NNSE: Nostrum Network-on-Chip Simulation Environment

Zhonghai Lu, Rikard Thid, Mikael Millberg, Erland Nilsson and Axel Jantsch
Laboratory of Electronics and Computer Systems
Royal Institute of Technology, Sweden
{zhonghai,thid,micke,erlandn,axel}@imit.kth.se

Abstract

A main challenge for Network-on-Chip (NoC) design is to select a network architecture that suits a particular application. NNSE enables to analyze the performance impact of NoC configuration parameters. It allows one to (1) configure a network with respect to topology, flow control and routing algorithm etc.; (2) configure various regular and application specific traffic patterns; (3) evaluate the network with the traffic patterns in terms of latency and throughput.

1 Introduction

Network-on-Chip (NoC) architects face many challenges coming from the huge architectural design space and time-to-market/time-in-market pressures. One major difficulty is to select a communication network that suits a specific application or a range of specific applications with the constraints of cost, power and performance.

Design decisions are typically made on the basis of simulation before resorting to emulation or implementation since it is cheap and flexible. To make a right decision on the network architecture, a simulation tool should enable to (1) faster explore the architectural design space; (2) assess design quality regarding performance, cost, power and reliability etc.; (3) evaluate extensively with various regular traffic patterns and application-oriented traffic. Currently no public simulation tool exists to aid NoC designers to make the decision. Our configuration and simulation tool NNSE aims to fill this gap. NNSE stands for Nostrum NoC Simulation Environment in which Nostrum is the name of our NoC concept [1]. By parameterizing network and traffic configurations, NNSE allows one to configure networks and evaluate them with various traffic configured.

The rest of the paper is structured as follows. We first describe the simulation kernel in Section 2, then introduce the network configuration in Section 3 and the traffic configuration in Section 4. The network evaluation flow is given in Section 5 with the assistance of an evaluation example. Finally we conclude the paper in Section 6.

2 The simulation kernel

The tool NNSE logically comprises a NoC simulation kernel wrapped with a graphical user interface (GUI) written in Python. The kernel called Semla (Simulation EnvironMent for Layered Architecture) [2] provides a layered network simulation engine developed in SystemC. The benefit of layering lies in that changing the implementation of a layer does not interfere with the layer above provided that the interface maintains. Semla is programmable as to network topology and size, process-to-node mapping, and traffic generation etc.

Following ISO's OSI model, the kernel implements five communication layers, namely, the physical layer (PL), the data link layer (LL), the network layer (NL), the transport layer (TL) and the application layer (AL). The upper three layers are shown in Figure 1, where TG/S stands for traffic generator/sink, and glue is the TL component which does packetization/packet-assembly, message queuing etc.

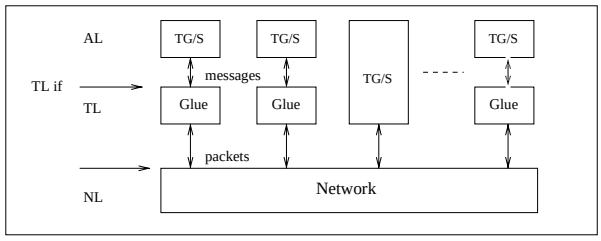

Figure 1. Communication layers

The transport layer offers transaction-level communication primitives as interface to enable communication via channels between application processes. A channel is a transaction-level modeling entity which allows simplex communication from a source process to a destination process. In Semla, we defined and implemented a set of communication primitives for message passing as follows:

- *ch* open(int src_pid, int dst_pid)*: it opens a channel between a source process *src_pid* and a destination process *dst_pid*. The method returns a channel handler which is a class including a unique channel identity number *cid* upon successfully opening the channel. The current implementation opens channels statically during compile time and the opened channels are never closed through simulation.

- *bool nb_write(int cid, void msg)*: it writes *msg* to channel *cid*. The size of messages is bounded. It returns the status of the write. The write is nonblocking.

- *bool nb_read(int cid, void *msg)*: it reads channel *cid* and writes the received protocol data unit to the address starting at *msg*. It returns the status of the read. The read is nonblocking.

Each layer may be configured with a set of parameters that captures its characteristics. The simulation tool currently supports to configure the network layer and the application layer. The GUI provides a convenient way to specify parameters for the network and traffic configuration.

3 Network Configuration

A network is characterized by topology, flow control scheme (switching mode) and routing algorithm etc. Each of them has a large design space on its own. With the tool, all of these characteristics are parameterized, as shown by the configuration tree in Figure 2.

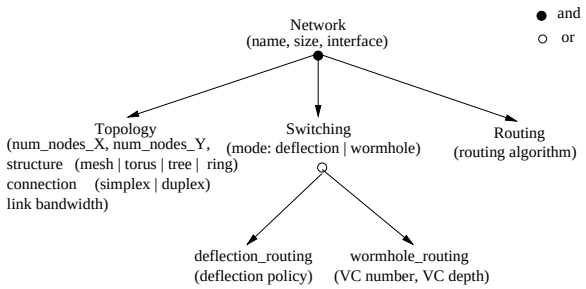

Figure 2. Network configuration tree

We consider on-chip packet-switched networks with regular topologies such as 2D meshes/tori, rings, trees etc. The benefit of the topological regularity is that the network nodes can be identified more structurally and with less bits. So far the tool realizes only a limited set of configurations, which are as follows:

- Topology: 2D mesh and 2D torus.

- Flow control: Wormhole routing and deflection routing. For wormhole routing, one can choose virtual channel (VC) parameters like the number and depth of VCs. For deflection routing, one can specify a deflection policy.

4 Traffic Configuration

4.1 The traffic configuration tree

Network messages (traffic) can typically be characterized and constructed by considering their distributions along the three dimensions: *spatial distribution*, *temporal characteristics*, and *message size specification*. The spatial distribution gives the communication partnership between sources and destinations. The temporal characteristics describe the message generation probability over time. The size specification defines the length of communicated messages. We use a traffic configuration tree to express the elements and their attributes of traffic in Figure 3.

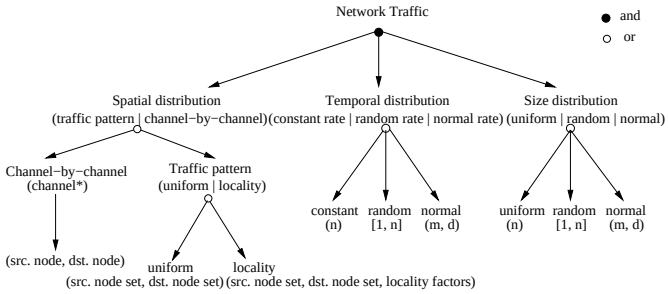

Figure 3. Traffic configuration tree

By the spatial distribution, traffic is classified into two categories: *traffic pattern* and *channel-by-channel* traffic. Traffic patterns consist of uniform and locality traffic. Each simulation cycle, the destinations of a traffic pattern may vary. That is to say, the same source node may send messages to a different channel. With a traffic pattern, all the channels share the same temporal and size parameters. In contrast, channel-by-channel traffic consists of a set of channels with each channel taking its own temporal and size parameters. The temporal distribution has a list of items such as constant rate (periodic), random rate, and normal rate etc. The size distribution has a list of items such as uniform, random, and normal. As can be observed, these lists are just examples of possible distributions. Other useful distributions can be integrated into the tree with their associated parameters. By the tree, each traffic configuration can be set with a list of parameters.

Please note that, although we have divided the traffic configuration into three independent axes, it also allows one to configure traffic by jointly considering two axes. For example, the configuration of burstiness traffic may involve both the time and size axis.

4.2 Traffic Patterns

Two types of traffic are considered in the tree. One is *uniform* traffic , the other *locality* traffic. In order to express them formally, we use a uniform representation.

Suppose the distance between a source node (x_s, y_s) and a destination node (x_d, y_d) is d, we define communication distribution probability $DP_{(x_s,y_s)>(x_d,y_d)}$ as a relative probability to a common probability factor P_c ($0 \leq P_c \leq 1$) in Equation 1 and 2:

$$DP_{(x_s,y_s)>(x_d,y_d)} = coef \cdot P_c \quad (1)$$

$$coef = 1 + \frac{\alpha}{d+1} \quad (2)$$

where *coef* is the *distribution coefficient*; α is called *locality factor*. Since $DP_{(x_s,y_s)>(x_d,y_d)} \geq 0$, $\alpha \geq -(d+1)$. Particularly when $\alpha = -(d+1)$, $DP_{(x_s,y_s)>(x_d,y_d)} = 0$; when $\alpha = 0$, $DP_{(x_s,y_s)>(x_d,y_d)} = P_c$. Besides, when $-(d+1) < \alpha < 0$, $DP_{(x_s,y_s)>(x_d,y_d)}$ is proportional to the distance d; When $\alpha > 0$, $DP_{(x_s,y_s)>(x_d,y_d)}$ is inversely proportional to the distance d. In order to have a symmetric coefficient *coef* when $\alpha > 0$ and $\alpha < 0$, we constrain the distribution coefficient *coef(d)* to be not greater than 2, then the locality factor falls into the region $[-(d+1), (d+1)]$ and $0 \leq P_c \leq 0.5$.

If all the source nodes' locality factors α are zero, their distribution coefficients *coef* are one. In this case, the traffic is uniformly distributed.

4.3 Channel-by-channel traffic

Channel-by-channel traffic differs from the traffic patterns in that the traffic's spatial pattern is built on per-channel basis and static. This type of traffic is used to construct application-oriented workloads. The temporal characteristics and message size specification can be approximated using analysis or communication traces. The set of traffic parameters of a channel is $\{s\_node, d\_node, \mathcal{T}, \mathcal{S}\}$, where s_node represents the source node, d_node the destination node, $\mathcal{T}$ its temporal characteristics, and $\mathcal{S}$ is its message size specification.

5 Performance evaluation

5.1 The network evaluation flow

After configuring a network and a traffic pattern, one can evaluate the network with the traffic pattern. The evaluation is based on the kernel simulation results which can be shown as figures and textual statistics in NNSE. The network evaluation flow may be iterative until satisfaction as illustrated in Figure 4. The main performance measures are latency and throughput.

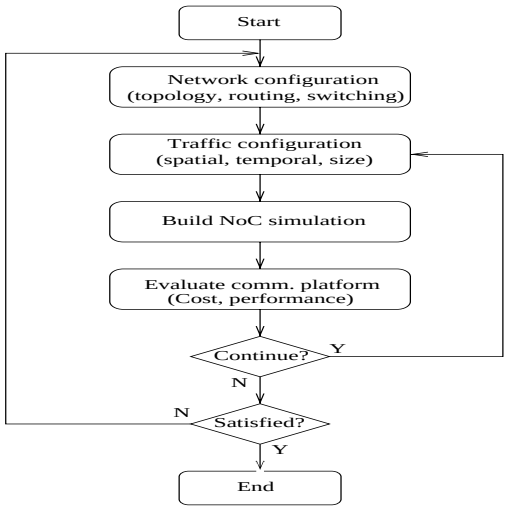

Figure 4. Network evaluation flow

We denotate the number of network nodes as M, the link capacity as C, the number of simulation cycles as T, the number of flits injected/offered into the network is N_{in}/N_{of}, the number of flits ejected from the network is N_{out}. Suppose that the shortest distance a flit i travels is D_i, then the total shortest distance to be traveled by all offered flits is $D_{of} = \sum_{i=1}^{N_{of}} D_i$; the total shortest distance traveled by all ejected/received flits is $D_{out} = \sum_{i=1}^{N_{out}} D_i$. We define the following terms for performance evaluation:

$$offered\_load = \frac{D_{of}}{C \cdot T} \quad (3)$$

$$link\_utilization = \frac{D_{out}}{C \cdot T} \quad (4)$$

$$flit\_injection\_rate = \frac{N_{in}}{M \cdot T} \quad (5)$$

$$throughput = \frac{N_{out}}{M \cdot T} \quad (6)$$

For a 2D $K \times K$ mesh, $M = K^2$, $C = 4K(K-1)$; For a 2D $K \times K$ torus, $M = K^2$, $C = 4K^2$.

5.2 An evaluation example

Figure 5 shows the performance of a 4x4 mesh network under uniform and locality traffic. The network employs wormhole-based virtual-channel (VC) flow control with dimension-ordered XY routing, which is deterministic and deadlock free. Switches operate synchronously with each hop taking one cycle. The number of VCs per physical channel is 4 and the depth of a VC is 2. The network diameter is 6. All the network nodes are both message sources and sinks. The source nodes inject packets into the network via

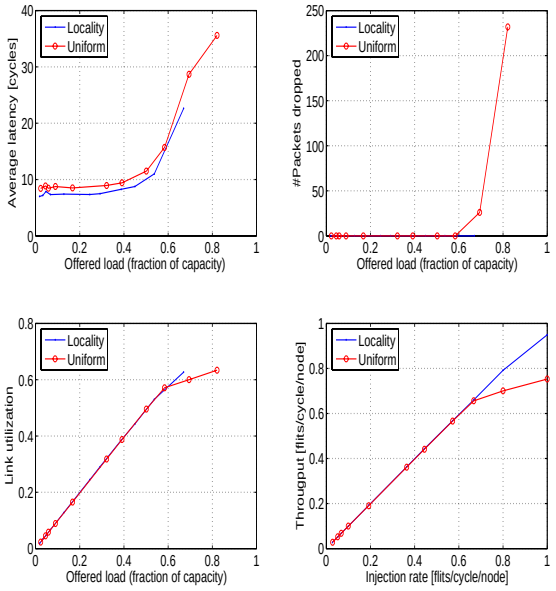

Figure 5. A performance evaluation example

bounded FIFOs with a constant rate, as illustrated in Figure 6. One message contains only one packet and each packet is split into four flits.

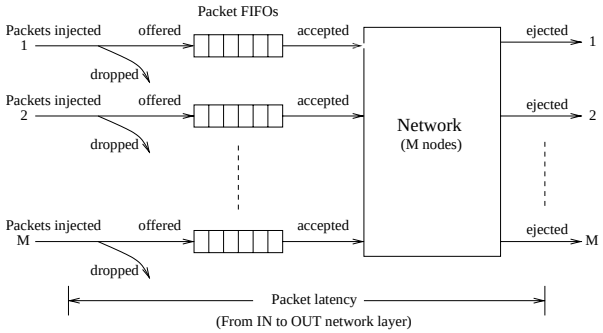

Figure 6. The packet injection model

By the traffic configuration tree, both traffic belong to periodic traffic with a fixed message size. With the uniform traffic, a network node sends packets to other nodes with the same probability. With the locality traffic, a network node sends packets to its *nearer* nodes with higher probability. We list the traffic's locality factors α and calculated distribution coefficients *coef* in Table 1.

The four figures in Figure 5 show average packet latency v.s. offered load, number of packets dropped v.s. offered load, link utilization v.s. offered load, and throughput v.s. flit injection rate, respectively. As can be seen, the network shows significant performance improvement if the traffic is more locally distributed. We can conclude that, with the dimension-order routing, the wormhole network can efficiently benefit from traffic locality. This implies that mapping to achieve traffic locality is crucial.

TRAFFIC	d	0	1	2	3	4	5	6
Locality	α	-1	0	-1.2	-2.4	-4.0	-5.4	-6.3
	coef	0	1	0.6	0.4	0.2	0.1	0.1
Uniform	α	-1	0	0	0	0	0	0
	coef	0	1	1	1	1	1	1

Table 1. Traffic specifications

6 Conclusion

We have presented our Nostrum Network-on-Chip simulation environment (NNSE). It enables to flexibly configure not only networks but also traffic, and then evaluate the networks with various traffic patterns. Although we list only a limited set of alternatives for network and traffic configuration, the trees could be expanded given the support from the simulation kernel. At this point, it is meaningful to point out several interesting directions for future extensions:

- Fledge the network and traffic configuration tree. Particularly, as the simulation kernel is upgraded, virtual-circuit service is to be integrated. Consequently configuring QoS traffic needs to be supported.
- Integrate cost and power model into NNSE. Design quality has to be evaluated from multiple angles.
- Support investigations on the TL, LL and PL layer. For example, from the TL, we can explore end-to-end flow control; from the LL, we can explore the link bandwidth, asynchrony etc.; from the PL, we can examine physical integrity.
- Incorporate mapping network clients onto network nodes. Mapping potentially impacts network performance and power consumption.
- Develop an implementation flow using, for example, VHDL. In NNSE, network and traffic configurations are stored in an intermediate data format using XML, thus can be shared by the SystemC and VHDL flow.

References

[1] Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network on chip. In *Proceedings of the Design Automation and Test Europe Conference (DATE)*, 2004.

[2] Richard Thid, Mikael Millberg, and Axel Jantsch. Evaluating NoC communication backbones with simulation. In *Proceedings of the IEEE NorChip Conference*, November 2003.