A hybrid interconnect network-on-chip and a transaction level modeling approach for reconfigurable computing

Lenart, Thomas; Svensson, Henrik; Öwall, Viktor

Published in:
[Host publication title missing]

DOI:
10.1109/DELTA.2008.85

2008

A Hybrid Interconnect Network-on-Chip and a Transaction Level Modeling approach for Reconfigurable Computing

Thomas Lenart, Henrik Svensson and Viktor Öwall
Department of Electrical and Information Technology, Lund University
Box 118, SE-221 00 Lund, Sweden
Email: {thomas.lenart,henrik.svensson,viktor.owall}@eit.lth.se

Abstract

*This paper presents a hybrid interconnect network consisting of a local network with dedicated wires and a global hierarchical network. A distributed memory approach enables the possibility to use generic memory banks as routing buffers, simplifies the implementation and reduces the area requirements of routers. A SystemC simulation environment (*SCENIC*) has been developed to simulate and instrument models, and to setup different topologies and scenarios. Modules are designed as transaction level models to improve design time and simulation speed.*

1. Introduction

Reconfigurable architectures (RA) have proven to effectively exploit parallelism and reuse available resources in a variety of application domains [4]. These architectures use flexible interconnects to allow communication between resources distributed over the chip. Interconnect systems are usually based on switches, which is reconfigured for every new task and set up by a global supervisor. In contrast, the Network-On-Chip (NoC) concept propose highly flexible interconnect topologies based on intelligent routing networks, where any two nodes in the system can initiate communication without global supervision [7]. However, network routers are complex and therefore area consuming. To bridge the gap between traditional RAs and NoC, this paper presents a processing array which combines local communication on dedicated links with a global hierarchical routing network. Hierarchical routing reduces the router complexity since there is only a single path between processing nodes, while the dedicated local links provide high speed communication between neighbor resources.

The complexity of NoC systems require advanced simulation platforms to evaluate performance and discover bottlenecks. Design of a reconfigurable architecture needs to be addressed at a system-level, and simulation-based performance exploration is required to evaluate the impact of design parameters. Simulation-based performance exploration is limited by simulation run-time and the required design effort to change functionality. In this paper we will present a simulation environment that extends SystemC [6] to enable interactive design exploration, and how to use Transaction Level Modeling (TLM) to simulate reconfigurable system.

2. Related work

Topology and routing - Recent work compares many popular NoC architectures, including *2D mesh*, *ring*, *torus*, *folded torus*, and *spidergon* networks [1] [7]. *Hybrid topologies* are discussed in [2] as a way of aggregating bandwidth between adjacent processing cells, but introduce longer latency and require bridges to convert between protocols. The 2D mesh architecture in Figure 1(a) is a popular and well researched architecture in academia, with several variations such as torus shown in Figure 1(b) and the folded torus. Mesh architectures are both regular and scalable, but comes with the downside of poor global communication and often large router logic overhead. Global communication is routed along the horizontal and vertical wires from source to destination, consuming local bandwidth along the way. Each processing cell connects to the two-dimensional mesh using a network router, which contains buffer queues to store and forward packets from all directions. Since every router can potentially handle communication from any source to any destination, the router becomes unnecessarily complex. Hence, pure mesh architectures face problems concerning both bandwidth and high complexity.

Common switching techniques are *packet switching*, *circuit switching*, and *wormhole switching*, using adaptive or deterministic routing [7]. Wormhole switching divides large packets into smaller units called *flits*. This ensures that every flit is routed through the same established path as the

0-7695-3110-5/08 $25.00 © 2008 IEEE
DOI 10.1109/DELTA.2008.85

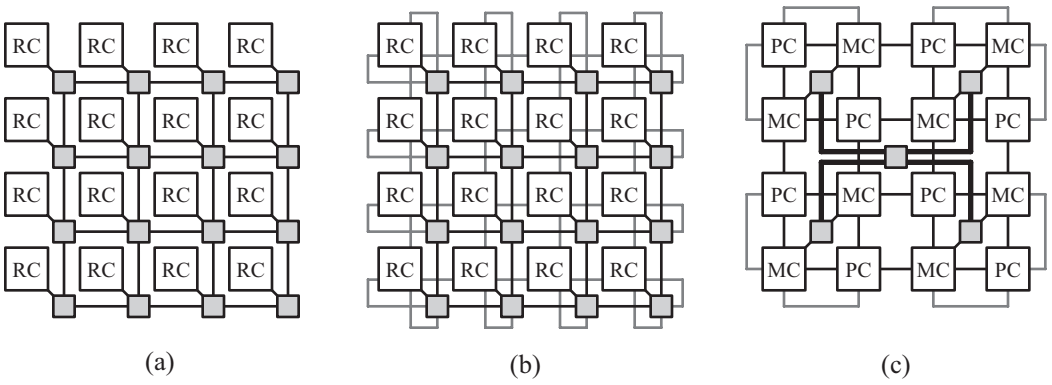

Figure 1. (a) 2D mesh with resource cells, (b) Torus, (c) Proposed architecture with nearest neighbor and hierarchical communication. In this example, four basic cells are connected together using dedicated wires for local communication and five network routers for global communication.

first flit in each packet, which reduces the required buffer size and avoids packet reordering at the destination. Since flits can not be interleaved over a physical channel, *virtual channels* are commonly introduced to increase the channel utilization by inserting multiple queues for each physical link. However, this will increase the total hardware requirements and pitfalls of wormhole switching has been presented in [3].

We propose a hybrid network that uses a common protocol and increases the throughput using dedicated local connections. Our approach is to keep communication simple, using the main part of the area resources to implement memory and processing cells. The path between sender and receiver is deterministic, which simplifies the router implementation. We also propose the use of distributed memory to handle communication between processing cells, avoiding large routing buffers in the network.

Simulation methodology - Traditionally, computer network simulations have been used to simulate NoC communication [5]. However, this approach focuses only on network communication and omits other system aspects required for design exploration, such as implementation of processing elements and accurate hardware modeling. To cope with the complexity and diversity of the system design aspects, a suitable description language and simulation environment is required. SystemC is one of the candidates to address all of these aspects [6]. Over the past years, SystemC has gained more interest in both academia and industry due to the capability of co-simulating software and hardware and a simple way of refining abstract models down to clock cycle accurate hardware. Simulation frameworks and NoC models based on SystemC has been presented in [11] and [8], but focusing mainly on communication. In contrast, we are currently developing a simulation and exploration tool based on the Open SystemC Initiative (OSCI) SystemC library to build, configure, and explore NoC architectures. Modules are currently being developed to construct larger Network-on-Chip architectures, which will be described in Section 4.2. We propose using a design methodology based on SystemC Transaction Level Modeling (TLM). As a result of raised abstraction level, model descriptions become more flexible and simulates faster.

3. Our approach

This section presents a hybrid interconnect network and shows how resource cells are instantiated and connected. Our proposed topology is an interleaved mesh of *resource cells* (RC), divided into *processing cells* (PC) and *memory cells* (MC). Processing cells can consist of simple dataflow components or more complex architectures such as computational kernels with a local routing network or processor like building blocks. Resource cells in general have local communication links from all four sides, connecting to its neighboring cells. The memory cell has an additional port connecting to a local router which forward communication to other parts of the systems. The structure constituting of two PC, two MC and a router is referred to as a *basic cell*, connected together to form larger structures. Figure 1(c) shows our proposed architecture consisting of four basic cells and totally five routers. The following section will describe the hybrid interconnects, memory organization, and communication approach.

3.1. Hybrid interconnect network

We propose a topology that splits the interconnect network into local communication and a global network. Local communication use dedicated links to connect neighboring

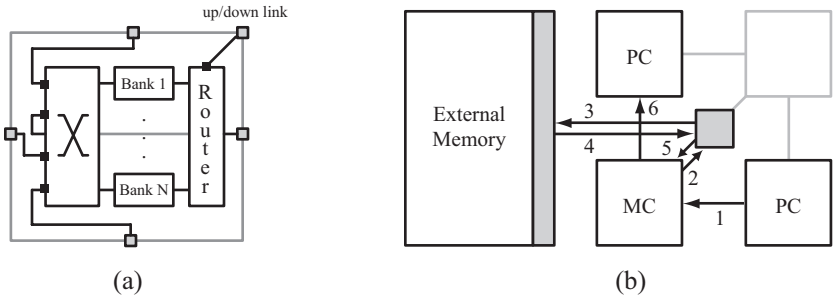

Figure 2. (a) Internal building blocks in the memory cell, consisting of a port switch and four memory banks. Each memory bank has a unique ID for routing data over the network. (b) Two memory cells using the external memory interface to transparently emulate a larger buffer.

cells. The global network connects any two cells using hierarchical routing. The motivation for this topology is the fact that an optimally mapped DFG often results in a high degree of local communication. In contrast, the mesh architecture has a shared network for both local and global communication, where even neighboring cells communicate over the global network. It also means that global communication consume bandwidth from local communication. The proposed topology can be viewed as two independent networks interacting through network routers, shown in Figure 1(c).

3.2. Distributed memory cells

Most algorithms require storage in form of buffering, data reorder, or delay feedback loops. The reason to introduce distributed memories in our proposed architecture is twofold. First, having memories placed close to the processing cells reduce both contention for the global network and costly communication to any external memory. Second, keeping data in distributed memories improves data delivery required to utilize the processing resources, since distributed processing advocates distributed memories. To fully utilize the computational power of the processing cells, the distributed memory cells need to handle multiple simultaneous requests. This is accomplished by placing multiple memory banks inside each memory cell which further improves the communication capacity, as shown in Figure 2(a). Each memory bank is assigned a unique ID and traffic is always routed from one memory bank to another memory bank. A switch is used to control data flow between the four ports and the memory is also connected to the local router in the basic cell.

Another advantage having memories distributed is that they can function as an alternative to large output queues in the network routers. Instead of dividing the total memory resources in two static pools, i.e. local memory and router memory, local memory will be assigned for either of these tasks.

In some of the studied architectures, memory is assumed to be either inside the processing cell, on the boundary of the array, or as external memory banks. Placing memories inside the processing cell results in poor memory utilization, since it can never be shared between cells. Placing memory on the boundary results in that memory transactions are going over the global network, which also puts restrictions on resource mapping. These internal/external memory approaches will not offer the same advantages as distributed memory. However, external memory can function as a way to emulate larger buffer by connecting it between two memory cells. Data is streamed to external memory from one memory bank and streamed back from external memory to another memory bank, as shown in Figure 2(b). More about external memory will be presented in Section 3.4.

3.3. Communication

In the proposed architecture, memory cells are used to separate processing from communication. Each port on a processing cell is connected to a neighboring memory cell, which handles the communication with other parts of the system. A memory bank can be configured to forward incoming data to any arbitrary memory bank (global communication), or to simply supply the neighbor processing cell with the received data (local communication).

Multiple input and output ports are required to implement many signal processing algorithms. One example is the butterfly and delay feedback structure found in pipeline FFTs, which requires streaming from two input ports to two output ports, as shown in Figure 3(a). Mapping this structure to a traditional mesh topology results in poor utilization of processing cells, since the cell I/O becomes the bottleneck of local communication. In the proposed architecture the throughput of communication and processing is balanced, resulting in full utilization. Figure 3(b-c) shows

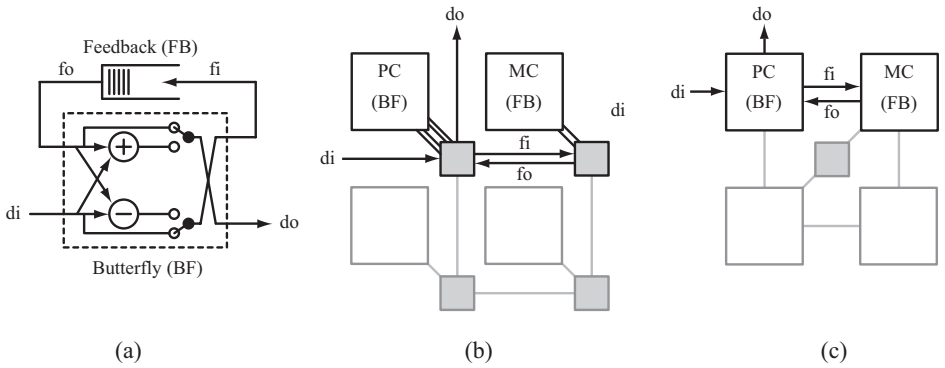

Figure 3. (a) Butterfly and delay feedback memory, (b) Mapped to a mesh architecture, the communication bandwidth limits the performance and resulting in a utilization of 50%, (c) Mapped to the proposed architecture, the communication and processing is balanced resulting in 100% utilization.

the mapping of the butterfly structure.

Several proposed router architectures contain output buffer queues to temporarily store packets traveling through the network [2]. When the communication load increases the buffer queues fills up, resulting in dropped packets or poor throughput. The reason is that a packet inside the communication network is only half-delivered, still waiting for the next router to accept outgoing packets. In case the receiving node is unable to accept a packet, it is trapped inside the communication network, resulting in increased latency in other parts of the network. In many cases the receiving node is assumed to always accept data, a scenario that is not realistic. In our approach there is only one single valid path to route network traffic. To change the routing path, the sender must select a different receiving memory bank to reach its final destination. This means that routing is a simple task of forwarding data between memory cell and network routers. If an incoming address is not known by the router, the packet is send to the default port which is upwards in the hierarchical routing network, as shown in Figure 4(a). Once the path is recognized by a router, the packets are propagated downwards until reaching its final destination.

3.4. Connecting to external memory

Local memory can store data during computations, but the data to be processed is usually located in on-chip scratch pads or in external memory. To fully benefit from the computational power of the mesh, data has to be streamed in and out at high speed. External memory interfaces can be connected to any router in the network and accessed using a unique ID in the same way as memory banks, which is shown in Figure 4(b). Every memory bank communicating with the external memory can setup a transfer to or from the

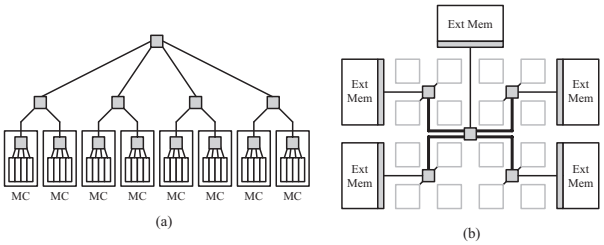

Figure 4. (a) Hierarchical connection of routers to memory cells, (b) On-chip scratch pads or external memory connected to routers. The designer specifies where to connect memory controllers for external memory.

memory. This transfer can be either a linear memory access or a more complex stream specified using stride, span, and skip parameters. The external memory controller handles the memory access mode and provides the memory bank with data.

External memory also functions as an extended buffer when required. Local memory is limited and sometimes larger memory buffers are required between computations. A transfer between two processing cells is usually routed from one memory cell to the other, but can transparently be connected through an external memory controller to act as a large intermediate buffer, shown in Figure 2(b).

4. SystemC TLM modeling

This section presents an extension to the OSCI SystemC simulator to enable user interactive simulations. It also presents the architectural modeling approach using the

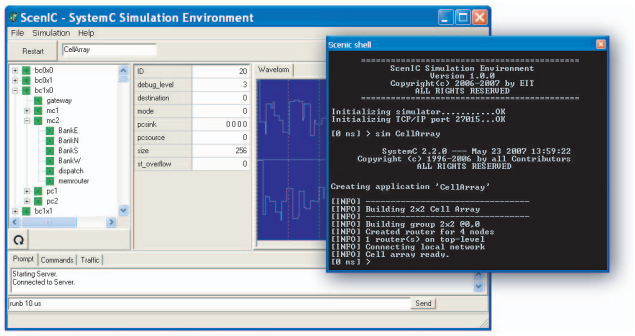

Figure 5. (a) Graphical user interface connecting to SCENIC using a TCP/IP socket, (b) SCENIC console window with both command line and scripting interface.

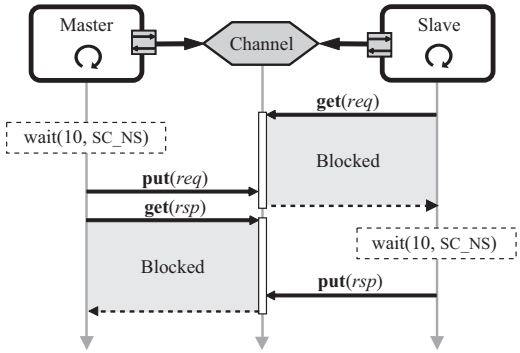

Figure 6. Communication between a master and slave module connected with a master/slave channel (containing two FIFO instances). Function calls are from the module to the channel. The OSCI-TLM library provides channels implementing functions to exchange transactions between modules.

OSCI Transaction Level Modeling (TLM) library.

4.1. Simulation environment

A SystemC Environment with Interactive Control (SCENIC) has been constructed to evaluate NoC architectures. SCENIC is based on the OSCI SystemC library and extends the functionality with features to for example control the simulator, interact with simulation modules, and to extract performance information during run-time. The simulator is controlled through a command line user interface, script files, or through a socket connection to a graphical user interface or Matlab. The graphical user interface and the SCENIC command shell are shown in Figure 5(a-b) respectively.

By extending the functionality of basic SystemC modules (SC_MODULE), the user modules can be configured and accessed from the user interface. An extended SystemC module class (SCI_MODULE) encapsulates features to monitor internal variables in user modules, including both current and historical values. The controllability and observability enables fast scripting to setup different scenarios and to evaluate the system performance.

The SCENIC environment also supports dynamic reconfiguration, useful for dynamically loading processing elements with new configurations. It is based on a reconfigurable module which allows dynamic modules (SCI_DYNAMIC_MODULE) to be inserted during run-time. Reconfigurable modules can be configured either from command line and script, or within the simulation itself.

4.2. Simulation models

Simulation models are described as Transaction Level Models (TLM). The major advantages with TLM are high simulation speed and simulation models that are easy to modify compared to Register Transfer Level (RTL) models. In TLM, pin-accurate models are replaced with an abstract communication layer, which uses *transactions* to transfer data. A transaction is a data structure that contains information about the transfer, for example a structure with memory address, transfer size and payload. The simulation speed increases since processes are activated or resumed only when transactions are exchanged between modules.

Our modules communicate using channels and interfaces specified by the OSCI TLM standard [9]. These channels provide a communication mechanism based on blocking and non-blocking *put* and *get* functions. The get function moves transactions from the channel to the module, while the put function moves transactions from the module to the channel. Channels are implemented as first-in-first-out (FIFO) queues with a configurable buffer depth. Blocking functions will block the calling process until the condition for moving the transaction is satisfied. In the non-blocking versions, a boolean value is returned to indicate if the call was successful, but returns immediately even if the condition is not satisfied. There are also convenience functions to peek the channel FIFO without consuming any data, and special events that are triggered when transactions are sent and received.

Figure 6 shows a point-to-point communication example between a master and a slave device, using separate FIFO channels in each direction. The slave module blocks on a call to the get function until the master puts a *request* after 10 ns. The master blocks while waiting for a *response*, which is provided by the slave after another 10 ns. Note that function calls are always made from the module to the chan-

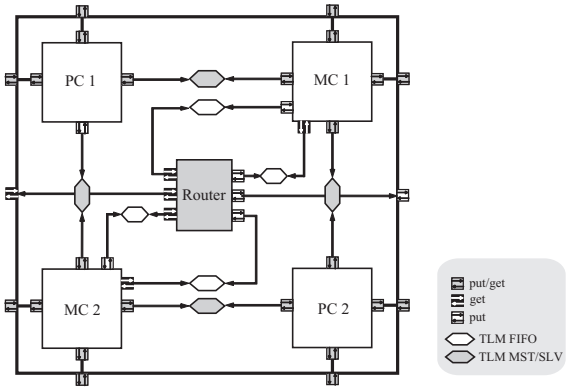

Figure 7. A basic cell containing two processing cells, two memory cells and a router for global communication. All modules in the system communicate through master-slave or FIFO channels. Master-slave channels contain two FIFO channels to support bidirectional communication.

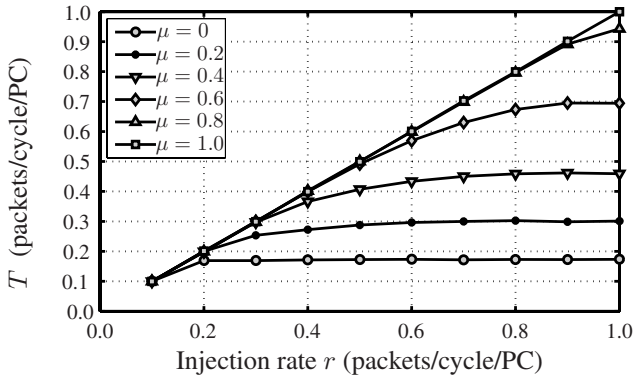

Figure 8. Accepted traffic as a function of injection rate and localization.

nel, but the flow of information depends on the function.

Figure 7 shows modules, ports, bindings, and channels inside a basic cell. TLM FIFO is one-directional and contains only a single FIFO, while TLM Master/Slave contains one FIFO in each direction. Modeling of channels as distinct modules is a design abstraction used to separate communication and behaviour. As a model is refined to a synthesizable description the functionality implemented in channels are separated into master and slave part and moved into the respective module [10]. The buffer size in all channel FIFOs seen in Figure 7 is one transaction. However, channels connecting to routers store a number of transactions to improve throughput. A transaction contains a destination address, a payload and a payload type specification. The type specification is needed as the payload could be data, memory address, dynamic configuration, or information exchanged between network routers.

Each module contains a set of parameters that can be used for design exploration. These parameters are for example, link and router capacity, clock period and latency. Some parameters directly relate to physical entities such as clock period to execution time and memory size to hardware area. The goal of exploration is to understand how the system parameters affect performance, area and functionality of the architecture.

5. Experiments and Results

This section presents simulation results and performance analysis of the proposed hybrid interconnect network. SCENIC is used to configure modules to define a scenario and to reconfigure modules during simulation (controllability). Throughput, latency, and other performance metrics are extracted during simulation to evaluate the system (observability).

Processing cells are configured to operate as traffic generators, which send random packets to neighboring cells and to the global network. Packets are annotated with timestamps to keep track of when packets were produced and injected into the network. The traffic generators also monitor the incoming traffic, counting the number of received packets, and calculating the transport latency as the number of clock cycles from successful injection to final consumption. The throughput is measured by counting packets consumed at the final destination. Since communication is both local and global, a localization factor μ is defined as the ratio between local and global communication, where $\mu = 1$ corresponds to pure local communication and $\mu = 0$ corresponds to fully global communication.

The traffic generators inject packets into the network according to the Bernoulli process, which is a commonly used injection process to characterize a network. For traffic injected into the global network the traffic pattern is modeled as uniform spatial distribution, which means that every processing cell communicates with every other processing cell with equal probability. *Injection rate*, r, is the number of packets per clock cycle and per processing cell injected into the local or global network. Throughput is the *accepted traffic*, T, measured in packets per clock cycle and per processing cell. Ideally, accepted traffic should increase linearly with the injection rate. However, due to traffic contention in the global network the amount of accepted traffic will saturate at a certain level.

Figure 8 shows accepted traffic for a network with 2×2 *basic cells* were all routers and links between routers have a capacity of one packet per clock cycle. Accepted traffic is

measured for $0.1 \leq r \leq 1$ and $0 \leq \mu \leq 1$. When $\mu = 1$ there is no traffic contention and the network achieves the optimal linear relationship between injection rate and accepted traffic. When $\mu = 0$, the saturation point for accepted traffic is $T = 0.17$ packets/cycle/PC. Assuming a realistic localization factor, $\mu = 0.8$, throughput is 94% of the optimal performance.

Figure 9 shows the average transport latency which is measured using the same injection process and traffic pattern as for the throughput measurement. The average transport latency is defined as $L = \sum L_i/N$ where L_i is the transport latency for packet i and N is the total number of packets consumed at destination after local or global transport over the interconnect network. Transport latency is measured as the number of clock cycles between successful injection into the network and consumption at the final destination. As shown in Figure 9, the average transport latency saturates at $L = 100$ clock cycles for a localization factor $\mu = 0$. The latency is bound by the available buffer capacity inside the network. Since routers use round robin arbitration, a packet is guaranteed to be delivered within $\sum B_i$ clock cycles, where B_i is the buffer capacity in router i.

In data-driven and streaming applications, latency has small impact on system performance. However, due to random access and feedback loops, the impact of latency can not be omitted in realistic applications. Hence, for these applications latency is an important parameter for the overall performance. With the realistic localization factor $\mu = 0.8$, the average transport latency for the proposed network is only 4 clock cycles.

The experiments show the advantages when combining high-performance local communication with a flexible global network for realistic localization factors. This is useful in reconfigurable computing, were the flexibility can be utilize during algorithm mapping to achieve a high localization factor. Functional units with high communication rate are mapped to adjacent processing cells, or to closely located processing cells on the hierarchical network.

6. Conclusions

We have presented our ongoing work on high-performance reconfigurable architectures and a SystemC simulation environment for design exploration. We have presented concepts on interconnect networks, memory distribution and communication, and proposed a simulation methodology based on transaction level models, which significantly improve design time and increase simulation speed. Experiments show that the hybrid network achieves a high throughput for realistic localization factors. In reconfigurable computing, high localization factors can be achieved by mapping functional units with high communication rate to adjacent processing cells.

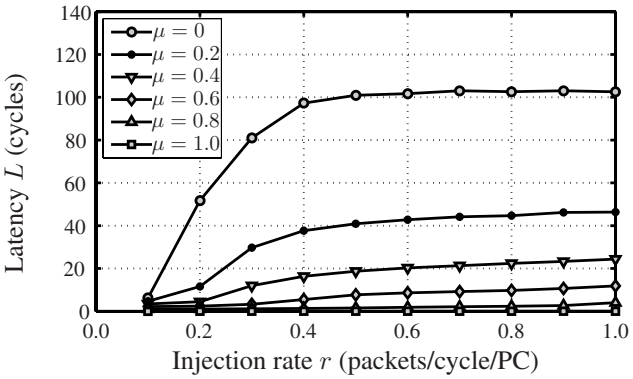

Figure 9. Average transport latency as a function of injection rate and localization.

References

[1] L. Bononi and N. Concer. Simulation and analysis of network on chip architectures: ring, spidergon and 2D mesh. In *Proc. of the 6th International Conference on Design Automation and Test in Europe*, 2006.

[2] J. Chan and S. Parameswaran. NoCGen: A Template Based Reuse Methodology for Networks on Chip Architectures. In *Proc. of the 17th International Conference on VLSI Design*, pages 717–720, 2004.

[3] Y. Chen. Cell Switched Network-on-Chip – Candidate for billion-transistor System-on-Chip. In *Proc. of the International SOC Conference*, pages 57–60, 2006.

[4] K. Compton and S. Hauck. Reconfigurable computing: A survey of systems and software. *ACM Computing Surveys*, 34(2):171–211, 2002.

[5] A. Hegedus, G. M. Maggio, and L. Kocarev. A ns-2 simulator utilizing chaotic maps for network-on-chip traffic analysis. In *Proc. of ISCAS*, pages 3375–3378, 2005.

[6] Open SystemC Initiative (OSCI). OSCI SystemC 2.2 Open-source Library. http://www.systemc.org.

[7] Partha Pratim *et al.* Performance Evaluation and Design Trade-Offs for Network-on-Chip Interconnect Architectures. *IEEE Trans. Comput.*, 54(8):1025–1040, Aug. 2005.

[8] A. Portero, R. Pla, and J. Carrabina. SystemC Implementation of a NoC. In *Proc. of IEEE International Conference on Industrial Technology*, pages 1132–1135, 2005.

[9] A. Rose, S. Swan, J. Pierce, and J.-M. Fernandez. *Transaction Level Modeling in SystemC*. OSCI-TLM WG, 2006.

[10] J. A. Rowson and A. Sangiovanni-Vincentelli. Interface-based design. In *Proc. of Design Automation Conference*, pages 178–183, 1997.

[11] Xu Ningyi *et al.* A SystemC-based NoC Simulation Framework supporting Heterogeneous Communicators. In *Proc. of the 6th International Conference on ASIC*, pages 1032–1035, 2005.