-time application mapping for homogeneous nocs with multiple voltage levels. In *Proceedings of the 5th IEEE/ACM international conference on Hardware/software codesign and system synthesis*, CODES+ISSS '07, pages 161–166, New York, NY, USA, 2007. ACM.

[18] Chen-Ling Chou, Uemit Y. Ogras, and Radu Marculescu. Energy- and performance-aware incremental mapping for networks on chip with multiple voltage levels. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 27(10):1866–1879, 2008.

[19] Jonathan Corbet. Vfs scalability patches in 2.6.36. `http://lwn.net/Articles/401738/`, August 2010. Article.

[20] W. J. Dally. Virtual-channel flow control. *IEEE Trans. Parallel Distrib. Syst.*, 3:194–205, March 1992.

[21] M. El-Taha and J. R. Heath. Queueing network models of credit-based flow control. *Comput. Math. Appl.*, 50:393–398, August 2005.

[22] Lavina Jain et al. Nirgam: a simulator for noc interconnect routing and application modeling. In *Workshop on Diagnostic Services in Network-on-Chips, Design, Automation and Test*, in DATE '07, 2007.

[23] D. N. Jayasimha, Bilal Zafar, and Yatin Hoskote. On-chip interconnection networks: Why they are different and how to compare them. *Technical Report, Intel Corp.*, 2006.

[24] Zhonghai Lu, Ming Liu, and Axel Jantsch. Layered switching for networks on chip. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 122–127, New York, NY, USA, 2007. ACM.

[25] Robert Mullins, Andrew West, and Simon Moore. Low-latency virtual-channel routers for on-chip networks. *SIGARCH Comput. Archit. News*, 32:188–, March 2004.

[26] Li-Shiuan Peh and William J. Dally. A delay model for router microarchitectures. *IEEE Micro*, 21:26–34, January 2001.

[27] Gaurav Kumar Singh, Mythri Alle, Keshavan Vardarajan, S K Nandy, and Ranjani Narayan. A generic graph-oriented mapping strategy for a honeycomb topology. *International Journal of Computer Applications*, 1(21):91–98, February 2010. Published By Foundation of Computer Science.

[28] Awet Yemane Weldezion, Matt Grange, Dinesh Pamunuwa, Zhonghai Lu, Axel Jantsch, Roshan Weerasekera, and Hannu Tenhunen. Scalability of network-on-chip communication architecture for 3-d meshes. In *Proceedings of the 2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, NOCS '09, pages 114–123, Washington, DC, USA, 2009. IEEE Computer Society.

[29] David Wentzlaff and Anant Agarwal. Factored operating systems (fos): the case for a scalable operating system for multicores. *SIGOPS Oper. Syst. Rev.*, 43:76–85, April 2009.

[30] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-chip interconnection architecture of the tile processor. *IEEE Micro*, 27:15–31, September 2007.

[31] Silas B. Wickizer, Haibo Chen, Rong Chen, Yandong Mao, Frans Kaashoek, Robert Morris, Aleksey Pesterev, Lex Stein, Ming Wu, Yuehua Dai, Yang Zhang, and Zheng Zhang. Corey: An operating system for many cores. In *Proceedings of the 8th USENIX Symposium on Operating Systems Design and Implementation (OSDI '08)*, San Diego, California, December 2008.

[32] Lingfu Xie and Du Xu. The two-level-turn-model fault-tolerant routing scheme in tori with convex and concave faults. In *Proceedings of the 2009 Sixth International Conference on Information Technology: New Generations*, pages 107–113, Washington, DC, USA, 2009. IEEE Computer Society.