NoCArc

First International Workshop on

Network on Chip Architectures

(NoCArc 2008)

In conjunction with the
41st Annual IEEE/ACM International Symposium on Microarchitecture
(MICRO-41)
November 8th, 2008
Lake Como, Italy

PROCEEDINGS

Foreword

The papers in this volume forms the proceedings of the First International Workshop on Network on Chip Architectures (NoCArc 2008) held in conjunction with the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-41), November 8th, 2008, Lake Como, Italy.

The workshop aims to provide a focused forum for researchers to present and discuss innovative ideas and solutions related to design and implementation of complex Systems on Chip (SoCs) which use a packet switched on-chip network as interconnection or communication infrastructure. Such on-chip networks are commonly referred as Networks on Chip (NoC). NoC architecture research requires combining knowledge from many areas like circuit design, computer architecture, data communication and design automation. This interdisciplinary nature of research provides many challenges and opportunities for NoC architecture designers. Over the last 7-8 years, there has been an explosion of proposals related to NoC topologies, router designs and routing algorithms. In spite of this, we feel we are still in an exploratory stage. Due to application specific nature of SoCs used in embedded systems, it is unlikely that a single NoC architectural proposal can be declared as the best for all designs. We feel such focused workshops can help collect good solutions (or information and links about good solutions) currently available which can be useful to the SoC and embedded systems industry.

The eight accepted papers in the workshop are divided into three themes, namely, micro-architectural aspects of NoC routers, performance evaluation of NoC-based SoCs, and prospective architectural proposals for on-chip interconnection systems. There were 18 submissions. Their authors were from 11 countries spread over the whole world. Each submitted paper has been reviewed by three to four reviewers. The deliberations were conducted by the Program Committee electronically. The Program Committee selected 8 papers for presentation at the Workshop (an acceptance rate of 44%). The papers were judged based on originality, quality and relevance to the subject area of the Workshop.

We would like to thank all of those who submitted papers for consideration, the Program Committee members for their invaluable contributions, and the MICRO-41 Organizing Committee for giving us the opportunity to host NoCArc 2008 Workshop. We hope that this workshop will become an annual event. Unfortunately, this workshop did not receive papers related to fault tolerance, memory architectures and reconfiguration aspects of NoC. We also hope that the future versions of the workshop will be able to attract good papers in these very important aspects for on-chip networks.

Lake Como, 8th November 2008

Maurizio Palesi Shashi Kumar

Chairs of NoCArc 2008 Workshop

Technical Program Committee

- Federico Angiolini, *iNoCs, Switzerland*
- Davide Bertozzi, *University of Ferrara, Italy*
- Giorgos Dimitrakopoulos, *FORTH, Greece*
- José Flich Cardo, *Universidad Politécnica de Valencia, Spain*
- Ahmed Hemani, *Royal Institute of Technology, Sweden*
- Rickard Holsmark, *Jönköping University, Sweden*
- Anshul Kumar, *Indian Institute of Technology (Delhi), India*
- Shashi Kumar, *Jönköping University, Sweden*
- Marcello Lajolo, *NEC Laboratories America, NJ, USA*
- Zhonghai Lu, *Royal Institute of Technology, Sweden*
- Srinivasan Murali, *iNoCs, Switzerland*
- Juan Manuel Orduña Huertas, *Universidad de Valencia, Spain*
- Maurizio Palesi, *University of Catania, Italy*
- Davide Patti, *University of Catania, Italy*
- Partha P. Pande, *Washington State University, USA*
- Timothy M. Pinkston, *University of Southern California, USA*
- Carlo Pistritto, *STMicroelectronics, Italy*
- Tor Skeie, *University of Oslo, Norway*
- Juha-Pekka Soininen, *VTT, Finland*
- Vittorio Zaccaria, *Politecnico di Milano, Italy*

Table of Contents

Keynote Talk

Managing Heterogeneity in Future NoCs

Jose Duato

Parallel Architectures Group
Technical University of Valencia, Spain
jduato@disca.upv.es

Jose Duato received the MS and PhD degrees in electrical engineering from the Technical University of Valencia, Spain, in 1981 and 1985, respectively. Currently, Dr. Duato is Professor in the Department of Computer Engineering (DISCA) at the same university. He was also an adjunct professor in the Department of Computer and Information Science, The Ohio State University.

His current research interests include interconnection networks and multiprocessor architectures. Prof. Duato has published over 400 refereed papers. He proposed a powerful theory of deadlock-free adaptive routing for wormhole networks. Versions of this theory have been used in the design of the routing algorithms for the MIT Reliable Router, the Cray T3E supercomputer, the on-chip router of the Alpha 21364 microprocessor, and the IBM BlueGene/L supercomputer. Prof. Duato also developed RECN, the only truly scalable congestion management technique proposed to date, and a very efficient routing algorithm for fat trees that has been incorporated into Sun Microsystem's 3456-port InfiniBand Magnum switch.

Prof. Duato is the first author of the book “Interconnection Networks: An Engineering Approach”. Dr. Duato served as a member of the editorial boards of IEEE Transactions on Parallel and Distributed Systems, IEEE Transactions on Computers, and IEEE Computer Architecture Letters. He has been the General Co-Chair for the 2001 International Conference on Parallel Processing, the Program Committee Chair for the Tenth International Symposium on High Performance Computer Architecture (HPCA-10), and the Program Co-Chair for the 2005 International Conference on Parallel Processing. Also, he served as Co-Chair, member of the Steering Committee, Vice-Chair, or member of the Program Committee in more than 55 conferences, including the most prestigious conferences in his area (HPCA, ISCA, IPPS/SPDP, IPDPS, ICPP, ICDCS, Europar, HiPC).

Managing Heterogeneity in Future NoCs

Jose Duato
Parallel Architectures Group
Technical University of Valencia, Spain
email: jduato@disca.upv.es

***Abstract*—Although most research on NoCs has assumed the use of regular topologies like 2D meshes, some current trends in chip architecture, combined with expected technology limitations and usage models, will very likely oblige designers to consider less regular topologies to provide the best cost-performance trade-off. Moreover, the set of nodes interconnected by those NoCs will also be heterogeneous, including computational cores of different sizes and computing power, cache blocks and local stores, accelerators of different kinds, and memory controllers. The memory wall problem will likely be addressed by using 3D integration, which will increase heterogeneity significantly, due to the need for locating the hottest cores in the top layer.**

Therefore, in order to deliver the best cost-performance trade-off while minimizing resource and power consumption and providing the maximum flexibility, heterogeneity needs appropriate hardware support in the NoC. This talk motivates the need for efficiently supporting heterogeneity, and sketches some results along this direction, describing power-efficient routing algorithms that provide support for multiple heterogeneous, possibly overlapping regions (e.g. virtual machines, coherence domains) in the presence of faulty components.

I. Introduction

Most theoretical studies of interconnection networks assume homogeneous systems with regular topologies. Homogeneity simplifies the design of topologies, routing and load balancing strategies. It also allows the use of a single switch design as a building block for implementing larger switch fabrics. As the number of cores per chip increases, buses are becoming a bottleneck, and researchers started to consider switched networks with point-to-point links in order to make communication bandwidth more scalable. In doing so, the initial proposals for networks on chip (NoCs) inherited many properties of the interconnection networks that were proposed in the 80s. The reason for this is that they share a common goal: minimizing packet latency while devoting the minimal amount of resources to the network. In the case for the interconnection networks developed in the 80s, the goal was to implement single-chip routers. In the case for NoCs, a goal is to minimize silicon area requirements.

Therefore, it is not surprising that the initial proposals for NoCs are based on wormhole switching, two-dimensional meshes, and dimension-order routing (DOR). Wormhole switching delivers low latency with very small buffers, and hence, with small silicon area and power consumption. DOR is easy to implement with a finite-state machine, and leads to very compact and fast routers. Finally, 2D meshes not only were considered the optimal topology for wormhole networks in the 80s. They also match the 2D layout of current chips, enabling the lowest communication latency among neighbors and minimizing wiring complexity. Interestingly enough, some theoretical studies in the 80s concluded that the 2D mesh is the optimal topology under the assumption of constant bisection bandwidth but such a constraint never became true in practice. However, it may become true for NoCs, therefore emphasizing the use of 2D topologies.

II. Sources of Heterogeneity

A. Architectural Sources

Current chips have significantly more complex requirements than just minimizing latency and power consumption. First of all, they need to be connected to the external components (e.g. DRAM modules), and most current processors already integrate one or more on-chip memory controllers. Even if the number of memory controllers is increased as the number of cores increases, there will always be a smaller amount of memory controllers than processing cores or cache blocks. As an example, current graphics processing units (GPUs) integrate several hundred cores and less than ten memory controllers. This introduces heterogeneity in the topology of the interconnection network, which must properly interconnect those devices. Moreover, it also introduces asymmetry in the traffic patterns since memory controllers are usually located near the chip edges. Additionally, since memory bandwidth will become a scarce resource, traffic destined to memory controllers will very likely produce congestion within the NoC, no matter how overdimensioned it is.

Another potential source of heterogeneity for the NoC is the fact that the devices attached to it have different functionality. In addition to processing cores, there are also cache blocks, which are either shared by all the cores or by subsets of them. Those cache blocks will likely differ in size and shape from the cores, possibly introducing irregularities in the topology of the NoC. Even if caches and cores have a similar size, they will generate different traffic patterns that may recommend some asymmetry in the bandwidth of the different links, or even the use of separate networks for different purposes. For example, a design may implement a 2D mesh topology for the transmission of cache lines and a binary tree (or even a fat tree) for the ordered transmission of coherence commands. This kind of solutions have already been implemented in off-chip networks (e.g. Sun E10000), and therefore, are not unexpected for NoCs as well.

Although the number of cores per chip is increasing at a steady rate, many applications are still sequential. Therefore, manufacturers are wondering whether increasing the number of cores per chip for the desktop, laptop and mobile markets is a good idea. In this situation, the best way of using the increasingly higher number of transistors per chip that will become available is by integrating more functionality into the processor chip. The next large component that will be integrated is the GPU, at least for application areas that do not require top graphics performance (e.g. all the desktop and laptop applications except for gaming and engineering design). Both Intel and AMD announced plans for this kind of integration. As a consequence, they need to figure out how to effectively use those GPUs in the server market so as to reuse CPU designs. The solution to this problem has already been provided by Nvidia, and consists of using the GPU as accelerator for numerical applications. High-performance computing (HPC) platforms and datacenters will make good use of those accelerators, not only to drastically increase computing power at a relatively low cost, but also to dramatically enhance the Flops/watt ratio. Again, the implementation of a GPU into the processor chip will introduce significant amount of heterogeneity, due to both the different size of this component and the very different traffic requirements with respect to general-purpose processing cores.

Another way of increasing the Flops/watt ratio is by replacing a small number of complex out-of-order cores with deep pipelines by a large number of simple in-order cores with shallow pipelines. However, manufacturers did not make that move because sequential applications would run much slower, and therefore, the number of sales for multicore chips would have dropped dramatically. A possible approach for increasing the Flops/watt ratio while being able to run sequential applications very fast consists of implementing a small number of complex cores (e.g. one or two) and a large number of simple cores, possibly with the same instruction set architecture (ISA). This is the approach followed by the Cell processor, although in this case the ISA is different for simple and complex cores, thus making it more difficult to generate code for this processor. Again, this strategy constitutes another source of heterogeneity, not only because of the different size of the different kinds of cores, which will force the use of irregular topologies, but also because of the different traffic requirements.

B. Technology Sources

In addition to the above mentioned architectural sources of heterogeneity, manufacturing processes will also force designers to adopt solutions that will end up introducing even more heterogeneity. One of the problems is that, as integration scale increases, the number of manufacturing defects is expected to increase. Therefore, yield will drop dramatically unless future designs incorporate support for fault tolerance. Fortunately, interconnection networks can implement relatively cheap solutions to increase fault tolerance by using the alternative paths provided by the network topology. Such a use, however, introduces asymmetries in the way links can be used, both to avoid deadlocks and also because fault-free regions will have more alternative paths, and therefore, will be less heavily loaded.

Another source of heterogeneity is the expected increase in manufacturing process variability. Up to now, chips are tested to determine the clock frequency at which they can safely run, and therefore, clock frequency is determined by the slowest devices in the chip. This is acceptable because variability is still relatively small. As process variability increases, the former approach becomes less interesting, and researchers are proposing different ways to allow different parts of the chip to run at different speeds. When those techniques are applied to a NoC, they result in links and/or switches that require a variable number of clock cycles to transmit information, depending on where they are located.

It is also predicted that VLSI technology will reach a point in which it will be feasible to integrate more transistors as long as they are not all active at the same time. Future processor chips will implement sophisticated temperature controllers able to dynamically adjust clock frequency for independent clock domains, thus introducing functional heterogeneity even in completely homogeneous systems. But this will affect performance guarantees for different virtual machines (see discussion at the next section). Moreover, pipelined switching techniques like wormhole and virtual cut-through perform quite poorly when some links and/or switches in the path are slower than others, because traffic accumulates at the buffers located before entering the slower regions and may even produce congestion. So, this problem needs to be addressed in order not to waste bandwidth.

The temperature problem will be aggravated with the introduction of 3D stacking technology. Effectively, 3D stacking is considered to be the most promising technology to alleviate the memory bandwidth problem of future multi-core chips. By stacking multiple DRAM chips together with a multi-core chip, it will be possible to drastically reduce the pressure on external memory bandwidth as well as memory access latency. However, those stacked chips will need to dissipate heat, and that heat must go through the already very hot, temperature-throttled multi-core chip. In addition to this, 3D stacking is an important source of heterogeneity. Not only chips in the stack will be different. They will also have very different communication requirements. Moreover, communication with each chip will be significantly faster than communication from chip to chip in the stack, due to the much larger number of wires in the metal layers with respect to the number of vias between chips.

C. Usage Model Sources

Finally, the usage model is also a source of heterogeneity. Current market trends, including outsourcing of IT services, have led to the massive adoption of virtualization as a solution to the problem of running applications from different customers in the same computer while guaranteeing security and resource availability. Moreover, in order to optimize utilization, resources are dynamically assigned to different virtual machines according to customer application requirements. As a beneficial side effect, virtualization splits a given computer into smaller chunks, thus reducing the severity of the problem of developing parallel code. Providing efficient virtualization support at the chip level is not trivial because splitting the set of cores into smaller disjoint subsets is not enough. It is also necessary to guarantee that each partition forms a region whose internal traffic does not interfere with traffic from other regions. This implies that regions should be formed by a contiguous set of nodes and that the routing algorithm is properly defined, which is not trivial because certain paths must be forbidden to avoid deadlocks. Moreover, each region should contain cores and caches in order to be as independent as possible from each other region, which again has some implications on the component layout and introduces heterogeneity at a finer granularity. Anyway, shared caches will introduce some interference among regions. Even if all the cache levels within the chip were private, memory controllers must be shared. This also implies that memory controllers must be an integral part of each and every region, thus introducing even more heterogeneity in the definition of regions.

Finally, there are application areas in which the application to run is fixed (e.g. some embedded devices). In those cases, it is very likely that the application generates non-uniform traffic, sometimes even using only a subset of the links in the NoC. In those cases, efficiently supporting heterogeneity can lead to significant savings in silicon area and power consumption. For instance, an application specific design may implement only the links and switches that are really needed. Also, different links may provide different bandwidth, according to the application requirements.

III. Some Proposed Solutions

Although there exists no generic solution addressing all of the problems mentioned in the previous section, several solutions have been proposed to address one or more of those problems. Some of those solutions are sketched below.

A. *Efficient Unicast and Multicast Support for CMPs*

bLBDR [2], an enhanced version of Logic-Based Distributed Routing (LBDR) [1], is a flexible and efficient unicast and multicast routing method that removes the need for using routing tables (both at end-nodes and switches), and provides support for disjoint and overlapped regions (or domains) in a NoC, thus enabling the concept of virtualization at the NoC level. bLBDR fulfills several of the requirements mentioned in the previous section, including support for virtualization, partitionability, fault tolerance, traffic isolation and broadcast across the entire network as well as constrained to coherency domains or regions. In particular, bLBDR allows the definition of deadlock-free unicast and multicast routing algorithms for irregular topologies and non-rectangular regions that deliver high performance, provide efficient support for cache coherence protocols by implementing collective communication operations, provide very efficient support for partitioning and virtualization by allowing the definition of multiple regions with traffic isolation, and provide support for reconfiguration and fault tolerance. It also provides support for accessing shared components like shared caches and memory controllers by supporting the definition of overlapped regions. All of this is achieved by a small and power efficient routing logic that delivers low latency as well as 4X area savings and 17X power reduction when compared to a routing table in an 8×8 mesh NoC. In fact, this design is almost as compact as the one for switches based on finite state machines (e.g. DOR), while providing almost the same degree of flexibility as the designs based on routing tables.

B. *Application Specific Routing Algorithms for NoCs*

A general-purpose multi-core processor is very flexible but may not be optimal for embedded systems that execute a single application. One way to specialize a general-purpose multi-core chip built using NoC principles is to provide a mechanism to configure an application-specific deadlock-free routing algorithm in the underlying NoC.

In [3], the authors present a methodology to specialize the routing algorithm in table-based NoC routers. It tries to maximize the communication performance while ensuring deadlock-free routing for an application. The paper demonstrates through analysis that routing algorithms generated by this methodology have higher adaptivity as compared to turn-model based deadlock-free routing algorithms for a mesh topology. Performance evaluation with traffic generated by real applications shows that the routing algorithms generated by the proposed methodology achieve an improvement in delay close to 50% and 30% over deterministic XY-routing algorithm and adaptive Odd-Even routing algorithm, respectively.

References

[1] J. Flich, S. Rodrigo, and J. Duato, “LBDR: Efficient Routing Implementation in NoCs”, in Workshop on Interconnection Network Architectures On-Chip, Multi-Chip, 2008.

[2] S. Rodrigo, J. Flich, J. Duato, and M. Hummel, “Efficient Unicast and Multicast Support for CMPs”, in 41st Annual IEEE/ACM International Symposium on Microarchitecture, 2008.

[3] M. Palesi, R. Holsmark, S. Kumar, V. Catania, “A methodology for design of application specific deadlock-free routing algorithms for NoC systems”, Proceedings of the 4th international Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS), pp. 142-147, 2006.

Session I

Router Microarchitecture

Session Chair: Maurizio Palesi*, University of Catania, Italy*

Planar Adaptive Router Microarchitecture for Tree-Based Multicast Network-on-Chip

Faizal A. Samman, Thomas Hollstein and Manfred Glesner
Institute of Microelectronic Systems
Darmstadt University of Technology
Karlstr. 15. D-64283 Darmstadt
Email: faizal.samman, thomas, glesner@mes.tu-darmstadt.de

***Abstract*— Adaptive tree-based multicast routings for networks-on-chip (NoC) in a mesh planar router architecture are presented in this paper. Multicast packets are routed and scheduled in the NoC using a local Identity-based multiplexing technique with wormhole switching. The identity-tag attached to every flit allows different flits of different packets to be mixed in the same queue and enables to implement a fair flit-by-flit round arbitration to share communication links. Hence, deadlock in intermediate nodes as a main problem in the tree-based multicast routing can be handled efficiently and effectively. Some static and planar adaptive routing schemes are implemented to evaluate the impact of the routing algorithms over the NoC performance. The router prototypes have been also synthesized using 130-nm and 180-nm standard-cell technologies.**

I. Introduction

Services in terms of efficient routing and scheduling are critical to the performance of the NoC-based multicore processor systems. Historically, the first generation multicomputers supported only unicast communication (a single PE sends a message to a single PE unit). Nowadays, the recent multicomputers have begun to implement collective communication services. Collective communication services embrace *multicast* (the same message is sent from a source node to an arbitrary number of destination nodes), *scatter* (different messages are sent from a source node to an arbitrary number of destination nodes), and *broadcast* (the same message is sent from a source node to all nodes in the network). With software implementation, a multicast message can be injected into the network by sending a separate copy of the messages from the source to every destination node (unicast-based multicast delivery). However, this approach is unefficient in terms of communication latency and energy.

The multicast delivery service has been intensively used in large-scale multiprocessor systems, and has been a fundamental service of some data parallel computer languages. The following points present the need for multicast services in parallel computing and multicomputer systems [1].

- Numerous parallel algorithms, e.g. parallel search and parallel graph algorithms, has been shown to benefit from the use of multicast service.
- In a single-program multiple-data (SPMD) mode of computation, multicast communication is of benefit. The same program is executed on different processors with different data, and several data are proceeded in parallel.
- In a data parallel mode of computation, a variety of process control operations and global data movement such as *reduction*, *replication*, *permutation segmented scan* and *barrier synchronization* requires collective communication models.
- In a distributed shared-memory paradigm, multicast services may be used to efficiently support shared-data invalidation and updating.

Some NoC-based chip multiprocessors such as RAW machine from *MIT* [2], Tile64 processor from *Tilera* [3], Teraflops from *Intel* [4] and TRIPS chip [5] have been recently published. The RAW machine comprises 16 tiles, where each computing resource tile is connected to programmable routers in a 2D mesh 4x4 topology. The Tile64 processor architecture consists of a 2D 8x8 grid of identical compute elements (tiles). The Teraflops processor architecture contains 80 tiles arranged in a 2D array and connected by a mesh 8x10 network. The TRIPS prototype chip contains two data networks, an on-chip network (OCN) and an operand network (OPN). The OPN consists of two TRIPS processors and is connected to the OCN comprising memory tiles in a 2x8 array structure.

Programming models of the parallel computing systems can be divided into shared-memory, threads, message passing and data parallel programming models. A hybrid parallel programming model can be also developed by combining two or more programming models, e.g. shared-memory model on a distributed memory machine. Most of the multiprocessor systems mentioned before are designed to support thread-level (multithreads) parallelism, shared-memory and message passing programming models. The RAW processor compiler [6] for instance uses the multithreaded program written in a high-level programming language and map it onto RAW hardware. While Tile64 is equiped with C-based interconnect library [3] that provides programmers with a set of commonly used communication primitive such as MPI-like message passing interface for ad hoc messaging.

A framework for automatic parallelization has been introduced in [7]. The aggressive automatic thread extraction framework will let programmers achieve the performance of parallel programming via a simpler sequential programming model. The new era of parallel computation running on a single chip multiprocessor systems is now coming and will be a hot topic. The multicast delivery services whose benefits that have been explored previously will be also an interesting issue for integrating the parallel computing models on the NoC-based multiprocessor systems.

II. Related Works and Motivations

Multicast messages can be routed in the network using *path-based* [1], [8], [9] or *tree-based* [10], [11], [12] multicast routing. However, the multicast networks presented in the abovementioned works are not dedicated for single-chip networks. Indeed, the routing hardware units presented in those works are very complex, and may also increase the logic area after gate-level synthesis. In our NoCs, the adaptive routing algorithms used to route unicast and multicast packets are the same, resulting in a very efficient routing function gate-level implementation.

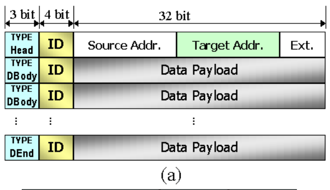
The NoC presented in [13] has introduced the path-based multicast routing to avoid multicast deadlock in the destination nodes by reserving virtual channels and giving priority for the multicast message over the unicast message on arbitration of link bandwidth. Experiments in the work show that the proposed multicast technique improves throughput, and does not exhibit significant impact on unicast performance in a network with mixed unicast-multicast traffic "only if" the network is not saturated. Our proposed multicast scheduling does not give priority for multicast messages (fair flit-by-flit arbitration between the unicast and multicast messages). Our multicast technique does not also present significant impact on the unicast performance "even if" the network is saturated because of the implementation of flit flow control at link-level. Indeed, the NoC in [13] has not been synthesized into logic gate level.

The NoC presented in [14] uses a time-space-time switch designed for time-division-multiplexing (TDM-based) NoCs. Slot map tables as central components are used as time slot interchangers to directly control the read and write operation to random access frame buffers. Unfortunately, although this work has mentioned the feasibility of implementing the multicast scheduling technique, a concrete multicasting procedure, system-level or RTL-level simulations for measuring the NoC performance over multimessage multicast traffics and the NoC capability to handle the multicast deadlock problem are not presented in the paper.

Æthereal NoC [15] and Nostrum [16] have used a time-division-multiplexing approach in order to be able to support the multicast services for further implementation in their NoC architectures. However, experiments by analysing multicast traffics and the NoCs performances over multicast deadlock problem have not been released so far. As far as we know, our NoC, which is called *XHINoC* (e*X*tendable *H*ierarchical and *I*rregular *NoC*) is the first gate-level synthesizable NoC supporting the multicast services. The XHINoC has proposed a new approach for a deadlock-free tree-based multicast routing that can be disjointed into various NoC topologies with specific router microstructures. Hopefully, our investigation under XHINoC infrastructure could make one step forward on the integrated research synergy between on-chip multiprocessor (CMP) in NoC platforms (hardware-level) and parallel computing (software-level) in the future.

III. Tree-Based Multicast Routing

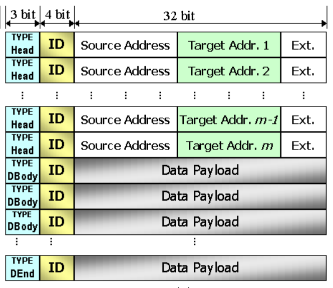
In the tree-based multicast routing, the header ordering in source nodes is not required (the order of the destination addresses can be freely determined). The multicast routing will form communication paths like branches of trees connecting the source node with the destination nodes at the end points of the tree branches. A higher possibility that multicast deadlock occurs in intermediate nodes has alleviated the intentions of the most of network designers to use this method. However, we have introduced a new multicast scheduling for tree-based multicast routing to solve effectively and efficiently the multicast deadlock, which can change the intention of the network designers.



(a)

Type of Flit	Bin. code	Dec. code
Not FLIT	000	0
Header Flit (Head)	001	1
DataBody Flit (DBody)	010	2
Last Flit (DEnd)	011	3

(b)



(c)

Fig. 1. The packet format for (a) unicast and (c) multicast, and (b) binary encoding of the flit types.

A. Multicast Packet Format

The packet format used in our NoC is presented in Fig. 1. Fig. 1(a) presents the packet format for unicast messages. The 39-bit packet consists of a header flit followed by payload flits. The additional heads for each flit are 3-bit flit type and 4-bit packet ID (Identity). The Type can be *header*, *data body*, and the *end of databody* (*last/tail flit*) as shown in Fig. 1(b). Flits belonging to the same message have the same local ID number in a local queue, and vary over different communication links to support scalable concept and wire-share flexibility. Fig. 1(c) shows the packet format for multicast messages. The *m* number of the embedded packet headers is the same as the number of targeted *m* multicast destinations.

B. Routing and Multicasting Procedure

Routing engine (RE) units in our NoC consist of combination of a router hardware logic (*RHL*) unit and a routing lookup table (*LUT*) unit. The combination is aimed at supporting a *runtime link interconnect configuration*. If the *RE* units identify a header flit in the output of a FIFO buffer, then the *RHL* unit will find a routing direction based on destination address stated in the header flit and current address of the router, and assigns the routing direction in a register of the *LUT* unit, and then index it based on its ID-tag. In the next time periods, when the *RE* units identify payload flits with the same ID-tag number with the previously forwarded header flit, then their routing direction will be taken up directly from the *LUT* unit in accordance with their ID-number indexed before.

There are three main steps to deliver a multicast message into multi destination processing elements. Firstly, forwarding all header flits for the multicast tree routing setup and ID-slot reservation. Secondly, broadcasting the payload flits to follow the path set up by the header flits. And the last, setting free the reserved local ID-slot by the tail flit. As shown in Fig. 1(c), each header represents one address of a multicast node.

C. ID-Based Multicast Scheduling

1) ID Slot Allocation: In our NoC, unicast and multicast messages are multiplexed at each outgoing link based on an ID slots allocation technique. As a counterpart of a Time-Division

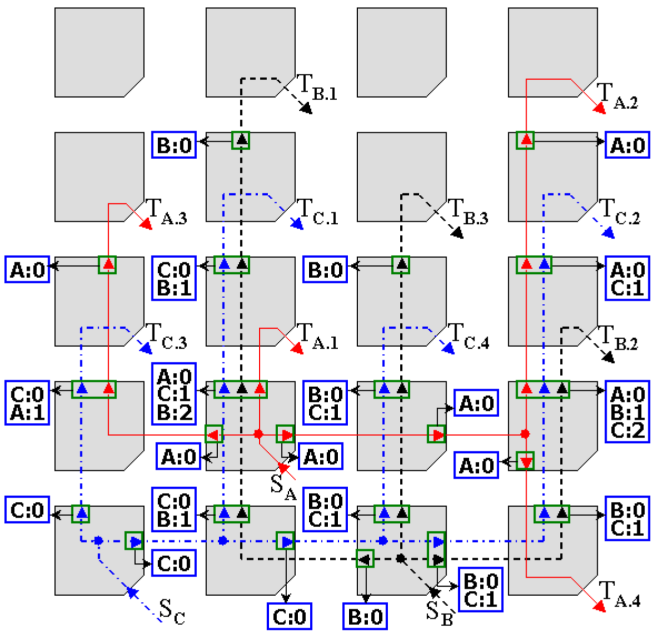


Fig. 2. ID-tag slot allocation of tree-based multicast packet routing.

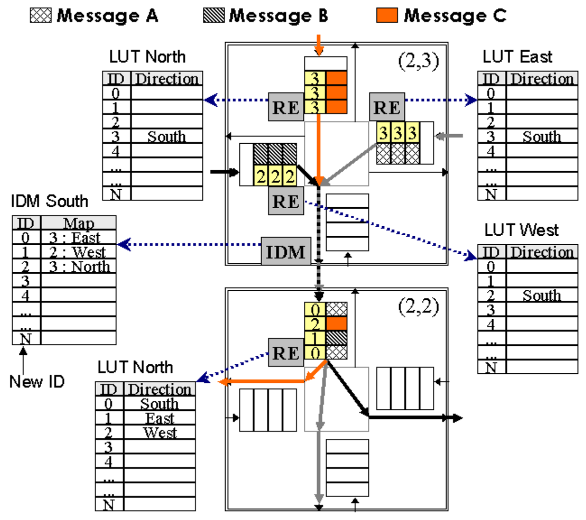


Fig. 3. ID-based routing and packet interleaving.

Multiplexing (TDM) technique, our ID-based Multiplexing technique provides more flexible and optimistic solution for scheduling unicast or multicast message in networks at run-time. There is also no need for a global network view if the link would be scheduled at design time.

Fig. 2 shows how ID-slots of each outgoing link are allocated for each tree-branch of the multicast packets A, B and C which are injected into the mesh 4x4 NoC topology. As shown in the figure, each tree-branch of the multicast message has different local ID-tag. The local ID-tags are updated over the links by using an ID mapping management technique as later explained in Subsections III-C.2 and III-C.3. Some multicast messages have also contentions to access the same outgoing links in the figure, in which multicast deadlocks are performed. In order to overcome that problem, we introduce a fair flit-by-flit hold-release scheduling policy as presented later in Subsection III-C.5.

2) ID-Based Routing Organization: The ID-based scheduling technique enables us to mix different flits of different messages into the same queue and to perform a fair flit-by-flit round arbitration to access outgoing ports. Fig. 3 presents three messages (Messages A, B and C coming from EAST, WEST and NORTH port, respectively) in the router node (2,3) that are switched to the router node (2,2) through the same SOUTH outgoing port. For the sake of simplicity, only routing tables (LUT) of the routing engine (RE) units of the occupied incoming ports are presented in the figure. Message A, B and C have local ID-tag 3, 2 and 3, respectively. Hence, the SOUTH routing direction are indexed and addressed in the routing tables based on the ID-number.

An ID management (IDM) unit at the SOUTH outgoing link as shown in Fig. 3 is used to update the local ID-tag of each packet into a new ID-tag before entering the next downstream router. Each new packet is allocated into a free ID slot and indexed/mapped based on its old local ID-tag and from which port it comes. As presented in the figure, Message A, B and C are mapped into new local ID-tags 0, 1 and 2, respectively. The following subsection will explain how the local ID-tag is updated.

3) ID-Slot Updating and Management: Fig. 4 shows how a packet header coming from NORTH port with ID-tag 3 which is just switched from crossbar switch is updated. The ID update process is described into 4 steps. In the 1^{st} step, the IDM detects a new incoming packet header and then looks for a free ID slot by checking the ID-state table. In this case, the ID-tag 2 is free and then in the 2^{nd} step, the ID is assigned as the new ID-tag for the new packet. In the 3^{rd} step, the ID-slot 2 is indexed based on the old local ID-tag 3 and NORTH data from which it comes. Hence, every time a payload flit coming from NORTH port with ID-tag 3 will have the new ID-tag 2. In the 4^{th} step, ID-tag 2 state is set from "free" to "used", and the number of used ID (UID) is incremented. When the UID is the same as N number of available ID slots, then "empty free ID flag" is set. When a tail flit (the end of databody) is passing through the outgoing port, then the related ID-tag 2 state is set from "used" to "free", the UID is decremented and the information related to the tail flit ID-number is then deleted from the ID Slot Table.

4) ID Slots Requirement: Our current implementation uses 4-bit ID field in each flit. Hence, a maximum number of 16 packets (2^4) can be in flight on the same link. The number of available ID slots can be increased by increasing the number of ID field bits as presented in the packet format, resulting in an increase of the routing table size and ID slot table size in the ID management unit. The number of required ID slots is application-dependent and can not be increased anymore if the NoC had been implemented on ASIC. Hence, an optimal post-manufacture application mapping should be made, in order to avoid that more than 16 packets interfere with each other across the same link.

In coarse-grain multiprocessor applications, where computation to communication ratio is high, it seems that 16 ID slots per channel are enough to run several applications. But, if the computation to communication ratio is low (fine-

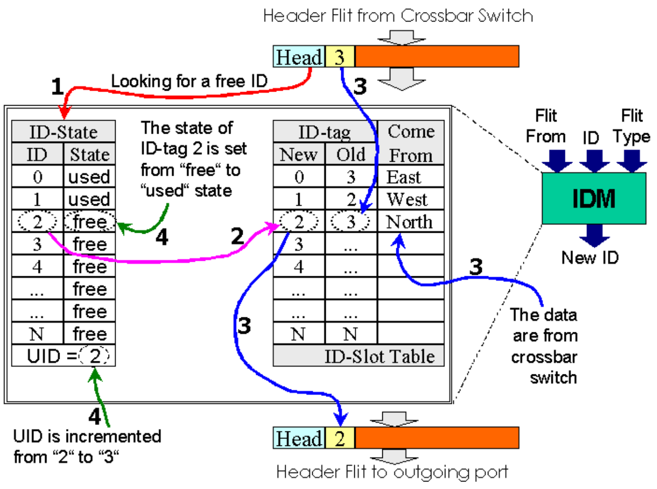


Fig. 4. ID-tag updating and mapping management.

grain), then the number of available ID slots per channel must be strongly considered. The user must ensure that each channel will not be burden with excessive communication loads. Because, even when a channel were shared by 8 to 10 packets, the bandwidth requirements may be not satisfied anymore, and the performance of the application may degrade accordingly. The situation can be compared with a bus system requested by 8 to 10 components simultaneously. Fortunately, traffic behaviors in the context of embedded system-on-chip and multiprocesor applications are predictable. Hence, it is possible to map an application in NoC-platform in such a way that all packets will be able to reserve ID slots to perform requested communications with fulfilled bandwidth requirements.

5) Hold-Release Multicast Fair Scheduling Policy: The tree-based multicast routing is prone to deadlock. The deadlock occurs in a intermediate node when one or more outgoing links are simultaneously requested by the same multicast packets. Therefore, we propose a new methodology to handle the multicast deadlock. Fig. 5 presents 6 Snapshots of the proposed multicast scheduling method and a fair flit-by-flit round arbitration of a so called *hold-release multicast fair scheduling policy* for the deadlock handling mechanism.

- In **Snapshot 1**, three multicast packets, i.e A coming from EAST, B from WEST and C from SOUTH ports request different and the same outgoing links. NORTH and WEST outgoing links are requested by Packets A and C. The other outgoing links are only requested by one Packet, i.e. EAST and SOUTH outgoing links are requested by Packet B, and LOCAL link by Packet C. The flits $A1^0$, $B1^0$ and $C1^0$ represent the flits with local ID-tag 0.
- Although the outgoing links are requested by more than one packet, one by one of the flits of all packets can be granted as a winner to access the outgoing link at every stage as shown in **Snapshot 2**. In this stage, we assume that flit $C1$ is firstly selected to access the NORTH outgoing link, while flit $A1$ is granted as the winner to access the WEST outgoing link. The other outgoing links i.e., EAST, SOUTH and LOCAL, select also their single request from flits in the incoming port.
- In the next stage as presented in **Snapshot 3**, all granted flits are accepted in the outgoing links. However, the states of all flits in incoming are different and depend on whether their multicast requests have been granted by their required outgoing ports. For instance, all multicast request of Packet B to access EAST and SOUTH ports have been granted by these ports. Hence, flit $B1$ (with R state) can be released from FIFO buffer in WEST inport and its request is now replaced by the request of the new incoming flit $B2$. But flits $A1$ and $C1$ (with H state) must be still withheld in input buffers, because their other requests (presented in dashed lines) to access another port have not been granted in this stage. In this stage, all ID-tags of the packets are mapped and updated with new ID-tag 0.
- In the next stage as shown in **Snapshot 4**, by using the flit-by-flit round arbitration method, NORTH and WEST outgoing arbiters change now their selection to other flits, which also request the ports. NORTH port selects now flit $A1$, while WEST port selects flit $C1$. EAST and SOUTH outgoing ports select again the flit coming from WEST incoming port (i.e. flit $B2$), because these ports are only requested by Packet B from WEST incoming port. But the LOCAL outgoing port will not grant again flit $C1$, because flit $C1$ has been granted in the previous stage. This decision is made to avoid flit C1 being transferred two times into the LOCAL outgoing port (avoiding improper multicast replication).
- In the next stage as presented in **Snapshot 5**, flits $A1$, $B2$ and $C1$ are transferred to the outgoing links, and can be released from EAST, WEST and SOUTH input buffers (with R state) respectively, because their multiple requests have been granted previously step by step in **Snapshot 2** and **Snapshot 4**. Their request are now replaced by the requests of new incoming flits i.e., flits $A2$, $B3$ and $C2$. Because ID-tag 0 has been used by packet C in the NORTH and by packet A in the WEST outgoing links, then packet A in the NORTH and packet C in the WEST outgoing links are assigned with new local ID-tags 1 ($A1^1$ and $C1^1$).
- **Snapshot 6** shows generally the same mechanism with the situation shown in **Snapshot 2**.

6) Link-Level and Dynamic Injection Rate Control: Because of using a wormhole packet switching, our NoC is equipped with control mechanisms for flit flow at link-level and dynamic injection rate. Before contentions of the messages to acquire the same communication channels occur, the messages are always injected from the source nodes with a maximum injection rate. The maximum injection rate is in accordance with the allowed maximum data frequency. When the contention occurs, then the FIFO queues in incoming ports occupied by the contenting messages will be congested (full). The congestion (full condition) signals are then traced back to the upstream nodes, and soon or later, the congestion signals will attain the source nodes. By using "request-grant" methodology, in which an *Injection State Controller* (*ISC*) unit in the on-chip network interface will not give an acknowledge signal to inject a new flit into a FIFO queue in a LOCAL port of the on-chip router until one space register of the

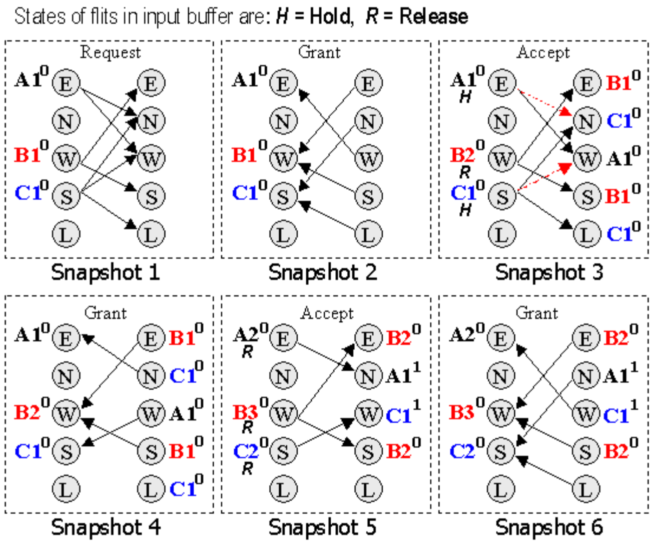


Fig. 5. Hold and Release Multicast Scheduling.

queue is free, then the injection rates at the source nodes can be controlled automatically and dynamically. The same mechanism is applied at link-level (inter-router data transfer), where a *Link State Controller* (*LSC*) unit (as later depicted in Fig. 8) will control the flit transfer from outgoing port to the next downstream FIFO through a communication channel.

IV. On-Chip Router Microarchitecture

A. *2-D Mesh Planar Topology*

Fig. 6 presents an example of the 2-D mesh planar 4x4 network. The network is physically divided into two subnetworks i.e., $X+$ (depicted in solid line arrows) and $X-$ subnetworks (depicted in dashed line arrows). If the x-distance between source and target nodes ($x_{offs} = x_{target} - x_{source}$) is zero or positive, then packets will be routed through the physical channels of the $X+$ subnetwork. If x_{offs} is zero or negative, then the packets will be routed through the physical channels of the $X-$ subnetwork. We can assume that the ports connected with vertical links of $X+$ and $X-$ subnetworks are denoted by (North1, South1) and (North2, South2) ports, respectively. Hence, the packets routed through the $X+$ subnetwork will have adaptivity to make West–North1, West–South1, North1–East and South1–East turns as well as West–East, North1–South1 and South1–North1 non-turn routing. While the packets routed through the $X-$ subnetwork will have adaptivity to make East–North2, East–South2, North2–West and South2–West turns as well as East–West, North2–South2 and South2–North2 non-turn routing directions.

The planar adaptive routing on a mesh topology is firstly introduced in [17] and deadlock-free. Instead of using virtual channels to implements the link interconnect between NORTH and SOUTH port as made in [17], we prefer to implement two physical channels to separate the NORTH–SOUTH link interconnects for $X+$ and $X-$ subnetworks. The objectives of this approach are to maintain the router performance and to increase the network bandwidth capacity. If the virtual channels are implemented in the NORTH and SOUTH ports, then we need to add two virtual queues at both incoming and

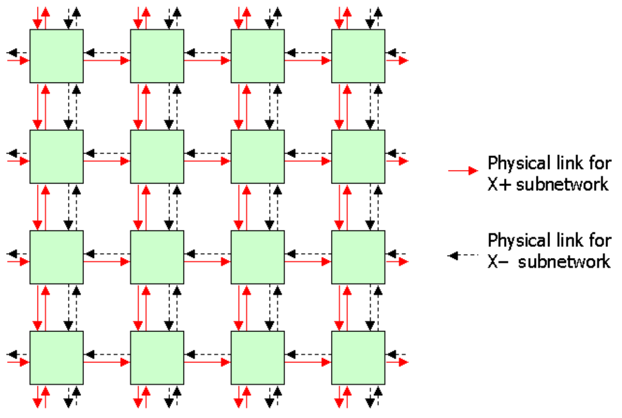

Physical link for X+ subnetwork
Physical link for X− subnetwork

Fig. 6. 2-D Mesh Planar Topology.

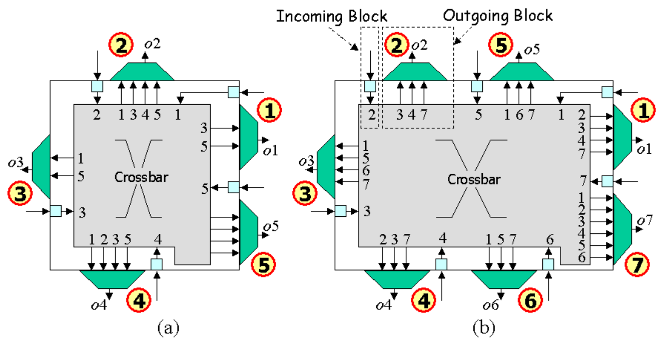


Fig. 7. Switch/Router Structure. (a) Mesh with Static XY routing, (b) Mesh Planar with Adaptive Routing.

outgoing ports. Rather than using such virtual queues, which can degrade router performance characteristic or increase data transfer latency, we substitute them by adding two additional ports (NORTH2 and SOUTH2 ports) in the existing mesh router as presented in Fig. 7(b). In this approach, the number of additional queues is similar to the virtual channel implementation but it maintains the router performance. Nevertheless, the number of input-output pins is certainly increased.

B. *Generic Modules and Modular-Based Design*

Fig. 7 shows the switch structures for networks with standard mesh and mesh planar topology. In the mesh standard, the EAST, NORTH, WEST, SOUTH and LOCAL ports are represented by port numbers 1, 2, 3, 4 and 5, respectively. While in the mesh planar, the EAST, NORTH 1, WEST, SOUTH 1, NORTH 2, and SOUTH 2 and LOCAL ports are represented by port numbers 1, 2, 3, 4, 5, 6 and 7, respectively. The numerical numbers in the crossbar area represents the connectivity between links from the incoming ports to the outgoing multiplexors.

The XHINoC router microarchitecture is developed based on modular units and is grouped into incoming block and outgoing block components. Each module contains generic codes, which are strongly related to the number of input-output connectivities of each port. Fig. 8 shows the incoming and outgoing components in the Port 2 (NORTH 1 port) of the mesh planar router for instance. In the incoming block, we need 3-input GMC (*Grant-Multicasting Controller*) and RDec

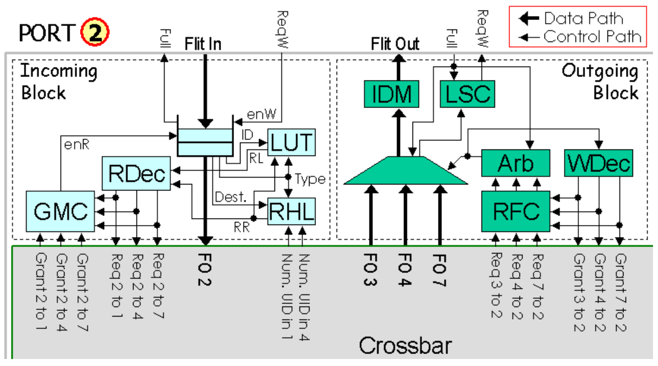


Fig. 8. Components in Port 2 (NORTH 1 Port).

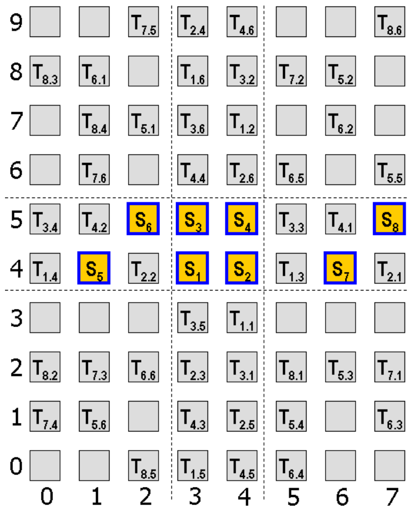


Fig. 9. The selected traffic scenario.

(*Request Decoder*), because the data coming from Port 2 is only connected to outgoing Port 1, 4 and 7. The GMC itself is an important unit to control the multicast flit release from the FIFO buffer. In the outgoing block, we need 3-input Arb (*Arbiter*), RFC (*Request-Feedback Controller*), WDec (Winner-out-Decoder) and 3-input outgoing multiplexor, because the data going out to Port 2 are from incoming Port 3, 4 and 7. The RFC unit is used to control multicast requests for avoiding improper multicast flit replications.

The design of the XHiNoC routers are fully customized on demand. However, each VHDL entity contains generic codes, which enable us to derive a new VHDL module with different behavioral architecture and the number of input/output pins according to the specification. The custom-generic modular-based design approach enables us to develop easily irregular NoC topologies.

C. *Routing Algorithms*

In this paper, we will evaluate our proposed mesh topologies by using four mesh prototypes with different routing algorithms. The first prototype (mesh XY) uses a static XY routing algorithm in the standard mesh topology (using a router as in Fig. 7(a)). The remaining three prototypes use 2-D planar adaptive routing algorithms in the mesh planar topology (using router as in Fig. 7(b)). In the second prototype (mesh PA XP), packets from the LOCAL port that can be routed adaptively to horizontal or vertical direction will be prioritized to select the horizontal direction, while in the third and fourth prototypes (mesh PA YP and PA ZZ), the packets from the LOCAL port will be prioritized to select the vertical direction. However, in the second and the third prototypes, the packets prefer to make non-turn routing direction from any port (except from LOCAL port) if the packets can be routed adaptively to horizontal or vertical direction. While in the fourth prototype, packet will make a zig-zag routing selection. The impact of such routing algorithms over the NoC performance will be explored in Section V as follows.

V. Experimental Results

A. *Selected Traffic Scenario*

Fig. 9 exhibits the traffic pattern used to verify our proposed scheduling methodology and algorithms. Although the traffic pattern does not represent an example of a real application, we are sure that the scenario can be accepted as one of among our best-case scenario to verify our methodology. This pattern performs some multi deadlock configurations because of contentions of some multicast messages to acquire the same outgoing links. Synthesized and randomly selected source and target nodes are mixed in the traffic in such as a way that multicast conflicts occur in the NoC.

Eight multicast messages (denoted with S_n in Fig. 9) are injected to the NoC at the same time and is replicated, broadcasted and flows in the networks. Each message has six multicast destination ($T_{n.m}$ symbols denote the target node m of a multicast message injected from source node S_n). As described in Subsection III-C.6, the messages are firstly injected with a maximum injection rate. The injection rate is then decreased and increased automatically and dynamically depending on the existing contention in the NoC routers. The more multicast messages are involved in a contention to acquire the same outgoing channel in a certain router node, the smaller the injection rates of the involved multicast message will be.

In each S_n node, 2048 flits are injected, resulting in a total number of 16384 (8 × 2048) flits are injected to the source nodes, and a total number of 98064 flits are ejected from the multicast destination nodes. Each message injected from a certain node is encoded to recognized it from other multicast messages. Every flit is numbered in-order to enables us to check the flits one-by-one in our testbench program, whether any flit looses or is replicated improperly in the network or is accepted out-of-order in the destination nodes.

B. *Performance Measurement Result*

The experiment result has also proved, that all flits of the messages are accepted in-order (the same order as in injection nodes). The out-of-order problem has been successfully handled because of the packet choice and the working organization between router hardware logic and routing look-up table units at each incoming port and the IDM units at each outgoing (See again Fig. 3). Each message is associated as a single packet (even if the message size is extremely large, a stream data for instance) with one header flit for each destination nodes. A routing decision (statically or adaptively) in each router node will be made once by the header flit, then all payload flits will just follow the path set up by the header flit. It means that the message will “not be divided” into several packets. Hence, the out-of-order problem will not appear in our NoCs.

The experiment has also exhibited that there is no improper flit replication. It resumes that our “hold-release” scheduling policy has successfully control and manage the existing multicast conflicts at each intermediate node. Table I shows the measurement of the tail flit (the end of payload flit) acceptance latency. The table reports the transfer latencies to accept the tail flits in target nodes $T_{n.m}$ of each message injected from S_n node. The transfer latency is measured based on the number of clock cycle periods to transfer tail flits (the end of packet/message bodies) starting from injection nodes until ejection (destination) nodes.

The RTL-simulations are run with similar clock periods to evaluate all NoC prototypes. As explained in Subsection III-C.6, all multicast messages are injected at the first-time with a maximum injection rate. The injection rate at each source node can change then dynamically and automatically due to the contention of the message injected from the source node with other messages to shares the requested channels. Therefore, transfer latency of the tail flit of the message will increase because of the reduced injection rate. The maximum tail flits transfer latencies of the multicast messages injected from S_n are then reported in Fig. 10. The figures shows that the PA YP prototype (See also Subsection IV-C) exhibits the best performance over the other prototypes.

Generally, the mesh planar (PA XP, PA YP and PA ZZ) prototypes give better performance than the mesh XY prototype specifically in this scenario, because the mesh planar prototypes have higher bandwidht capacity in the double vertical links connecting the NORTH and SOUTH ports. The mesh planar PA YP presents also the best performance in this scenario, because routing multicast packets firstly to vertical direction will reduce the possibility of multicast contentions to occur. The general result of this measurement is only valid in this scenario. When the traffic is low or medium high, the planar adaptive routing algorithms give always better performance. In a certain traffic pattern (especially if the traffic is very high), the performance of the static tree-based multicast is often better than the planar adaptive multicast routing. However, one general result that satisfies our expectation is, that the multicast deadlock configuration can be successfully tackled regardless of the multicast routing algorithm choice.

TABLE I
TAIL FLITS ACCEPTANCE LATENCY MEASUREMENT

		Target $Tn.m$					
S_n	Alg.	$T_{n.1}$	$T_{n.2}$	$T_{n.3}$	$T_{n.4}$	$T_{n.5}$	$T_{n.6}$
S_1	XY	14318	14334	14320	14322	14318	14314
	PA XP	12264	12272	12276	12272	12264	12268
	PA YP	9538	9556	9546	9554	9538	9540
	PA ZZ	12274	12278	12282	12280	12270	12274
S_2	XY	14322	14326	14326	14336	14320	14320
	PA XP	12270	12266	12276	12292	12270	12272
	PA YP	9546	9548	9546	9556	9544	9540
	PA ZZ	12264	12272	12264	12274	12262	12258
S_3	XY	14332	14332	14320	14320	14324	14330
	PA XP	16360	16360	16352	16352	16352	16358
	PA YP	9548	9554	9540	9546	9542	9546
	PA ZZ	12278	12276	12274	12270	12266	12266
S_4	XY	18402	18398	18404	18392	18390	18396
	PA XP	18400	18396	18402	18390	18388	18394
	PA YP	9556	9556	9556	9546	9544	9548
	PA ZZ	12278	12278	12278	12266	12266	12270
S_5	XY	14320	14304	14342	14344	14348	14350
	PA XP	8190	8180	8210	8210	8216	8210
	PA YP	12272	12256	12272	12272	12292	12292
	PA ZZ	12274	12242	12260	12260	12290	12294
S_6	XY	14322	14320	14360	14358	14340	14348
	PA XP	8186	8182	8216	8212	8198	8202
	PA YP	12270	12266	12290	12286	12278	12284
	PA ZZ	12270	12266	12300	12296	12278	12286
S_7	XY	14300	14322	14340	14348	14348	14344
	PA XP	12260	12280	12292	12302	12302	12294
	PA YP	8176	8202	8206	8214	8218	8196
	PA ZZ	8176	8184	8206	8214	8200	8192
S_8	XY	18402	18386	18422	18414	18422	18422
	PA XP	18400	18384	18420	18412	18420	18420
	PA YP	8196	8188	8220	8198	8216	8216
	PA ZZ	8196	8170	8194	8186	8216	8216

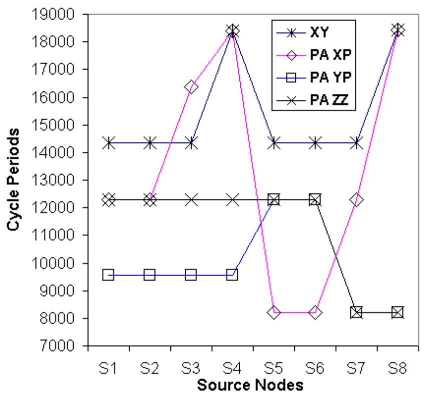


Fig. 10. The maximum tail flit transfer latency per multicast message.

VI. Synthesis Results

Our router prototypes have been synthesized using CMOS 130-nm and 180-nm standard-cell libraries from *UMC* (United Microelectronics Corporation). Table II presents the synthesis reports of the number of consumed logic cells and estimated logic cell area for standard multicast mesh router using static XY routing algorithm and extended multicast mesh router using planar adaptive routing algorithm with Y-direction adaptive priority (PA YP prototype). It looks that the cell area overheads to synthesis mesh PA YP prototype over mesh XY prototype are 47% and 49% using 130-nm and 180-nm UMC technology respectively. The significant area overheads are due to the use of adaptive routing mechanism and extra I/O ports in the mesh planar adaptive router microarchitecture.

Fig. 11 presents the circuit layout of the mesh planar PA YP prototype using *Cadence Silicon Encounter* tool. The cell area of the IDM units are highlighted in the figure with bright color. In the future, the power dissipations of the NoCs over various traffic scenarios will be analyzed. In our previous investigation [18], we have evaluated that the area overhead to update the NoC from unicast to multicast with 8-register buffer size and the same static XY routing algorithm and the same standard mesh router is only about 15%.

TABLE II
SYNTHESIS RESULT

Mesh Router Type	Mesh MC XY		Mesh MC PA YP	
UMC CMOS techn.	130-nm	180-nm	130-nm	180-nm
Num. of cells	7607	7418	11104	11038
Num. of nets	7855	7757	11444	11460
Total cell area (mm^2)	0.107	0.187	0.157	0.278
Num. of ports (pin)	420	420	584	584

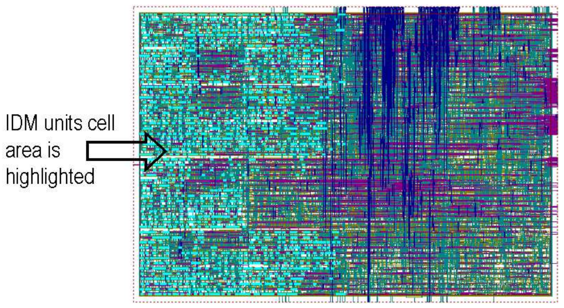


Fig. 11. Automatic place and route of the multicast mesh router PA YP prototype using 180-nm standard-cell library from UMC.

VII. CONCLUSION

The novel scheduling mechanism for tree-based multicast routing using deadlock-free static and partially planar adaptive routing algorithms has successfully solved the multicast deadlock configuration problem in the intermediate nodes of the NoC. By using the hold-release scheduling rule and the ability of the on-chip router to interleave flits of different messages in the same queue, the multicast deadlock problem can be solved easily. The methodology can also guarantee lossless flit acceptance in multiple destination nodes even if the size of the multicast messages is very long (e.g., a streaming data in video application). There is also no out-of-order delivery problem, even if the adaptive routing algorithms are used because of the packet format choice and the dedicated working organization of the combined router hardware logic and routing look-up table units.

In general, our proposed NoC prototypes with planar adaptive routing algorithm can give better performance using a certain test traffic scenario because of the higher bandwidth capacity of the NoC in double vertical links connecting NORTH and SOUTH ports. Nevertheless, this performance gain must be paid by logic area overhead to implement the mesh planar router architecture.

The development of low-level message passing and distributed shared-memory programming models for a multiprocessor system under XHINoC platform is now in progress. Some open core MIPS processors are connected to XHINoC routers via a customized network-interface. We can now involve multicast instructions at higher protocol layer, because the multicast service has been implemented in the network and data-link layers. In the future, we will also develop a heterogeneous NoC-based system by integrating some IP components and open embedded processor cores to evaluate the efficiency of the multicast communication for certain applications.

REFERENCES

[1] X. Lin, P. K. McKinley and L. M. Ni, “Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers,” *IEEE Trans. on Parallel and Distributed Systems,* vol. 5, no. 8, pp. 793–804, Aug. 1994.

[2] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, et. al., “The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs,” *IEEE Micro,* vol. 22, issue 2, pp. 25–35, Mar/Apr. 2002.

[3] D. Wentzlaff, P. Griffin, H. Hoffmann, L. Bao, B. Edwards, C. Ramey, M. Mattina, et. al., “On-Chip Interconnection Architecture of the Tile Processor,” *IEEE Micro,* vol. 27, no. 5, pp. 15–31, Sep/Oct. 2007.

[4] Y. Hoskote, S. Vangal, A. Singh, N. Borkar and S. Borkar, “A 5-GHz Mesh Interconnects for A Teraflops Processor,” *IEEE Micro,* vol. 27, no. 5, pp. 51–61, Sep/Oct. 2007.

[5] P. Gratz, C. Kim, K. Sankaralingam, H. Hanson, P. Shivakumar, S. W. Keckler and D. Burger, “On-Chip Interconnection Networks of the TRIPS Chip,” *IEEE Micro,* vol. 27, no. 5, pp. 41–50, Sep/Oct. 2007.

[6] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, et. al., “Baring it All to Software: Raw Machines,” *Computer,* vol. 30, issue 9, pp. 86–93, Sep. 1997.

[7] M. J. Bridges, N. Vachharajani, Y. Zhang, T. Jablin and D. I. August, “Revisiting the Sequential Programming Model for the Multicore Era,” *IEEE Micro,* vol. 28, no. 1, pp. 12–20, Jan/Feb. 2008.

[8] J. Duato, “A Theory of Deadlock-Free Adaptive Multicast Routing in Wormhole Networks,” *IEEE Trans. on Parallel and Distributed Systems,* vol. 6, no. 9, pp. 976–987, Sep. 1995.

[9] R. V. Boppana, S. Chalasani and C. S. Raghavendra, “Resource Deadlocks and Performance of Wormhole Multicast Routing Algorithms,” *IEEE Trans. on Parallel and Distributed Systems,* vol. 9, no. 6, pp. 535–549, June 1998.

[10] M. Barnett, D. G. Payne, R. A van de Geijn and J. Watts, “Broadcasting on Meshes with Worm-Hole Routing,” *Journal of Parallel and Distributed Computing,* vol. 35, no. 2, pp. 111–122, 1996.

[11] M. P. Malumbres, J. Duato and J. Torrelas, “An Efficient Implementation of Tree-Based Multicast Routing for Distributed Shared-Memory Multiprocessors,” *in Proc. of the 8th IEEE Symposium on Parallel and Distributed Processing,* pp. 186–189, 1996.

[12] D. R. Kumar, W. A. Najjar and P. K. Srimani, “A New Adaptive Hardware Tree-Based Multicast Routing in K-Ary N-Cubes,” *IEEE Trans. on Computers,* vol. 50, no. 7, pp. 647–659, July 2001.

[13] Z. Lu, B. Yi and A. Jantsch, “Connection-oriented Multicasting in Wormhole-switched Network-on-Chip,” *Proc. IEEE Comp. Society Annual Symposium on VLSI (ISVLSI’06),* 6 pp., 2006.

[14] J. Liu, L.-R. Zheng and H. Tenhunen, “Interconnect intellectual property for Network-on-Chip (NoC),” *Journal of Systems Architecture,* vol. 50, issue 2–3, Feb. 2004, pp. 65–79.

[15] E. Rijpkema, K. Goossens, A. Radulescu, J. Dielissen, J. van Meerbergen, P. Wielage and E. Waterlander, “Trade-offs in the design of a router with both guaranteed and best-effort services for networks on chip,” *IEE Proc. Computers and Digital Techniques,* vol. 150, no. 5, pp. 294-302, Sep. 2003.

[16] M. Millberg, E. Nilsson, R. Thid and A. Jantsch, “Guaranteed Bandwidth using Looped Containers in Temporally Disjoint Networks within the Nostrum Network on Chip,” *Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE’04),* pp. 890–895, 2004.

[17] A. A. Chien and J. H. Kim, “Planar Adaptive Routing: Low-Cost Adaptive Networks for Multiprocessors”, *in Proc. of the 19th Int’l Symp. on Computer Architecture*, pp. 268–277, May 1992.

[18] F. A. Samman, T. Hollstein and M. Glesner, “Multicast Parallel Pipeline Router Architecture for Network-on-Chip,” *in Proc. Design, Automation and Test in Europe Conf. and Exhibition (DATE’08),* pp. 1396–1401, March, 2008.

DMesh: a Diagonally-Linked Mesh Network-on-Chip Architecture

Wen-Hsiang Hu, Seung Eun Lee, and Nader Bagherzadeh
Department of Electrical Engineering and Computer Science
University of California, Irvine
Irvine, CA 92697 USA
{wenhsiah, seunglee, nader} @uci.edu

Abstract— In this paper, we propose a new mesh-typed NoC architecture which aims at enhancing network performance while keeping implementation cost feasible. The result is a diagonally-linked mesh (DMesh) NoC that uses wormhole packet switching technique. Together with the proposed adaptive quasi-minimal routing algorithm, DMesh improves average latency and saturation traffic load. In addition, logic synthesis results show that adding diagonal links is a more area-efficient way for increasing network performance than using large buffers.

I. INTRODUCTION

As semiconductor technology continues its phenomenal growth and follows the Moore's Law, the amount of computation power and storage that can be integrated on a chip increases. There have been previous articles reporting a single chip that incorporated 64 cores [1] and another one with 80 cores [2]. While the computation logic grows, the performance of on-chip interconnections does not scale as well. Starting with 0.25μm CMOS technology, wire delay dominates gate delay and the gap between wire delay and gate delay becomes wider as process technology improves. In addition, human design productivity can not keep up with the growth rate of available circuits on a single chip. These issues call for a well-structured design approach, modularized design methodology, clear programming model and predictable behavior of the system [5]. There is a need for a new on-chip interconnection architecture to solve these design challenges.

Network-on-chip (NoC) interconnection scheme is proposed as a unified solution for the design problems faced in advanced process technology [3][4]. With NoC, we can apply wire segmentation and wire sharing design techniques to resolve the performance bottleneck due to wire delay. NoC uses a distributed control mechanism, resulting in a scalable interconnection network. The use of standardized sockets enables modular design and intellectual property (IP) reuse and the system predictability can be obtained by using guaranteed service provided by NoC. Therefore, there is growing interest in NoC research [5][6] and NoC is considered as a practical approach for the next-generation on-chip interconnection.

We have recently developed a multi-processor system platform called Network-based Processor Array (NePA) [7] in which the processors are interconnected by using an on-chip two-dimensional (2D) mesh network. The NePA NoC is a deadlock-free and livelock-free network that implements the wormhole packet switching technique and utilizes an adaptive minimal routing algorithm. To further improve the performance of NePA NoC, we propose in this paper to add diagonal links to the 2D mesh network, because of the emergence of X-architecture routing technique in chip manufacturing [8][9]. The diagonal links not only reduce the distance between a source node and a destination node but alleviate traffic congestion in the network so that the network performance is enhanced. Our proposed NoC architecture is referred to as DMesh: Diagonally-linked Mesh. Simulation results from self-similar traffic show that DMesh improves the average latency and the saturation traffic load on both 4x4 and 8x8 mesh networks. In addition, logic synthesis results in TSMC 65nm CMOS process show that adding diagonal links is a more area-efficient way to improve network performance than increasing buffer size.

The rest of this paper is organized as follows: Section 2 presents the background knowledge of NoC architecture and related researches. The proposed DMesh NoC architecture is discussed in Section 3. Section 4 presents experimental results. Finally, brief statements conclude this paper in the last section.

II. BACKGROUND

In this section, we discuss the background of NoC architecture and provide a review of some related works in this field, as well as an overview of the NePA platform.

A. *NOC Architecture*

The function of an on-chip network is to deliver messages from source node to destination node and there exist many design alternatives to accomplish this job. Depending on the application requirements, how to choose suitable network architecture remains an open problem in this field of research. Here we discuss the network properties that need to be considered when devising an NoC architecture for specific application needs.

1) *Switching policy*

There are two major switching techniques: circuit switching and packet switching. Circuit switching establishes a link between source node and destination node either virtually or physically before a message is being transferred. The link is held until all the data is transmitted. The major advantages of circuit switching are that there is no contention delay during message transmission and its behavior is more predictable, so circuit switching is usually employed when Quality of Service (QoS) is considered. Examples of using this technique are [15] and [16].

On the other hand, packet switching transfers messages on a per-hop basis. With packet switching, messages are divided into packets at the source node and then sent into a network. Packets move along a route determined by the routing algorithm and traverse through a series of network nodes and finally arrive at the destination node. Packet switching is utilized in most of NoCs because of its potential for providing simultaneous data communication between many source-destination pairs. Readers are referred to [6] for a list of NoCs utilizing packet switching techniques. Packet switching can be further classified into three classes: store and forward (SAF), virtual cut through (VCT), and wormhole switching. The most popular one for NoC based architectures is wormhole switching because it only requires a buffer size of one flit (flow control unit) so that the area cost of a router can be kept low. In contrast, SAF and VCT require a buffer size of the whole packet which prohibits their adoption.

2) *Topology*

Topology defines how nodes are placed and connected, affecting the bandwidth and latency of a network. Many different topologies have been proposed, [6], such as mesh, torus, binary tree, Octagon, mixed and custom topology, as shown in Fig. 1. Some researchers have proposed the application-specific topology that can offer superior performance while minimizing area and energy consumption [17][18]. The most common topologies are 2D mesh and torus due to their grid-type shapes and regular structure which are the most appropriate for the two dimensional layout on a chip.

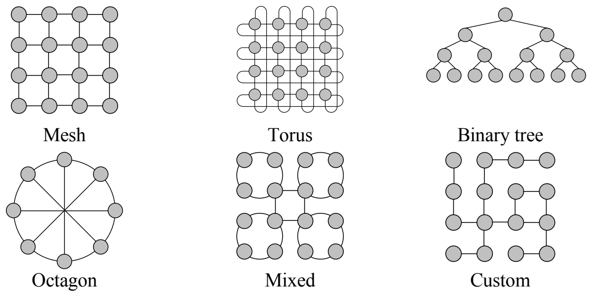


Figure 1. NoC topologies.

3) *Routing*

Routing is the mechanism responsible for determining the path that a packet traverses from the source node to the destination node. Routing algorithms such as deterministic and adaptive ones have been proposed. With deterministic routing, the path between source-destination pair is fixed, regardless of the current state of the network. On the other hand, an adaptive routing algorithm takes the network state into account when deciding a route, resulting in variation of the routing path with time. For example, it may choose an alternative path if a certain link is congested, therefore, an adaptive routing algorithm has the potential of supporting more traffic for the same network topology. However, most of the proposed packet-switched NoCs use deterministic routing because of its simplicity and the low area overhead in router design.

B. *NePA*

We provide an overview of the NePA architecture in this section. NePA implements the wormhole packet switching technique and the topology of NePA is based on a 2D mesh as shown in Fig. 2. Each node in NePA consists of a router and a local IP which can be a CPU, DSP, memory block, or application-specific logic. The router connects with its four neighboring routers via six bidirectional links. A key feature of the NePA architecture is the use of two separate vertical links which are employed to construct a deadlock-free network [19]. The NePA network is actually composed of two disjoint sub-networks. One sub-network is responsible for delivering east-bounded packets while the other one is for west-bounded packets. Therefore, cycles in the resource dependence graph [14] and prevent deadlocks from happening. This design technique reduces the design complexity of the router because there is no need for a deadlock aware routing algorithm. To increase network performance, NePA utilizes an adaptive XY routing algorithm. When an output port is congested, or the output buffer is full, the router selects an alternative output port for packets. Therefore, the link utilization is balanced and network performance improves.

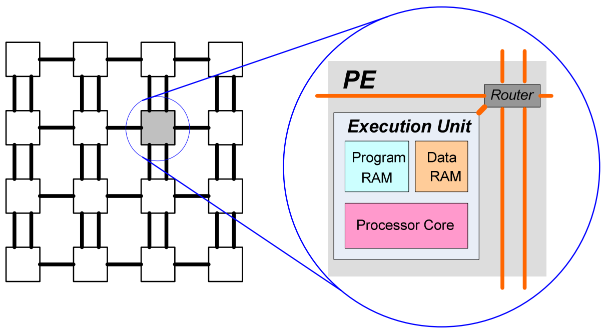


Figure 2. A 4x4 NePA network and its node composition.

III. DMesh ARCHITECTURE

A. *Topology*

The DMesh network is constructed by integrating diagonal links to NePA, as presented in Fig. 3. Each node has 10 64-bit bidirectional links connecting with its neighbors so the DMesh router has 10 output ports (*N1/N2/S1/S2/E/W/NE/NW/SE/SW-out*) and 10 input ports (*N1/N2/S1/S2/E/W/NE/NW/SE/SW-in*). Additionally, there are three ports for connection with local PEs: *IntR*, *IntL* and *Int*. Fig. 4 depicts the input and output ports of NePA router and DMesh router. The DMesh network is composed of two sub-networks: *E-subnet* and *W-subnet*, represented in dashed arrows and solid arrows in Fig. 3, respectively. The E-subnet is responsible for transferring

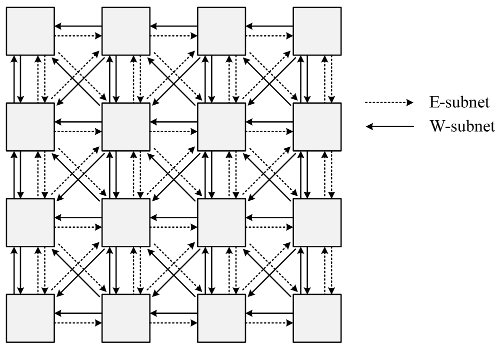


Figure 3. Topology and links of DMesh.

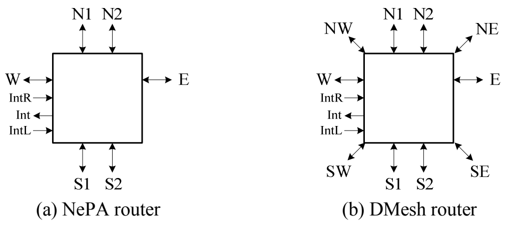


Figure 4. Ports of NePA router and DMesh router.

packets eastward while the W-subnet is for transmitting westward traffic. When source PE starts packet transmission, it injects packets into the network via IntR or IntL port, depending on the direction of destination PE. The IntR port is in charge of injecting packets into the E-subnet and the IntL port is for the W-subnet. Then the packets traverse in one of the sub-networks to their destinations. When packets arrive at the destination node, they are ejected from the Int port.

B. *Packet format*

With wormhole packet switching, DMesh packets are composed of 64-bit flits. We utilized the same packet format defined for NePA in [19]. There are four types of packets defined. The single data transfer packet consists of one flit and is used for transferring 32-bit data. The single command packet is for building control specific protocols between processor elements (PEs) or between a PE and a router.

DMesh also supports multiple data packets, which are used for transmitting more than one 32-bit words at a time, because multiple data transmission has better performance in terms of communication overhead than the single data transmission. Two different block transfers are defined. One is block program transfer packet which is used for programming each PE. The other is block data transfer packet used for transferring multiple data words between PEs. The block program/data transfer packet consists of a header flit and a series of body flits which contains the actual program/data to be transmitted. The number of body flits is encoded in the header flit.

The address of destination PE is represented in the X-dir field and Y-dir field in a relative distance format. For instance, if the destination node is on the east side of the source node the X-dir field has a positive value. The X-dir field has a negative value if the destination node is on the west side of the source node. This technique of relative address representation helps reduce router design effort because a same router can be applied to all network nodes without any modification. We also incorporated the seq_num field in the packet for reordering out-of-order delivery. The single/block data transfer packet has the sourcePE_address field and the application-dependent data_ID field in order for the destination PE to identify received data.

C. *Routing*

We devised a distributed adaptive routing algorithm for DMesh. With distributed routing, the selection of the next hop is decided at the current node and the path selection is based on a quasi-minimal routing technique. Take Fig. 5 for example, if there is a packet being transferred from node S to node D, there are three alternative paths: *a*, *b*, and *c*. Clearly, path *a* is the shortest path. However, from our preliminary simulation, if a minimal routing algorithm is adopted and we always choose the shortest path there will be severe congestion on diagonal links and low utilization on vertical and horizontal links. Thus, the network performance is impacted. Our quasi-minimal routing relaxes the output port selection. In this example, it allows packets to take path *b* or path *c* depending on the network state, if path *a* is congested. Although the packet may traverse a longer path, this approach helps balance link load and relieve congestion. In order to solve contention at an output port, we employed a fixed-priority scheme for arbitration. In general, the diagonal input ports are given the highest priority, then the horizontal and vertical input ports, and IntR and IntL have the lowest priority.

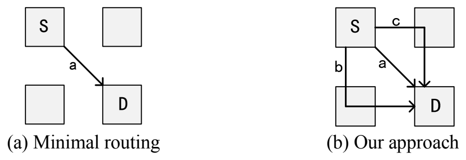


Figure 5. Example of route selection

IV. Performance and Cost Evaluation

Here we describe the methodology used to analyze the performance and area cost of DMesh architecture and present the results.

A. *Performance evaluation*

To evaluate DMesh performance and compare it with NePA, we constructed a SystemC based cycle accurate simulator called eNoC. In eNoC, we can change various network configurations, such as network size, topology, buffer size, routing algorithm, priority scheme for router arbitration, and traffic pattern. There are four different traffic patterns used for measuring the performance: uniform random, bit complement, bit reverse and matrix transpose traffic patterns. These patterns define the spatial distribution of packets.

As for the temporal distribution of packets, we adopted the self-similar traffic generation techniques. Self-similar traffic has been found in the traffic between on-chip modules in MPEG-2 video applications [10] and conventional computer networks [11]. Researchers [12] have shown that self-similar traffic can be generated by aggregating a large number of packet sources which exhibit a long-range dependence property. We used the modeling method proposed in [13] to produce the self-similar traffic. During simulation, each source node is either in the ON or OFF state. A source node generates packets when it is in the ON state and it does not generate any packets when in the OFF state. The length of time a node spends in the ON or OFF states is determined by the Pareto distribution ($F(x) = 1 - x^{-\alpha}$, $1<\alpha<2$). The equations for calculating ON and OFF times are

$$T_{ON} = U^{-1/\alpha_{ON}} \quad (1)$$

$$T_{OFF} = U^{-1/\alpha_{OFF}} \quad (2)$$

U is a uniformly distributed value in the range of (0, 1], α_{ON} = 1.9 and α_{OFF} = 1.25.

We used a standard interconnection network measurement setup described in [14]. After a packet is generated, it is stored in an infinite queue at the source node and waits for being injected into the network. This mechanism referred to as the open-loop measurement configuration isolates the packet generation from the network behavior, i.e. the packet generation is independent of the network condition. Each simulation executes 10,000 clock cycles for warm-up and then continues for 100,000 cycles during which performance measurements are conducted.

Two performance metrics are of importance to us: latency and throughput. In order to compute latency information, each flit in eNoC is declared as a SystemC object that carries four latency related private variables: generation time (T_g), injection time (T_i), arrival time (T_a) and inter-node distance (D). Inter-node distance is represented in terms of the number of hops between source-destination pairs. With this information, the latency, queuing delay, and blocking time of each flit can be calculated by the following equations:

$$\text{Latency} = T_a - T_g \quad (3)$$

$$\text{Queuing delay} = T_i - T_g \quad (4)$$

$$\text{Blocking time} = T_a - T_i - (D * \text{clock cycle time}) \quad (5)$$

The average inter-node distance is shown in Table I. We can see that our routing algorithm makes efficient use of diagonal links so that the inter-node distance is reduced in all traffic patterns. Matrix transpose traffic has the largest improvement and makes the most of the diagonal links because the source-destination pairs are all symmetric to the diagonal in a matrix.

The comparison of average latency in 4x4 and 8x8 networks under four different traffic patterns is shown in Fig. 8. In both 4x4 and 8x8 network size, DMesh outperforms NePA. In particular, the 4x4 network under bit reverse and matrix transpose traffic, the latency in DMesh is a constant because the network is capable of resolving all routing resource contentions. That is, each source-destination pair can obtain an alternative path that is not occupied by other packets. In Fig. 6, we compare the queuing delay, traverse latency, and blocking time for 4x4 and 8x8 networks under random traffic. For different traffic loads, all of these delays are decreased in DMesh. The saturation load (the point where throughput no longer grows linearly with traffic load) in various configurations are summarized in Table III. It can be observed that DMesh is able to sustain higher load than the NePA. For random traffic, the improvement in the 8x8 network is more than the 4x4 network which implies that DMesh has a greater impact on systems with more nodes. From Table III, it can be seen that the increase in FIFO sizes does not help much with the saturation load.

B. *Area cost and power consumption evaluation*

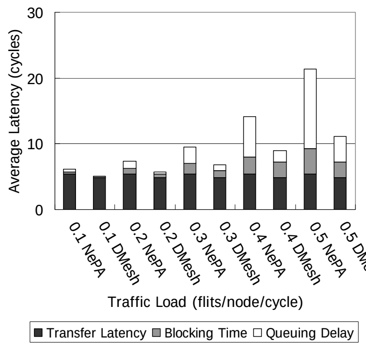
In order to estimate hardware cost, we implemented the NePA router, the DMesh router and the FIFO buffer in Verilog and performed logic synthesis by using the Synopsys Design Compiler to get gate count information. Various buffer sizes were also evaluated. For the NePA router, we followed the architecture described in [7]. The block diagram of DMesh router is presented in Fig. 7. The DMesh router has three sub-routers for processing traffic in the E-subnet, W-subnet and Int output port. There is a FIFO associated with each input port. Header processing unit (HPU) extracts destination information from the header flit and routing logic (RL) is used to decide routing path, perform arbitration and control the crossbar switch.

We used TSMC 65nm CMOS generic process technology in logic synthesis. The target clock rate is set to be 800 MHz and is met in all configurations. The results are listed in Table II. From the table, we can observe that the gate count and power consumption of the DMesh router with a FIFO depth of 4/8 is roughly equal to or less than those of the NePA router with a FIFO depth of 8/16. Performance comparisons of these four configurations are shown in Fig. 9. It is clear that DMesh has a shorter latency than NePA with similar hardware cost. For example, for a 8x8 mesh in random traffic, DMesh with a FIFO depth of 4 has a shorter latency than NePA with a FIFO depth of 8 and 16. All other configurations have similar results. Fig. 9 also shows that the improvements from diagonal links are more than those from larger buffers. Therefore, DMesh is a more area-efficient architecture.

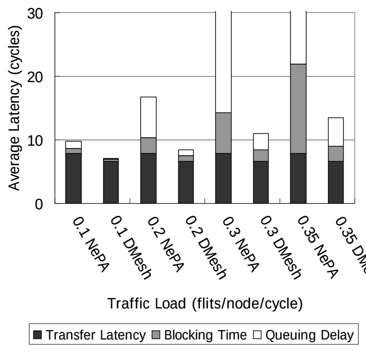
TABLE I. COMPARISON OF AVERAGE INTER-NODE DISTANCE

Network	**4x4 network**			
	Random	***Bit complement***	***Bit reverse***	***Matrix transpose***
NePA	2.38	4.33	3.32	3.38
DMesh	1.83	2.55	1.99	1.79
Reduction	23.1%	41.1%	40.0%	47.0%

Network	**8x8 network**			
	Random	***Bit complement***	***Bit reverse***	***Matrix transpose***
NePA	4.89	9.75	6.19	7.33
DMesh	3.66	5.80	4.04	3.86
Reduction	25.1%	40.5%	34.7	47.3



(a) 4x4 mesh network



(b) 8x8 mesh network

Figure 6. Comparison of various delays in random traffic

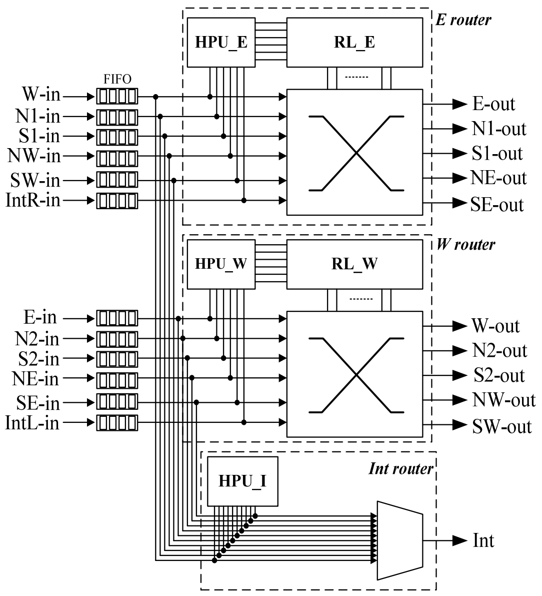


Figure 7. Block diagram of DMesh router.

TABLE II. GATE COUNT (EQUIVALENT 2-INPUT NAND GATE) OF NEPA AND DMESH ROUTER NODE (FIFOS INCLUDED)

FIFO depth (flits)	NePA		DMesh	
	Gate Count	Dynamic Power (mW)	Gate Count	Dynamic Power (mW)
2	18368	6.99	32750	11.53
4	28654	12.29	46598	20.40
8	47038	22.32	75382	36.88
16	85362	42.03	134479	69.45
32	163330	81.27	250559	134.36
64	316173	159.33	490820	261.95

V. CONCLUSION

We developed a novel DMesh NoC architecture and demonstrated its performance enhancement over the previous work. Hardware cost evaluation also shows that our approach is more area-efficient than previously reported results. With more links in the network, we anticipate that DMesh has the potential of supporting better QoS and fault tolerance capability.

REFERENCES

[1] S. Bell et al., "TILE64 Processor: A 64-Core SoC with Mesh Interconnect," Solid-State Circuits Conference, 2008. Digest of Technical Papers. IEEE International, pp. 88-598, 2008.

[2] S. Vangal et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," Solid-State Circuits Conference, 2007. Digest of Technical Papers. IEEE International, pp. 98-589, 2007.

[3] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," Design Automation Conference, 2001. Proceedings, pp. 684-689, 2001.

[4] L. Benini and G. De Micheli, "Networks on chip: a new paradigm for systems on chip design," Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings, pp. 418-419, 2002.

[5] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of Network-on-chip," ACM Computing Surveys, vol. 38, pp. 1, 2006.

[6] E. Salminen et al., "Survey of Network-on-Chip proposals," White Paper, OCP-IP, March 2008.

[7] J. H. Bahn, S. E. Lee, Y. S. Yang, J. Yang and N. Bagherzadeh, "On Design and Application Mapping of a Network-on-Chip(NoC) Architecture," Parallel Processing Letters, vol. 18, pp. 239-255, 2008.

[8] M. Igarashi, T. Mitsuhashi, A. Le, S. Kazi, Y. T. Lin, A. Fujimura and S. Teig, "A Diagonal-Interconnect Architecture and Its Application to RISC Core Design," IEIC Technical Report (Institute of Electronics, Information and Communication Engineers), vol. 102, pp. 19-23, 2002.

[9] S. L. Teig, "The X architecture: Not your father's diagonal wiring," Proceedings of the 2002 International Workshop on System-level Interconnect Prediction, pp. 33-37. 2002.

[10] G. Varatkar and R. Marculescu, "Traffic analysis for on-chip networks design of multimedia applications," in DAC '02: Proceedings of the 39th Conference on Design Automation, 2002, pp. 795-800.

[11] W. E. Leland, M. S. Taqqu, W. Willinger and D. V. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," Networking, IEEE/ACM Transactions on, vol. 2, pp. 1-15, 1994.

[12] M. S. Taqqu, W. Willinger and R. Sherman, "Proof of a fundamental result in self-similar traffic modeling," SIGCOMM Comput. Commun. Rev., vol. 27, pp. 5-23, 1997.

[13] D. R. Avresky, "Performance evaluation of the ServerNet(R) SAN under self-similar traffic," Parallel and Distributed Processing, 1999. 13th International and 10th Symposium on Parallel and Distributed Processing, 1999. 1999 IPPS/SPDP. Proceedings, pp. 143-147, 1999.

[14] W. J. Dally, Principles and Practices of Interconnection Networks. Morgan Kaufmann, 2004.

[15] K. Chang, J. Shen and T. Chen, "Evaluation and design trade-offs between circuit-switched and packet-switched NOCs for application-specific SOCs," in DAC '06: Proceedings of the 43rd Annual Conference on Design Automation, 2006, pp. 143-148.

[16] P. Marchal, D. Verkest, A. Shickova, F. Catthoor, F. Robert and A. Leroy, "Spatial division multiplexing: a novel approach for guaranteed

throughput on NoCs," Hardware/Software Codesign and System Synthesis, 2005. CODES+ISSS '05. Third IEEE/ACM/IFIP International Conference on, pp. 81-86, 2005.

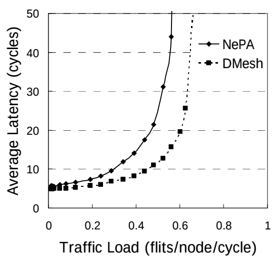
[17] J. Hu, Y. Deng and R. Marculescu, "System-level point-to-point communication synthesis using floorplanning information," in ASP-DAC '02: Proceedings of the 2002 Conference on Asia South Pacific Design automation/VLSI Design, 2002, pp. 573.

[18] D. Bertozzi, A. Jalabert, S. Murali, R. Tamhankar, S. Stergiou, L. Benini and G. D. Micheli, "NoC Synthesis Flow for Customized Domain Specific Multiprocessor Systems-on-Chip," vol. 16, pp. 113-129, 2005.

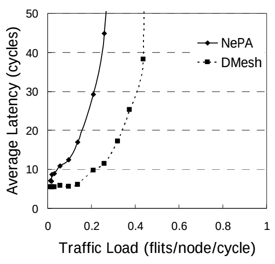
[19] J. H. Bahn, S. E. Lee and N. Bagherzadeh, "On Design and Analysis of a Feasible Network-on-Chip (NoC) Architecture," Information Technology, 2007.ITNG'07.Fourth International Conference on, pp. 1033-1038, 2007.

TABLE III. COMPARISON OF SATURATION LOAD OF NEPA AND DMESH

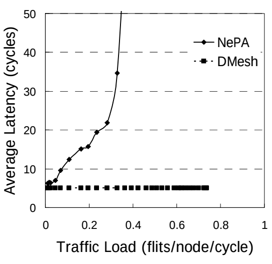
Network		**FIFO depth = 2**	**FIFO depth =4**	**FIFO depth =8**	**FIFO depth =16**	**FIFO depth =32**	**FIFO depth =64**
4x4 Random	NePA	0.519	0.595	0.626	0.659	0.686	0.695
	DMesh	0.597	0.688	0.752	0.803	0.828	0.855
	Improvement	15.0%	15.6%	20.1%	21.8%	20.6%	23.0%
4x4 Bit-complement	NePA	0.360	0.361	0.359	0.362	0.365	0.371
	DMesh	0.611	0.504	0.545	0.530	0.556	0.537
	Improvement	69.7%	39.6%	51.8%	46.4%	52.3%	45.1%
4x4 Bit-reverse	NePA	0.388	0.384	0.385	0.386	0.387	0.387
	DMesh	1.000	1.000	1.000	1.000	1.000	1.000
	Improvement	157.7%	160.4%	159.7%	159.0%	158.3%	158.3%
4x4 Matrix transpose	NePA	0.422	0.395	0.392	0.392	0.390	0.387
	DMesh	0.709	1.000	1.000	1.000	1.000	1.000
	Improvement	68.0%	153.1%	155.1%	155.1%	156.4%	158.3%
8x8 Random	NePA	0.298	0.353	0.394	0.423	0.436	0.441
	DMesh	0.459	0.509	0.550	0.596	0.627	0.650
	Improvement	54.0%	44.1%	39.5%	40.8%	43.8%	47.3%
8x8 Bit-complement	NePA	0.090	0.090	0.090	0.090	0.090	0.144
	DMesh	0.232	0.221	0.230	0.239	0.244	0.246
	Improvement	157.7%	145.5%	155.5%	165.5%	171.1%	70.8%
8x8 Bit-reverse	NePA	0.178	0.174	0.175	0.187	0.199	0.204
	DMesh	0.312	0.309	0.304	0.313	0.307	0.311
	Improvement	75.2%	77.5%	73.7%	67.3%	54.2%	52.4%
8x8 Matrix transpose	NePA	0.167	0.174	0.174	0.181	0.184	0.184
	DMesh	0.317	0.322	0.327	0.373	0.379	0.445
	Improvement	89.8%	85.0%	87.9%	106.0%	105.9%	141.8%



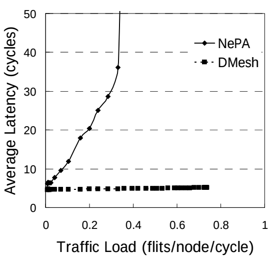
(a) 4x4 random traffic



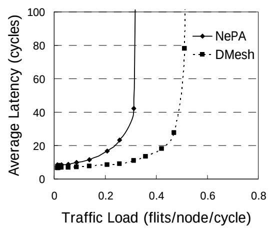
(b) 4x4 bit complement traffic



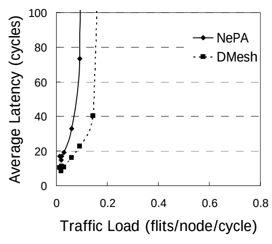
(c) 4x4 bit reverse traffic



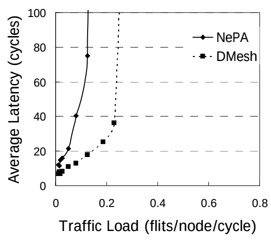
(d) 4x4 matrix transpose traffic



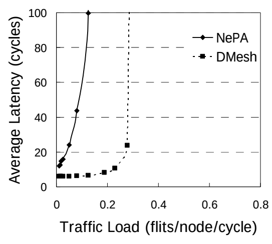
(a) 8x8 random traffic



(b) 8x8 bit complement traffic

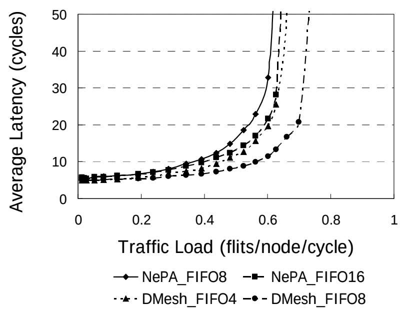


(c) 8x8 bit reverse traffic

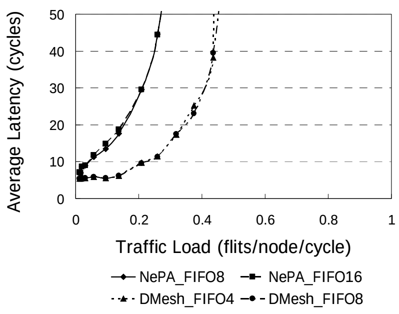


(d) 8x8 matrix transpose traffic

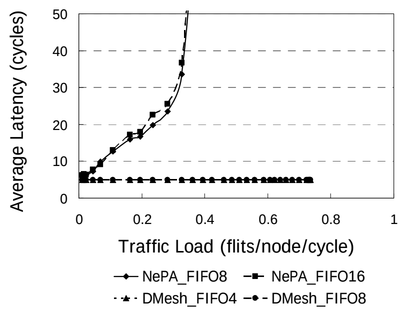
Figure 8. Comparisons of average latency in 4x4 and 8x8 mesh networks (FIFO depth = 4).



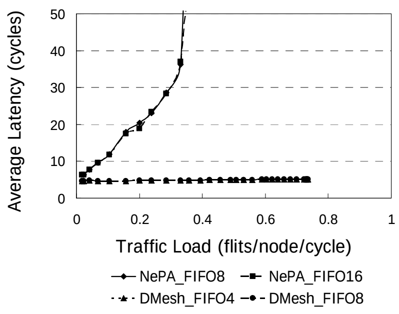
(a) 4x4 random traffic



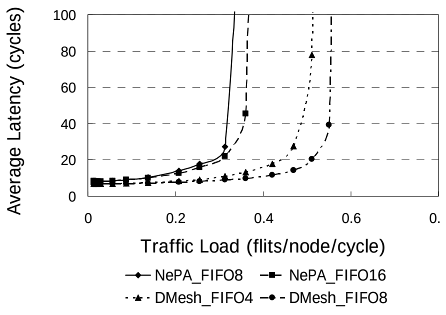
(b) 4x4 bit complement traffic



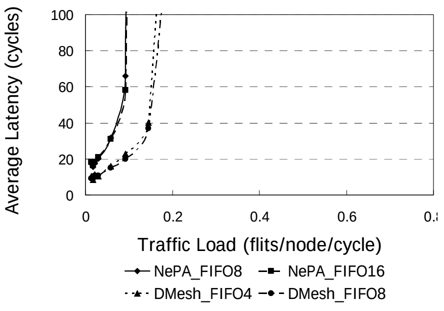
(c) 4x4 bit reverse traffic



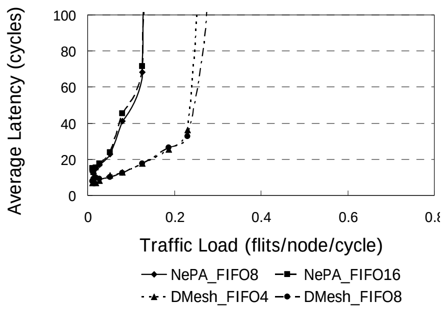
(d) 4x4 matrix transpose traffic



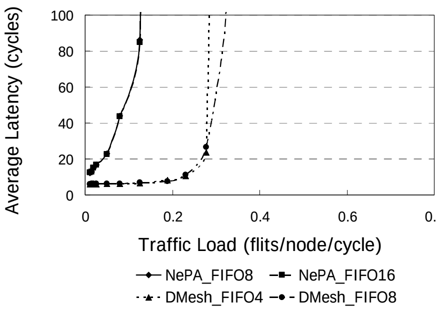
(a) 8x8 random traffic



(b) 8x8 bit complement traffic



(c) 8x8 bit reverse traffic



(d) 8x8 matrix transpose traffic

Figure 9. Comparisons of average latency in 4x4 and 8x8 mesh networks with similar router cost.

Session II

Performance Evaluation

Session Chair: Ahmed Hemani, *Royal Institute of Technology, Sweden*

A Generic Traffic Model for On-Chip Interconnection Networks

Jun Ho Bahn
Qualcomm CDMA Technologies
Qualcomm Inc., 5775 Morehouse Drive
San Diego, CA 92121-1714
Email : jbahn@qualcomm.com

Nader Bagherzadeh
EECS, University of California, Irvine
325 Engineering Tower
Irvine, CA 92697-2625
Email : nader@uci.edu

***Abstract*— On-chip interconnection networks or Network-on-Chips (NoCs) are becoming the *de-facto* scaling communication techniques in Multi-Processor System-on-Chip (MPSoC) or Chip Multiprocessor (CMP) environment. However, the current traffic models for on-chip interconnection networks are insufficient to capture the traffic characteristics as well as evaluate the network performance. As the technology scaling enables the increase of available on-chip resources and innumerable new network architectures are proposed, there is a need to make NoCs more application-specific. Therefore, a traffic model to characterize such an application-specific network is necessary. In this paper, we propose a generic traffic model for on-chip interconnection networks. Our traffic model is based on three empirically-derived statistical characteristics using temporal and spatial distributions. With captured parameters, our model can generate accurate traffic patterns recursively to show similar statistical characteristics of the observed on-chip networks. Therefore, using the proposed traffic model defined by captured statistics, any kind of on-chip interconnection traffic patterns can be reproduced.**

I. Introduction

As the number of integrated IP cores in the current System-on-Chips (SoCs) keeps increasing to meet the design requirements for computation-intensive applications and highly integrated low power solutions, communication requirements among cores can not be sufficiently satisfied using either traditional or multi-layer bus architectures because of their poor scalability and bandwidth limitation on a single bus. While new interconnection techniques have been explored to overcome such a limitation, the notion of utilizing Network-on-Chip (NoC) technologies for the future generation of high performance and low power chips for myriad of applications, in particular for wireless communication and multimedia processing, has been of great importance [1]. By applying network-like communication which inserts routers in-between each communication object, the interconnection network improves scalability and freedom from the limitation of complex wiring. Replacement of SoC busses by NoCs will follow the same path as data communication systems where from the economics point of view NoC can potentially reduce SoC manufacturing cost, time to market, time to volume, and design risk and at the same time improve performance. According to [2], the NoC approach has a clear advantage over traditional busses and most notably as far as the system throughput is concerned. Though hierarchies of crossbar or multi-layer busses have characteristics somewhere in between traditional busses and NoC, they still fall far short of the NoC with respect to performance and complexity. Recently many researchers have proposed various routing algorithms as well as different router architectures appropriate for on-chip interconnection network environments.

In order to evaluate the performance of either these routing algorithms or their routers, including implementations, many researchers have used conventional traffic patterns [3], [4] or some limited number of real traffic traces. Even though these static traffic patterns exhibit similar patterns of some particular applications, there is a fundamental limit in covering complete traffic patterns of real applications. For this reason, some researchers have used real traffic patterns extracted from real applications to evaluate the performance of their proposed routing algorithm or router based on more practical benchmarks [5], [6].

In traditional networks such as Internet, Ethernet, and wireless LANs transporting TCP/IP, HTTP, and FTP traffic among others, network traffics have been traced and analyzed to understand the traffic behavior of these networks and characterize them. Therefore, various extensive traffic models for diverse networks have been developed [7], [8], [9], [10], [11], [12], [13], [14]. These models provide not only meaningful insight into understanding the traffic behavior of these networks, but have also been effectively used in evaluating current and newly designed networks. Because on-chip interconnection network is a new class of networks where the overall communications occur in a single chip, a similar approach to understanding the behavior of NoC traffic and evaluating networks in NoC environment is needed.

In this paper, we propose a generic traffic model for NoC environments. The proposed model is based on the spatial/temporal profile of traffic using three statistical parameters. These three statistical parameters construct node burstiness, node injection rate, and the distribution of source-to-destination pairs. Different from the other previous proposals where statistical parameters were extracted from overall nodes and formulated in a single statistic model of each component, each statistical parameter is extracted from each node and the associated statistic model is constructed per node. Therefore, the degree of accuracy of the proposed traffic model

TABLE I
CONVENTIONAL STATIC TRAFFIC PATTERNS [3]

Name	*Pattern*
Random	$\lambda_{sd} = 1/N$
Permutation	
Bit permutations	
Bit complement	$d_i = \sim s_i$
Bit reverse	$d_i = s_{b-i-1}$
Bit rotation	$d_i = s_{i+1 \bmod b}$
Shuffle	$d_i = s_{i-1 \bmod b}$
Transpose	$d_i = s_{i+b/2 \bmod b}$
Digit permutations	
Tornado	$d_x = s_x + (\lceil k/2 \rceil) \bmod k$
Neighbor	$d_x = s_x + 1 \bmod k$

in emulating real traffic situation is much higher than the previous work.

The organization of this paper is as follows. Section 2 provides previous related works in the field of traffic modeling and motivation of this paper. Next, Section 3 explains an overview of our traffic model with three different statistical components for NoC. In Section 4, the details of each component are presented and the overall procedure of generating traffic pattern with the given parameters is described. Section 5 validates the accuracy of the proposed traffic model by comparing it with real traffic traces. Finally, Section 6 concludes this paper.

II. Related Work

Conventional traffic patterns consider the spatial distribution of messages in interconnection networks. Therefore, the distribution between source nodes and destination nodes is defined depending on the type of conventional traffic patterns. Table I lists some common static traffic patterns used to evaluate interconnection networks. In Table I, In conventional traffic patterns, random traffic is described by a traffic matrix with all fraction of traffic sent from node λ_{sd} as $1/N$. Permutation traffic, in which all traffic from each source is directed to one destination, can be more compactly represented by a permutation function. Bit permutations are those in which each bit d_i of the b-bit destination address is a function of one bit of the source address, s_i. In digit permutations, each (radix-k) digit of the destination address d_x is a function of a digit s_y of the source address. Historically, several of these patterns are based on communication patterns that arise in particular applications. For instance, matrix transpose or corner-turn operations induce the transpose pattern, whereas fast Fourier transform (FFT) or sorting applications might cause the shuffle permutation, and fluid dynamics simulations often exhibit neighbor patterns [3].

While these models enable a network to be stressed with a regular, predictable pattern and provide NoC researchers with helpful insights, they do not cover real application traffics to explore a realistic NoC design-space. Until now, few researches have been able to present results in the field of realistic NoC traffic models. Varatkar and Marculescu [15] have reported the evidence of self-similarity in NoC burst traffic between on-chip modules in typical MPEG-2 video applications and captured traffic characteristics between pairwise nodes. Also using a generic tile-based communication architecture, they proposed a technique for synthetically generating traces having statistical properties similar to those obtained from real video clips. Soteriou et al. [16] proposed an empirically-derived model of NoC traffic based on traffic traces obtained from full system simulations. Their model comprehensively espouses the spatio-temporal characteristics of traffic with three dimensionless statistical components in a three-tuple model. Also they illustrate two potential uses of their traffic model: how it allows us to characterize and gain insights on NoC traffic patterns, and how it can be used to generate synthetic traffic traces that can drive NoC design-space exploration. Tedesco et al. [17] presented application driven traffic modeling for NoCs. In their work, applications are characterized according to their delivery requirements (e.g. real-time streaming and block transfer) and QoS service levels (e.g. CBR and VBR). Also they identify three methods to model traffic: constant injection rate is the most commonly used, but least accurate. Probabilistic methods are normally used in simulation for applications with variable rates. Finally trace-based traffic models are more suitable for emulation.

III. Overview

We propose a generic traffic model for NoC based on traffic traces obtained from full system simulation or real system devices. This model combines the spatio-temporal characteristics of traffic with three independent components, (H_s, λ_s, $\delta_{(s,d)}$) where s and d represent the indices of source node and destination node, respectively. With three independent components, the given traffic can be analyzed and characterized in a statistical manner. Different from the approach used in [16], each statistical component is derived per node. To define the burstiness of each node, the Hurst exponent H_s for source node s, is adopted. As a component of the characteristics of self-similarity, H_s determines the temporal burstiness of traffic at each node, that is, the peak size of injection packets and their injection patterns of arrival time. To define one of spatial properties in NoC traffic traces, the distribution of average injection rate at every node, denoted by λ_s is captured. Finally $\delta_{(s,d)}$ representing the distribution of traffic ratio from s node to d node in the given injection rate λ_s is extracted.

For each component of our (H_s, λ_s, $\delta_{(s,d)}$) traffic model, we analyze and extract the proposed statistical distribution against 8 traffic traces used in [18]. Those are SPLASH-2 [19] traces gathered by running the benchmarks on Bochs [20], a multiprocessor simulator with an embedded Linux 2.4 kernel. Each benchmark was run in Bochs with 49 (= 7×7) concurrent threads, and the memory trace is captured. This memory trace is then applied to a memory system simulator that models the classic MSI (Modified, Shared, Invalid) directory-based cache coherence protocol, with the home directory nodes statically assigned based on the least significant bits of the tag, distributed across all processors in the entire chip.

IV. Traffic Modeling

In this Section, we explain the details of our (H_s, λ_s, $\delta_{(s,d)}$) traffic model. Based on the extracted parameters, the procedure for generating a synthetic traffic trace will be provided as well.

A. Temporal Burstiness: H_s

In classic networks, self-similarity is one of the key features to characterize burstiness as well as long-range dependence (LRD) of traffic in the temporal sense. To measure such a burstiness and LRD, the Hurst parameter H is used where $H \in (1/2, 1)$ indicates the presence of LRD. As many communication traffics are proven to be statistically self-similar, some researchers already showed that the traffic in NoC also has a self-similar characteristic [15], [16]. Thus, we parameterize such a degree of burstiness or LRD using H. Furthermore, in order to be accurate, this parameter indicating the burstiness is analyzed on every injection node.

Because the definitions of self-similarity are well described in the literature, in this Section, a brief description of self-similarity will be introduced. For more details, the reader is recommended to read several references [7], [8], [9], [21].

Considering a cumulative process $Y(t)$ with stationary increments, let X_t be its corresponding incremental process:

$$X_t = Y(t) - Y(t-1) \tag{1}$$

The process $X_s^{(m)}$ is defined as an aggregated process of X_t if

$$X_s^{(m)} = [X_{sm-m+1} + X_{sm-m+2} + \ldots + X_{sm}]/m \tag{2}$$

Process X_t is *self-similar* if X_t is indistinguishable from $X_s^{(m)}$. Because this is a very restrictive definition, usually *second-order self-similarity* is considered for traffic analysis, i.e. *auto-covariance* of the original and aggregated processes should be same:

$$\gamma^{(m)}(k) = \gamma(k) \tag{3}$$

$$\lim_{m\to\infty} \gamma^{(m)}(k) = \gamma(k) \tag{4}$$

where $\gamma(k) = E[(X_t - \mu)(X_{t+k} - \mu)]$ and $\gamma^{(m)}(k) = E[(X_s^{(m)} - \mu)(X_{s+k}^{(m)} - \mu)]$. The process X_t is *exactly* second-order self-similar or *asymptotically* second-order self-similar if Eq. (3) or Eq. (4) is satisfied, respectively.

In order to measure the degree of self-similarity, the Hurst parameter H is used where a process is self-similar with parameter $H (0 < H < 1)$ if:

$$Y(t) = k^H Y(kt), \forall k > 0, t \geq 0 \tag{5}$$

which means that the original and normalized aggregated processes should have the same distribution. In other words, the self-similarity can be understood as the ability of an aggregated process to preserve the burstiness of the original process, i.e. the property of *slowly decaying variance*:

$$var(X^{(m)}) \sim m^{2H-2} \tag{6}$$

In this paper, Eq. (6) is computed to measure the Hurst parameter H. Table II shows the measured H value per node for eight different traces.

B. Injection Rate: λ_s

As one of the spatial parameters in our traffic model, traffic injection rate determines the distribution of injection load per node. In [16], this spatial injection distribution is parameterized by the standard deviation σ of the injection distribution with an actual coordinate assignment. In that approach, it assumes that the actual results possess Gaussian-type distributions. Even though that approach can help the injection distribution be quantified using single σ value, the mapping to Gaussian-like distribution is not alway accurate in real NoC traffic situation. Also it requires large amount of computation to find out the exact coordinate assignment. Hence, in this paper, the original distribution of injection rate on every node is kept as it is. This enables more accurate synthetic traffic generation than σ-based Gaussian-like distribution. Figure 1 shows injection rate distributions for traffic traces in a 7×7 mesh.

C. Spatial Distribution: $\delta_{(s,d)}$

Another spatial distribution $\delta_{(s,d)}$ represents the traffic ratio from source node s to destination node d based on the injection rate λ_s. In [16], spatial hop distribution p is adopted. In order to formulate the hop count distribution model, they applied the mechanism so that the mapping should not choose a receiver whose distance is d hops from the sender unless it cannot choose any other node whose distance to the sender is less than d. Also, in that model, there is no concern about the geometry of destination nodes. In other words, all nodes with same d-hop distance from the source node are considered to have the same statistical characteristics. Thus, this assumption is somehow far from the actual NoC traffic regardless of the optimal communication mapping. However, our model considers the difference of location of destination nodes within same distance of hops when the traffic ratio between source and destination node is analyzed. Moreover, the matrix of traffic ratio from each source node is constructed in order to characterize the spatial distribution of source/destination pairs. Figure 2 illustrates the distribution of traffic ratio for each node in the `barnes` application.

D. Synthetic Traffic Generation

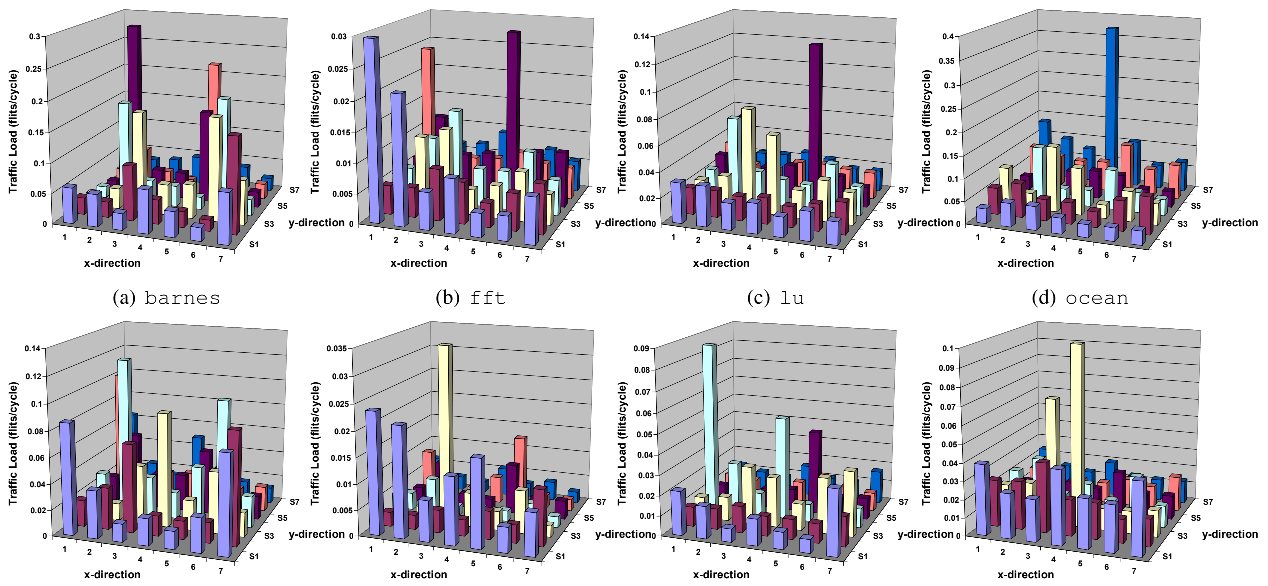
To describe how our (H_s, λ_s, $\delta_{(s,d)}$) traffic model can generate synthetic network traffic, we implemented *tgNePA*, a tool that automatically generates NoC traffic of the given network topology from the configured (H_s, λ_s, $\delta_{(s,d)}$) traffic model. Figure 3 shows the traffic generation flow in *tgNePA*.

***tgSelfSimilar*: Traffic generation based on (H_s, λ_s).** To generate self-similar NoC traces, *tgNePA* uses the method described in [22]. In this method, the synthetic self-similar traffic is obtained by aggregating multiple sub-streams, each consisting of alternating Pareto-distributed on/off periods. Pareto distribution is defined by a heavy-tailed distribution with the probability-density function $f(x) = ab^\alpha / x^{\alpha+1}, x \geq b$ where α is a shape parameter, and b is a location parameter. Pareto distribution with $1 < \alpha < 2$ has a finite mean and

TABLE II

MEASURED HURST PARAMETER FOR TRAFFIC TRACES IN 7×7 MESH

barnes							fft						
0.95322	0.930023	0.976069	0.976115	0.931223	0.958229	0.967662	0.975095	0.941273	0.948791	0.964113	0.951594	0.956773	0.961565
0.953165	0.958007	0.906468	0.927599	0.932508	0.952065	0.938917	0.95084	0.951059	0.966805	0.966816	0.960732	0.952544	0.963932
0.961447	0.950251	0.960211	0.976686	0.923968	0.9375	0.864216	0.95335	0.978645	0.977989	0.962314	0.958788	0.968619	0.963367
0.958195	0.890648	0.918889	0.93756	0.958898	0.927886	0.904769	0.961431	0.973687	0.992332	0.959887	0.96177	0.968292	0.964245
0.956649	0.985442	0.939089	0.920304	0.904582	0.938582	0.885551	0.964755	0.972209	0.972016	0.97097	0.990788	0.968756	0.974325
0.961257	0.88265	0.938839	0.95921	0.96127	0.917485	0.928582	0.993044	0.972676	0.971063	0.975506	0.976879	0.973728	0.970238
0.869166	0.965638	0.895383	0.906786	0.909061	0.883905	0.88992	0.975961	0.980018	0.977654	0.973719	0.968751	0.964043	0.960366
lu							ocean						
0.924199	0.953505	0.956322	0.953416	0.955267	0.961355	0.949458	0.880499	0.808062	0.794881	0.845817	0.799616	0.816938	0.832904
0.958744	0.95534	0.953154	0.954404	0.954908	0.952774	0.956247	0.7324	0.784654	0.830542	0.826276	0.77571	0.753008	0.795864
0.954391	0.967936	0.985967	0.980213	0.953937	0.95738	0.956906	0.901085	0.865756	0.723628	0.749781	0.781642	0.752834	0.742008
0.959172	0.982929	0.961082	0.954406	0.95408	0.967851	0.956651	0.802515	0.757628	0.798828	0.75681	0.760965	0.790886	0.750362
0.963854	0.9776	0.95748	0.965053	0.992639	0.962783	0.954991	0.82159	0.810766	0.726558	0.748242	0.777665	0.78399	0.78296
0.967356	0.950398	0.952623	0.964804	0.954611	0.961498	0.962472	0.818765	0.770188	0.809174	0.785652	0.803659	0.762584	0.782661
0.956253	0.960129	0.957397	0.95746	0.957518	0.95611	0.954621	0.837008	0.761783	0.87089	0.763205	0.77494	0.78582	0.778135
radix							raytrace						
0.902839	0.944403	0.96101	0.972136	0.964553	0.947608	0.985433	0.942255	0.961226	0.940355	0.962855	0.95834	0.932357	0.963871
0.957888	0.942072	0.971374	0.973585	0.966459	0.964246	0.977311	0.945377	0.948391	0.943875	0.936158	0.964887	0.943996	0.961554
0.964332	0.957286	0.980439	0.98841	0.983089	0.985034	0.974669	0.945114	0.957771	0.974867	0.950478	0.960309	0.968862	0.945402
0.969146	0.983704	0.971503	0.977166	0.96415	0.981327	0.957322	0.92635	0.967857	0.956754	0.949919	0.951917	0.974327	0.968231
0.973206	0.982417	0.97069	0.960063	0.975664	0.963702	0.969817	0.929483	0.961821	0.964149	0.936166	0.974707	0.941642	0.946531
0.93137	0.966252	0.969756	0.975648	0.958885	0.963826	0.96038	0.949944	0.93797	0.93593	0.943694	0.958023	0.934166	0.947076
0.946714	0.966286	0.961873	0.944223	0.947943	0.965468	0.962096	0.963812	0.963292	0.934751	0.959196	0.958551	0.960313	0.949267
water-nsquared							water-spatial						
0.938858	0.954619	0.961653	0.961435	0.964605	0.959879	0.983422	0.954124	0.941421	0.96231	0.980993	0.970395	0.968765	0.979926
0.958101	0.944468	0.963525	0.957574	0.95899	0.957685	0.966726	0.967107	0.964892	0.976365	0.954031	0.954496	0.95555	0.951143
0.954599	0.952399	0.978571	0.973547	0.959643	0.974414	0.978607	0.956379	0.958845	0.985977	0.992441	0.946157	0.954781	0.949645
0.995104	0.974171	0.963846	0.987655	0.952222	0.966112	0.952725	0.950117	0.957826	0.949018	0.943914	0.948824	0.960112	0.94836
0.951494	0.971262	0.954555	0.958582	0.982684	0.958186	0.965041	0.94065	0.939393	0.957809	0.954648	0.962962	0.949257	0.950307
0.958905	0.95965	0.959323	0.961995	0.965036	0.963629	0.962181	0.95947	0.95342	0.956035	0.95468	0.955787	0.955492	0.955214
0.961457	0.955258	0.957095	0.965474	0.952312	0.959847	0.965554	0.969339	0.965527	0.963964	0.966033	0.961421	0.960948	0.962182



(a) barnes (b) fft (c) lu (d) ocean

(e) radix (f) raytrace (g) water-nsquared (h) water-spatial

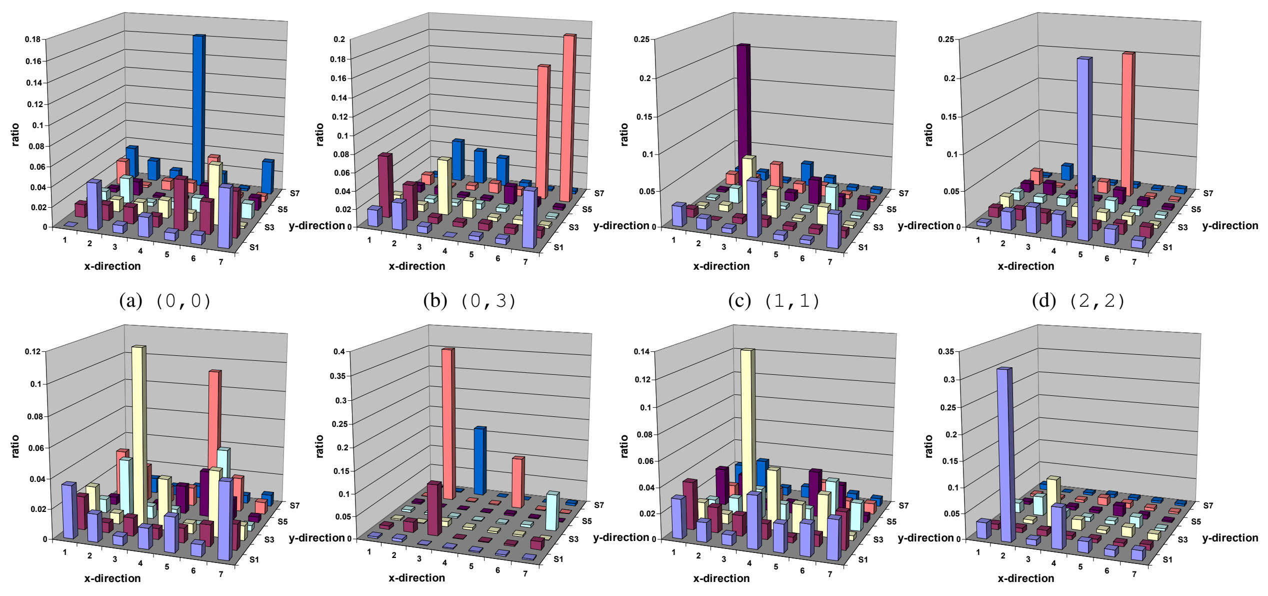
Fig. 1. Injection rate distributions for traffic traces in 7×7

an infinite variance. To generate Pareto-distributed values, the following formula is used: $X_{Pareto} = b/[U^{1/\alpha}]$ where U is a uniform random variable ($0 \leq U \leq 1$). The Hurst parameter H of self-similar trace generated by this method can be derived by $H = (3 - \alpha)/2$ [22], [23].

Additionally, while generating Pareto-distributed values, the injection rate for each sub-stream can be controlled. Therefore, by applying λ_s to each generation of self-similar stream for the corresponding node s, the (H_s, λ_s) configured self-similar traffic can be obtained.

Depending on the method of self-similar traffic generation, its accuracy may be varied. To minimize the error between the expected (H_s, λ_s) and the measured value from the generated traffic, a recursion is applied as shown in the first phase *tgSelfSimilar* of Figure 3. Along with generating self-similar traffic with the expected (H_s, λ_s) configuration, (H'_s, λ'_s)-tuple components of the generated traffic are measured. If the error of the expected (H_s, λ_s) and the measured (H'_s, λ'_s) is acceptable, then the generated self-similar traffic is delivered to the next step *splitPE*. Otherwise, the generation of self-similar traffic with the similar configuration is repeated.

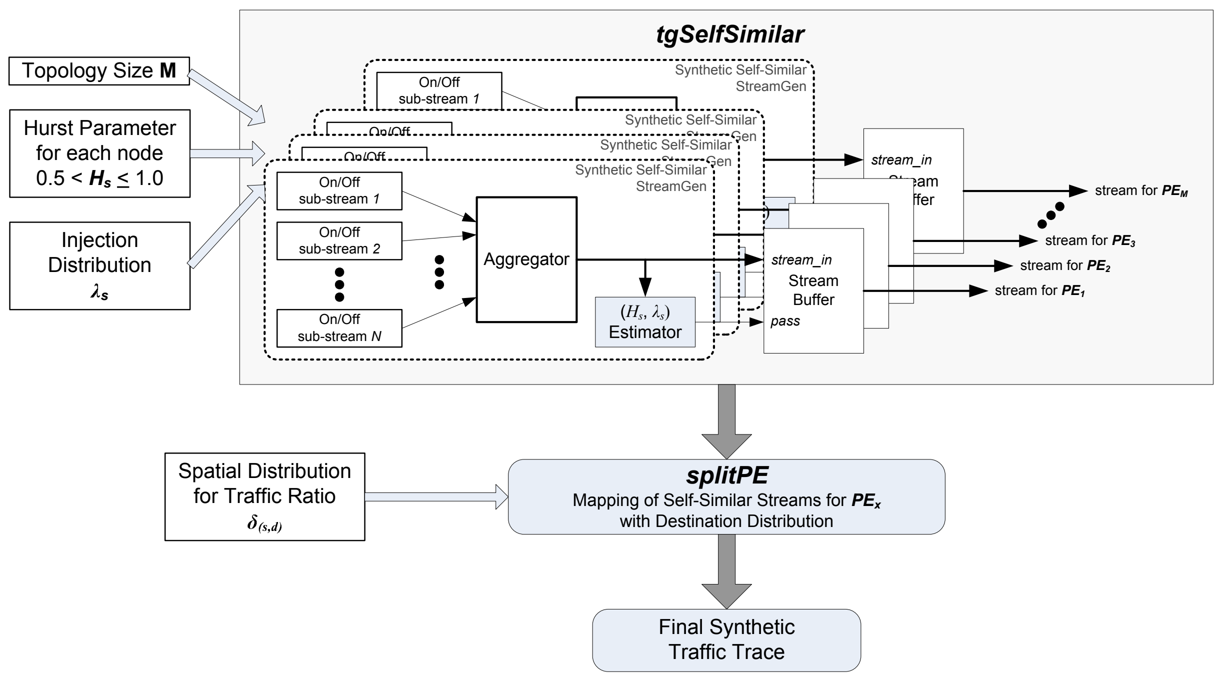
***splitPE*: Traffic generation based on $\delta_{(s,d)}$.** The second phase generates the destination node upon the generated self-similar traffic of each node. Because the ratio of traffic from each source node s to each destination node d is already provided by the distribution of $\delta_{(s,d)}$, the generation of destination



(a) (0,0) (b) (0,3) (c) (1,1) (d) (2,2)

(e) (3,4) (f) (4,1) (g) (5,3) (h) (6,1)

Fig. 2. Distributions of traffic ratio on selected nodes for barnes traffic trace in 7×7



tgSelfSimilar
Topology Size M
Hurst Parameter for each node 0.5 < Hs ≤ 1.0
Injection Distribution λs
Synthetic Self-Similar StreamGen
On/Off sub-stream 1
On/Off sub-stream 2
On/Off sub-stream N
Aggregator
(Hs, λs) Estimator
stream_in
Stream Buffer
pass
stream for PEM
stream for PE3
stream for PE2
stream for PE1
Spatial Distribution for Traffic Ratio δ(s,d)
splitPE
Mapping of Self-Similar Streams for PEx with Destination Distribution
Final Synthetic Traffic Trace

Fig. 3. *tgNePA* traffic generation flow diagram

node for each instance of traffic from the corresponding source node s can be accurately controlled randomly. Different from the Trident's approach [16], the ratio of traffic for each pair of source and destination is separately assigned. Therefore, the distribution of source/destination pairs can be more accurately emulated.

V. Validation

Each synthetic traffic is generated using the analyzed (H_s, λ_s, $\delta_{(s,d)}$) for each application mentioned in the previous Section. In order to control the recursion of *tgSelfSimilar*, we set the marginal error bound of H_s and λ_s to 5%. In recursive generation of self-similar traffic for each node, the Hurst parameter H_s can be easily matched with the given marginal percentage. However, in matching the lower injection rate λ_s, it requires excessive computation time. For that reason, to reduce such a large computation time in matching the injection rate, a proportional margin value is applied as an alternative approach. That is, in relatively higher injection rate, the tighter margin value is applied. Reversely, in relatively lower injection rate, the lighter margin value is applied. For instance, by using logarithmic scale of injection rate, the marginal value can be scaled by multiplying $|\log_{10}(\lambda_s)|^{|\log_{10}(\lambda_s)|}$. Because the higher injection rate is dominant, the effect of larger error at source nodes with lower injection rates can be minimized. Table III and Table IV show the measured Hurst parameter and injection rates of synthetic traffic according to the analyzed traffic model of each application. For H_s parameter in synthetic traffic, the accuracy is in the range of 2.7% to 4.3% average error bound. However, the accuracy of λ_s is varied depending on the level of injection rates of applications because the propotional margin value to the level of injection rates is applied in matching the injection rate during the first phase of traffic generation. For instance, in `barnes` application, the average of injection rates of original traffic is 0.065 and the ratio of average error in injection rates is 6.8%. On the other hand, in `fft` application, the average of injection rates of original traffic is 0.0089 and the ratio of average error is 26%. In this case, the level of injection rates is relatively low, i.e. the scale factor to apply a propotional margin is 27 (=3^3) during the recursion. Therefore, the resultant synthetic traffic has relatively large error from the original injection rates.

For source/destination distribution $\delta_{(s,d)}$ of synthetic traffic, its accuracy is almost 100% as shown in Figure 4.

Finally, throughout the cycle accurate NoC simulation [24], [25] using original traffic traces as well as synthetic traffic traces, the accuracy of overall network performance is observed. As shown in Table V, the synthetic trafic patterns for applications except for `fft` and `raytrace` have maximum 17% error from the perspective of the offered load. For two exceptional applications with high error ratio in the offered load, their offered load is relatively low. Therefore, even a small difference results in a large percentage of error ratio.

VI. Conclusion and Future Works

We proposed a generic traffic model for on-chip interconnection networks. To keep the temporal and spatial distribution of traffic traces, every statistical information is measured per node. In order to characterize the burstiness of injection nodes, the Hurst parameter H_s is selected. For specifying the temporal statistics, the distribution of injection rates λ_s and ratio of source/destination pairs $\delta_{(s,d)}$ on the given source node are used. With the proposed traffic model, we also introduced a recursive traffic generation method to minimize the error of statistical components, and allow synthetic traffic traces with similar temporal and spatial statistics to be generated. Throughout detailed comparison of each component and performance simulation, our proposed traffic model can reconstruct traffic patterns with a similar tendancy of real NoC traffic and provide insights into NoC traffic.

As the future works, an advanced methodology needs to be developed to validate the accuracy of synthetic traffic patterns. In this paper, only the statistical measurement such as comparing average parameters, which does not evaluate the accuracy in time, is used. To be scalable, the proposed traffic model should be tested in different size or type of networks.

References

[1] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *DAC '01: Proceedings of the 38th Conference on Design Automation.* New York, NY, USA: ACM, 2001, pp. 684–689.

[2] ARTERIS, "A comparison of network-on-chip and busses," http://www.arteris.com/noc_whilepaper.pdf.

[3] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* San Francisco, CA, USA: Morgan Kaufmann Publishers, 2004.

[4] K. Lahiri, S. Dey, and A. Raghunathan, "Evaluation of the traffic-performance characteristics of system-on-chip communication architectures," in *VLSID '01: Proceedings of the 14th International Conference on VLSI Design.* Washington, DC, USA: IEEE Computer Society, 2001, pp. 29–35.

[5] J. Hu and R. Marculescu, "Dyad: smart routing for networks-on-chip," in *DAC '04: Proceedings of the 41st annual conference on Design automation.* New York, NY, USA: ACM, 2004, pp. 260–263.

[6] M. Palesi, R. Holsmark, S. Kumar, and V. Catania, "A methodology for design of application specific deadlock-free routing algorithms for noc systems," in *CODES+ISSS '06: Proceedings of the 4th International Conference on Hardware/Software Codesign and System Synthesis.* New York, NY, USA: ACM, 2006, pp. 142–147.

[7] P. Doukhan, G. Oppenheim, and M. S. Taqqu, *Theory and Applications of Long-Range Dependence.* Birkhäuser Boston, December 2002.

[8] K. Park and W. Willinger, *Self-Similar Network Traffic and Performance Evaluation.* New York, NY, USA: John Wiley & Sons, Inc., September 2000.

[9] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson, "On the self-similar nature of Ethernet traffic (extended version)," *IEEE/ACM Trans. Network*, vol. 2, no. 1, pp. 1–15, February 1994.

[10] V. Paxson and S. Floyd, "Wide-area traffic: The failure of poisson modeling," in *SIGCOMM '94: Proceedings of the Conference on Communications Architectures, Protocols and Applications.* New York, NY, USA: ACM, 1994, pp. 257–268.

[11] A. Balachandran, G. M. Voelker, P. Bahl, and P. V. Rangan, "Characterizing user behavior and network performance in a public wireless lan," *SIGMETRICS Performance Evaluation Review*, vol. 30, no. 1, pp. 195–205, June 2002.

[12] I. Y. Bucher and D. A. Calahan, "Models of access delays in multiprocessor memories," *IEEE Trans. Parallel Distributed Systems*, vol. 3, no. 3, pp. 270–280, May 1992.

[13] F. Darema-Rogers, G. F. Pfister, and K. So, "Memory access patterns of parallel scientific programs," in *SIGMETRICS '87: Proceedings of the 1987 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems.* New York, NY, USA: ACM, 1987, pp. 46–58.

[14] S. W. Turner, "Performance analysis of multiprocessor interconnection networks using a burst-traffic model," Ph.D. dissertation, Champaign, IL, USA, 1995.

[15] G. V. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for mpeg-2 video applications," *IEEE Trans. Very Large Scale Integration (VLSI) Systems*, vol. 12, no. 1, pp. 108–119, January 2004.

[16] V. Soteriou, H. Wang, and L.-S. Peh, "A statistical traffic model for on-chip interconnection networks," in *MASCOTS '06 : Proceeding of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation.* Washington, DC, USA: IEEE Computer Society, 2006, pp. 104–116.

[17] L. Tedesco, A. Mello, L. Giacomet, N. Calazans, and F. Moraes, "Application driven traffic modeling for nocs," in *SBCCI '06: Proceedings of the 19th Annual Symposium on Integrated Circuits and Systems Design.* New York, NY, USA: ACM, 2006, pp. 62–67.

[18] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Towards the ideal interconnection fabric," in *ISCA '07: Proceedings of the 34th Annual International Symposium on Computer Architecture.* New York, NY, USA: ACM, 2007, pp. 150–161.

[19] "Splash-2," http://www-flash.stanford.edu/apps/SPLASH/.

[20] K. P. Lawton, "Bochs: A portable pc emulator for unix/x," *Linux Journal*, vol. 1996, no. 29es, p. 7, September 1996.

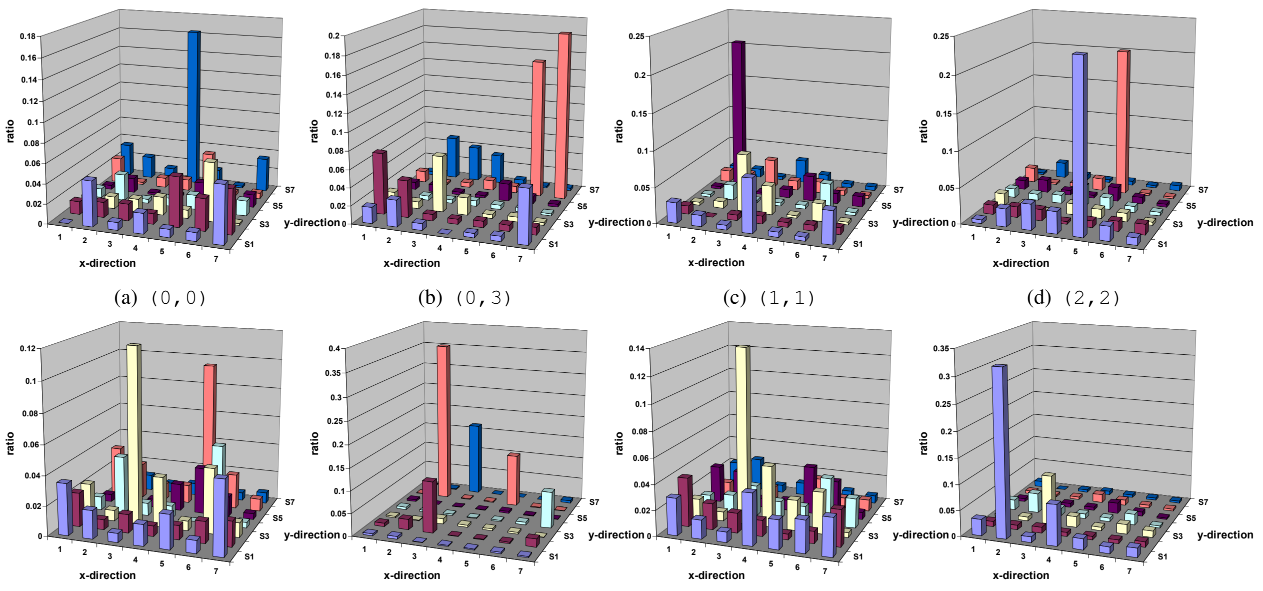
[21] B. Tsybakov and N. D. Georganas, "Self-similar processes in communications networks," *IEEE Trans. Information Theory*, vol. 44, no. 5, pp. 1713–1725, September 1998.

[22] M. S. Taqqu, W. Willinger, and R. Sherman, "Proof of a fundamental result in self-similar traffic modeling," *SIGCOMM Computer Communication Review*, vol. 27, no. 2, pp. 5–23, April 1997.

TABLE III

MEASURED HURST PARAMETER FOR SYNTHETIC TRAFFIC TRACES IN 7×7 MESH

`barnes`							`fft`						
0.922604	0.900864	0.929619	0.932654	0.887087	0.924292	0.924685	0.930469	0.901261	0.904385	0.919863	0.905215	0.920039	0.927163
0.910217	0.92359	0.894023	0.892669	0.900064	0.922199	0.916118	0.918589	0.930192	0.932514	0.93347	0.921322	0.913654	0.953305
0.913906	0.924976	0.918239	0.93298	0.878349	0.890662	0.87693	0.907441	0.929731	0.931737	0.91638	0.911257	0.935479	0.91756
0.933494	0.873282	0.90397	0.912061	0.924956	0.891938	0.887996	0.927311	0.927169	0.950335	0.933219	0.934989	0.923255	0.925058
0.926644	0.949166	0.894207	0.888905	0.879505	0.917606	0.898869	0.930802	0.92372	0.95463	0.924153	0.942931	0.920807	0.942047
0.940055	0.898484	0.894776	0.911445	0.925797	0.888928	0.922339	0.950361	0.932204	0.937753	0.935888	0.929717	0.925567	0.928409
0.831892	0.917855	0.917632	0.892862	0.879119	0.886606	0.862894	0.944464	0.931497	0.933646	0.927396	0.928349	0.932712	0.920281
						average error ratio = 0.032							average error ratio = 0.041
`lu`							`ocean`						
0.920638	0.913734	0.926406	0.914993	0.912014	0.929957	0.930625	0.8483	0.783373	0.770411	0.875487	0.766814	0.828058	0.824162
0.915871	0.913395	0.923574	0.93993	0.914495	0.925637	0.910338	0.739303	0.77524	0.798285	0.8164	0.80788	0.719764	0.780451
0.919504	0.95263	0.937157	0.935734	0.908091	0.915819	0.916393	0.895681	0.849855	0.709589	0.728271	0.758329	0.756766	0.705397
0.917747	0.936518	0.91593	0.915539	0.912704	0.94054	0.92322	0.841761	0.745178	0.807255	0.721676	0.736948	0.787291	0.71473
0.919705	0.933109	0.91106	0.918234	0.944434	0.923524	0.913384	0.837015	0.799498	0.699795	0.717817	0.815538	0.754053	0.804242
0.926757	0.904367	0.90686	0.927362	0.912072	0.916251	0.91677	0.805689	0.780835	0.790708	0.818814	0.764959	0.742495	0.753796
0.917779	0.929412	0.920856	0.918243	0.920673	0.918731	0.924098	0.799607	0.76778	0.853409	0.795997	0.752484	0.785473	0.752427
						average error ratio = 0.039							average error ratio = 0.027
`radix`							`raytrace`						
0.887586	0.927803	0.92179	0.927197	0.920037	0.902051	0.94445	0.90344	0.923164	0.94793	0.921658	0.919907	0.930598	0.916547
0.912924	0.912437	0.92907	0.935315	0.930807	0.920588	0.93943	0.933545	0.915805	0.938897	0.945765	0.93872	0.899207	0.93716
0.94104	0.929403	0.936362	0.946848	0.956438	0.937805	0.931888	0.908462	0.912697	0.931622	0.937748	0.918021	0.927636	0.909152
0.941251	0.936482	0.925917	0.930899	0.918615	0.940462	0.915708	0.953294	0.941418	0.915519	0.924081	0.906276	0.946177	0.926767
0.928504	0.94163	0.923772	0.921857	0.933592	0.916538	0.9224	0.919059	0.916655	0.923873	0.925912	0.949737	0.902823	0.905981
0.892508	0.92093	0.927384	0.932629	0.915048	0.933493	0.91771	0.953268	0.903407	0.921178	0.905038	0.914827	0.890978	0.909914
0.923702	0.926957	0.916387	0.906687	0.904082	0.917981	0.93198	0.934584	0.917112	0.917526	0.934793	0.918665	0.937771	0.920748
						average error ratio = 0.041							average error ratio = 0.032
`water-nsquared`							`water-spatial`						
0.898308	0.923437	0.940954	0.922098	0.923039	0.913735	0.934255	0.924825	0.907196	0.917813	0.935485	0.928298	0.923866	0.931682
0.937719	0.915814	0.925254	0.911514	0.913656	0.911644	0.930135	0.91907	0.921237	0.929094	0.907	0.929373	0.920545	0.915879
0.913194	0.919129	0.936486	0.927012	0.919423	0.930479	0.932936	0.924795	0.913452	0.938763	0.944932	0.8999	0.923134	0.906942
0.946839	0.928766	0.920046	0.945132	0.909967	0.92056	0.91277	0.907262	0.918599	0.904109	0.905521	0.91763	0.927629	0.90155
0.913586	0.923798	0.90929	0.913297	0.93518	0.913151	0.921679	0.901858	0.903175	0.930372	0.908927	0.917512	0.912481	0.926686
0.917045	0.914046	0.917961	0.918059	0.917218	0.919949	0.918271	0.920825	0.909482	0.909743	0.915424	0.927488	0.922289	0.920409
0.916328	0.948779	0.910059	0.925256	0.912018	0.919631	0.91887	0.931546	0.919134	0.919032	0.919456	0.925384	0.922665	0.92463
						average error ratio = 0.043							average error ratio = 0.041



(a) `(0,0)` (b) `(0,3)` (c) `(1,1)` (d) `(2,2)`

(e) `(3,4)` (f) `(4,1)` (g) `(5,3)` (h) `(6,1)`

Fig. 4. Distributions of traffic ratio on selected nodes for synthetic traffic trace of `barnes` application in 7×7

[23] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, "Self-similarity through high-variability: Statistical analysis of ethernet lan traffic at the source level," *SIGCOMM Computer Communication Review*, vol. 25, no. 4, pp. 100–113, October 1995.

[24] J. H. Bahn, S. E. Lee, and N. Bagherzadeh, "On design and analysis of a feasible network-on-chip (noc) architecture," in *ITNG '07: Proceedings of the International Conference on Information Technology*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 1033–1038.

[25] ——, "Design of a router for network-on-chip," *International Journal of High Performance Systems Architecture*, vol. 1, no. 2, pp. 98–105, 2007.

TABLE IV

COMPARISON OF INJECTION RATES BETWEEN ORIGINAL AND SYNTHETIC TRAFFIC TRACES IN 7×7 MESH

original traffic							synthetic traffic						
barnes													
0.059230	0.053724	0.027742	0.072139	0.042431	0.022027	0.083564	0.082931	0.055818	0.034412	0.067748	0.044069	0.032461	0.099190
0.028138	0.026441	0.091737	0.040663	0.027105	0.018616	0.157872	0.034190	0.034794	0.100266	0.042767	0.025398	0.023744	0.175239
0.021208	0.034636	0.167460	0.051668	0.056595	0.171639	0.074054	0.023813	0.043012	0.187606	0.060602	0.057168	0.164082	0.080500
0.018661	0.168760	0.039418	0.035184	0.021129	0.190589	0.027948	0.023320	0.160879	0.043212	0.038176	0.021441	0.183314	0.036472
0.015542	0.290371	0.044580	0.043946	0.154056	0.052408	0.025951	0.017888	0.324491	0.062108	0.059061	0.153222	0.052371	0.026421
0.024136	0.064680	0.028955	0.018944	0.225996	0.028720	0.025712	0.025948	0.067938	0.031171	0.026097	0.254005	0.033386	0.029015
0.112588	0.033496	0.037716	0.046235	0.030914	0.038005	0.022475	0.115044	0.045602	0.046282	0.056193	0.035571	0.060112	0.027397
													average error ratio = 0.066
fft													
0.029666	0.021464	0.006244	0.008981	0.003974	0.004094	0.007642	0.044174	0.022187	0.011228	0.013225	0.007996	0.008160	0.011472
0.004875	0.005072	0.008658	0.006987	0.004060	0.006191	0.008248	0.009111	0.007835	0.015021	0.013979	0.008017	0.011784	0.015275
0.005504	0.012254	0.013928	0.004325	0.005509	0.008301	0.005121	0.009686	0.016122	0.018494	0.008304	0.009961	0.013941	0.009420
0.005197	0.010923	0.015967	0.006519	0.006589	0.010280	0.006026	0.010128	0.021058	0.019755	0.011395	0.010854	0.019347	0.010866
0.005839	0.013510	0.007209	0.007880	0.028822	0.009014	0.009372	0.010099	0.016749	0.013635	0.014502	0.029346	0.017603	0.017984
0.024218	0.006013	0.005391	0.005813	0.007263	0.005757	0.005479	0.028910	0.011271	0.013258	0.013190	0.017048	0.009654	0.010647
0.010977	0.006434	0.006770	0.009269	0.005420	0.007020	0.005331	0.020612	0.011626	0.012168	0.016830	0.010672	0.011232	0.010410
													average error ratio = 0.27
lu													
0.031522	0.031486	0.020856	0.023430	0.015734	0.022615	0.017442	0.035190	0.033321	0.024563	0.027929	0.016210	0.027098	0.020902
0.020890	0.021314	0.019032	0.020641	0.016205	0.020843	0.024755	0.022980	0.024165	0.022151	0.022552	0.018941	0.023302	0.029638
0.020656	0.026310	0.080935	0.062681	0.022026	0.032159	0.025184	0.024147	0.031387	0.084642	0.073180	0.025317	0.037863	0.028438
0.023138	0.066632	0.025918	0.021733	0.019832	0.038455	0.022993	0.026949	0.076103	0.028331	0.023800	0.021964	0.045441	0.027337
0.029276	0.062923	0.023083	0.026969	0.124827	0.022259	0.015869	0.034131	0.067290	0.025380	0.030227	0.131003	0.026182	0.018190
0.033716	0.017579	0.016078	0.023511	0.024204	0.022110	0.021272	0.039662	0.020108	0.017948	0.027392	0.027782	0.023304	0.024658
0.021180	0.021210	0.022325	0.025828	0.022391	0.016425	0.016723	0.024829	0.024846	0.026269	0.030037	0.026304	0.018605	0.019671
													average error ratio = 0.13
ocean													
0.032656	0.052407	0.053066	0.034972	0.028506	0.029530	0.030476	0.035787	0.057154	0.057344	0.040709	0.032434	0.034281	0.034655
0.059482	0.076336	0.048041	0.047850	0.035247	0.066863	0.082566	0.063344	0.083863	0.051858	0.053600	0.039598	0.070091	0.087853
0.086836	0.035133	0.147355	0.105736	0.030472	0.067765	0.047462	0.096221	0.039389	0.150070	0.109829	0.033589	0.066591	0.050848
0.033205	0.123656	0.033226	0.037146	0.090287	0.046643	0.037019	0.036830	0.128103	0.038558	0.038684	0.093770	0.050305	0.038934
0.031955	0.035925	0.056059	0.063624	0.060362	0.034188	0.035604	0.037264	0.041204	0.057429	0.063482	0.064157	0.036824	0.038539
0.086679	0.068636	0.064304	0.067920	0.113985	0.062790	0.078873	0.098110	0.074006	0.071210	0.077321	0.119170	0.066208	0.081037
0.134312	0.096478	0.077706	0.371114	0.106355	0.055866	0.069098	0.139972	0.103311	0.085479	0.383716	0.110420	0.059998	0.076108
													average error ratio = 0.067
radix													
0.085885	0.036942	0.013971	0.020770	0.013779	0.026615	0.075784	0.088893	0.044055	0.015655	0.023982	0.014646	0.031706	0.087981
0.020545	0.032627	0.068298	0.015802	0.014502	0.016102	0.086488	0.023399	0.039103	0.071430	0.017994	0.016906	0.018508	0.098817
0.015641	0.013742	0.045850	0.088507	0.023404	0.047885	0.018831	0.017750	0.016331	0.052414	0.106077	0.027570	0.055200	0.021370
0.029354	0.122007	0.030738	0.020711	0.043255	0.096830	0.024630	0.031458	0.125732	0.036056	0.021926	0.049505	0.114297	0.028689
0.021256	0.057330	0.027209	0.028700	0.050168	0.035776	0.018404	0.022906	0.065534	0.032163	0.032332	0.060077	0.040638	0.021660
0.100576	0.014770	0.022440	0.024386	0.019568	0.015721	0.019553	0.105422	0.016417	0.026233	0.027580	0.022311	0.018466	0.023020
0.063424	0.022736	0.014779	0.049019	0.025052	0.015360	0.012427	0.074280	0.026441	0.015996	0.056243	0.028725	0.017785	0.014806
													average error ratio = 0.13
raytrace													
0.023664	0.021465	0.007970	0.013069	0.016971	0.004876	0.008253	0.026600	0.024533	0.015101	0.015422	0.018870	0.007748	0.013036
0.002765	0.002812	0.004423	0.003225	0.005511	0.003513	0.010775	0.006357	0.006265	0.008340	0.005851	0.009994	0.006027	0.012393
0.004837	0.002790	0.034439	0.006664	0.003718	0.008369	0.002179	0.006846	0.005459	0.039667	0.013511	0.006941	0.016444	0.004676
0.003348	0.006765	0.001843	0.002009	0.002106	0.004217	0.002200	0.007861	0.013654	0.004120	0.003902	0.004186	0.008048	0.004354
0.002909	0.008605	0.002654	0.002212	0.009745	0.003546	0.003611	0.006384	0.012687	0.005938	0.004182	0.012975	0.007668	0.008029
0.009046	0.002361	0.002462	0.005211	0.013867	0.002560	0.002273	0.019626	0.005373	0.005123	0.007910	0.014637	0.004454	0.004494
0.006204	0.002753	0.002233	0.005590	0.002890	0.003757	0.002269	0.013494	0.004253	0.004473	0.008239	0.005175	0.007742	0.004435
													average error ratio = 0.55
water-nsquared													
0.022009	0.016074	0.006600	0.013258	0.008507	0.006761	0.032313	0.023661	0.018801	0.009436	0.014747	0.010829	0.008875	0.035534
0.009806	0.010507	0.013351	0.009940	0.010067	0.009624	0.014857	0.013117	0.012558	0.015723	0.014398	0.010760	0.011149	0.016496
0.010534	0.012138	0.029008	0.025092	0.013357	0.027807	0.032449	0.012128	0.013980	0.034044	0.028754	0.015152	0.031198	0.038481
0.085102	0.025618	0.011492	0.051211	0.008794	0.012639	0.009679	0.096370	0.027683	0.013569	0.057958	0.010880	0.014441	0.013205
0.008935	0.018494	0.009593	0.010994	0.042124	0.009718	0.010690	0.012895	0.020918	0.011417	0.012446	0.049889	0.013163	0.012201
0.010824	0.009237	0.008589	0.010808	0.015551	0.010871	0.009337	0.012715	0.012761	0.011246	0.012833	0.016926	0.013036	0.012725
0.012572	0.009745	0.008497	0.015741	0.010567	0.009653	0.016860	0.014897	0.012041	0.009559	0.018343	0.012647	0.013809	0.019778
													average error ratio = 0.18
water-spatial													
0.039134	0.024893	0.023346	0.041009	0.027349	0.025811	0.039816	0.045553	0.029164	0.026863	0.046777	0.031364	0.030894	0.046882
0.025982	0.026905	0.038835	0.020124	0.020240	0.013104	0.015139	0.027518	0.029762	0.044945	0.023105	0.023776	0.014670	0.017940
0.019024	0.021667	0.069799	0.099872	0.012199	0.012508	0.014645	0.022233	0.024674	0.080859	0.111213	0.014298	0.014812	0.016507
0.022947	0.031320	0.012389	0.019785	0.012086	0.027014	0.012403	0.025280	0.036956	0.013214	0.022760	0.013890	0.031608	0.014042
0.012658	0.021574	0.014751	0.012530	0.023434	0.012939	0.013315	0.014053	0.023360	0.016113	0.012756	0.024740	0.014915	0.015571
0.015907	0.011776	0.011851	0.011625	0.015237	0.012631	0.019781	0.018469	0.013941	0.013507	0.013421	0.016467	0.014772	0.021343
0.023773	0.014841	0.012444	0.019787	0.012952	0.012253	0.013045	0.026972	0.017753	0.014759	0.021032	0.014798	0.013726	0.014950
													average error ratio = 0.14

TABLE V

COMPARISON OF CYCLE ACCURATE NOC SIMULATION BETWEEN ORIGINAL AND SYNTHETIC TRAFFIC TRACES IN 7×7 MESH

	original traffic		synthetic traffic		error ratio (%)	
application	offered load	avg. latency	offered load	avg. latency	offered load	avg. latency
barnes	0.06522	7.26	0.06951	7.32	6.58	0.86
fft	0.00889	8.20	0.01125	7.95	26.63	2.94
lu	0.03560	7.67	0.03240	7.52	8.99	2.17
ocean	0.06881	7.65	0.07345	7.78	6.75	1.70
radix	0.03690	7.96	0.04177	8.00	13.18	0.42
raytrace	0.00636	7.99	0.00987	7.95	55.21	0.53
water-nsquared	0.01649	7.93	0.01939	7.77	17.60	2.22
water-spatial	0.02221	7.99	0.02529	7.76	13.83	2.89

A System-C based Microarchitectural Exploration Framework for Latency, Power and Performance Trade-offs of On-Chip Interconnection Networks

Basavaraj Talwar and Bharadwaj Amrutur
Electrical and Communication Engineering Department, Indian Institute of Science, Bangalore.
Email: {bt,amrutur}@ece.iisc.ernet.in

***Abstract*— We describe a System-C based framework we are developing, to explore the impact of various architectural and microarchitectural level parameters of the on-chip interconnection network elements on its power and performance. The framework enables one to choose from a variety of architectural options like topology, routing policy, etc., as well as allows experimentation with various microarchitectural options for the individual links like length, wire width, pitch, pipelining, supply voltage and frequency. The framework also supports a flexible traffic generation and communication model. We provide preliminary results of using this framework to study the power, latency and throughput of a 4x4 multi-core processing array using mesh, torus and folded torus, for two different communication patterns of dense and sparse linear algebra. The traffic consists of both Request-Response messages (mimicing cache accesses) and One-Way messages. We find that the average latency can be reduced by increasing the pipeline depth, as it enables higher link frequencies. We also find that there exists an optimum degree of pipelining which minimizes energy-delay product.**

I. Introduction

On-chip interconnection networks (ICN) are critical elements of modern system-on-chip as well as multi-core designs. These chips have a multiplicity of communicating entities like programmable processing elements, hardware acceleration engines, memory blocks as well as off-chip interfaces. The communication patterns between these entities is very application dependent and diverse in terms of connectivity, burstiness, latency and bandwidth requirements. With power having become a serious design constraint, there is a great need for designing ICN which meets the target communication requirements, while minimizing power using all the tricks available at the architecture, microarchitecture and circuit levels of the design.

Many simulation tools have been developed to aid designers in ICN space exploration [1] [2]. These tools usually model the ICN elements at a higher level abstraction of switches, links and buffers and help in power/performance trade-off studies [3]. These are used to research the design of Router architectures [4] [5] and ICN topologies [6] with varying area/performance trade-offs for general purpose SoCs or to cater to specific applications. Kogel et. al. [1] present a modular exploration framework to capture performance of point-to-point, shared bus and crossbar topologies. Impacts of varying topologies, link and router parameters on the overall throughput, area and power consumption of the system (SoCs and Multicore chips) using relevant traffic models is discussed in [7]. Orion [2] is a power-performance interconnection network simulator that is capable of providing power and performance statistics. Orion model estimates power consumed by Router elements (crossbars, fifos and arbiters) by calculating switching capacitances of individual circuit elements. Most of these tools do not allow for exploration of the various link level options of wire width, pitch, serialization, repeater sizing, pipelining, supply voltage and operating frequency.

On the other hand, tools exist to separately explore these low level link options to various degrees as in [8], [9] and [10]. Work in [8] explores use of heterogeneous interconnects optimized for delay, bandwidth or power by varying design parameters such as a buffer sizes, wire width and number of repeaters on the interconnects. Courtay et. al [9] have developed a high-level delay and power estimation tool for link exploration that offers similar statistics as Intacte does. The tool allows changing architectural level parameters such as different signal coding techniques to analyze the effects on wire delay/power. Intacte [10] provides a similar capability to explore link level design options and is used in this research.

It is clear from works like [11] that there is a need for a co-design of interconnects, processing elements and memory blocks to fully optimize the overall system-on-chip performance. This necessitates a simulation framework which allows a co-simulation of the communicating entities along with ICN simulation. Additionally, to optimize power fully, one also needs to incorporate the link-level microarchitectural choices of pipelining etc. Hence we are developing a System-C framework which enables one to hook up actual communicating entities, along with the ICN and also allows for exploration of architectural and microarchitectural parameters of the ICN, in order to obtain the latency, throughput and power trade-offs. Results of trade-off studies in this paper consider Energy-Delay product (of the NoC) as the optimization parameter. Effects of wire density and area of NoC have not been taken into account in our experiments. We defer this study for future work.

We report on the design of this framework in System-C in Section II. We are using this framework to study the network design of a multi-core chip, supporting various communication patterns as in [12] for different classes of parallel computing benchmarks. We use a mix of Request-

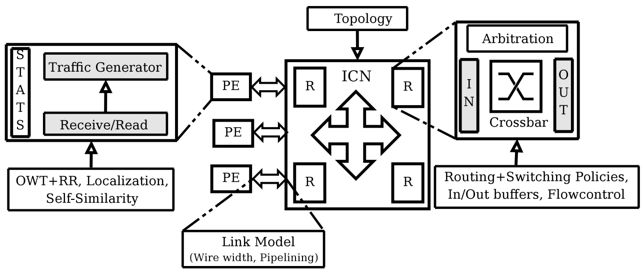

Topology
STATS
Traffic Generator
Receive/Read
OWT+RR, Localization, Self-Similarity
PE
PE
PE
ICN
R
R
R
R
Arbitration
IN
OUT
Crossbar
Routing+Switching Policies, In/Out buffers, Flowcontrol
Link Model (Wire width, Pipelining)

Fig. 1. Architecture of the SystemC framework.

Response and One-way traffic generation model to mimic realistic traffic patterns generated by these benchmarks. We use two benchmarks, Dense Linear Algebra (DLA) and Sparse Linear Algebra (SLA) benchmark communication patterns on three NoC topologies (2D Mesh, 2D Torus and Folded 2D Torus) to determine the average latency, throughput and power under different amounts of link pipelining and present some preliminary results in Section III. We draw some conclusions and outline future work in Section IV.

II. NoC Exploration Framework

The NoC exploration framework (Figure 1) has been built upon Open Core Protocol-IP models [13] using OSCI SystemC 2.0.1 [14] on Linux (2.6.8-24.25-default). The framework contains Router, Link and Processing Element (PE) modules and each can be customized via various parameters and will be described in more detail next. The NoC modules can be interconnected to form a desired NoC. The PE module represents any communicating entity on the SoC and not just the processing element. We can either hookup an actual executable model of the entity or some abstract model representing its communication characteristics. For abstract models, we support many different traffic generation and communication patterns. The link module can be used to customize the bit-width of the links as well as the degree of pipelining in the link. A single run (Figure 2) uses these models to run a communication task and outputs data files of message transfer logs. From these log files, one-way and round trip flit latency, throughput and link capacitance activity factors are extracted. Intacte is then used to obtain the final power numbers for different operating frequency and supply voltage options. Table I summarizes the various parameters that can be varied in the framework.

A. NoC Elements

1) Traffic Generation and Distribution Models (PE): To test NoCs on realistic multi-core applications we setup traffic generation and distribution to mimic various communication patterns. We support Request-Response (RR) and One-Way Traffic (OWT) generation. For example in multi-core chips, the former can correspond to activities like cache line loads and the latter can correspond cache line write backs.Traffic distribution input is given using two matrices of sizes N×N, where N is the number of communicating entities. Item (i, j) in a matrix gives the probability of communication of PE i with j in the current cycle. Two separate matrices correspond to Request-Response (RR) and One-Way traffic (OWT) generation. The probability of choosing among the two matrices depends on a global input to decide the percentage of RR traffic to be generated for the simulation run. This model can be further expanded to capture burst characterisitics as well as message size and is something we plan to add in the future. The communication packets are broken into a sequence of Flit transfers. The Flit header format is shown in Figure 3. The SQ field is used to identify in order arrival of all flits. Response flits have first 2 bits set to 11. SRCID, DSTID and FlitID fields are preserved in Response flit for the sake of error checking and latency calculations in the framework. The traffic receiver will read the header to determine if the flit type is RR or not (flag RQ). If RQ is set, then the Traffic

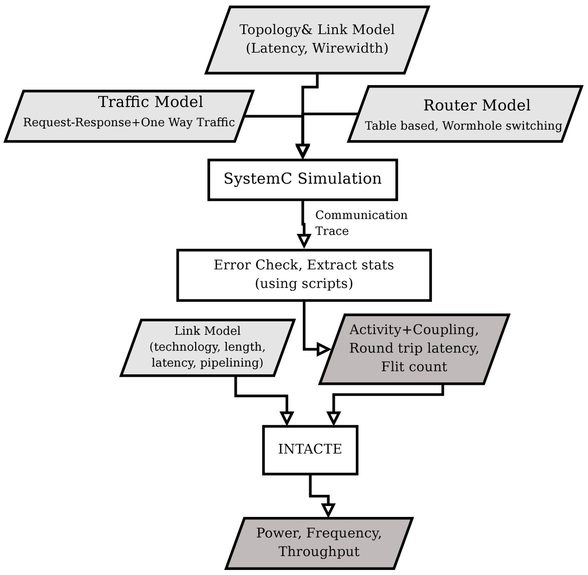

Topology& Link Model (Latency, Wirewidth)
Traffic Model
Request-Response+One Way Traffic
Router Model
Table based, Wormhole switching
SystemC Simulation
Communication Trace
Error Check, Extract stats (using scripts)
Link Model (technology, length, latency, pipelining)
Activity+Coupling, Round trip latency, Flit count
INTACTE
Power, Frequency, Throughput

Fig. 2. Flowchart depicting simulation steps.

TABLE I
ICN exploration framework parameters.

Parameter	*Description*
NoC Parameters	
Routing Algorithms	Source Routing and Table based routing
Switching Policy	Packet, Circuit, Wormhole, VC switching
Traffic Paradigm	Request-Response & One-Way Traffic
Traffic Generation Scheme	Deterministic, Self-Similar
Traffic Distribution Scheme	Deterministic, Uniformly random HotSpot, Localized, First Matrix Transpose
Router Microarchitecture	
No. of Input/Output Ports	2-8 (based on topology to be generated)
Input/Output buffer sizes	Flit-level buffers
Crossbar Switching capacity	In terms of flits (default=1)
Link Microarchitecture	
Length of interconnect	Longest link in mm
Bit width of the interconnect	
Circuit Parameters	
Frequency, Supply Voltage	

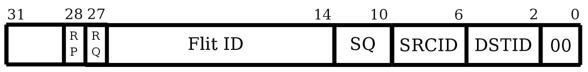


Fig. 3. Flit header format. DSTID/SRCID: Destination/Source ID, SQ:Sequence Number, RQ & RP: Request and Response Flags and a 13 bit flit id.

TABLE II
TRAFFIC GENERATION/DISTRIBUTION MODEL AND EXPERIMENT SETUP.

Parameter	Values	
Communication Patterns	DLA Traffic	SLA Traffic
NoCs Simulated	2D Mesh, Torus and Folded Torus	
Localization Factor	0.7	0.5
Traffic injection rate	20%	
RR Factor	0.03	0.1
Size of phit (Wire width)	32 bits	
OW and RR Request Flit	1 flit (2 phits)	
RR Response Flit	3 flits (6 phits)	
Simulation Time	40000 cycles	
Process	45nm	
Environment	Linux (2.6.8-24.25-default)+ OSCI SystemC 2.0.1 + Matlab 7.4	

generator is notified and the flit header is sent to the Traffic generator. RR traffic has priority over OW traffic and hence the request will be immediately serviced (without breaking an OW flit). Response flit to the request flit has RP set and RQ reset. In a received flit, if RQ is not set, then no action is taken. Table II lists out parameters used in our traffic model and in experiments. The framework is also capable of generating Deterministic, Uniformly Random, Hotspot and First Matrix Transpose traffic distributions.

2) Router Model: The router model is a parameterized, scalable module of a generic router [7]. *Router microarchitecture* parameters include number of Input/Output ports, sizes of input/output buffers, switching capacity of the crossbar (no. of bits that can be transfered from input to output buffers in a cycle) etc (Table I). Flow control is implemented through sideband signals [13].

B. Power Model

Intacte [10] is used for interconnect delay and power estimates. Design variables for Intacte's interconnect optimization are wire width, wire spacing, repeater size and spacing, degree of pipelining, supply (V_{dd}) and threshold voltage (V_{th}). Activity and coupling factors are input to Intacte from the System-C simulation results. Intacte arrives at a power optimal number of repeater, sizes and spacing for a given wire to achieve a desired frequency. The tool also includes flop and driver overheads for power and delay calculations. Intacte outputs total power dissipated including short circuit and leakage power values. We arrive at approximate wire lengths using floorplans. Other physical parameters are obtained from Predictive Technology Models [15] models for $45nm$.

Power consumed by routers have not been included in the results presented in the paper and will be added in the future.

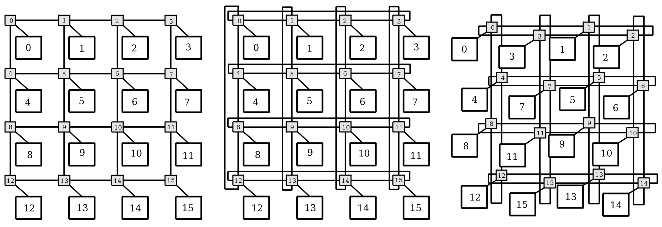


Fig. 4. Schematic of 3 compared topologies (L to R: Mesh, Torus, Folded Torus). Routers are shaded and Processing Elements(PE) are not.

However we can still draw some useful conclusions about those aspects of the ICN design which relate to the links like the degree of pipelining and optimal toplogy.

III. SIMULATION AND RESULTS

We study a 4x4 multi-core platform for three different network topologies of Mesh, Torus and Folded-torus. We use two communication patterns from [12] of Dense Linear Algebra (DLA) and Sparse Linear Algebra (SLA) benchmarks. DLA applications exhibit highly localized communication. The traffic model for DLA generates 70% traffic to immediate neighbors and remaining traffic is distributed uniformly to other nodes. SLA communication is reproduced using 50% localized traffic and rest of the traffic is destined to half of the remaining nodes. Further we assume all RR traffic to be localized. For eg. 10% of generated traffic over the simulation per PE will be of Request type if RR=0.1. All Request flits are destined to immediate neighbors. 70% of flits generated by any PE over the simulation time are destined to immediate neighbors if localization factor is 0.7(as in case of DLA).

Experiments are designed to calculate latency (clock cycles), throughput (Gigabits/sec) and power (milliWatts) of various topologies. Table II lists some of the simulation setup parameters used in the following experiments.

A. NoC Topologies

In this work we consider three similar topologies for tradeoff studies. Router and processing elements are identical in all three topologies. In fact the same communication trace is played out for all the different ICN parameter explorations. The schematic of the three NoCs is shown in Figure 4 with the Floorplans largely following the schematics. The floorplans are used to estimate the wire lengths which are then input to Intacte. Processing elements sizes are estimated by scaling down the processor in [16] to $45nm$to be of size 2.25×1.75mm. The routers are of size 0.3×0.3mm. The length of the longest links in the Mesh, Torus and Folded Torus are estimated as $2.5mm$, $8.15mm$ and $5.5mm$ respectively. The longest link in the torus connect the routers at the opposite sides. The routing policy for all topologies is table based. Routing tables are populated such that longer links have minimum activity. Lengths of links in each of the topologies and pipelining factors is illustrated in Table III. Pipelining factor corresponds to the longest link in the NoC. Pipelining factor of 1 means the longest link is unpipelined, P=2 indicates it has a two cycle latency and so on.

TABLE III

LINKS AND PIPELINING DETAILS OF NOCS

Topology	Length in mm (no. of links)	Pipelining							
2D Mesh	2.5 (24)	1	2	3	4	5	6	7	8
	2.0 (56)	1	2	3	4	4	5	6	7
2D Torus	8.15(8)	1	2	3	4	5	6	7	8
	6.65(8)	1	2	3	4	4	5	6	7
	2.5 (24)	1	1	1	2	2	2	3	3
	2.0 (56)	1	1	1	1	2	2	2	2
Folded 2D Torus	5.5 (16)	1	2	3	4	5	6	7	8
	4.5 (16)	1	2	3	4	5	5	6	7
	2.75(16)	1	1	2	2	3	3	4	4
	2.25(16)	1	1	2	2	3	3	3	4
	2.0 (32)	1	1	2	2	2	3	3	3

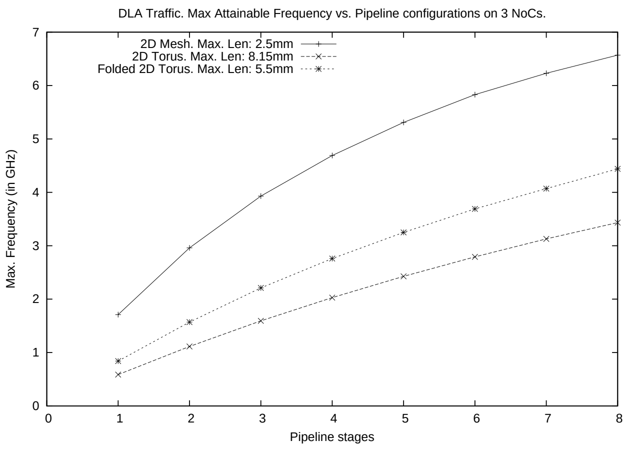


Fig. 5. Max. frequency of links in 3 topologies. Lengths of longest links in Mesh, Torus and Folded 2D Torus are $2.5mm$, $8.15mm$ and $5.5mm$.

B. Round Trip Flit Latency & NoC Throughput

Round trip flit latency is calculated starting from injection of the first phit (physical transfer unit in an NoC) to the reception of the last phit. In the case of OW traffic latency is one way. In case of RR traffic it is the delay in clock cycles of beginning of request injection to completion of response arrival. Communication traces are analysed using error checking (for phit loss, out-of-order reception, erroneous transit etc.) and latency calculation scripts to ensure functional correctness of the system.

Total throughput of the NoC (in $bits/sec$) is calculated as total number of bits received $((flit_r * bits_{flit}))$ at sink nodes divided by total (real) time $((\frac{1}{f} * sim_{cycles}))$ spent (Eqn 1).

$$Th_{total} = \frac{flit_r * bits_{phit}}{\frac{1}{f} * sim_{cycles}} \tag{1}$$

Max achievable frequency of a wire of given length is obtained using Intacte(Figure 5). Max throughput of each NoC running DLA traffic at P=1 is shown in Figure 6.2D Mesh has the shortest links and highest achievable frequency and hence the highest throughput.

Average round trip latencies in nano-seconds over various pipeline configurations in all 3 NoCs is shown in Fig. 7. Results show overall latency of flits actually decrease to a certain point by pipelining. Avg. latencies are larger for RR type of traffic and it also has a larger number of flits involved (1 Req + 3 Response). Clearly, there is a latency advantage by pipelining links in NoCs upto a point. This is because as the number of pipe stages increase, the operation frequency can also be increased as the length of wire segment in each pipe stage decreases. Real time latencies do not vary much after pipelining configuration P=5, as delay of flops start to dominate and there is not much marginal increase in frequency. Throughput and Latency behaviour for SLA traffic are identical (not shown here).

C. NoC Power/Performance/Latency Tradeoffs

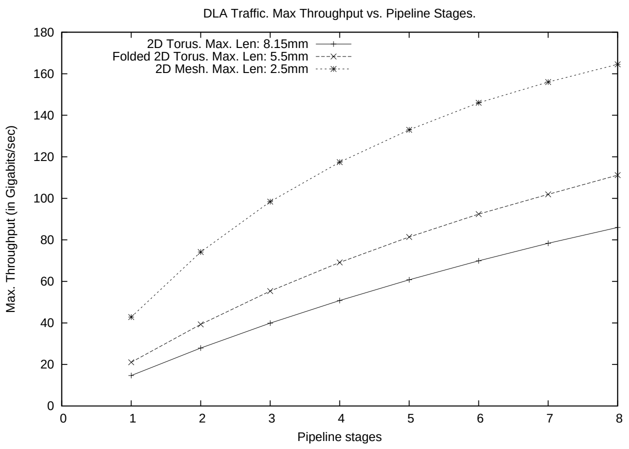


Fig. 6. Total NoC throughput in 3 topologies, DLA traffic.

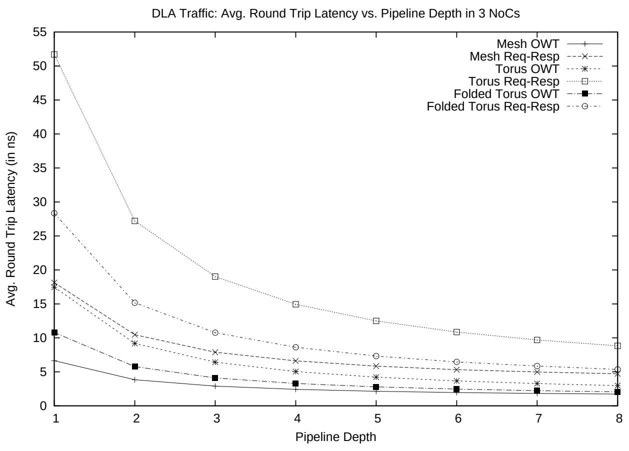


Fig. 7. Avg. round trip flit latency in 3 NoCs, DLA traffic.

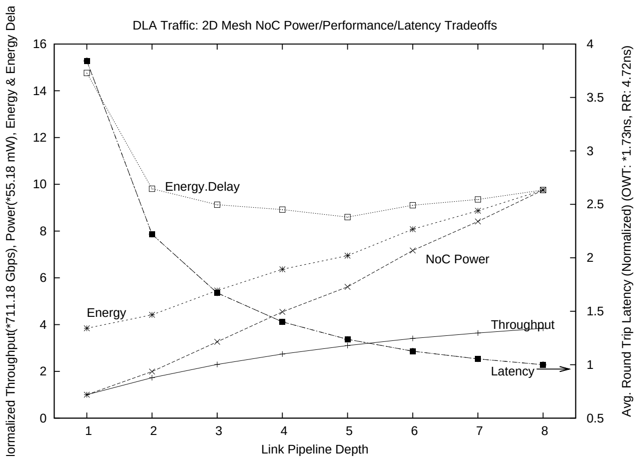


Fig. 8. 2D Mesh Power/Throughput/Latency tradeoffs for DLA traffic. Normalized results are shown.

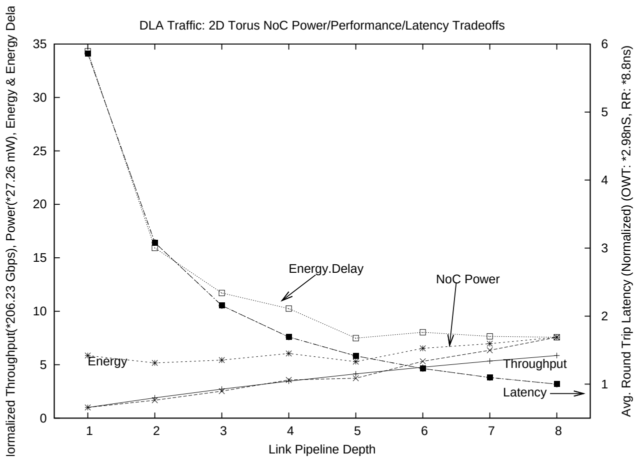


Fig. 10. DLA Traffic, 2D Torus Power/Throughput/Latency tradeoffs. Normalized results are shown.

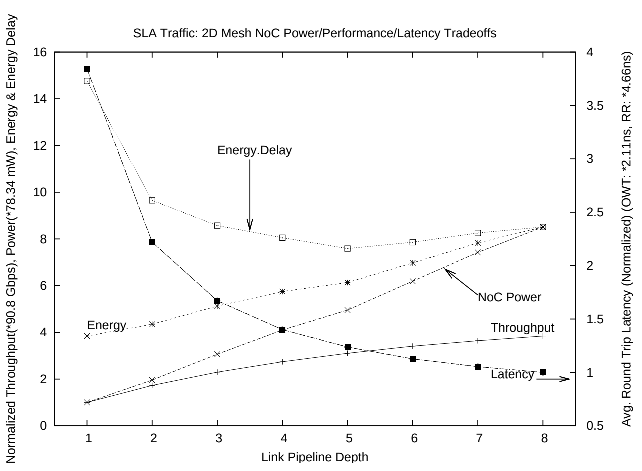


Fig. 9. 2D Mesh Power/Throughput/Latency tradeoffs for SLA traffic.

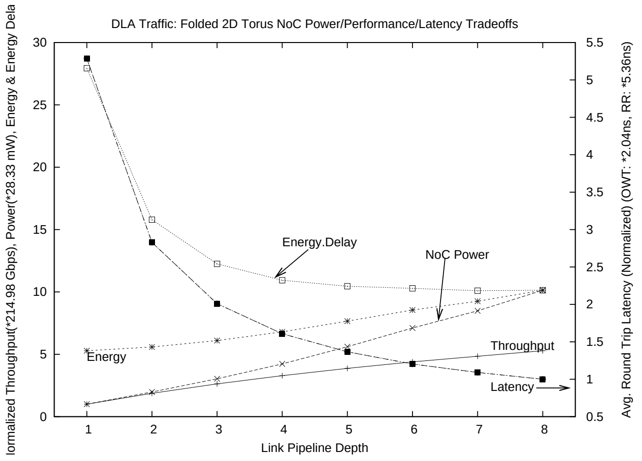


Fig. 11. DLA Traffic, Folded 2D Torus Power/Throughput/Latency tradeoffs. Normalized results are shown.

1) 2D Mesh: Figure 8 and 9 shows the combined normalized results of NoC power, throughput and latency experiments on a 2D Mesh for DLA and SLA traffic. Throughput and power consumption are lowest at P=1 and highest at P=8. Normalized avg. round trip flit latency for both OW and RR traffic is shown (the curves overlap). From the graph it is seen that growth in power makes configurations more than P=5 less desirable. Link pipelines with P=1,2 and 3 are also not optimal with respect to latency in both these benchmarks. Rise in throughput also starts to fade as configuration of more than P=6 are used. The optimal point of operation indicated by the results from both communication patterns is P=5. Energy curve is obtained as the product of normalized Latency and Power values. Energy for communication increases with pipeline depth. Energy Latency (Energy.Delay) is the product of Energy and Latency values. Quantitatively the optimal point for operation is when the longest link has pipeline segments (P=5). In DLA traffic, Avg. round trip flit latency of flits in the NoC is 1.23 times minimum and 32% of maximum possible. NoC power consumed is 57% of max and throughput 80.5% of max possible value.

2) 2D Torus and Folded 2D Torus: Similar power, throughput and latency tradeoff studies are done on both communication patterns on 2D Torus (Fig. 10) and Folded 2D Torus (Fig. 11) NoCs. Results obtained in 2D Torus experiments indicate that growth in power makes configurations more than P=5 is not desirable. Latencies of flits in pipeline configurations P=1-4 are large. Rise in throughput also starts to fade as configurations after P=5 are used. The optimal point of operation indicated by the Energy Delay curves in both DLA and SLA traffic (not shown here) for 2D Torus is P=5. In DLA traffic, this configuration shows power consumed by the NoC is 50% of the value consumed at P=8 and throughput is 70.5% the max value. Avg. Round Trip latency of flits for both OW & RR traffic is 1.4 times minimum and 24% of the maximum (when P=1).

Tradeoff curves for the Folded 2D Torus show similar trends as in the 2D Torus. Avg. round trip flit latency reduction and throughput gain after P=6 is not considerable. There is no single optimum obtained from the Energy Delay curve.

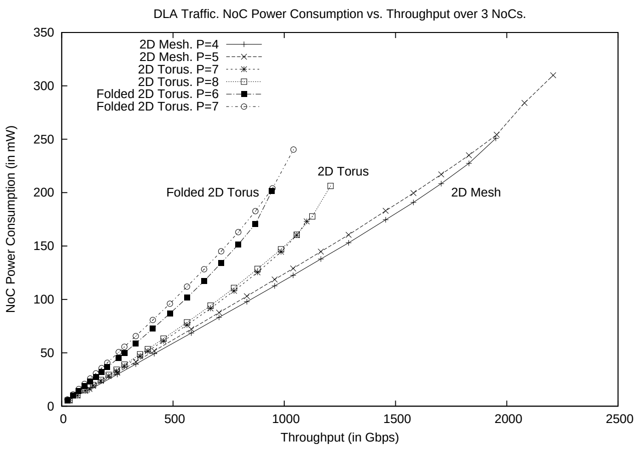


Fig. 12. Frequency scaling on 3 topologies, DLA Traffic.

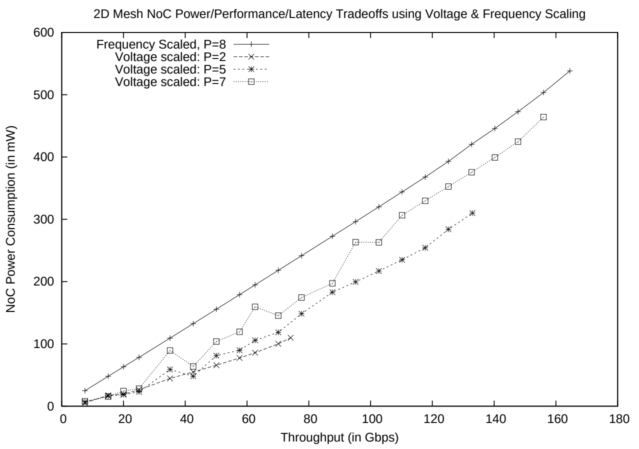


Fig. 13. Dynamic voltage scaling on 2D Mesh, DLA Traffic. Frequency scaled curve for P=8 is also shown.

Pipeline configurations from P=5 to P=7 present various throughput and energy configurations for approximately same Energy Delay product.

D. Power-Performance Tradeoff With Frequency Scaling

We discuss the combined effects of pipelining links and frequency scaling on power consumption and throughput of the 3 topologies (Figure 4) running DLA traffic. Maximum possible frequency of operation at full supply voltage $(1.0V)$ is determined using Intacte.

Figure 12 shows NoC power consumption for 3 example topologies over a pair of pipelining configurations along with frequency scaling (at V_{dd}). As observed from the graph, power consumption of a lower pipeline configuration exceeds the power consumed by a higher configuration after a certain frequency. Larger buffers (repeaters) are added to push frequencies to the maximum possible value. Power dissipated by these circuit element start to outweigh the speed advantage after a certain frequency. We call this the “crossover” frequency. The graph shows 3 example pairs from each NoC from each of the topologies to illustrate this fact.

Maximum frequency of operation of an unpipelined longest link in a 2D Mesh $(2.5mm)$ is determined to be 1.71GHz. This maximum throughput point is determined in each pipeline configuration in each topology. Frequency is scaled down from this point and power measurements are made for NoC activity obtained using the SystemC framework for DLA traffic.At crossover frequencies it is advantageous to switch to higher pipelining configurations to save power and increase throughput. For example in a 2D Mesh, link frequency of 3.5GHz can be achieved by pipelining configuration of 3 and above. NoC power consumption can be reduced by 54% by switching to a 3 stage pipeline configuration from 8 stage pipeline configuration.In other words, a desired frequency can be acheived by more than one pipeline configuration. For example, in a 2D Torus frequency (throughput) of 2.0GHz can be achieved by using pipeline configurations from 4 to 8. NoC power consumption can be reduced by 13.8% by switching from P=8 to P=4 and still achieve similar throughput.

TABLE IV
DLA TRAFFIC, FREQUENCY CROSSOVER POINTS IN 2D MESH

Pipe	Trip Frequency (in GHz)		
Stages	Mesh	Torus	Folded Torus
1-2	1.7	0.25	0.45
2-3	2.96	0.7	1.5
3-4	3.93	1.1	2.0
4-5	4.69	2.0	2.76
5-6	5.31	2.2	3.2
6-7	5.83	2.8	3.69
7-8	6.23	3.0	4.07

E. Power-Performance Tradeoff With Voltage and Frequency Scaling

In each topology, frequency is scaled down from the maximum and the least voltage required to meet the scaled frequency is estimated using Intacte and power consumption and throughput results are presented. Voltages are scaled from $1.0V$ till 0.1GHz is met for each pipelining configuration in each NoC.Similar to the frequency scaling results there exists a crossover frequency in a pipelining configuration after which it is power and throughput optimal to switch to a higher pipelining stage (Table IV).Figure 13 compares Power and Throughput values obtained by voltage and frequency scaling with a frequency scaled P=8 curve for 2D Mesh with DLA traffic. Scaling voltage along with frequency compared to scaling frequency alone can result in power savings of upto 14%, 27% and 51% in cases of P=7, P=5 and P=2 respectively.

Comparison of all 3 NoCs is presented in Table V.

IV. CONCLUSION

Consideration of low level link parameters like pipelining, bit widths, wire pitch, supply voltage, operating frequency etc, along with the usual architectural level parameters like router type, topology etc., of an ICN enables better optimization of the SOC. We are developing such a framework in System-C

TABLE V

COMPARISON OF 3 TOPOLOGIES FOR DLA TRAFFIC.

Topology	Pipe Stages	Power (mW)	Performance (Gbps)
Mesh	1	55.18	42.82
	2	109.87	74.12
	4	250.83	117.44
	7	464.16	156.00
Torus	1	27.26	14.67
	2	45.71	27.89
	4	97.48	50.78
	7	206.22	78.33
Folded Torus	1	28.32	21.03
	2	55.95	39.31
	4	119.75	69.11
	7	287.18	101.91

since it can allow co-simulation with models for the communicating entities along with the ICN.

Preliminary studies on a small 4x4 multi-core ICN for three different topologies and two different communication patterns indicate that there is an optimum degree of pipelining of the links which minimizes the average communication latency. There is also an optimum degree of pipelining which minimizes the energy-delay product. Such an optimum exists because increasing pipelining allows for shorter length wire segments which can be operated either faster or with lower power at the same speed.

We also find that the overall performance of the ICNs is determined by the lengths of the links needed to support the communication patterns. Thus the mesh seems to perform the best amongst the three topologies we have considered in this study. This opens up interesting research opportunities for reconfigurable ICNs with heterogenous links which can support different patterns efficiently.

It also points to an overall optimization problem that exists in the architecture of the individual PEs versus the overall SOC, since smaller PEs lead to shorter links between PEs, but more traffic, thus pointing to the existence of a sweet spot in terms of the PE size.

ACKNOWLEDGMENT

We thank Shailesh Kulkarni for help with the initial development of this framework. We acknowledge funding support from Texas Instruments, India.

REFERENCES

[1] T. Kogel et. al., "A modular simulation framework for architectural exploration of on-chip interconnection networks," in *Proc. of, Hardware/Software Codesign and System Synthesis, 2003 (CODES+ISSS'03). Intl. Conf. on*, pp. 338–351, Oct. 2003.

[2] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: A power-performance simulator for interconnection networks," in *Proc. of, MICRO 35*, 2002.

[3] P. Gupta, L. Zhong, and N. K. Jha, "A high-level interconnect power model for design space exploration," in *Proc. of, Computer Aided Design (ICCAD '03). Intl. Conf. on*, pp. 551–558, 2003.

[4] K. Lee, S.-J. Lee, and H.-J. Yoo, "Low-power network-on-chip for high-perforamance soc design," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, pp. 148–160, Feb. 2006.

[5] S. E. Lee, J. H. Bahn, and N. Bagherzadeh, "Design of a feasible on-chip interconnection network for a chip multiprocessor (cmp)," in *Proc. of, Computer Architecture and High Performance Computing. Intl. Symp. on*, pp. 211–218, 2007.

[6] F. Karim et. al., "An interconnect architecture for networking systems on chips," *IEEE Micro*, vol. 22, pp. 36–45, Oct. 2002.

[7] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," *IEEE Transactions on Computers*, vol. 54, pp. 1025–1040, Aug. 2005.

[8] R. Balasubramonian, N. Muralimanohar, K. Ramani, and V. Venkatachalapthy, "Microarchitectural wire management for performance and power in partitioned architectures," in *Proc. of, High-Performance Computer Architecture. HPCA-11. 11th International Symposium on*, pp. 28–39, Feb. 2005.

[9] A. Courtey, O. Sentieys, J. Laurent, and N. Julien, "High-level interconnect delay and power estimation," *Journal of Low Power Electronics*, vol. 4, pp. 1–13, 2008.

[10] R. Nagpal, M. Arvind, Y. N. Srikanth, and B. Amrutur, "Intacte: Tool for interconnect modelling," in *Proc. of 2007 Intl Conf. on Compilers, Architecture and Synthesis for Embedded Systems(CASES 2007)*, pp. 238–247, 2007.

[11] R. Kumar, V. Zyuban, and D. M. Tullsen, "Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling," in *Proc. of, Computer Architecture. ISCA '05. 32nd International Symposium on*, pp. 408–418, 2005.

[12] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," Tech. Rep. UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[13] http://www.ocpip.org/socket/systemc/, "Ocp-ip, systemc ocp models."

[14] http://www.systemc.org/, "Open systemc initiative."

[15] http://www.eas.asu.edu/∼ptm/, "Predictive technology models."

[16] G. Konstadinidis et. al., "Implementation of a third generation 16-core, 32-thread, cmt sparc processor," in *ISSCC '08: Processor of the International Solid-State Circuits Conference*, pp. 84–85, IEEE, 2008.

Application Specific Buffer Allocation for Wormhole Routing Networks-on-Chip

Wang Liwei[1], Cao Yang[1,2], Li Xiaohui[1], Zhu Xiaohu[1]
[1]School of Electronic Information
[2]State Key Laboratory of Software Engineer
Wuhan University
Wuhan, China
wangliwei@mail.whu.edu.cn, caoyang@whu.edu.cn

***Abstract*—A buffer allocation algorithm for wormhole routing networks-on-chip was proposed. When the total budget of the available buffering space is fixed, the proposed algorithm automatically assigns the buffer depth for each input channel, in different routers across the chip, according to the traffic characteristics of the target application. The simulation results show that the buffer allocation result is more reasonable and lower average packet latency can be achieved compared to the uniform buffer allocation.**

I. Introduction

As technology scales and chip integrity grows, on chip communication is playing an increasing dominant role in System-on-Chip (SoC) design. The NoC approach was proposed as a promising solution to complex intraSoC communication problems[1-3]. It consists of a grid of nodes where each node can be a SoC, an IP, a DSP, etc. Compared to traditional bus interconnection architecture, NoC is much more extensible and parallelizable.

Fig. 1 shows a 4×4 2D mesh NoC which consists of 16 nodes and a typical router architecture. There are buffers at every input channels of router which significantly affect the system performance.

Compared to a computer-network, an on-chip network is much more resource limited. In order to minimize the implementation cost, the interconnection network should be implemented with very little area overhead. In a packet switched network, the route processing logic occupies only a small portion of area(about 6.6%) in each router[2], but the input buffers take a significant one[6], consequently, their size should be carefully minimized.

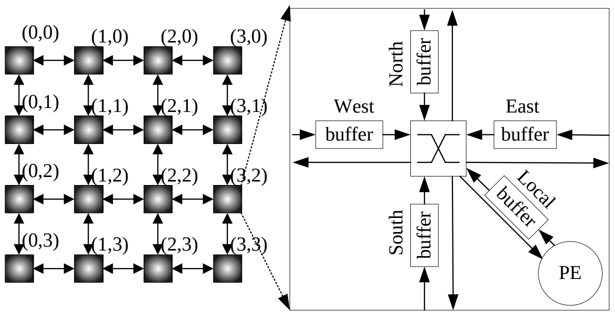


Figure 1. Block diagram of 4X4 mesh-based NoC

Wormhole routing is one of the most popular switching techniques in NoC and it's more suitable for implementing NoCs compared to store-and-forward and virtual cut-through switching[4,5]. In wormhole switching, a data packet (message) is divided into small flits for transmission and flow control. The header flit governs the route of the packet and the reaming data flits follow it in a pipeline fashion. When the header flit is blocked due to contention for output channels or due to insufficient buffer space, all other data flits wait at their current nodes forming a chain of flits that spans over multiple nodes. Wormhole switching makes the end-to-end delay insensitive to the packet destination due to the pipelining of flits, and routers require only small amount of buffer space.

II. Related Work

The input buffer size significantly affects system performance, area and power consumption of NoC. Traditionally, all input channels of every router are assigned with the same amount of buffer. But because of the imbalance of the traffic pattern in most real NoCs, uniform buffer size allocation may not be the most effective way to use the silicon area. To utilize buffering resources more efficiently, one better buffer allocation way is to allocate the buffer size according to the traffic pattern of the target application, allocate more buffering resources only to the heavy loaded channels. Therefore, the buffer allocation problem of NoC can be described as: under total buffering resources constraints, calculate the buffer size of each input channel to minimize the average packet latency of NoC[6].

To solve buffer allocation problem, it is necessary to evaluate the NoC performance. Traditional work on performance evaluation uses either simulation[11] or analytical models[7,8,12,13]. The network simulation method is straight and easy to understand, but time consuming, while using analytical model is more suitable to solve buffer allocation problem. In[13], a wormhole-based mesh network is modeled as a closed queuing network to calculate the average packet latency, but it can apply only to networks with single-flit buffers. In[7,12], two analytical models are proposed, but it can apply only to networks with infinite buffers. In[8], a analytical model for networks with finite buffers is proposed, but it's constrained to uniform traffic conditions.

In[9-10], the authors present two dynamic buffer allocation methods, their main idea are similar: all buffering resources in a router are placed together as a buffer pool, when there is a data packet enter the input channel, some amount of buffering resources are assigned to store this packet temporally; after the packet leave the router, those buffers are released to buffer pool in case for the next use. Dynamic allocation can not solve buffer allocation problem if the size of buffer pool is not big enough.

In[6], the authors describe and provide an efficient way to solve the buffer allocation problem for NoC designs which use store-and-forward or virtual cut-through switching for the first time based on the queuing theory. But in most NoC designs, the wormhole switching is more suitable. In this paper, we study the buffer allocation in wormhole routing NoCs.

III. Buffer Allocation Algorithm

First, a wormhole router analytical model is presented. Using this model, the system performance bottleneck among the different channels can be detected. In this paper, performance bottleneck is defined as such an input channel which owns the FIFO that has the highest probability to be "full". After finding the system performance bottleneck, the buffer allocation algorithm iteratively adds extra buffering space to the bottleneck channels until the total buffer budget is reached, which leads to the maximum improvement in performance.

A. The wormhole router analytical model

The model in this paper is based on the following assumptions, which are commonly used in similar studies[6-8,12,13].

i. Nodes generate traffic independently of each other, and according to a Poisson process.

ii. NoC uses XY routing algorithm.

iii. Packet length is fixed at M flits. Each flit takes one cycle to advance from one to the next. Buffer width equals to the bit widths of a flit.

iv. The local queue at the injection channel in the source node has infinite capacity; moreover, packets are transferred to the local PE as soon as they arrive at their destinations through the ejection channel.

The basic parameters used in this paper are summarized in Table I.

TABLE I. PARAMETER NOTATION

Param.	Description
M	The size of a data packet (M flits)
T_H	The time needed by a router to process the header flit
dir	Direction, i.e.: *North, East, South, West, Local*
$b_{x,y,dir}$	The probability of the buffer at *dir* input channel of *Node(x,y)* being full
$\rho_{x,y,dir}$	The utilization factor of *dir* input channel at *Node(x,y)*
$\lambda_{x,y,dir}$	The packet arrival rate at *dir* input channel of *Node(x,y)*
$\mu_{x,y,dir}$	The packet service rate at *dir* input channel of *Node(x,y)*
$l_{x,y,dir}$	The buffer size of *(x,y) dir* input channel at *Node(x,y)*
$a_{x,y}$	The packet injection rate of *Node(x,y)*
$d_{(x,y)(x',y')}$	The probability of a packet generated by *Node(x,y)* to be delivered to *Node(x',y')*
$T_{x,y,dir}$	The packet service time at *dir* input channel of *Node(x,y)*
$TB_{x,y,dir}$	The blocking delay at *dir* input channel of *Node(x,y)*
$\theta_{x,y,dir}$	The probability of a packet get blocked at the head of *dir* input channel of *Node(x,y)*
$\omega_{x,y,dir}$	The mean waiting time that a packet needs to wait in the event of blocking
$\lambda_{x,y,dir\to dir'}$	The packet arrival rate which is forwarded from *dir* input channel to *dir'* input channel of *Node(x,y)*

Resorting to the theory of finite queuing networks[15], every input channel buffer can be modeled as a M/M/1/K finite queue. Therefore, the probability of the buffer at *dir* input channel of *Node*(*x,y*) being full can be calculated using the following equations:

$$b_{x,y,dir} = \frac{1-\rho_{x,y,dir}}{1-\rho_{x,y,dir}^{\,l_{x,y,dir}+1}} \times \rho_{x,y,dir}^{\,l_{x,y,dir}} \quad (1)$$

$$\rho_{x,y,dir} = \frac{\lambda_{x,y,dir}}{\mu_{x,y,dir}} = \lambda_{x,y,dir} \times T_{x,y,dir} \quad (2)$$

The packet arrival rate at *dir* input channel of *Node(x,y)* can be calculated with the following equation:

$$\lambda_{x,y,dir} = \sum_{\forall j,k}\sum_{\forall j',k'} a_{j,k} \times d_{(j,k)(j',k')} \times \Re(j,k,j',k',x,y,dir) \quad (3)$$

In (3), the $\Re(j,k,j',k',x,y,dir)$ is the routing function, it equals 1 if the packet from *Source Node(j,k)* to *Destination Node(j',k')* uses the *dir* channel of *Node(x,y)*; it equals 0 otherwise. In this paper, we use XY deterministic routing algorithm, it's easy to calculate $\lambda_{x,y,dir}$ according to the Eq.(3).

B. Calculation of the packet service time

The packet service time at *dir* channel of *Node(x,y)* is given by

$$T_{x,y,dir} = T_H + M + TB_{x,y,dir} \quad (4)$$

In (4), $T_H + M$ represents the service time per packet in a router without contention; when there is a contention, the waiting time that a packet needs to wait in a buffer can be modeled by $TB_{x,y,dir}$. In[8], the authors present a method to calculate this blocking delay:

$$TB_{x,y,dir} = \theta_{x,y,dir} \times \omega_{x,y,dir} \tag{5}$$

In (5), $\theta_{x,y,dir}$ represents the probability that a packet may get blocked at the head of the buffer due to the contention for the same output channel; $\omega_{x,y,dir}$ represent the mean waiting time that a message needs to wait in the event of blocking.

C. *Calculation of the mean waiting time*

To determine the mean waiting time, each router is treated as an M/M/1 queuing system where the arrival rate is $\lambda_{x,y,dir}$, the packet service time is $T_{x,y,dir}$. Resorting to the queuing theory[15], the mean waiting time becomes

$$\omega_{x,y,dir} = \frac{\lambda_{x,y,dir}}{\mu_{x,y,dir}(\mu_{x,y,dir} - \lambda_{x,y,dir})} = \frac{\lambda_{x,y,dir} \cdot T^2{}_{x,y,dir}}{1 - \lambda_{x,y,dir} \cdot T_{x,y,dir}} \tag{6}$$

D. *Calculation of the blocking probability*

For convenience, we use θ_{dir} to represent $\theta_{x,y,dir}$. First, we have to calculate the Forwarding Probability Matrix(FPM) $\boldsymbol{F}$

$$F = \begin{bmatrix} 0 & f_{NE} & f_{NS} & f_{NW} & f_{NL} \\ f_{EN} & 0 & f_{ES} & f_{EW} & f_{EL} \\ f_{SN} & f_{SE} & 0 & f_{SW} & f_{SL} \\ f_{WN} & f_{WE} & f_{WS} & 0 & f_{WL} \\ f_{LN} & f_{LE} & f_{LS} & f_{LW} & 0 \end{bmatrix}, \text{where} \quad f_{ij} = \frac{\lambda_{i \to j}}{\lambda_i} \tag{7}$$

In (7), f_{ij} is the probability that a packet arrives at *dir i* and leaves the router through *dir j*. Every elements of the forwarding probability matrix $\boldsymbol{F}$ can be calculated using Equation (3).

Now let us calculate the blocking probability and use North direction as an example.

$$\theta_N = \sum_{\forall Dir} f_{NDir} \cdot \theta_{NDir} \tag{8}$$

In (8), f_{NDir} is the forwarding probability of $N \to Dir$, while θ_{NDir} is the probability that a packet forwarded from *N* to *Dir* may get blocked.

$$\theta_{NDir} = f_{NDir} \cdot \sum_{\forall Dir', Dir' \neq N} f_{Dir'Dir} \tag{9}$$

Combining Eq.(7), Eq.(8) and Eq.(9) together, $\theta_{x,y,dir}$ can be calculated. Combining $\theta_{x,y,dir}$, Eq.(6), Eq.(5) and Eq.(4), we can finally build a nonlinear equation about $T_{x,y,dir}$, this equation can be solved to determine $T_{x,y,dir}$. Combining $T_{x,y,dir}$, Eq.(1) and Eq.(2), we can calculate $b_{x,y,dir}$ and detect the system performance bottleneck.

E. *The buffer allocation algorithm*

In this paper, we propose a greedy algorithm to solve this problem based on the aforementioned analytical model. The flow of the algorithm is shown in Fig. 2.

1) Given the system parameters(such as T_H, M, size of Mesh and total buffer budget) and traffic parameters(such as $a_{x,y}$ and $d_{(x,y)(x',y')}$), the algorithm (written in C++) can calculate $\lambda_{x,y,dir}$, $\boldsymbol{F}$ and $\theta_{x,y,dir}$ respectively using Eq.(3), Eq.(7) and Eq.(8) and build a nonlinear equation about $T_{x,y,dir}$.

2) The *Matlab Math Library* is used to solve the given equation(more specifically, the `roots` utility from *Matlab* is used as the nonlinear equation solver); at the same time the algorithm also generates the initial buffer configuration which assigns the buffer in all the used channels ($\lambda_{x,y,dir} \neq 0$) to be one flit large.

3) The algorithm determine $b_{x,y,dir}$ for each input channel using Eq.(1) and select the channel with the largest $b_{x,y,dir}$ as the system performance bottleneck.

4) The buffer size of the bottleneck channel is incremented by one flit, while the system free buffer is decremented by one flit.

5) Repeat (3)~(4), until system free buffer is 0.

6) Output the final buffer allocation results.

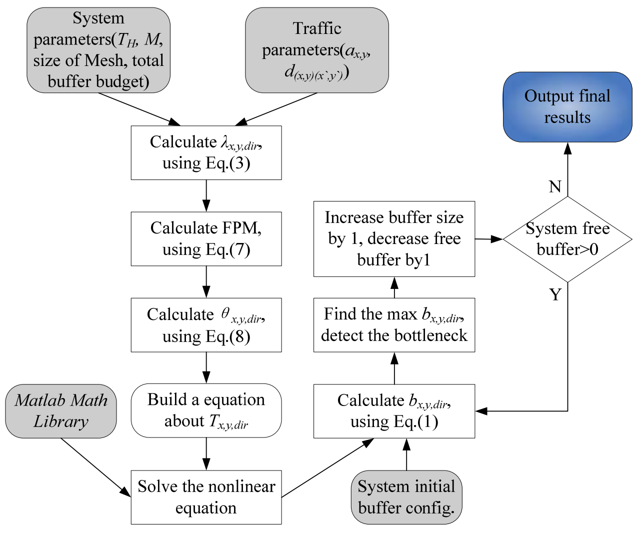

System parameters(T_H, M, size of Mesh, total buffer budget)
Traffic parameters($a_{x,y}$, $d_{(x,y)(x',y')}$)
Output final results
Calculate $\lambda_{x,y,dir}$, using Eq.(3)
Calculate FPM, using Eq.(7)
Increase buffer size by 1, decrease free buffer by1
System free buffer>0
N
Y
Calculate $\theta_{x,y,dir}$, using Eq.(8)
Find the max $b_{x,y,dir}$, detect the bottleneck
Matlab Math Library
Build a equation about $T_{x,y,dir}$
Calculate $b_{x,y,dir}$, using Eq.(1)
Solve the nonlinear equation
System initial buffer config.

Figure 2. The buffer allocation algorithm flow

IV. EXPERIMENT RESULT

To validate the proposed algorithm, we build a wormhole routing NoC simulation platform with OMNet++[14]. OMNet++ is a public source, generic and flexible simulation environment. It allows a fast and high-level simulation environment for NoC exploration.

Some parameters used in simulation are: $M = 16$, a data packet is divided into 16 flits; $T_H = 2$, the router needs 2 clock cycle to make routing decision for the header flit; the size of Mesh is 4, NoC is 4×4 2D Mesh; system total buffers are 240 flits, which means every used channel is assigned with 5 flits large buffer. For simplicity, every node has the same packet injection rate.

In the experiments, we applied our algorithm to applications with two typical traffic models: uniform random traffic model and hotspot traffic model. Under the uniform traffic pattern, a PE sends a packet to any other node with equal probability (this probability is 1/15 ≈ 0.0666). Under hot spot traffic pattern, one or more nodes are chosen as hot spots which receive an extra proportion of traffic (in this paper, the probability is 0.2) in addition to the regular uniform traffic. The efficiency of the algorithm is evaluated through latency-throughput curves. The average packet delay is defined as the mean amount of time from the generation of a packet until the last data flit reaches the local PE at the destination node.

Three hotspot traffic patterns used in this evaluation are hotspot1, hotspot2 and hotspot3. Traffic pattern hotspot1 have only one hot spot, while hotspot2 and hotspot3 are the hot spot traffic patterns which have two and three hot spots respectively. In each simulation experiment, a total number of about 80000 packets are delivered to their destinations. To avoid the distortions due to start-up conditions, the first 10000 packets are ignored.

In simulation results, NoCs with uniformly allocated FIFO buffers are denoted by UNOC, and systems that customized by our buffer allocation algorithm are denoted by CNOC. Their average packet latency should be compared.

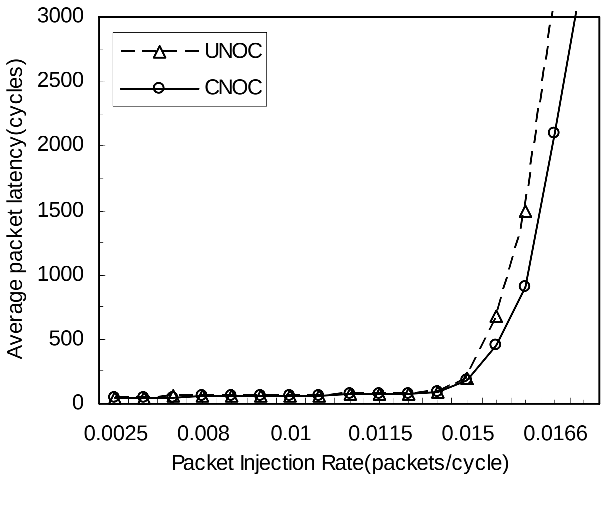


Figure 3. Performance under uniform traffic

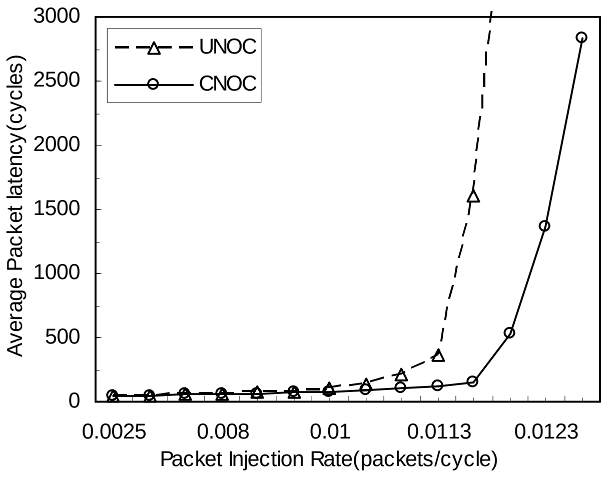


Figure 4. Performance under hotspot1 traffic

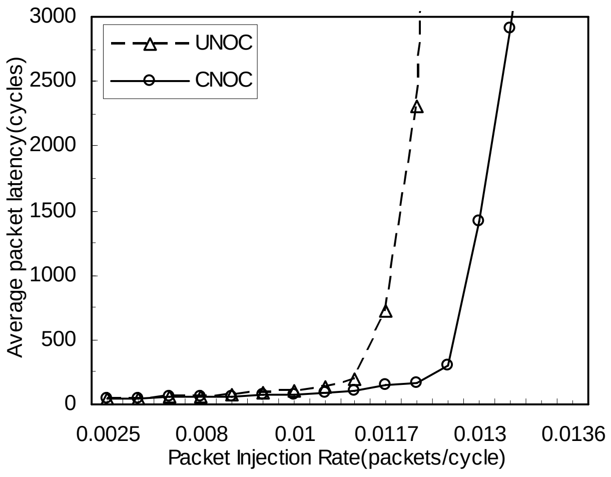


Figure 5. Performance under hotspot2 traffic

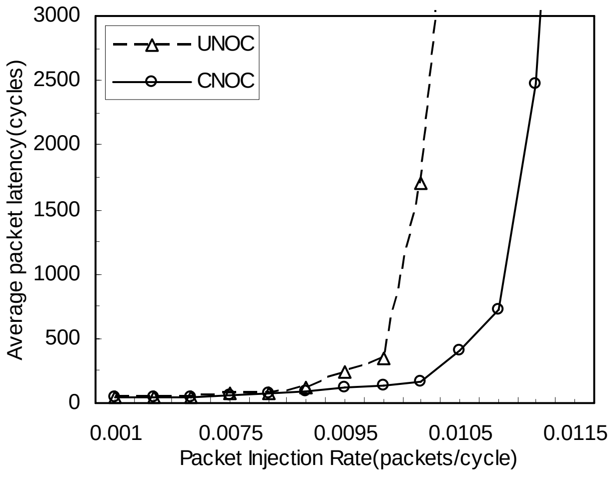


Figure 6. Performance under hotspot3 traffic

Fig. (3) shows the performance of UNOC and CNOC under the uniform traffic. The X-axis represents the packet injection rate per node, and the Y-axis represents the average packet latency of the NoC. In Fig. (3), we can see that, when packet injection rate is low(< 0.015 packets/cycle) their performance almost the same, but CNOC is better than UNOC. As the packet injection rate increases, network congestion happens in both UNOC and CNOC, and the packet latency rises dramatically, but the time that congestion happens in CNOC is later than UNOC. CNOC performs significantly better than UNOC when the packet injection rate is the same. This is because after customized by our buffer allocation algorithm, CNOC is assigned more buffering resources at performance bottleneck channels than UNOC, this customization makes the network average packet latency of CNOC is lower than UNOC.

Fig. (4) shows the performance of UNOC and CNOC under the hotspot1 traffic. Traffic pattern hotspot1 have only one hot spot, which is located at *Node(0,1)*. When packet injection rate is low(<0.015packets/cycle) their performance almost the same, but CNOC is better than UNOC. As the packet injection rate increases, congestion happens in UNOC at 0.0113 packets/cycle, while happens in CNOC at 0.012 packets/cycle. At any packet injection rate, the CNOC performs better than the UNOC.

Fig. (5) and Fig. (6) show the performance of UNOC and CNOC under the hotspot2 and hotspot3 traffic. Traffic pattern hotspot2 have two hot spots, which are located at *Node(2,1)* and *Node(0,2)*. Traffic pattern hotspot3 have three hot spots, which are located at *Node(1,1)*, *Node(2,2)* and *Node(1,3)*. The CNOC also performs better than the UNOC under these two traffic patterns, the results confirm the system behavior and conclusions discussed for uniform and hotspot1 traffic.

In all cases, CNOC performs much better than UNOC, which validate our buffer allocation algorithm. Comparing Fig. (6) with Fig. (3), it is clear that the buffer allocation is much more beneficial for hotspot3 traffic. The reason is compared to the uniform traffic pattern, the hotspot3 traffic pattern is more unbalanced, which makes the results of buffer allocation much better.

V. Conclusion

In order to minimize the implementation cost and maximum system performance, it's necessary to carefully allocate buffering resources to all used channel of NoC. A buffer allocation greedy algorithm for wormhole routing NoC is proposed in this paper, which can automatically assigns the buffer depth for each input channel, in different routers, according to the traffic characteristics of the target application. The simulation results show that the NoC customized by our algorithm has lower average packet latency than those whose buffering resources are uniformly allocated. Extending this research to NoCs that support adaptive routing and virtual channels will be our future work.

References

[1] L. Benini and G. D. Micheli, "Networks on chips: a new soc paradigm", Computer, Vol.35, pp.70-78, 2002.

[2] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks" in Proc. Design Automation Conference, Las Vegas, USA, pp.684-689, June 18-22, 2001.

[3] S. Kumar, A. Jantsch, J. P. Soininen, et al, "A network on chip architecture and design methodology" in Proc. IEEE Symposium on VLSI, Pittsburgh, USA, pp.117-124, Apri. 25-26, 2002.

[4] T. Bjerregaard and S. Mahadevan, "A survey of research and pratices of network-on-chip", ACM Computing Surveys, Vol.38, pp.71-121, 2006

[5] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques in direct networks", IEEE Tran. on Computers, Vol.26, pp.62-76, 1993.

[6] J. Hu, U. Y. Ogras and R. Marculescu, "System-level buffer allocation for application-specific networks-on-chip router design", IEEE Tran. on Computer-Aided Design Of Integrated Circuits And Systems, Vol.25, pp.2919-2933, 2006.

[7] U. Y. Ogras and R. Marculescu, "Analytical router modeling for networks-on-chip performance analysis" in Proc. Design, Automation and Test in Europe, Nice, France, pp.1-6, 2007

[8] N. Alzeidi, M. Ould-Khaoua, L. M. Mackenzie, et al, "Performance analysis of adaptively-routed wormhole-switched networks with finite buffers" in Proc. IEEE International Conference on Comunication, Glasgow, Scotland, pp.38-43, June 24-28, 2007.

[9] R. Dobkin, R. Ginosar and I. Cidon, "QNoC asynchronous router with dynamic virtual channel allocation" in Proc. First International Symposium on Networks-on-Chip, New Jersey, USA, pp.218-218, May. 7-9, 2007.

[10] Y. Tamir and G. L. Frazier, "Dynamically-allocated multi-queue buffers for VLSI communication switches", IEEE Tran on Computers, Vol.41, pp.725-737, 1992.

[11] G. M. Chiu, "The odd-even turn model for adaptive routing", IEEE Tran. on Parallel and Distributed Systems, Vol.11, pp.729-738, 2000.

[12] W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks", IEEE Tran. on Computers, Vol.39, ppl775-785, 1990.

[13] V. S. Adve and M. K. Vernon, "Performance analysis of mesh interconnection networks with deterministic routing", IEEE Tran. on Parallel and Distributed Systems, Vol.5, pp.225-246, 1994.

[14] OMNet++ discrete event simulation system user manual, http://www.omnetpp.org/, 2005.

[15] F. Hillier and G. Lieberman, Introduction to operations research. NY: McGraw-Hill Companies, 2002.

Session III

Prospective Architectural Proposals

Session Chair: Shashi Kumar*, Jönköping University, Sweden*

Scalable CMOS-compatible photonic routing topologies for versatile networks on chip

Alberto Scandurra
On Chip Communication Systems
STMicroelectronics
Catania, Italy
alberto.scandurra@st.com

Ian O'Connor
Lyon Institute of Nanotechnology
Ecole Centrale de Lyon
Lyon, France
Ian.Oconnor@ec-lyon.fr

***Abstract*— Optical network on chip (ONoC) architectures are emerging as potential contenders to solve both physical (routing, wire congestion) and performance (bandwidth, latency) issues in future computing architectures. In this work, we present a scalable and fully connected ONoC topology for multiple-core and heterogeneous SoCs. We show that it is possible, through careful design of network interfaces, to use the ONoC directly with existing protocols, while still exploiting specific optical properties and improving overall performance metrics, most notably that of congestion.**

I. INTRODUCTION

The shift to very high performance distributed Multi-Processor Systems-on-Chip (MPSoC) as mainstream computing devices is the recognized route to address, in particular, power issues by reducing individual processor frequency while retaining the same overall computing power. This rationale answers the need for flexible and scalable computing platforms capable of (i) achieving future required application performance in terms of resolution (audio, video and computing) and CPU power / total MIPS (real-time encoding-decoding, data encryption-decryption), and (ii) of working with multiple standards and with constrained power, which are both particularly important for mobile applications.

However, the move to such architectures requires organized high-speed communication between processors and therefore has an impact on the interconnect structure. It clearly relies upon the existence of an extremely fast and flexible interconnect architecture, to such a point that the management of communication between processors will become key to successful development. Aggregated on-chip data transfer rates in MPSoC, such as the IBM Cell processor [1], is critical and is expected to reach over 100Tb/s in the coming decade. As such, interconnects will play a significant role for MPSoC design in order to support these high data rates.

At the architectural level, networks on chip (NoC) overcome the limitations of bus-based platforms by providing each IP block, interfaced towards the network, with one or more reconfigurable channels of high-speed communication. NoC architectures are based on multiple data links interconnected by routers implementing packet switching for resource multiplexing. At the physical communication level, it is increasingly recognized that electrical interconnect will be highly inefficient in NoCs due to increasing power and silicon real estate concerns. One of the main replacement technologies currently under development consists of using integrated optical interconnect. Besides a huge data rate, optical interconnects also allow for additional flexibility through the use of wavelength division multiplexing. Exploring this aspect is necessary since it is not clear that a *direct* (single-wavelength) replacement of electrical links between switchboxes in a NoC topology by optical interconnect will achieve a significant performance gain, since this would require conversion between optical and electrical domains *at each switchbox*. Instead, through a shift in the routing paradigm (where the address of the target is not contained in the data packet but rather in the wavelength of the optical signal), it is possible to exploit this additional flexibility to design more intelligent interconnect systems, such as passive, wavelength-reconfigurable optical networks on chip (ONoC).

In section II we introduce the limit of classical electrical interconnect, and the need for an alternative solution. In section III an overview of a current NoC solution, the one developed by STMicroelectronics, is presented. Section IV details the architecture and principle of operation of the generic optical network on chip structure. Finally in section V, we cover the main communication scenarios for ONoCs.

It's important to point out that the focus of this paper is mainly on the topology of optical NoC, relying on the assumption that technology and design techniques allow to have an effective implementation of the physical layer, i.e. emitters, detectors and transport.

II. LIMIT OF ELECTRICAL INTERCONNECT

As design rules drop below 90 nm, a variety of challenges emerge such as RC delay, electromigration resistance, and heat dissipation exacerbated by increased chip power. The use of copper and thin barrier layers solves resistivity and electromigration problems but not for long due to electron scattering issues' increasing the apparent resistivity. Moreover, reliability issue with respect to an efficient diffusion barrier is a concern. Low k dielectrics allowing capacitance reduction have low thermal conductivity and hence poor heat dissipation

[1] This work was partially funded by FP7-ICT-2007-1-216405 "WADIMOS"

capability. Integration of copper and low k dielectrics is intensively studied worldwide [2].

Optical interconnects seem to be an alternative solution to overcome the issue of speed and power, providing much greater bandwidth, lower power consumption, decreased interconnect delays, resistance to electromagnetic interference and reduced signal crosstalk.

Photonic materials where light can be generated, guided, modulated, amplified and detected need to be integrated with standard CMOS integrated circuits in order to mix the information processing capability of electronics with the information transmission capability of photonics, providing a significant performance breakthrough within a cost effective engineering.

III. Network on Chip solution overview

The current ST NoC solution is based on a Network on Chip architecture called VSTNoC (Versatile STNoC), and evolves from the STBus approach [3]. It is in fact an interconnect system which has the same structure and functionality as the STBus, but uses a NoC-based protocol with appropriate interfaces and links. This approach enables higher performances and dramatic reductions in the number of pins and wires of the interconnect system, giving benefits in terms of area and facilitating rapid prototyping with FPGAs.

An STBus interconnect is composed of a set of building-blocks (nodes, converters and buffers) that can be cleanly assembled together in order to build almost any kind of architecture, from the simplest to the most complex one.

Figure 1. shows an interconnect built up with the VSTNoC, where network interfaces, nodes and buffers are used. In this figure we can see the uniformity in terms of both protocol (type) and bus size of the network with respect to the STBus; in fact all the required conversions are performed by the network interfaces where required, in order to adapt protocol, bus size and operation frequency to those of the network.

The VSTNoC solution belongs to the topology-dependent family. This means that, depending on the system topology (i.e. the number and type of initiators and targets of the system), the network topology can have different structures.

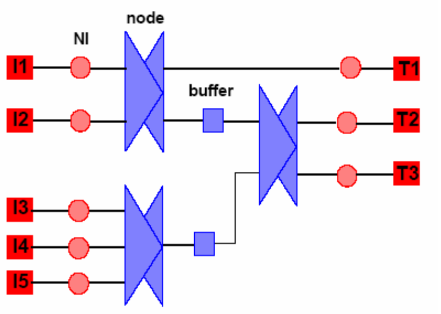


Figure 1. VSTNoC interconnect example

A. VSTNoC Protocol

The main features of the VSTNoC protocol are:

- a parametric header structure, the first field (IP_prot) of which identifies the protocol of the IP generating the traffic. According to the value of this field, the subsequent fields can differ in both meaning and size, depending on the IP native protocol;
- a NoC interface signal (aux/r_aux) carrying information about boundaries between possible elements characterizing different possible hierarchy levels of the IP native protocol (i.e. packet, chunk and message in STBus context, packet and burst in AMBA context);
- a flit identifier (flit_id) carrying information about the start and the end of a NoC transaction (a NoC transaction is a collection of NoC packets), determining the transaction or arbitration granularity (AG);
- an optional field in the response path carrying information about transaction status (r_flit_status), indicating whether errors have occurred, and which flits are affected.

B. VSTNoC Transactions

The VSTNoC transaction consists of the transmission of information from a traffic source to a destination according to a format that closely follows the one of usual network packets. It is the highest level transmission entity, marked by a start and an end, and can be chosen equivalent to an STBus message, chunk or even a single packet. The VSTNoC transaction is an atomic element, i.e. *it is not interruptible*.

A VSTNoC transaction is composed of VSTNoC packets, consisting of a header and a payload, the presence of which depends on specific conditions.

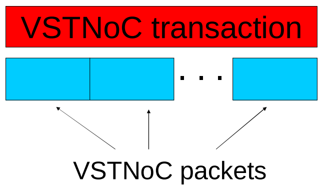


Figure 2. VSTNoC transaction structure

From a physical point of view, packets are split into basic units called flits (FLow control unITs). These represent the data link layer elements transmitted within a clock cycle in the case of synchronous transmission, or as asynchronous entities, the size of which is generally greater than the phyt (physical layer element) size. However it has been chosen equal to the phyt size in the VSTNoC context (i.e. in this work, flits and phyts are equivalent).

The flit is chosen to be sufficiently wide so as to be able to contain both data and byteenables transmitted over one single STBus cell in the request path. The following tables show the possible flit sizes that can be specified in an STNoC system for both the request and the response path, together with the different fields within a payload flit.

TABLE I. VSTNoC INTERCONNECT EXAMPLE

Flit size (bits)	Request flit field		Response flit field	
	be	data	be not used	r_data
36	<35:32>	<31:0>	-	<31:0>
72	<71:64>	<63:0>	-	<63:0>

The response flits are smaller than the request flits since in the response path there is no need to transport the byteenables signal, so fewer wires are required for the response interfaces.

C. *VSTNoC Building Blocks*

The VSTNoC communication system is based on the following building-blocks:

- Initiator Network Interface, responsible for IP to NoC traffic conversion and write posting response generation
- Target Network Interface, responsible for NoC to IP traffic conversion and internal errors (security and power down) management
- Node, responsible for buffering, arbitration, routing and wrong address errors
- Programming Module, allowing STNoC registers configuration
- Generic Converter, allowing to connect different NoC domains (with different flit size and/or frequency) and/or breaking long paths

In the next section, we will cover the description of the Optical Network On-Chip (ONoC), based on the VSTNoC.

IV. SCALABLE ONOC ARCHITECTURE

In an ONoC communication system, information is transmitted in the form of light, in opposition to the situation in classical electrical NoCs where the information is transmitted in the form of electrical charge (voltage levels on capacitors and currents for switching between voltage levels).

Communication relies on the ISO-OSI protocol stack, and can be seen as very close to the VSTNoC architecture, where the physical layer is replaced with a completely new one, exploiting optoelectronics in order to transmit information in form of light. The aim of this work is to demonstrate effective compatibility of the ONoC at the physical layer with the VSTNoC protocol.

The ONoC architecture consists of five main sets of building-blocks, as shown in Figure 3:

- **Initiator Network Interface** (**INI**): responsible for the conversion of the traffic generated by an initiator into a form suitable to be transmitted in form of light over the ONoC;
- **Transmitter**: responsible for the actual conversion of information from the electrical form into optical form, by means of information encoding for minimizing the power consumption by keeping the light emitter turned off as much as possible, serialization, emitter selection, emitter driving;
- **λ-Router (scalable passive integrated photonic routing structure)**: responsible for the actual propagation of optical information streams from sources to destinations;
- **Receiver**: responsible for the conversion of information from the optical form into electrical form, by means of photocurrent to voltage conversion, level adjustment, de-serialization, information decoding (for power consumption issue) and arbitration in case of multiple access from different traffic sources;
- **Target Network Interface** (**TNI**): responsible for the conversion of the traffic generated by the ONoC receiver into a form suitable to be received by the target.

Notice that INI and TNI are modules belonging to the electrical domain, while transmitter, receiver and l-router belong to the optical domain. Because of the serialization, the flit size in the electrical domain does not affect the optical network, but just the required storage at buffers in the electrical domain.

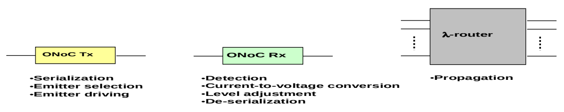


Figure 3. ONoC building blocks in optical domain

Such building-blocks can be assembled together to build proper on-chip communication architectures.

A. *Principle of operation*

An N×N ONoC, from a functional point of view, has the same behavior as an electrical N-port NoC: each initiator port (among N) can communicate simultaneously with one (or more, and possibly any number up to N) of N target ports. In this work, the quantity N represents the number of IP blocks to be connected through the communication structure; hence each IP block sends data through an initiator port and receives data through a target port. As previously mentioned, the ONoC is composed of a set of N transmitters and N receivers (one for each initiator port and target port respectively), and a scalable passive integrated photonic routing structure (λ-router). In this section, we will cover the principle of operation of this architecture and present results of physical and architectural evaluations from previous work.

Figure 4. shows an example of an 8×8 ONoC architecture.

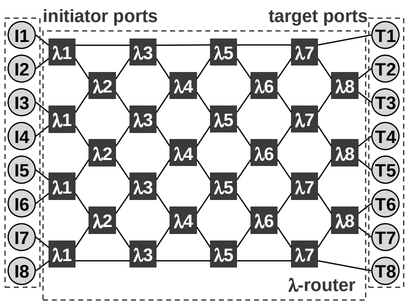


Figure 4. Full 8x8 ONoC topology schematic

In this representation, each initiator port I_i ($\forall i \in \{1,2,\ldots,8\}$) consists of a network interface (NI) and transmitter; and each target port T_j ($\forall j \in \{1,2,\ldots,8\}$) consists of a receiver and NI. Data is sent through the passive λ-router optically from each initiator to one or more targets by selecting a specific wavelength (for each initiator-target pair); in fact, only one physical path associated with a single wavelength exists between I_i and T_j. At any one time, a maximum of 8 (N) connections can exist in the network if each transmitter is equipped with a single, tunable-wavelength source; and a maximum of 64 (N^2) connections can exist in the network if each transmitter is equipped with N single- wavelength sources.

In the figure, each box containing λ_x represents a passive photonic component called an "add-drop filter" which can realize the key functionality of selecting and redirecting a signal based on its wavelength. There are many ways of realizing a photonic add-drop filter. In our work, we consider the use of passive microdisk resonators as shown in Figure 5. [3], for which the overall footprint can be considered to be approximately $10\times10\mu m^2$. Resonance in the individual microdisks occurs whenever the wavelengths of the optical signal carried by the neighboring waveguide corresponds to an integer number of lobes around the circumference of the microdisk, i.e. when the energy is distributed in the disk in *whispering gallery* modes. Because of this, the resonant wavelengths of a microdisk depend, for a given technology (and material parameters), on the radius of the microdisk.

As shown in the figure, the switching direction depends on the input wavelength λ and its relation to the resonant wavelength of the add-drop filter:

- when $\lambda=\lambda_n$ (within a given tolerance range depending on the quality factor of the microdisk) the signal will couple into the microdisk and then couple out into the waveguide in the same plane as the input. This is the *straight*, or *bar*, state.
- when $\lambda\neq\lambda_n$ the signal will propagate along the same waveguide and outputs in a different plane to the input according to the geometry of the waveguide. This is the *diagonal*, or *cross*, state.

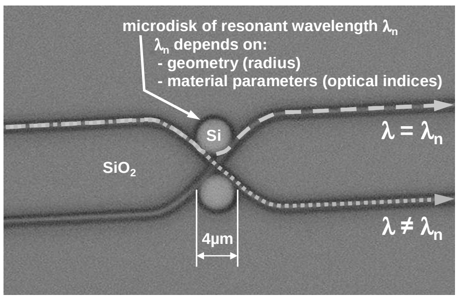


Figure 5. Si/SiO_2 microdisk-resonator based add-drop filter

When the WDM[2] technique is used, i.e. when multiple signals of various wavelengths are injected at the input (which is usually the case to increase the global throughput of the network), a cumulative state occurs, where individual signals simultaneously obey the routing characteristics of the add-drop filter according to their individual wavelengths. Because of this property and the fact that the four add-drop ports can be used simultaneously, a contention-free network can be built.

The overall passive λ-router network consists of N stages of alternately N/2 and (N/2)–1 add-drop filters (or, more generically, routing elements). Using microdisk resonators, the overall area required for the 8×8 passive network is around $3000\mu m^2$. The path followed by the optical signal in the overall network shown in Figure 4. depends only on the wavelength and can be obtained by equation (1). As an example, if the block at initiator port I_3 is to communicate with the block at target port T_5, then I_3 must send data through the λ-router with wavelength λ_1. It is thus clear that each IP block can "reconfigure" its communication paths by using different wavelengths.

The matrix shown in equation (1) displays two interesting properties. Firstly, it is symmetrical around both diagonals. This means that the set of communication properties of the top half of the network is the flipped mirror image of that of the bottom half of the network; and that the return path for communication is exactly the same as the transmission path. The second noteworthy property is the existence of non-resonant wavelengths in certain communication paths (shown in bold in the matrix). While specific wavelengths have been assigned in the matrix to these communication paths, *any* wavelength (other than the wavelengths used by the other communication paths) can be used. This is the case since these communication paths do not actually pass through a routing element corresponding to the assigned wavelength at all – they cross the (N/2)-1 routing stages at the top or at the bottom of the network and thus only pass through a waveguide, rather than a resonant routing element. In the full ONoC, the unused wavelengths are assigned to these communication paths in order to exploit the resources – however this property can also be exploited to reduce the number of wavelengths used [5] [6].

$$\begin{bmatrix} T_1\\T_2\\T_3\\T_4\\T_5\\T_6\\T_7\\T_8 \end{bmatrix} = \begin{bmatrix} \lambda_4 & \lambda_5 & \lambda_3 & \lambda_6 & \lambda_2 & \lambda_7 & \lambda_1 & \boldsymbol{\lambda_8} \\ \lambda_5 & \lambda_6 & \lambda_4 & \lambda_7 & \lambda_3 & \lambda_8 & \boldsymbol{\lambda_2} & \lambda_1 \\ \lambda_3 & \lambda_4 & \lambda_2 & \lambda_5 & \lambda_1 & \boldsymbol{\lambda_6} & \lambda_8 & \lambda_7 \\ \lambda_6 & \lambda_7 & \lambda_5 & \lambda_8 & \boldsymbol{\lambda_4} & \lambda_1 & \lambda_3 & \lambda_2 \\ \lambda_2 & \lambda_3 & \lambda_1 & \boldsymbol{\lambda_4} & \lambda_8 & \lambda_5 & \lambda_7 & \lambda_6 \\ \lambda_7 & \lambda_8 & \boldsymbol{\lambda_6} & \lambda_1 & \lambda_5 & \lambda_2 & \lambda_4 & \lambda_3 \\ \lambda_1 & \boldsymbol{\lambda_2} & \lambda_8 & \lambda_3 & \lambda_7 & \lambda_4 & \lambda_6 & \lambda_5 \\ \boldsymbol{\lambda_8} & \lambda_1 & \lambda_7 & \lambda_2 & \lambda_6 & \lambda_3 & \lambda_5 & \lambda_4 \end{bmatrix} \begin{bmatrix} I_1\\I_2\\I_3\\I_4\\I_5\\I_6\\I_7\\I_8 \end{bmatrix} \qquad (1)$$

B. Evaluated performance metrics

In prior work [6], a 4×4 passive λ-router was fabricated and measurements show that its operation agrees with the theory. Resonant wavelengths were measured between 1547-1583nm for Si/SiO_2 microdisks of radii from 1.0-2.5µm. The minimum

[2] Wavelength Division Multiplexing

free spectral range (FSR[3]) was measured to be 50nm, and quality factors around 500-800.

In parallel work, the design of a 16×16 ONoC virtual prototype was carried out at various abstraction levels using a top-down approach [8] from architecture to physical design, enabling an accurate estimation of various performance metrics. The source and detector characteristics were extracted from III-V device data, and transistor-level interface circuits sized with a 0.13 μm CMOS technology. In this context, the ONoC can achieve a data rate of up to 3.2Gb/s per port with a latency of 420ps and power consumption of 10mW per unidirectional link. The ONoC data rate is in fact limited by the interface circuits, mainly at the receiver. The SERDES circuits contribute greatly to power consumption at these frequencies.

More recently in [9], the impact of the low latency and absence of contention in the ONoC interconnect architecture was assessed for an 8-processor SoC running an MPEG-4 algorithm. When comparing a 100MHz ONoC against 200MHz STBus [10] and 2- and 5-CCL[4] crossbars, the ONoC demonstrated speedup factors of between 1.5 and 3.2, i.e. better performance, in terms of processing time, than any traditional electrical interconnect, even at half the operational frequency.

V. ONoC ARCHITECTURE COMMUNICATION SCENARIOS

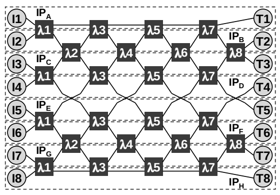
In this section, we cover the uses of ONoC in actual communication scenarios.

A. *Communication scenarios*

The optical waveguides within the ONoC are bidirectional. However, two-way communication between 2N IP blocks over a single ONoC is not feasible since this would require optical detectors and sources with identical wavelength selectivity to lie on the same waveguide with no interaction – this is clearly impossible. Additionally in this configuration there can be no communication among IP blocks which have been assigned ports situated (physically) on the same side of the passive routing network. In fact there are two scenarios for the use of the N×N ONoC, both using the ONoC for communication in a single direction only:

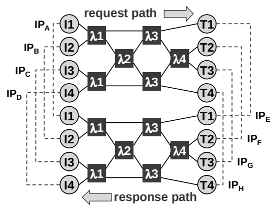
- in the first scenario, shown in Figure 6. (a) for 8 IP blocks, we consider that each IP block is assigned a pair of initiator/target ports. This leads to total connectivity between all N IP blocks, and to the non-use of wavelengths corresponding to communication paths I_i-T_j when i=j.

- in the second scenario, shown in Figure 6. (b) for 8 IP blocks, we consider that two identical (N/2)×(N/2) ONoCs are used for request/response type communications between two sets of N/2 IP blocks. In this case, no communication is possible between IP blocks in the same set, but this scenario does lead to reduced requirements on the overall number of wavelengths and routing elements.



$$\begin{bmatrix} IP_A \\ IP_B \\ IP_C \\ IP_D \\ IP_E \\ IP_F \\ IP_G \\ IP_H \end{bmatrix} = \begin{bmatrix} 0 & \lambda_5 & \lambda_3 & \lambda_6 & \lambda_2 & \lambda_7 & \lambda_1 & \lambda_8 \\ \lambda_5 & 0 & \lambda_4 & \lambda_7 & \lambda_3 & \lambda_8 & \lambda_2 & \lambda_1 \\ \lambda_3 & \lambda_4 & 0 & \lambda_5 & \lambda_1 & \lambda_6 & \lambda_8 & \lambda_7 \\ \lambda_6 & \lambda_7 & \lambda_5 & 0 & \lambda_4 & \lambda_1 & \lambda_3 & \lambda_2 \\ \lambda_2 & \lambda_3 & \lambda_1 & \lambda_4 & 0 & \lambda_5 & \lambda_7 & \lambda_6 \\ \lambda_7 & \lambda_8 & \lambda_6 & \lambda_1 & \lambda_5 & 0 & \lambda_4 & \lambda_3 \\ \lambda_1 & \lambda_2 & \lambda_8 & \lambda_3 & \lambda_7 & \lambda_4 & 0 & \lambda_5 \\ \lambda_8 & \lambda_1 & \lambda_7 & \lambda_2 & \lambda_6 & \lambda_3 & \lambda_5 & 0 \end{bmatrix} \begin{bmatrix} IP_A \\ IP_B \\ IP_C \\ IP_D \\ IP_E \\ IP_F \\ IP_G \\ IP_H \end{bmatrix}$$

(a)



$$\begin{bmatrix} IP_A \\ IP_B \\ IP_C \\ IP_D \\ IP_E \\ IP_F \\ IP_G \\ IP_H \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & \lambda_2 & \lambda_3 & \lambda_1 & \lambda_4 \\ 0 & 0 & 0 & 0 & \lambda_3 & \lambda_4 & \lambda_2 & \lambda_1 \\ 0 & 0 & 0 & 0 & \lambda_1 & \lambda_2 & \lambda_4 & \lambda_3 \\ 0 & 0 & 0 & 0 & \lambda_4 & \lambda_1 & \lambda_3 & \lambda_2 \\ \lambda_2 & \lambda_3 & \lambda_1 & \lambda_4 & 0 & 0 & 0 & 0 \\ \lambda_3 & \lambda_4 & \lambda_2 & \lambda_1 & 0 & 0 & 0 & 0 \\ \lambda_1 & \lambda_2 & \lambda_4 & \lambda_3 & 0 & 0 & 0 & 0 \\ \lambda_4 & \lambda_1 & \lambda_3 & \lambda_2 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} IP_A \\ IP_B \\ IP_C \\ IP_D \\ IP_E \\ IP_F \\ IP_G \\ IP_H \end{bmatrix}$$

(b)

Figure 6. Communication scenarios and corresponding connectivity matrices for ONoC in 8-IP block scenarios (a) single 8×8 ONoC for total connectivity between 8 IP blocks (b) 2 4×4 ONoCs for request/response connectivity between 2 groups of 4 IP blocks

In Table II, a comparison is made between various performance metrics for each scenario. These represent extremes for (a) total connectivity and (b) balanced communication between groups of IP blocks of equal numbers. In practice, it is unlikely that the required system connectivity will fall into either of these scenarios. However, the total connectivity scenario represents the default or reference scenario, while the grouped connectivity scenario makes clear that if total connectivity is not required in the system, significant reductions in complexity can be achieved.

TABLE II. COMPARISON BETWEEN PERFORMANCE METRICS FOR TOTAL CONNECTIVITY AND GROUPED CONNECTIVITY ONOC SCENARIOS

	(a) Total connectivity	**(b) Grouped connectivity**
IP blocks	N	N
Connections	N(N-1)	$(N/2)^2$
Required wavelengths per IP block n_λ	N-1	N/2
Number of routing elements n_r	N(N-1)/2	N(N-1)/4

B. *Physical considerations*

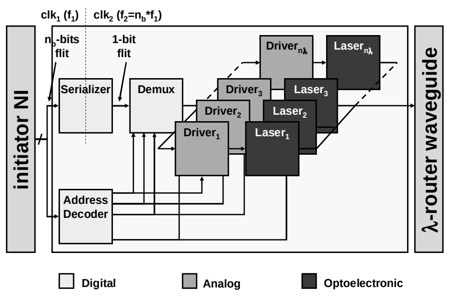
The comparisons mentioned in Table I are important for several reasons. Firstly, the number of routing elements n_r impacts directly on the overall size and complexity of the passive routing network. The size of the photonic communication layer is limited by the size of the CMOS chip. If several parallel λ-routers can fit into this area, then data rate could be increased (or power consumption reduced by running at a lower clock frequency).

Secondly, the required number of wavelengths n_λ per IP block will impact directly on the number of transmitters (and

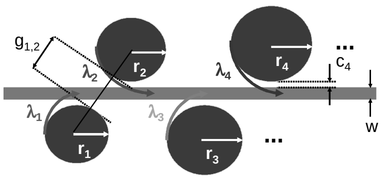
[3] FSR is defined as the difference between resonant wavelengths of a passive resonator. In the Si/SiO_2 microdisk resonators, FSR ≈ 50nm.

[4] Clock Cycle Latency

sources and wavelength multiplexers) and receivers (and wavelength demultiplexers and detectors) per IP block. The schematic of the transmitter structure and corresponding geometrical representation for the set of microdisk laser sources is shown in Figure 7. (a) and Figure 7. (b) respectively.



(a)



(b)

Figure 7. n$_\lambda$-laser source transmitter structure (a) schematic (b) corresponding geometrical representation for the set of microdisk laser sources

Since the laser source drivers are based on current modulation schemes, each source costs, in terms of static and dynamic power consumption, its bias current and modulation current respectively. As a consequence, the overall static and dynamic power consumption increases linearly with n$_\lambda$. In terms of the geometry and its impact on the size of the transmitter on the photonic layer, its area A$_t$ can be expressed as

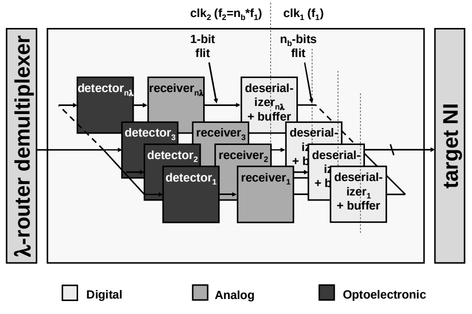
$$A_t = \left((n_\lambda - 1)\sqrt{(2\bar{r} + \bar{g})^2 - (w + 2(\bar{c} + \bar{r}))^2} + 2\bar{r} \right)(2(2\bar{r} + \bar{c}) + w) \quad (2)$$

where

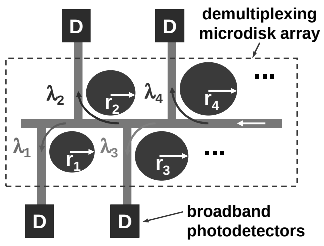
$$\bar{c} = \frac{\sum_{n=1}^{n_\lambda} c_n}{n_\lambda} \;;\; \bar{r} = \frac{\sum_{n=1}^{n_\lambda} r_n}{n_\lambda} \;;\; \bar{g} = \frac{\sum_{n=1}^{n_\lambda} g_{n,n+1}}{n_\lambda}$$

and c$_n$ represents the nominal source$_n$-waveguide distance (between 0.4-0.6µm), r$_n$ the nominal microdisk laser radius (between 1-10µm), g$_{n,n+1}$ the minimum source-source spacing (typically 3µm), and w the waveguide width (under 1µm).

At the target end, each IP block requires a separate receiver path for each wavelength received, in order to identify the origin of each incoming data flit and also in order to be able to buffer flits incoming simultaneously from different initiator ports. The schematic of the receiver structure and corresponding geometrical representation for the set of microdisk demultiplexers and broadband photodetectors is shown in Figure 8. (a) and Figure 8. (b) respectively.



(a)



(b)

Figure 8. n$_\lambda$-demultiplexing receiver structure (a) schematic (b) corresponding geometrical representation for the set of microdisk demultiplexing elements and broadband photodetectors

Finally, as shown in Figure 9. , since the maximum WDM window is approximately equivalent to the FSR of the microdisk resonators, a larger number of wavelengths will also lead to more stringent constraints on the selectivity (Q factor) of each resonator, and on the accuracy of the lithography techniques used to define the radius (and resonant wavelength) of each passive microdisk in the λ-router. With current process technology characteristics, a maximum of around 16 distinguishable and stable wavelengths can be achieved.

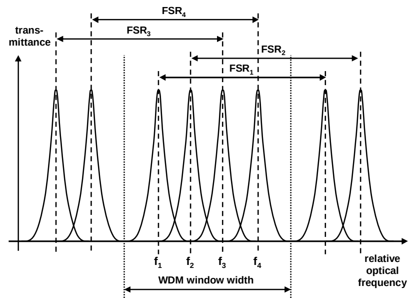


Figure 9. Relationship between microdisk resonator free spectral range and WDM window width

VI. Conclusions

In this work, we have described a scalable and fully connected N×N ONoC topology compatible with existing NoC paradigms. We have covered the main metrics that can be extracted from various communication scenarios for fully connected topologies, and in particular constraints on source wavelength accuracy and passive filter selectivity depending on the number of required wavelengths, and power and area issues depending on the number of active and passive devices.

REFERENCES

[1] M. Gschwind, "Chip multiprocessing and the Cell broadband engine," *ACM Int. Conf. Computing Frontiers*, Ischia, Italy, May 2006

[2] L. Pavesi, G. Guillot (Eds.), “Optical Interconnects”, Springer (2006)

[3] M. Coppola, C. Pistritto, R. Locatelli and A. Scandurra STNoC™: An Evolution Towards MPSoC Era”, NoC Workshop, march2006, DATE 2006

[4] A. Kazmierczak *et al.*, "Design, simulation, and characterization of a passive optical add-drop filter in silicon-on-insulator technology," *IEEE Photon. Technol. Lett.*, vol. 17, no. 7, pp. 1447-9, July 2005

[5] I. O'Connor *et al.*, "Reduction methods for adapting optical network on chip topologies to specific routing applications," *Proc. Design of Circuits and Integrated Systems (DCIS)*, November 12-14 2008, Grenoble, France

[6] L. Zhang *et al.*, "Generalized Wavelength Routed Optical Micronetwork in Network-on-chip", Proc. 18th Int. Conf. Parallel and Distributed Computing and Systems (IASTED), pp. 698-703, Dallas, TX, Nov. 2006

[7] A. Kazmierczak *et al.*, "Design and characterisation of optical networks on chip," *Proc. Eur. Conf. Integrated Optics (ECIO)*, Grenoble, France, April 2005

[8] M. Briere *et al.*, "Heterogeneous modelling of an Optical Network-on-Chip with SystemC," *Proc. Rapid System Prototyping Workshop (RSP)*, June 2005

[9] M. Brière *et al.*, "System Level Assessment of an Optical NoC in an MPSoC Platform," *Proc. Design Automation and Test in Europe (DATE)*, Nice, France, April 2007

[10] A. Bona, V. Zaccaria, and R. Zafalon, "System level power modeling and simulation of high-end industrial network-on-chip," *Proc. Design, Automation and Test in Europe (DATE) Conference and Exhibition*, Munich, Germany, February 2004

Move Logic Not Data : A Conceptual Presentation

Ahmed Hemani, Muhammad Ali Shami
School of ICT, Royal Institute of Technology, KTH
Stockholm, Sweden
hemani/shami@kth.se

***Abstract*—Memory and global interconnect dominate the cost, power and performance of Embedded System on Chip (SOC) architectures. We contend that most of the architectural innovations being pursued today do not directly address these challenges. Move Logic Not Data is a scalable architectural concept where the movement of data is minimal. The logic that transforms/creates data is instead brought to the data. This is implemented with the help of Networks on Chip (NOC) which allows us to create seamless portioning of memory and logic resources. Two additional innovations, further improve the efficacy of this fundamental innovation: Reduce the code size. Movement of code also costs in terms of energy and latency. We propose using ultra complex algorithmic size instructions in the form of reconfigurable logic. Conceptual arguments and architecture that would implement these innovations are presented backed by theoretical analysis of their impact. Research challenges are identified that would need to be overcome to implement the proposed architecture.**

I. INTRODUCTION

Nomadic products host a suite of applications, dominated by high performance wireless communication and multi-media algorithms. The performance requirement of each of these applications can be extreme - the PHY layer of the now mundane 802.11a/g standard is 5 GIPs [1] and encoding decoding a h.264 stream in the meager CIF standard would need the ARM's flagship ARM11 processor to tick at 1.6 GHz with no cache misses, which is quite theoretical as it is not possible to clock ARM11 processors at this speed even in the latest 65 nm technology node. Meeting such extreme performance demands, for not one but a suite of applications and to power them on battery, produce them for mass market and keep the engineering cost manageable is an extreme SOC engineering challenge.

In SOC designs embedded memory is increasing primarily because of rapid increase in the amount of data handled by communication and multimedia algorithms to achieve higher bandwidth or resolution/quality [2].

The potentially arbitrary communication among applications forces system architects to often adopt a shared memory model of communication. To satisfy the large storage need, the cost and process factors, these products almost always having a single large external SDRAM memory. Concurrent applications that need high bandwidth memory access to the external SDRAM creates a bottleneck and results in usage of expensive L1 and L2 caches to hide latency and this explains the secondary need for memory. The architecture efficacy gap between the energy and performance needs of applications - communication, multi-media and security - and what is afforded by technology scaling is increasing [3]. Not only do the SOC architects have to contend with vastly large amount of data, they also have to tackle moving this data among applications at high-speed, a particularly difficult task in view of the well known fact that while transistors become fast with technology scaling, the global interconnect is not scaling [4] [5] as shown in Table I.

TABLE I. ALU, MEMORY AND INTERCONNECT DELAYS [4]

Operation	Delay in 130 nm	Delay in 50 nm
32b ALU Operation	650 ps	250 ps
32b Register Read	325 ps	125 ps
Read 32b across chip RAM	780 ps	300 ps
Transfer 32b acros chip (10mm)	1400 ps	2300 ps
Transfer 32b acros chip (20mm)	2800 ps	4600 ps

II. RELATED WORK

To close this gap between performance and memory a range of techniques have been deployed from the simple measures like increasing the clock frequency, increasing the depth of pipelining to more sophisticated measures like Instruction level parallelism (ILP), thread level parallelism (TLP) have been tried and the returns are diminishing [6]. ILP is primarily an architectural technique directed at improving the computational efficiency, and imposes more stringent demands on memory efficiency.

A. MPSOC

The latest architectural trend is the move to multi-processing. Advanced architectures are exploring the possibility of using NOC together with Multi-Processors to alleviate the bus bottleneck. Like ILP, we contend that the move to MPSOC is well justified but the goal is to overcome computational bottleneck, it does not effectively deal with the memory and the global interconnect challenge, which we argue is the central challenge.

To drive home the point that MPSOC does not alleviate the memory and interconnect challenge and are in fact plagued by it, consider the example shown in Figure 1, a slightly modified schematic of a wireless multi-media platform test chip from NXP [7]. This platform is a state of the art MPSOC in 65nm and a flagship product of NXP. To meet the large combined storage need of these applications and to enable arbitrary data communication among them, a large external Low Power DDR (LPDDR) memory is instantiated to implement a shared memory model of communication. As illustrated in Figure 1 with black lines, all the application processors need to access the external LPDDR creating a huge bottleneck. To hide the latency, the processors have a sizeable L1 caches for instruction and data and potentially a system L2 cache. Other sub-systems have local buffers. Even a 1 GB/sec memory bus is barely able to sustain the worst case bandwidth requirement. Even with large, fast caches the processors - ARM1176 and Trimedia - typically operate effectively at one-third of their clocked frequency. In other words, if the ARM1176 is clocked at 400 MHz, the computational throughput is as if it was operating at 133 MHz with no cache misses. The inability to effectively handle large amount of data and large movement of data by this typical state-of-the-art architecture is the root cause of huge latencies, wastage of energy and silicon.

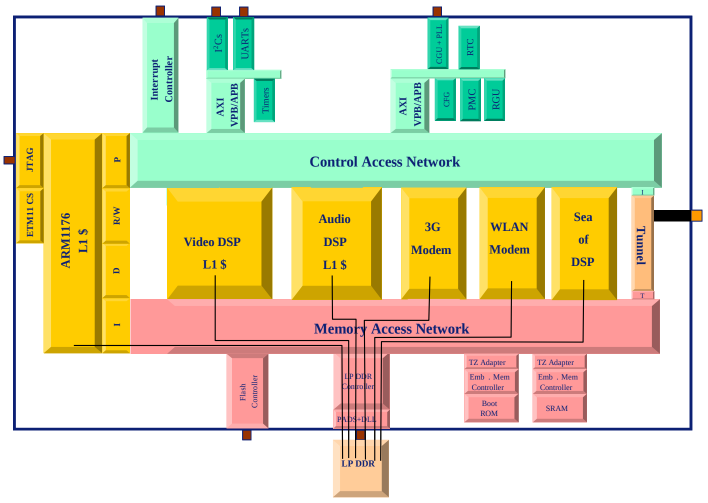


Figure 1. Memory Challenge In State of Art SOC

B. *Processor In Memory*

Memory is the central challenge has been recognized by a stalwart like Prof. David Patterson at Univ. of California Berkeley where a major project called Intelligent RAM (IRAM) has been launched. A vector media processor [8] for embedded systems is the first concrete outcome of this project. Solving the bottleneck to the external DRAM has motivated this work to incorporate on chip DRAM to get large bandwidth. While on chip DRAM in itself is not an innovation as it has been used by graphics chip designers in the past, the key contribution is to couple vector lanes to banks of DRAM via a fully connected inter-routing network in an architecture called VRAM. Recognizing that the fully connected network is overkill, a more optimized but less general version called CODE has also been developed. The focus of this project is to exploit the high on-chip memory bandwidth to fuel vector processor creating a complex on-chip communication network.

Flex RAM [9] is the name of an effort at Univ. of Illinois at Urbana Champaign that advocates Processor in Memory (PIM) approach to address the memory challenge. In this approach memory chips are replaced by 1 MB DRAM banks that includes a light processor. An on-chip interconnect connects 64 such memory bank processor together. Whereas IRAM targets embedded applications FlexRAM principally targets server applications. Flex RAM has not been commercialized mostly because the synchronization problem was not solved.

Imagine Stream Processor is another effort by Professor William Dally in university of Stanford to increase on-chip communication bandwidth in order to fuel many ALUs arranged in SIMD fashion [10]. In imagine processor, on chip memory called streaming memory and streaming register file is used to increase the communication bandwidth to 32Gbytes/s. Imagine processor utilizes the reference of locality principle and reduces the global bandwidth by having local buffers and stream register file. The intermediate results are stored in these local memories. Data movement between different kernels takes place through them as well instead of main memory, thus reducing global communication bandwidth requirement [11]. Once we have presented our architecture in detail in sections III and IV we will show the essential differences between imagine and MLND and show how this difference in moving logic vs moving data benefits both energy and performance.

III. Move Logic Not Data

The objective of this principle is to develop an architecture where the movement of data is minimal. The code/logic that transforms/creates data is instead brought to the data to transform it and/or create new data. Movement of code also costs in terms of energy and latency. We propose using ultra complex algorithmic size instructions in the form of reconfigurable logic. Traditionally, communication among applications/tasks has been achieved by either message passing or shared memory models. We propose a third alternative, a shared logic model.

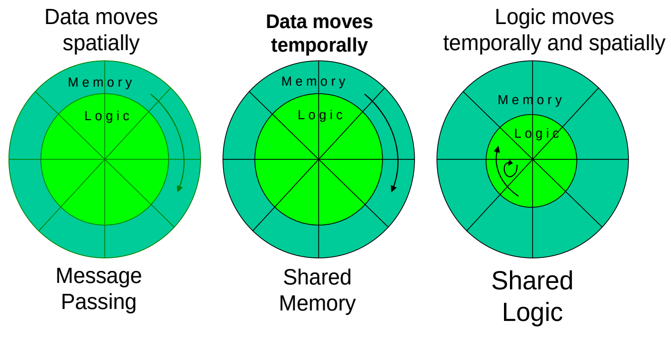


Figure 2. Visualizing the three models of communication.

Figure 2 provides an intuitive explanation of the differences between the three models. The inner circle represents logic and the outer circle represents the memory. Conceptually, the segments show the multi-processors hosting multiple applications and the alignment of segment shows a processor’s association with a memory partition. In the message model, the inner logic circle does not rotate, the outer memory circle does, implying that logic segment once it has transformed data, the data moves and gets associated with another logic segment. In the shared memory model, the segments in outer circle represents temporal windows into a shared memory, at any particular time only one temporal window opens and the data is fetched and stored in local buffers of the logic segments. Generally, more than one temporal window could be open representing more than one shared memory. In the model we propose, it is the logic circle that rotates, while the memory circle stays stationary. The logic circle is shown in relatively reduced size to underscore the fact that the code size is reduced. The reduction is both spatial - the code should take less space compared to the equivalent code in terms of assembly instructions and temporally - the code is changed less frequently as it represents ultra-large algorithmic size instruction. The logic also rotates in spatial and temporal sense. Spatial rotation implies that the same reconfigurable function can potentially move (its code) from one logic segment to another. Temporal rotation implies that different reconfigurable functions are sequentially loaded into the same logic segment.

IV. Quantitative Analysis

Consider an abstract chip shown in Figure 3(a), square in shape of side *a* units, dominated by memory with a small negligible area occupied by processor in the centre. Also assume that the memory is organized in *N* words. On an average these *N* words would travel approximately a*/2* distance. Now if we divide this chip into 4 equal parts and each part has its own processor in the middle. Assuming that the data in each partition stays within the partition and is only operated by the processor/logic in that partition, and then the average distance that these *N* words would need travel would be a/4. Generalizing, by dividing the memory into 'n' partitions, we reduce the average distance travelled by each of the *N* words by factor $n^{1/2}$ compared to the original un-partitioned case. Latency is dominated by interconnect and memory access. The delay in interconnect is directly scaled down by $n^{1/2}$ and also its switching capacitance. Since the delay in memory is related to its size by $(SIZE)^{1/4}$ [12], the memory delay scales down by $n^{1/4}$. Let $L_{Total} = L_c + L_i + L_m$, where L_{Total} is the total latency, L_c is the compute latency, L_i is the interconnect latency and L_m is the memory latency. Further assume that $L_i = 10L_c$ and $L_m = L_c$. This is partly based on data shown in Table I. Then $L_{Total} = 12 L_c$. Now if $L_{Total(n)}$ represents the Total Latency for the n-partitioned case

$$L_{Total(n)} = L_C + \frac{10L_C}{n} + \frac{L_C}{\sqrt[4]{n}} = L_C \frac{n + 10 + n^{3/4}}{n} \quad (1)$$

$$\frac{L_{Total}}{L_{Total(n)}} = \frac{12n}{n + 10 + n^{3/4}} = S \quad (2)$$

Where S is the scaling factor, which is the ratio of total latencies for the un-partitioned and partitioned case. Now we use the above arguments to see its impact on dynamic power consumption; the partitioning does not have any impact on static power consumption under the assumption that there is no change in the total area of the chip. The dynamic power consumption for the unpartitioned case is $P = CV^2f$. Now if we consider dynamic power for a single partition *P(1)* in the n-partitioned case, the switching capacitance scales by *n* and since the Total Latency $L_{Total(n)}$ has gone down by a factor S, we can scale down the operational frequency by S to maintain the same throughput.

$$P(1) = \frac{C}{n} V^2 \frac{f}{S} \Rightarrow n.P(1) = CV^2 \frac{f}{S} \quad (3)$$

$$\frac{n.P(1)}{P} = \frac{n + 10 + n^{3/4}}{12n} \quad (4)$$

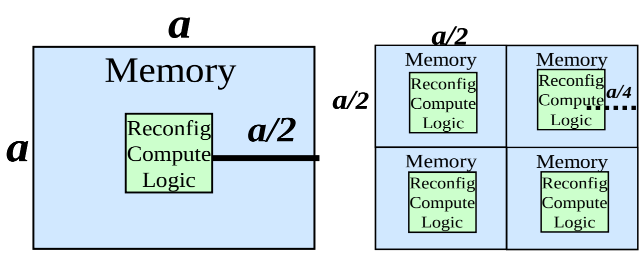


Figure 3. (a)An embedded soc (b)Embedded soc partitioned into 4 parts.

Equation 4 is the factor by which the dynamic power consumption goes down for the n-partitioned case and is shown in Figure 4. The above analysis though broadly accurate does not factor in the fact that with the scaling frequency, the V_{dd} would also scale down. Besides partitioning is a key complication. Getting a clean partitioning is a non-trivial problem and beyond a certain partitioning, the inter-partition communication will start to eat into the benefits of partitioning.

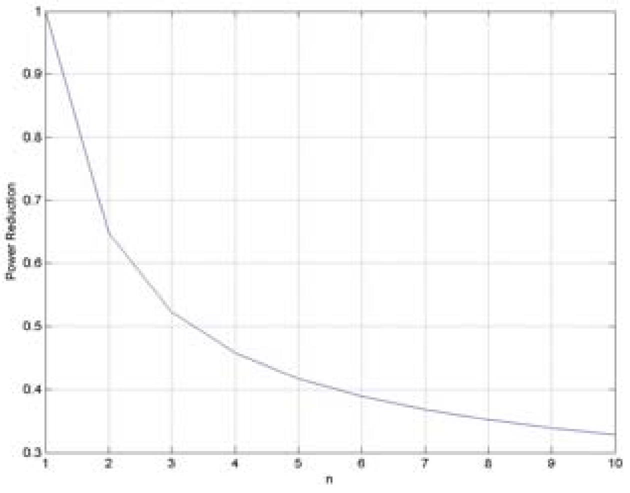


Figure 4. Power scaling as function of n

V. MLND Architecture

A conceptual diagram of the MLND architecture is shown in Figure 5 which is composed of the following components:

A. *Memory Pool*

Memory Pool is a pool of runtime partition-able memory. Each partition has the capacity to hold the dataset required for an application. While the MLND architecture provides hooks for implementing the partition, it is the software that characterizes an application for its memory needs based on bounded use case statements and manages the partitioning at runtime. Two kinds of memories are used. One that has higher bandwidth and relatively low capacity is meant for use by the physical (PHY) layer of the seven layer OSI model and the other that has relatively lower bandwidth but higher capacity is meant for the upper six layers of the OSI model.

B. Arithmetic Logic Pool

Arithmetic logic Pool is a pool of runtime partition-able reconfigurable arithmetic logic. This logic will be glued together to implement complex integer units to implement MACs, Butterflys etc. Like the memory partition, the arithmetic logic partition is dimensioned to fulfill the needs of an application. Depending on the performance constraint, it is the MLND compiler's task to determine the degree of parallelism, algorithmic level pipelining etc.

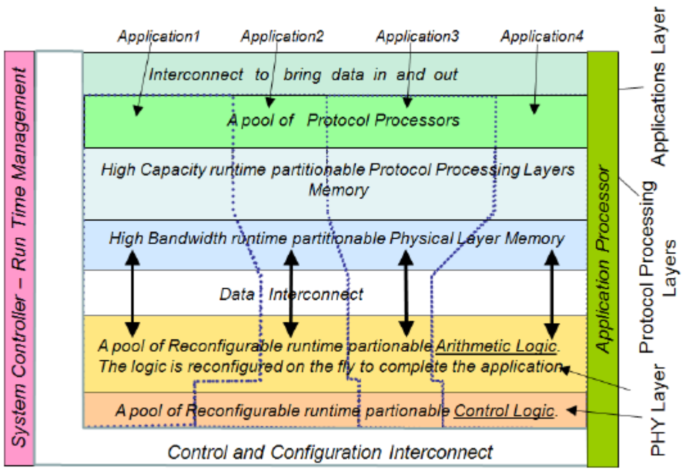

Application1
Application2
Application3
Application4
Interconnect to bring data in and out
A pool of Protocol Processors
High Capacity runtime partitionable Protocol Processing Layers Memory
High Bandwidth runtime partitionable Physical Layer Memory
Data Interconnect
A pool of Reconfigurable runtime partionable Arithmetic Logic.
The logic is reconfigured on the fly to complete the application
A pool of Reconfigurable runtime partionable Control Logic.
Control and Configuration Interconnect
System Controller – Run Time Management
Application Processor
Applications Layer
Protocol Processing Layers
PHY Layer

Figure 5. Conceptual view of Move Logic Not Data Architecture

C. Control logic

Like arithmetic logic, the pool of control logic provides the possibility of creating FSMs on the fly to control the arithmetic data path and memory operations. Essentially, the created FSM works like the hardware FSM and together with the arithmetic logic comes very close to the hardwired ASIC model of implementation, albeit with an overhead for some generality and the ability to partition. The separation of arithmetic and control logic is a key innovation to gain generality while maintaining efficiency and performance. This will be achieved by composing concurrent FSMs from a library of templates corresponding to various computational behaviors. The Arithmetic and Control Logic Pools together are dedicated to implementing the PHY layer of the OSI model.

D. Protocol Processor Pool

In the OSI model, the five immediate layers above the physical layer are characterized by control and memory intensive functionality and their memory access pattern is very irregular as compared to that of PHY layer so they are ideally served by a protocol processor. We intend to run the Application layer on a separate Application Processor. The protocol processors have access to high capacity memory and also control the transfer of data between the high-bandwidth PHY layer memory partitions and the Protocol processor memory partitions.

E. Interconnect

The NOC based interconnects implemented in MLND will give the flexibility to seamlessly partition the system. This kind of partitioning will make custom ASIC processors on the fly with its own memory unit, interconnect and data path The MLND architecture will have three kinds of NOC based interconnects as shown in Figure 6.

External data NOC brings external data into the chip and deposits it into the right memory partition and then once it has been processed takes it out to external memory. This kind of interconnect requires speed and flexibility and will be made up of high speed packet switched NOC.

Data NOC couples the memory partition to the arithmetic and control logic partition and it is this interconnect that MLND ensures is qualified as local interconnect. This interconnect will connect some memory to some ALU for a complete reconfigurable cycle and the interconnection will remain fixed for that reconfigurable cycle. This requires less flexibility, so we can implement this interconnect with high speed circuit switched NOC.

Control and Configuration NOC is used to control and configure the partitions and for operation and maintenance. This interconnect don't require much bandwidth but do require flexibility. So low speed packets switched NOC will be implemented for this kind of interconnect.

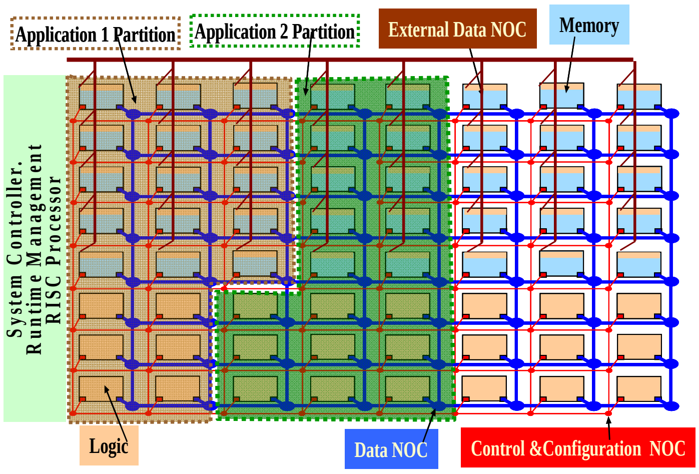


Figure 6. Conceptual view of Partitioning using NOC

F. System/Application Controller

The system controller provides the runtime management services of allocating memory and logic/arithmetic/protocol processor pools and partitioning. Figure 5 shows two RISC processors, System Controller and Application Processor, flanking the entire MLND structure. One of them is intended as a systems controller and the two would share the application layer functionality of the applications. The choice of two RISC processors is arbitrary at this stage; the actual number will be the outcome of the dimensioning of the MLND architecture by the design tool proposed as a research topic in this project.

G. Run Time Management

The MLND runtime support provides interface to the external world, manages resources in the MLND architecture and co-ordinates execution of the applications. The MLND Runtime System is conceptually made up of three interacting components. The External Interface manager interacts with the external world. This involves interrupts that trigger applications and peripherals responsible for exchange of data between external world and the MLND system. Resource manager, as the name suggests manages the resources like memory, arithmetic and control logic, bandwidth and energy/battery. When an external signal, a touch screen, a jog dial or a radio signal triggers an interrupt, the External Interface Manager passes on the request to the Resource Manager. The Resource Manager in turn analyzes the available resource, makes an allocation and passes on the constraints and requirement to the Application Manager. It is the Application Manager that instantiates the controllers: the application controller, the protocol processing code and the PHY layer controllers based on the constraints received from the Resource Manager. When an application is complete, it informs The Resource Manager, via the Application Manager resulting in an update of the available resources.

Besides the physical resources, the other key resource that the MLND Runtime Manager would have to handle is that of energy. The MLND architecture, from grounds up is built to implement the philosophy, if a resource is not being used, keep it shut. The other key energy management principle is to run the application at the optimal voltage frequency operating point, using the Dynamic Voltage Frequency Operating principle. The MLND architecture will introduce the novelty of having dynamic voltage islands and the RTM will play a critical role in its management

H. Methodology

MLND programming methodology would map a suit of applications, typified by modems and codecs, to the MLND architecture. While the details of the Design Environment (DE) are the objective of the proposed research, the conceptual steps and the components involved in the mapping process are shown in Figure 7.

1) Step 1. System Partitioning

In this step, the MLND DE partitions the application into three sets of functionality, each intended to run on a different kind of compute engine adapted to the nature of the functionality. The Application Layer on a RISC processor, the next five protocol processing layers on a customized protocol processor and the Physical layer on the reconfigurable arithmetic/control logic tiles. As a result of this step, we get the total application partitioned into the Application layer, the Protocol Processing Layers and the Physical Layer. These three partitions now communicate using the NOC based interconnect structures of the MLND architecture.

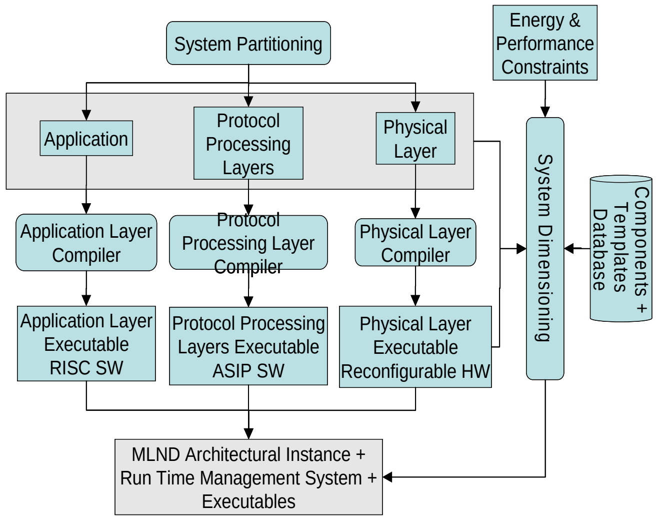
2) Step 2. System Dimensioning

This step identifies the overall storage need of the application, dimensions it and budgets the energy and performance constraints among the different partitions. We rely on the fact that the MLND concept primarily targets DSP oriented applications, where the nature of functionality, characterized by isochronous traffic, and the standards specification (from IEEE, ITU, MPEG etc) considerably helps gauge the storage needs. Further narrowing the search space are the architectural constraints that are imposed by the MLND architectural template and the memory technologies available. Lastly, the way application is modeled in terms of its data organization and access to it has a strong bearing on the storage dimensioning. The result of this step is the critical decision on what parts of dataset will be on chip and off-chip. Also decided in this step are the dimensions of the high bandwidth PHY layer memory and the high capacity Protocol layers memory. These decisions are also key inputs to the run time management system that needs to implement these partitions.

After the Steps 1 and 2, it should be possible to create a transaction level simulation model of the application, where the different partitions and storage interact using the MLND control, configuration and interconnect infrastructure. The next steps would refine the individual partitions.

3) Step 3. Protocol Layer Compilation

This step refines the protocol processing layers. The implementation style is software running on a RISC processor enhanced with custom instructions and internal memory structure to optimize the energy and performance of the protocol processing functionality. Creating such ASIPs (Application Specific Instruction Processors) is well researched with MESCAL [13] methodology as a prime example of this and will be the basis for this step. The logic for enhancing the RISC processor is built from reconfigurable tiles that can be configured to do various protocol processing functions. This makes the protocol processing pool homogenous and gives the runtime system freedom to place an application anywhere as required by the dynamic runtime situation.



System Partitioning
Energy & Performance Constraints
Application
Protocol Processing Layers
Physical Layer
System Dimensioning
Components + Templates Database
Application Layer Compiler
Protocol Processing Layer Compiler
Physical Layer Compiler
Application Layer Executable RISC SW
Protocol Processing Layers Executable ASIP SW
Physical Layer Executable Reconfigurable HW
MLND Architectural Instance + Run Time Management System + Executables

Figure 7. The MLND Design Flow

4) Step 4. Physical Layer Compilation

This step dimensions and instantiates the reconfigurable compute engine composed of arithmetic and control logic tiles. The key insight behind this step is that the physical layer functionality in most cases is composed of standard DSP functions whose architectural implementation space is pretty well understood. Examples of such functions are FFT, Viterbi, FIR filters etc. This understanding of architectural space is captured as templates and used to narrow the design space that the synthesis tool would otherwise have to search. For the few functions that do not have templates, we intend to use existing High Level Synthesis tools to create an implementation.

The results of this layer are the configuration codes that implement the arithmetic and address generation parts for the different algorithms/functions and the configuration codes for the control logic to implement the Finite State Machine (FSM) that controls the arithmetic and address generation logic. This layer also synthesize the PHY layer control and memory logic that glues together individual algorithms/functions that make up the PHY layer. This controller is again implemented as an FSM, and controls the individual algorithmic level controller synthesized in steps 3 & 2. More importantly, this controller is responsible for controlling the pipeline decision.

An additional key aspect regarding evaluation and estimation in the Steps 3 and 4 is that the MLND is an architecture built using regular tiles, and these tiles are built using full-custom macro implementation styles, this leaves little room for uncertainty in wiring delay as is common using the logic synthesis/standard cell based methodology. This approach, we believe will be key to our ability to achieve not only the best energy / performance metrics, but also the regularity of layout makes it possible to predict the energy and performance.

5) *Step 5. Application Layer Compilation*

In this step, the application layer is compiled to the RISC based application processor. While compilation to the RISC processor is straight forward, the application layer is essentially a controller that interfaces to the Protocol Processing layers via the MLND architectural elements. The Runtime Management System interacts with the Application Controller as the main agent for activating an application and knowing when an application is complete.

VI. Comparison with Previous Work

MLND is a flexible, scalable general purpose architecture which is not only suitable for nomadic products but equally suitable for high performance computing systems like base station and super computing. The computational requirements for a nomadic product may change with time and depends on the usage. For instance a nomadic user at times may be using MP3 player and browsing, at some other time he may only be using it as a GSM/3G phone, doing a video call and at most of the time the phone is sitting idle. So a nomadic product like cell phone requires flexibility so that it could offer the required computational power according to user's need and switch off/on additional resources.

MLND is a natural candidate for such a requirement. This ability to create runtime partitions of memory, arithmetic and control logic to implement custom ASIC like macros are the key to implementing the MLND theme: once the data come into the memory partition, the reconfigurable logic (arithmetic and control and protocol processing) implements a succession of algorithms to transform the data. The architecture guarantees that the memory partition and the arithmetic and control logic partitions are geometrically close enough that they qualify as being connected via local interconnects that does scale with technology as opposed to the global interconnects that do not scale with technology[5].

In MLND the separate control logic controls a set of partitioned memory and logic blocks called cluster of memory/logic. The partitioning will be done by using NOC [15]. Kernels will be implemented on this cluster of memory/logic. This kind of partitioning will make custom ASIC processors on the fly with its own memory unit, interconnect and data path. The data path unit will be parallel or serial as required. These custom processors will act like multi core/multi processors, exploiting (Instruction Level Parallelism), or DLP (Data level parallelism), algorithm level pipelining where all algorithms can be executed concurrently, and working in a pipelined fashion. Every cluster has its own individual control which makes it possible to clock them at different clock frequency, hence implementing dynamic voltage frequency scaling techniques to reduce power or switch them on off by resource manager.

The basic theme of MLND is to keep the wire distance between memory and logic minimum (local wire) so that the power consumption on interconnect is very small. In traditional architectures, ALUs are fueled by feeding data from memories which are far from them; hence dissipating a lot of power in interconnects. Such architectures do not scale with technology. In MLND the logic close to data memory is re-programmed to perform operation on the data stored in that memory. Imagine processor [10] [11] is also designed keeping interconnect power consumption in mind and is closest to MLND theme. A comparison of data flow of OFDM in MLND with Imagine processor is shown in Figure 8.

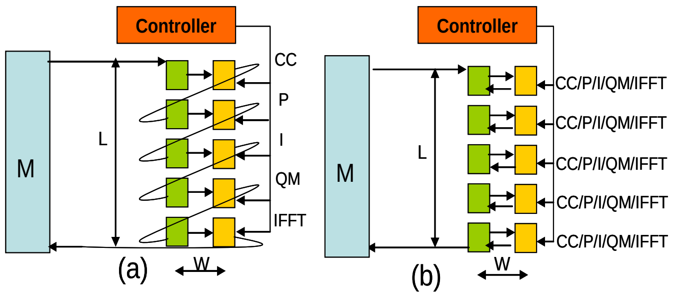


Figure 8. (a) Imagine Processor (b) MLND

In Imagine processor data enters into the first logic block i.e. Convolutional Coding (CC), processed and then saved into the memory as shown in Figure 8. From there it goes to Puncturing (P), then back to memory. It goes to Interleaving (I), QAM Mapping (QM) and IFFT in the same way before going to the main memory. The total logical distance travelled by the each data word, in case of Imagine processor, is L+10W. Assuming the arithmetic blocks are of same dimensions, in case of MLND the kernels are reconfigured instead of moving the data; reducing the total distance travelled by each data word to 10W. An OFDM symbol uses 64point FFT. A DVB standard uses 2048 points FFT. A reconfigurable DPU designed for MLND take 3-bits to configure. A radix-4 FFT butterfly uses 14 such DPUs. Suppose Z is the energy consumed by single bit to travel L distance shown in Figure 8. The Imagine processor configures the data path once and keeps its state for the life time of the application. On the other hand, MLND reconfigures the data path after a certain reconfiguration time. The number of bits needed to reconfigure the data path, travel on average L/2 distance. Assuming the arithmetic block of same dimensions, the energy comparison between moving data and code is done in Table II; which shows that it takes more energy to move data then moving the code as code size is much smaller then data size. Table II shows results for just one sample of FFT. Of course the hardware will operate on many more samples before undergoing reconfiguration which further confirms that code movement is cheaper then data movement in terms of energy. One may argue that code movement will be global and data movement will be local. According to [5] in 65nm global wires are 10 times slower then local wires, but the data in the Table II shows that energy for movement of code, in case of 64 point FFT, is 100 times less then energy required for movement of data. The figures are even better for 2048 point FFT.

TABLE II. ENERGY PER BIT FOR DATA MOVEMENT IN IMAGINE PROCESSOR VS CODE MOVEMENT IN MLND

FFT	Energy per bit for Data Movement	Energy Per Bit for Code movement
OFDM 64 points	64x16xZ=1024Z	14x3x5xZ/2=105Z
DVB 2048 points	2048x16xZ=33554432Z	14x3x5xZ/2=105Z

Minimizing the movement of data at PHY layer is a good thing but not sufficient. Because huge movement of data also happens at MAC layer and if that is left un-addressed the solution as a whole will still suffer from performance, energy and cost in-efficiencies. That is where the protocol processing layer takes over takes to minimize the data movement.

VII. CONCLUSION

MLND is an energy aware, scalable architecture, which minimizes the data movement inside the chip hence reducing power consumption. It has a regular structure and can be implemented in full custom. Ability to know exact wire lengths because of full custom implementation, and energy aware mapping and runtime system, makes it significantly different and better from the competitors. Traditional architectures lack this ability. MLND is flexible enough to be used in nomadic products as well as high end computing systems and super computers.

REFERENCES

[1] End to End Reconfigurability White Paper. Hardware Technology Exploration: Impact of Technology Evolution on End to End Reconfigurability. http://e2r.motlabs.com/whitepapers/E2R_WhitePaper_HardwareTechExploration_December05.pdf

[2] Erik Jan Marinissen, Betty Prince, Doris Keitel-Schulz, Yervant Zorian. Challenges in Embedded Memory Design and Test. Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05).

[3] Panagiotis Tsarchopoulos. Objective ICT-2007.3.4 Computing Systems. FP7 Information Day. March 7, 2007. Brussels.

[4] Jan Rabaey. Silicon Architectures for Wireless Systems. Tutorial Part 1. Hotchips Conf. 2001.

[5] Bjerregaard T. and Mahadevan S., "A Survey of Research and Practice of NoC", ACM Inc. New York, USA, 2006.

[6] Michael J. Flynn, Patrick Hung, Kevin W. Rudd. Deep-submicron Microprocessor Design Issues. IEEE Micro. July-August 1999.

[7] Hemani, A, Klapproth, P. Trends in SOC Architectures. Chapter in the book "Radio Design in Nanonmeter Technologies" Editied by Professor Mohammed Ismail and Delia Gonzales. Springer Verlag 2006/2007

[8] Christoforos Koszyrakis. Scalable Vector Media-processors for Embedded Systems. PhD Thesis. Univ. of California Berkeley. May 2002. Report No. UCB/CSD-02-1183

[9] Kang Yi el.al. FlexRAM: Toward more Advanced Intelligent Memory System. Proceedings ICCD Oct. 1999.

[10] Brucek Khailany, William J. Dally, Scott Rixner, Ujval J. Kapasi, Peter Mattson, Jinyung Namkoong, John D. Owens, Brian Towles, and Andrew Chang, "Imagine: Media Processor With Stream," IEEE Micro, March/April 2001, pp. 35-46.

[11] Scott Rixner, William J. Dally, Ujval J. Kapasi, Brucek Khailany, Abelardo Lopez-Lagunas, Peter Mattson, and John D. Owens, "A Bandwidth-Efficient Architecture for Media Processing," Proceedings of the 31st Annual International Symposium on Microarchitecture, Nov. 30 - Dec. 2, 1998, Dallas, Texas, pp. 3-13.

[12] Michael J Flynn, Patrick Hung. Microprocessor Design Issues: Thoughts on the Road Ahead. IEEE Micro. May-June 2005.

[13] J. Rabaey: Reconfigurable Computing: The Solution to Low Power Programmable DSP; Proc. ICASSP'97 Munich, Germany, April 1997.

[14] Marinissen Erik Jan, Prince Betty, Shultz D.K, Zorian, Yervant. Challenges in Embedded Memory Design and Test. Proceedings of DATE 2005.

[15] A Hemani, A Jantsch, S Kumar, A Postula, D Lindqvist, J Öberg, M Millberg. Networks on Chip: An architecture for the Billion Transistor Era. Proceedings of the IEEE Norchip Conference. October 2000.

[16] Reiner Hartenstein. Coarse Grain Reconfigurable Architectures. Proceedings of Asia South Pacific Design Automation Conference. 2001. Yokohama, Japan

Hierarchical Agent Architecture for Scalable NoC Design with Online Monitoring Services

Alexander Wei Yin, Liang Guang, Pasi Liljeberg, Pekka Rantala, Ethiopia Nigussie, Jouni Isoaho, Hannu Tenhunen
Department of Information Technology, University of Turku, Finland
{yinwei, liagua, pakrli, peaura, ethnig, jisoaho, hatenhu}@utu.fi

Abstract—**Hierarchical Agent Architecture is proposed to provide online monitoring services to NoC-based systems. Based on circuit conditions traced at the run-time, system settings are monitored adaptively by agents at each architectural level. This monitoring approach partitions various online diagnostic and management services onto hierarchical implementation levels so as to provide scalability and variability for large-scale NoC design. This paper explains the monitoring interaction between agent levels, and focuses on system optimization alternatives handled by different agent levels. It further quantitatively analyzes the feasibility and design alternatives in monitoring communication interconnection upon regular tile-based NoC layout. Though still under intensive research, the proposed architecture is endowed with promising potential for highly-integrated NoC design.**

I. Introduction

With continuous technology scaling, the size of NoCs (Network-on-chip) is constantly increasing. Parallelizing applications onto many processing elements leads to high potential speedup as demonstrated by the recently released TeraFLOPS processor [1] and TILE64 processor [2] which integrate 80 and 64 cores respectively on a single chip. In academia, thousand-core processors have been projected and discussed [3]. However, system designers are challenged with a number of daunting issues. Conventional concerns, such as power consumption, will continue to pose tough, if not stronger, constraints on design and implementation methods. Especially the dramatic increase of leakage power in sub-100nm technology requires urgent consideration from all architectural levels [4]. New design considerations including increasing influence from PVT (process, voltage and temperature) variations [5] and unpredictable hardware and software errors only exacerbate the design complexity. Variations and faults also worsen the power constraints as the design margin is lowered to tolerate parametric variations. To deal with these issues, the system-level design method should support online dynamic services at different implementation level, so as to achieve maximum system efficiency with run-time coarse/fine-granular tuning.

A few previous works have addressed system monitoring services on NoC platforms [6, 7, 8]. From them, several distinctive requirements for managing NoC structures in a scalable manner can be identified. Firstly, local circuits need to be provided with distributed monitoring modules. Distributed monitoring reduces the local operation delay or interconnect latency for urgent monitoring services, and it prevents the appearance of communication bottleneck. However, despite the system size, centralized monitoring is still an indispensable complement to localized monitoring schemes. Theoretically, a centralized monitor, with the knowledge of all on-chip resources, is able to coordinate and balance the functioning of all components with the aim of optimizing the overall system performance. In practice, as an example, [9] adopts a single processing unit for dynamic testing operations and a global-level scheduler. For the scenario of thousand-core NoCs with no concrete analysis available, an analogy to the overwhelmingly complex nervous system of human beings can help motivate the need of centralized monitors. The human nervous system is a large-scale monitoring network with numerous distributed neurons as local monitors. These neurons are coordinated by upper-level centralized monitors such as the spinal cord and the brain, which balance and optimize the general body function. For either distributed or centralized monitoring schemes, the energy efficiency of monitoring services should be maximized.

We propose a hierarchical agent architecture endowed with the required monitoring features. This architecture adds a monitoring layer of agent hierarchy onto the NoC platform. Agents are autonomous and adaptive monitors to be implemented with various approaches, and they are responsible for monitoring different architectural levels. Local agents provide fast and low-overhead monitoring services to individual functional components, and report low-level conditions and performance to higher-level agents. The latter supervise the general system performance on a coarse granularity. This architecture aims to achieve overall system performance by balancing the monitoring among all on-chip resources, while providing a wide design and synthesis space for the realization of agents at each level.

This paper examines the functional partition of agent levels and the monitoring interaction between them to perform monitoring services with an joint effort (Section II). Upon a tile-based NoC platform, we demonstrate the flexible incorporation of system optimization techniques with agent monitoring architecture (Section III). As an extra communication layer upon existing interconnect, alternatives in realizing agent communications are examined quantitatively in Section IV, which suggests an optimal design trade-off for monitoring communication interconnects. Section V concludes the paper.

II. Hierarchical Agent Monitoring Architecture

A. Agent Hierarchy

The architecture ranks the agents into four level: from the top to the bottom level, a single application agent, a single platform agent, distributed cluster agents (one per each cluster) and distributed cell agents (one per each cell) (Fig. 1). The application agent is a piece of software capturing application functionality and run-time performance requirements and constraints. The platform agent, based on the application specification and resource availability, utilizes appropriate resources, maps and schedules the instructions onto the acquired resources, configures the network, and monitors general system performance during application execution. Each cluster agent monitors a whole cluster, which is a group of processors with accompanying components (caches, scratchpad memories, switches, links, etc.). A cluster is logically divided into cells, each of which is a basic functional unit, such as a processing unit, a switch or a link. The cells are equipped with their own local monitors, the cell agents, which trace and adjust the local circuit conditions.

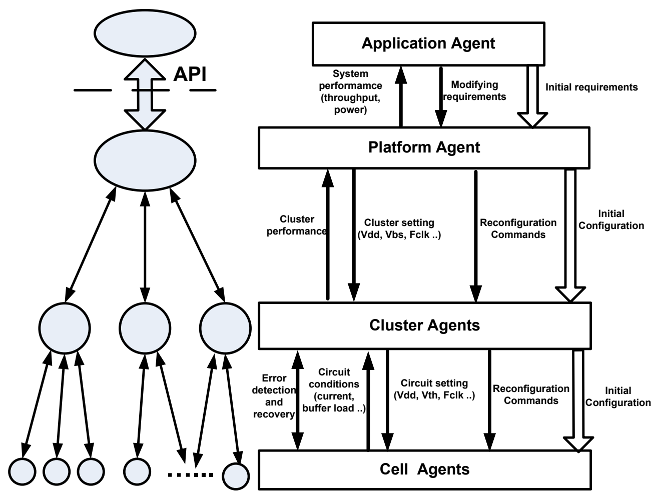


Figure 1. Hierarchical Agent Monitoring Approach

B. Hierarchical Monitoring Approach

The proposed architecture highlights hierarchical approaches to various monitoring services, for instance power-optimization and fault-tolerance, by the joint efforts from all levels of agents.

Before execution, the platform agent utilizes a number of resources and configures the network based on the initial application requirements with power and performance awareness [10]. A number of resources are reserved as spares in case of component failures. The initial configuration is enforced from the platform agent to the cluster and then cell agents.

After the application starts running, the cell agents are tracing their local circuit conditions, such as current (including leakage current for idle components), workload, and any faults or failures (for instance a link failure or a malfunctioning processing unit). Cell agents attempt to fix the errors if feasible (for example by retransmission in case of transient crosstalk-induced error [11]). The traced circuit conditions along with not-solved-yet failures are sent to cluster agents. Cluster agents attempt to adjust the cell settings based on these information. For instance they may scale the supplies of a certain cell (DVFS: dynamic voltage and frequency scaling) or the threshold voltage by using ABB (adaptive body biasing [12]). If a component has failed to work, they will acquire spare components and configure them into the cluster. Cluster agents send cluster performance to the platform agent. The information concerning cluster performance is represented at a coarser granularity than those sent between cluster and cell agents, for example, the power consumption of the cluster, or average network workload within the cluster, the error rate of the cluster. Based on these information, the platform agent may reconfigure the system, for instance assigning more spares into a failure-prone cluster, or scale down the voltage and frequency of a cluster with overwhelming power consumption. The overall system performance, for instance the throughput and the power consumption, is reported by the platform agent to the application agent, which may modify the real-time application requirements. Fig. 1 illustrates these hierarchical monitoring interactions.

The hierarchical agent-based monitoring approach is distinctive as being scalable and implementation-flexible for any-sized NoCs. The distributed cell agents, as physically adjacent to the functional units and exclusively responsible for local monitoring, can provide fast and fine-grained monitoring services to local circuits. The cluster agents are exclusively responsible for their own clusters, thus cluster-level monitoring is still low-latent and requires limited amount of processing capacity. The platform agent, though monitoring the whole system, only handles the resources at a coarse granularity. For instance, in terms of fault-tolerance, only errors which can not be fixed by low-level agents are reported to and handled by the platform agent. In this manner, no communication or processing bottleneck will appear in any large-scale platform. The supervision of higher level agents over lower-level ones ensures the optimal overall system performance. Hierarchical monitoring approach also provides a wide design and synthesis space for implementing various management algorithms and circuits. Low-level circuit optimization methods, such as power or clock-gating can be implemented as dedicated circuits triggered by cell agents. High-level component management methods, such as DVFS or ABB, can be enforced by cluster level agents. Low-level circuit optimization should be simple in terms of synthesis to offer fast operation with small overhead. High-level operations can require more processing power since they are typically much less frequent than low-level operations. As the highest-level monitor, the platform agent configures the system with optimal general settings, for instance, an appropriate network connection to reduce inter-cluster communication. Only with the concept of monitoring hierarchy can various optimization methods be implemented efficiently with different design and synthesis constraints.

III. Hierarchical Monitoring Services on NoCs

A. Agent Mapping on Regular NoC Platform

To discuss the feasible mapping of agents on NoC platform, we consider the regular tile-based mesh structure. A conventional tile comprises of a PE (processing element), a NI (network interface), a switch and the links. On such tile-based NoC platform, we naturally locate a cell agent for each tile, though distributed monitoring circuits may be located at particular places within the cell, for instance, a power-gating sleep-transistor on the link. The cell agent physically shares the space with a processing element. The cluster agent is located at fixed locations at design time, and cells are configured into the clusters dynamically at the run-time. Depending upon the complexity of cluster monitoring algorithm and maximum number of cells to be monitored, a cluster agent may physically replace a conventional PE or still shares the space with a PE. The application agent and the platform agent monitor over the whole system; without application-specific knowledge, we assume they are located together at the geographic center of the tiling area. Fig. 2 illustrates the feasible mapping of agents on the regular NoC structure.

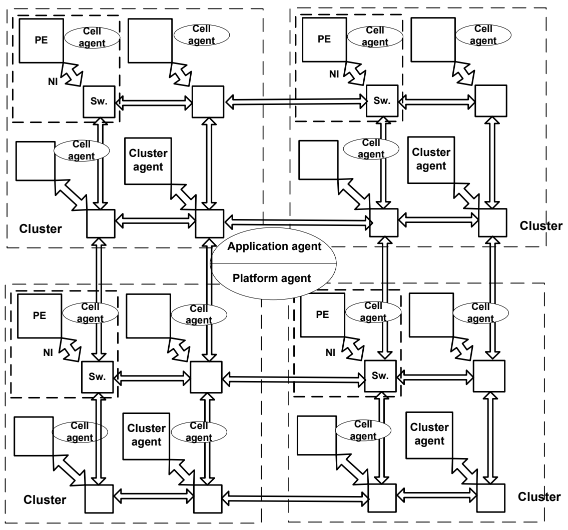


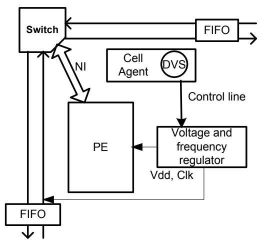
Figure 2. Illustration of Agent Mapping on NoCs

To offer scalability for thousand-core systems, clusters can be divided into hierarchical subclusters and similar monitoring functional partition will be applied. It conceptually originates from the manner a biosystem or human society organizes its overwhelming amount of resources.

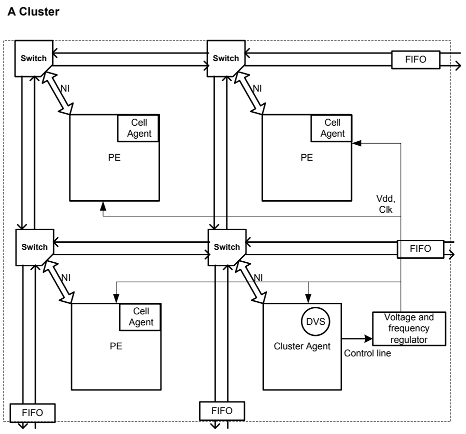
B. Low-power Optimization with Agents

In the hierarchical agent architecture, various monitoring services can be incorporated at different implementation level considering the specific trade-off on the actual platform. Here we explain the design consideration with dynamic power optimization as an example of various feasible services.

One of the major dynamic power saving techniques is DVFS, which is traditionally provided on a chip-wide domain



(a) Cell-level DVFS (showing one cell)



(b) Cluster-level DVFS (showing one cluster)

Figure 3. Power Optimization Services by Different Agent Levels

[13]. But chip-level single power domain is not able to utilize the local traffic variation in exploiting the supply scaling potential, thus per-core based DVFS is proposed [14]. In the cell-divided NoC platform, a cell can be conveniently set with a supply regulator with the cell agent in charge of the voltage and frequency adjustment (Fig. 3(a)). The overhead for per-cell based DVFS is significant. [15] reports $0.14mm^2$ area overhead and 83.2% peak efficiency of a DC-DC converter in $90nm$ technology. Each time the voltage is converted, extra energy will be consumed for the power regulation.

To alleviate the per-core-based DVFS overhead, the concept of voltage islands [16, 17] has been proposed. Up-to-date, voltage islands are statically determined at design time. To incorporate multiple voltage islands on the NoC platform, each cluster agent determines the voltage and frequency for its own cluster (Fig. 3(b)). The area and energy overhead is reduced proportional to the number of cells in a cluster. Per-cluster-based power optimization, however, does not support the reconfiguration of cells into different clusters at the run-time, though assigning spares into clusters initially still provides cell replacement possibilities against component failures.

The granularity of monitoring services is a design choice dependent on the size of the actual platform, the workload and constraints of the application. In terms of power optimization, per-cluster-based monitoring with lower implementation overhead seems to be more feasible in the long term with smaller-sized processing cores. In general, any monitoring service can be configured at the design time or execution time (with the support of reconfigurable platform) to be handled by different level of agents, correspondingly in various granularities.

IV. Design Trade-offs for Agent Communication

A. Monitoring Communication Interconnect Alternatives

Agents exchange monitoring information with their higher or lower counterparts as illustrated in Fig. 1. The monitoring communication needs to be reconfigurable so new cells can be incorporated to certain clusters at the run-time. Some conventional interconnection does not support reconfiguration (for instance, the star-like network as in Fig. 4). Instead, we consider three interconnect alternatives which all support run-time reconfiguration but have different area, energy and latency overheads. Throughput is not a prioritized design constraint, since the monitoring communication is low in data volume ([18] reports 8% and 5% debugging monitoring traffic overhead for two streaming applications).

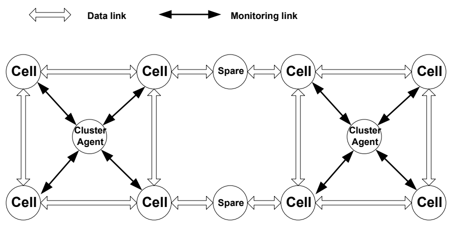


Figure 4. Non-reconfigurable Star Networks for Agent Monitoring Interconnect

The first alternative is to realize monitoring communication as TDM (Time-Division-Multiplexing)-based virtual channel upon existing links. This option incurs design complexity in virtual channel arbitration and allocation, increases the switch latency of both monitoring interconnect and data communication. The virtual channel arbitration and allocation also incur energy overhead. Wiring overhead, however, is kept to the minimum though the switch area is moderately increased.

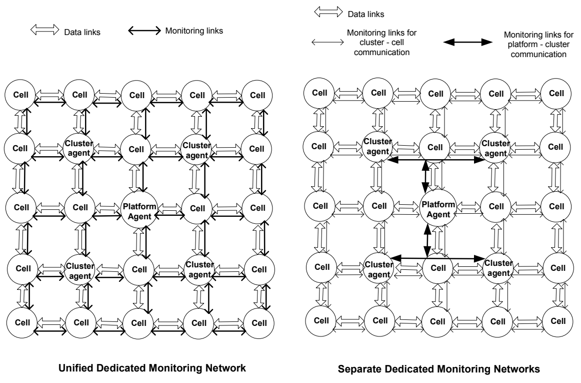

Data links
Monitoring links
Monitoring links for cluster - cell communication
Monitoring links for platform - cluster communication
Cell
Cluster agent
Platform Agent
Unified Dedicated Monitoring Network
Separate Dedicated Monitoring Networks

Figure 5. Alternative Dedicated Monitoring Interconnect Architectures

The second alternative is to adopt a "unified dedicated monitoring network" for monitoring communication (Fig. 5 on the left side). It is called "unified" as monitoring communication between both cluster-cell agents and platform-cluster agents is transmitted on the same dedicated network. This option utilizes more wiring resources but simplifies the switch arbitration between data and monitoring communication, thus reducing the communication energy and latency.

The third alternative is to adopt "separate dedicated monitoring networks" for monitoring communication (Fig. 5 on the right side). Compared to the unified monitoring network, this option adds another network connecting the single platform agent to a small number of cluster agents. As a result, the communication between platform and the cluster agents is simplified with very limited wiring overhead.

B. Quantitative Analysis of Monitoring Interconnects

To quantitatively compare the implementation overhead of three monitoring interconnect architecture, we model a network similar to the TeraFLOPS processor in the same 65nm technology. The network has 8*8 processing elements mapped on a regular tile-based mesh topology. We assume input-buffered pipelined switches with the structure suggested by [19] with matrix crossbar [20]. For TDM channels, each input buffer is 4-flit long while the unified separate network has 2-flit-long input buffer considering the higher traffic load of data communication. The other dedicated network for communication between cluster agents and the platform agent assumes no buffer since the traffic on this network is exclusive and infrequent. The arbitration assumes wormhole routing. NoC links are modeled as segmented wires with drivers and evenly inserted repeaters[1]. Data links are 32 bits wide and 2 mm long [2], and dedicated monitoring link is 8-bit wide and equally long. The locations of the platform agent, cluster agents and cells (with cell agents) are illustrated in Fig. 6. The whole NoC system is assumed to be mesochronous with network frequency as 1GHz and the supply voltage as 1V.

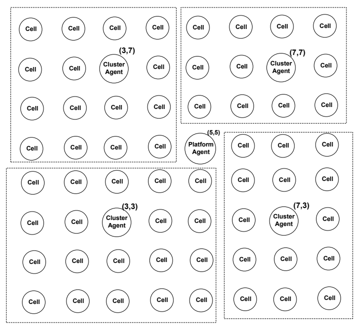


Figure 6. Locations of Platform, Cluster and Cell Agents in the Experimental Platform (with initial cluster boundary labeled)

We estimate the area and energy overhead of switches by simulating with Orion [21], a widely-used on-chip switch

[1]wire width: 210nm; spacing: 210nm; repeater interval: 0.25mm; repeater size: 10x minimal inverter size; driver size: 12x.

[2]TeraFLOPS uses 2mm * 1.5 mm tiles, while we simplify the tiles to be 2 mm * 2 mm squares

power simulator. The switch latency is estimated based on [19]. The wires are modeled and simulated by Cadence. The Orion simulator does not produce result for 65nm technology directly, thus we apply scaling factors (based on [22]) to the result of 70nm technology simulation using Orion. The scaling factors for energy, area, and latency are 0.86, 0.86 and 0.93 respectively. The energy of wires are simulated by Cadence. The latency in the switch buffer assumes an average 50% occupancy ratio.

1) Latency: The latency is calculated in cycles considering the longest distances between the platform agent and a cluster agent and between a cluster agent to one of its cell agent. From Fig. 6, we see that both distances are at maximum 4 hop counts with minimal routing. The wire latency is simulated to be 198ps, and each pipeline stage latency in switches is estimated to be lower than 300ps ([19], assuming an FO4 inverter delay to be 15ps in 65nm technology). With 1GHz frequency, each link and one router pipeline stage (virtual channel allocation, routing and decoding, crossbar traversal) take 1 cycle delay. Table I summarizes the latency comparison for monitoring communication in each interconnect architecture.

Interconnect Architecture	Delay (cluster <-> cell agents)	Delay (platform <-> cluster agents)
TDM-based	24 cycles	24 cycles
Unified Dedicated Network	16 cycles	16 cycles
Separate Dedicated Networks	16 cycles	8 cycles

Table I
LATENCY COMPARISON OF THREE MONITORING INTERCONNECT ARCHITECTURES (NETWORK WORKING AT 1GHZ)

2) Energy Consumption: The energy is calculated by the amount of energy consumed by a 8-bit flit (as we assume dedicated monitoring networks are 8-bit wide) traversing on the longest paths between the platform agent and a cluster agent, and between a cluster agent to one of its cell agent (4 hop counts as in Fig. 6 with no misrouting). Table II summarizes the energy consumption for monitoring communication in each interconnect architecture.

Interconnect Architecture	Energy (cluster <-> cell agents)	Energy (platform <-> cluster agents)
TDM-based	12.92 pJ	12.92 pJ
Unified Dedicated Network	5.40 pJ	5.40 pJ
Separate Dedicated Networks	5.40 pJ	2.31 pJ

Table II
ONE-FLIT MONITORING COMMUNICATION ENERGY OF THREE MONITORING INTERCONNECT ARCHITECTURES (NETWORK WORKING AT 1GHZ)

3) Area : We analyzed the total wiring and switch area for each interconnect architecture as a percentage of a TeraFLOPS chip (275mm^2) (Table III).

Interconnect Architecture	Area (mm^2)	Percentage (of a chip area)
TDM-based	7.44	2.71%
Unified Dedicated Network	8.95	3.26%
Separate Dedicated Networks	9.11	3.32%

Table III
AREA OVERHEAD OF THREE MONITORING INTERCONNECT ARCHITECTURES

C. Optimal Design Trade-off for Future NoCs

The estimated figures show that separate dedicated monitoring networks are the most energy-efficient and low-latency interconnection for monitoring communication. Compared to TDM-based interconnection, it reduces the latency by 66.7% and energy consumption 82.1% for the communication between the platform and cluster agents, while achieving the same latency and energy efficiency as unified dedicated network for the communication between the cluster and cell agents. However there is area penalty involved: the area overhead is increased from 2.71% to 3.32%. Nonetheless the wiring area overhead has become less of a design constraint as multi-layer fabrication process provides quite abundant wiring potential for on-chip systems ([8]; TILE64 processors incorporate 5 physically separate networks, each of them being 64-bit wide). With transistor feature size and wire dimension continue to decrease in the foreseeable future, the separate monitoring networks will provide the most optimal trade-off exploiting the on-chip wiring resources while minimizing the more critical power consumption and global interconnect latency.

V. CONCLUSIONS

Hierarchical agent monitoring architecture provides great scalability and design flexibility for future large-scale NoC systems. With an extra monitoring layer comprised of four levels of agents, the system is potentially able to achieve maximized efficiency with online monitoring services. This paper elaborately explains the hierarchical monitoring approaches enabled by the interactions of all levels of agents, and examines the design alternatives for low-power optimization of different granularities as an example of flexible functional partitions among agent levels. Quantitative analysis for agent interconnection alternatives suggests reasonable trade-offs between area, energy and latency overhead, and motivates separate dedicated monitoring networks for inter-agent communication. This work demonstrates the potential and feasibility of multi-level online monitoring layer upon the overwhelming amount of on-chip resources, which provides a great diversity of design options in a scalable manner.

At present, specific monitoring services on regular NoC platform with the proposed architecture is under intensive study and analysis.

REFERENCES

[1] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erra-

guntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w teraflops processor in 65-nm cmos. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, 2008.

[2] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, Matthew Mattina, Chyi-Chang Miao, Carl Ramey, David Wentzlaff, Walker Anderson, Ethan Berger, Nat Fairbanks, Durlov Khan, Froilan Montenegro, Jay Stickney, and John Zook. Tile64tm processor: A 64-core soc with mesh interconnect. In *Proc. Digest of Technical Papers. IEEE International Solid-State Circuits Conference ISSCC 2008*, pages 88–598, 2008.

[3] Shekhar Borkar. Thousand core chips: a technology perspective. In *DAC '07: Proceedings of the 44th annual conference on Design automation*, pages 746–749, New York, NY, USA, 2007. ACM.

[4] Jan M. Rabaey. Scaling the power wall: Revisiting the low-power design rules. Keynote speech at SoC 07 Symposium, Tampere, November 2007.

[5] K. Bernstein, D. J. Frank, A. E. Gattiker, W. Haensch, B. L. Ji, S. R. Nassif, E. J. Nowak, D. J. Pearson, and N. J. Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM Journal of Research and Development*, 50(4/5):433–449, 2006.

[6] C. Ciordas, T. Basten, A. Radulescu, K. Goossens, and J. Meerbergen. An event-based network-on-chip monitoring service. In *Proc. of the 9th IEEE International High-Level Design Validation and Test Workshop*, pages 149–154, 2004.

[7] C. Ciordas, K. Goossens, A. Radulescu, and T. Basten. Noc monitoring: impact on the design flow. In *Proc. IEEE International Symposium on Circuits and Systems ISCAS 2006*, pages 1981–1984, 2006.

[8] D. Wentzlaff, P. Griffin, H. Hoffmann, Liewei Bao, B. Edwards, C. Ramey, M. Mattina, Chyi-Chang Miao, J.F. Brown, and A. Agarwal. On-chip interconnection architecture of the tile processor. *IEEE MICRO*, 27(5):15–31, 2007.

[9] D. Sylvester, D. Blaauw, and E. Karl. Elastic: An adaptive self-healing architecture for unpredictable silicon. *IEEE Design & Test of Computers*, 23(6):484–490, 2006.

[10] Jingcao Hu and R. Marculescu. Energy and performance-aware mapping for regular noc architectures. *IEEE Transactions on COMPUTER-AIDED DESIGN of Integrated Circuits and Systems*, 24(4):551–562, 2005.

[11] Teijo Lehtonen, Pasi Liljeberg, and Juha Plosila. Online reconfigurable self-timed links for fault tolerant noc. *VLSI Design*, 2007:13, 2007.

[12] S.M. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proc. IEEE/ACM International Conference on Computer Aided Design ICCAD 2002*, pages 721–725, 2002.

[13] Canturk Isci, Alper Buyuktosunoglu, Chen-Yong Cher, Pradip Bose, and Margaret Martonosi. An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget. In *Proc. of 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-39)*, pages 347–358, 2006.

[14] W. Kim, M. S. Gupta, G.-Y. Wei, and D. Brooks. System level analysis of fast, per-core dvfs using on-chip switching regulators. In *International symposium on high-performance computer architecture*, Feb. 2008.

[15] P. Hazucha, G. Schrom, Jaehong Hahn, B.A. Bloechel, P. Hack, G.E. Dermer, S. Narendra, D. Gardner, T. Karnik, V. De, and S. Borkar. A 233-mhz 80%-87% efficient four-phase dc-dc converter utilizing air-core inductors on package. *IEEE Journal of Solid-State Circuits*, 40(4):838–845, 2005.

[16] Lap-Fai Leung and Chi-Ying Tsui. Energy-aware synthesis of networks-on-chip implemented with voltage islands. In *Proc. 44th ACM/IEEE Design Automation Conference DAC '07*, pages 128–131, 2007.

[17] D.E. Lackey, P.S. Zuchowski, T.R. Bednar, D.W. Stout, S.W. Gould, and J.M. Cohn. Managing power and performance for system-on-chip designs using voltage islands. In *Proc. IEEE/ACM International Conference on Computer Aided Design ICCAD 2002*, pages 195–202, 2002.

[18] C. Ciordas, K. Goossens, T. Basten, A. Radulescu, and A. Boon. Transaction monitoring in networks on chip: The on-chip run-time perspective. In *Proc. International Symposium on Industrial Embedded Systems IES '06*, pages 1–10, 2006.

[19] L.-S. Peh and W.J. Dally. A delay model and speculative architecture for pipelined routers. In *Proc. of The Seventh International Symposium on High-Performance Computer Architecture*, pages 255–266, 19–24 Jan. 2001.

[20] Hangsheng Wang. a detailed architectural-level power model for router buffers, crossbars and arbiters. Technical report, Department of Electrical Engineering, Princeton University, 2004.

[21] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and S. Malik. Orion: a power-performance simulator for interconnection networks. In *Proc. 35th Annual IEEE/ACM International Symposium on (MICRO-35) Microarchitecture*, pages 294–305, 2002.

[22] W. Haensch, E. J. Nowak, R. H. Dennard, P. M. Solomon, A. Bryant, O. H. Dokumaci, A. Kumar, X. Wang, J. B. Johnson, and M. V. Fischetti. Silicon cmos devices beyond scaling. *IBM Journal of Research and Development*, 50(4/5):339–361, 2006.

Author Index