HNOCS: Modular Open-Source Simulator for Heterogeneous NoCs

Yaniv Ben-Itzhak[1] Eitan Zahavi[1] Israel Cidon[2] Avinoam Kolodny[2]
Electrical Engineering Department
Technion – Israel Institute of Technology
Haifa, Israel
[1]{yanivbi, ezahavi}@tx.technion.ac.il [2]{cidon, kolodny}@ee.technion.ac.il

***Abstract*— We present HNOCS (Heterogeneous Network-on-Chip Simulator), an open-source NoC simulator based on OMNeT++. To the best of our knowledge, HNOCS is the first simulator to support modeling of heterogeneous NoCs with variable link capacities and number of VCs per unidirectional port. The HNOCS simulation platform provides an open-source, modular, scalable, extendible and fully parameterizable framework for modeling NoCs. It includes three types of NoC routers: synchronous, synchronous virtual output queue (VoQ) and asynchronous. HNOCS provides a rich set of statistical measurements at the flit and packet levels: end-to-end latencies, throughput, VC acquisition latencies, transfer latencies, etc. We describe the architecture, structure, available models and the features that make HNOCS suitable for advanced NoC exploration. We also evaluate several case studies which cannot be evaluated with any other exiting NoC simulator.**

Keywords-NoC simulator; Heterogeneous NoC

I. INTRODUCTION

Network-on-Chip (NoC) has emerged as a new on-chip communication approach. It offers better scalability, throughput, overall latency and area compared to bus-based on-chip interconnect. However, the NoC design space is very large and high dimensional. It includes the optimization of topology, routing mechanism, congestion control methodologies, link capacities, number of buffers and virtual channels per link, etc. Furthermore, the NoC research area is still at its infancy. Consequently new architectures, techniques and ideas are being proposed, developed and evaluated. Hence, simulators are essential tools in evaluating the performance of different NoC designs and new proposals. Therefore, in order to be able to cover the existing and future NoC diversities, NoC simulators should be modular, scalable, extendible and fully parameterizable.

In this paper we introduce HNOCS, a modular open-source OMNeT++ based NoC simulator. To the best of our knowledge, HNOCS is the first NoC simulator to support heterogeneous NoCs with variable link capacities and number of VCs per each unidirectional port. Heterogeneous NoCs [7] [8] [9] offer better performance compared to homogeneous NoCs since SoCs and CMPs are heterogeneous in terms of module-to-module traffic requirements. HNOCS allows researching and exploring new paradigms and phenomena of heterogeneous NoCs [10].

HNOCS supports parallelism[1] and arbitrary topologies. It includes synchronous, synchronous virtual output queue and asynchronous NoC routers. Several previous NoC simulators

TABLE I. NOC SIMULATORS COMPARISON

Simulator	Framework	Availability	Parallelism	Topologies	Open-Source	**Heterogeneous Support**	Synchronous / Asynchronous
SICOSYS [1]	C++	+	-	Limited	+	**-**	Synchronous
Noxim [2]	SystemC	+	-	Mesh	+	**-**	Synchronous
NNSE [3]	SystemC	+	-	Mesh/Torus	+	**-**	Synchronous
Nirgam [4]	SystemC	+	-	All	+	**-**	Synchronous
gpNoCsim [5]	Java	+	-	All	+	**-**	Both
[6]	OMNeT++	-	+	All	-	**-**	Synchronous
HNOCS	**OMNeT++**	+	+[1]	**All**	+	+	**Both**

[1] In order to obtain parallelism in HNOCS, one must change the types of links to unidirectional [11].

have been presented. To the best of our knowledge, none of them satisfies all the aforementioned requirements that are necessary to cover the design and research space. TABLE I presents a comparison between previous NoC simulators and HNOCS. HNOCS is based on OMNeT++ [11], which is an extensible, modular, open-source component-based C++ simulation library and framework, primarily aimed at building network simulators.

OMNeT++ provides several important advantages utilized by HNOCS. It offers easy traceability, debug utilities to reduce debug-time, and built-in parallelism support to reduce simulation run-time. Moreover, it supports flexible and efficient topology definition using NED, the OMNeT++ topology description language. NED has a simple syntax, yet it is very powerful for defining arbitrary topologies (see Fig. 3). All these advantages give HNOCS the potential to become a highly beneficial and extensible open source simulation platform for the service of the NoC research community.

Availability. HNOCS is available at the OMNeT++ website in the simulation models page [12] and at the following link: http://webee.technion.ac.il/matrics/software.html.

Currently, it supports three router types: synchronous, synchronous virtual output queue (VoQ) and asynchronous. It includes full model documentation and several examples.

The rest of this paper is organized as follows: section II presents the simulator architecture and the available models. In section III, we describe in details the models of HNOCS. Section IV presents performance evaluation examples. We evaluate the end-to-end latency and throughput for uniform traffic pattern. We also present a synthetic non-uniform traffic and heterogeneous NoC examples. In section IV.D, we discuss possible extensions of HNOCS.

II. Simulator Architecture

HNOCS architecture is optimized for extendibility and comparative architecture evaluation. This requirement is translated to several major features of HNOCS: the use of module interfaces, message classes, modules directory, support for arbitrary topologies and module selection as a simulation parameter. In this section we describe each one of these features and the way, in which they are provided by HNOCS.

HNOCS is designed so that different implementations of NoC architecture building blocks could be gradually extended to finally form a rich set of alternative architectures. Therefore, it is required that parts of the model should be easily replaced by new implementations in a manner that does not require the author of the new model to know the details of other blocks internals. It is crucial for such a system to use well defined interfaces between the parts of the model. Such interfaces should be extendible in such a way that several new modules would be enabled to exchange new types of information without breaking the rest of the modules functionality. OMNeT++ provides two main features that enable such architecture: module-interfaces and message-inheritance.

An OMNeT++ module represents a hardware or a software entity that is capable of receiving messages (from itself or other modules) and implements self-contained logic. Modules are declared by specifying their configurable properties and “ports” – where messages can arrive or leave from. A module-interface is a skeleton that has no real module implementation behind it. Modules may declare their adherence to such an interface (meaning the set of ports and properties) and thus may be used in place of the skeletons in a dynamic manner. The list of available interfaces in HNOCS is presented in TABLE II. Each line of the table describes a module interface with its properties and a list of ports. A port which represents a set of indexed ports is marked by “[]”.

Generally, a NoC is built from two main modules: *Routers* and *Network Interfaces* (*NI*) as shown in Fig. 1(a) (named “core[*]” and “router[*]" respectively). Internally the *NI* contains sources and sinks (Fig. 1(c)). Routers hierarchically built as a collection of connected ports. The hierarchical approach allows HNOCS to support heterogeneity; each port is configured according to its capacity and number of VCs. The routers consist of ports (Fig. 1(b)). The ports include input-port (*inPort*), output-port selector (*opCalc*), VC-allocator (*vcCalc*), and scheduler (*sched*) (Fig. 1(d)). The input-port stores the incoming flits in the proper VC buffer. Once a head-flit is in the head of the FIFO buffer, the output-port selector implements the routing decision. Then, the VC-allocator allocates an output VC. The scheduler employs arbitration between the different packets needed to be transmitted through the proper out-port. Actually, this is an implementation of the switch allocation pipe-stage. Section III.A provides more details about the process of flits through the NoC router.

TABLE II. NoC Module Interfaces

Interface	Properties	Ports	Description
NI_Ifc	id	in, out	Network interface
Source_Ifc	srcId	Out	Source of traffic
Sink_Ifc	numVCs	In	Sink of traffic
Port_Ifc	numPorts	in, out, sw_in, sw_out, sw_ctrl_in, sw_ctrl_out	Hierarchical router port
Router_Ifc	id, numPorts	ports in[], out[]	NoC router
Sched_Ifc	numVCs	ctrl[], in[], out[]	Scheduler/Arbiter
InPort_Ifc	numVCs	in, out[], ctrl[], calcVC, calcOp	Router input port
OPCalc_Ifc		Calc	Routing calculation
VCCalc_Ifc		Calc	VC Allocator

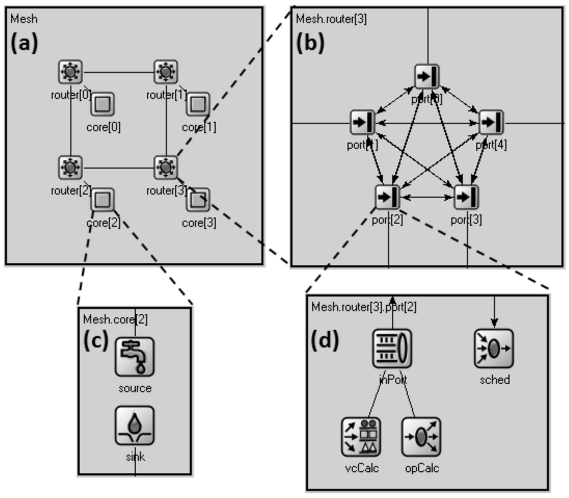

Figure 1. 2x2 mesh NoC: (a) a complete mesh built from routers and cores; (b) a hierarchical router of 5 ports; (c) a core structure; (d) a single port structure.

Fig. 2 depicts the HNOCS modules directory structure for each module shown in Fig. 1. For instance, one can select to use the *opCalc* module as static (i.e. deterministic routing). In order to use a new routing scheme, one should implement new module for *opCalc*.

HNOCS uses “message” objects as its scheduled events. “Messages” are callback functions to be invoked when the event reaches the top of the event wheel. In addition, they carry topological information, such as the source module, the output port, the message sent on, the current module, the message arrived to, etc. In addition, OMNET++ provides a rich set of utilities and APIs acting on messages. These are, for example, time-stamping (used for latency calculations), tracking ownership, management, visualization and logging. A standard set of messages is defined in HNOCS in order to facilitate extendibility. Similarly to module-interfaces, OMNeT++ provides means to sub-class messages using a mechanism similar to C++ inheritance. All HNOCS modules should be designed to accept a small set of message types on their NoC link ports. Several modules may share message sub-class definition and extend the data delivered by the standard messages without affecting other modules operation. HNOCS

```
src/                  all modules of the project
├cores/               the network interfaces
│ ├sources/           packet generators injecting into the network
│ └sinks/             packet collectors
├routers/             the packet switching logic within the network
│ ├flat/              routers with no internal hierarchy
│ └hier/              hierarchical routers built from a set of "ports"
│   ├sched/           arbiters for selecting which packet wins the out-port
│   │ └wormhole/      schedulers for wormhole packet switching
│   ├inPort/          input port logic including the queues
│   ├opCalc/          routing function – selecting the packet`s out-port
│   │ └static/        deterministic routing
│   └vcCalc/          VC allocator
│     └free/          allocation VC from the set of free VCs
└topologies/          collection of common topologies
```

Figure 2. HNOCS modules directory structure and classification

```
network Mesh {
   parameters: string routerType;  string coreType; int columns;  int rows;
   submodules:
      router[columns*rows]: <routerType> like Router_Ifc {
         parameters: numPorts = 5;  id = index;
         gates: in[5];  out[5];
      }
      core[columns*rows]: <coreType> like NI_Ifc {  parameters: id = index;  }
   connections allowunconnected:
      for r=0..rows-1, for c=0..columns-1 {
         router[r*columns+c].in[2]   <--> Link <--> router[(r+1)*columns+c].out[0] if r!=rows-1;
         router[r*columns+c].out[2] <--> Link <--> router[(r+1)*columns+c].in[0] if r!=rows-1;
         router[r*columns+c].in[3]   <--> Link <--> router[r*columns+c+1].out[1] if c!=columns-1;
         router[r*columns+c].out[3] <--> Link <--> router[r*columns+c+1].in[1] if c!=columns-1;

         router[r*columns+c].in[4] <--> Link <--> core[r*columns+c].out;
         router[r*columns+c].out[4] <--> Link <--> core[r*columns+c].in;
      }
}
```

Figure 3. A parameterized mesh topology definition in NED language

defines two types of messages that represent traffic on the NoC links: *NoCFlitMsg* representing the data flits and *NoCCreditMsg* the flow control signals. *NoCFlitMsg* messages represent a single flit with its associated small data size. However, they also carry simulation attributes to enable different types of algorithms within the NoC. Flexible and efficient topology definition is natively supported by OMNeT++. For example the topology definition of a flexible size mesh is provided in Fig. 3. The provided mesh definition demonstrates the power of the NED language in the ability to iterate and declare connections using iterator values. The NI and Router modules are selected using the *coreType* and *routerType* parameters which are evaluated during the simulation run. As opposed to the common practice where on each change, a new simulation executable should be built and maintained. When the number of possible combinations for different module implementations grows, the number of executables explodes, thus turning the run-time parameterization into key-factor in delivering a platform for many NoC architectures simulation.

OMNeT++ simulates a wire by describing its latency and capacity. In HNOCS we use the capacity parameter for links that carry flits and the latency parameter for zero size control messages such as credits or router control messages. This allows us to describe a synchronous clocked implementation for the router internal scheduler with fine grain control over exact number of cycles of each function. OMNeT++ protects the data links by enforcing the delivery of a single message at a time preventing any overflow of the link’s defined capacity.

III. Models Description

This section describes the main modules available in the current version of HNOCS.

A. Hierarchical Router

The hierarchical router structure consists of ports which are fully connected (see Fig. 1(c)); each port consists of in-port, VC allocator (*vcCalc*), routing module (*opCalc*), and scheduler (*sched*) (see Fig. 1(d)).

Fig. 4 depicts the process for a new head-flit arrival. When a head-flit arrives at an in-port, it is queued in the proper VC buffer and the routing module calculates the packet`s out-port (1). When the head-flit is in the head of the buffer, the VC allocator assigns a VC over the out-port (2). Then, the in-port sends a request message to the scheduler of the out-port (3). Once the scheduler can transmit the head-flit (depends on the capacity of the port`s outgress link), it sends a grant message to the in-port (4). In return, the in-port sends a credit message to the scheduler of the previous router (5) and also sends the head-flit to the out-port (6). Then, the scheduler sends it to the in-port of the next-router (7). The rest of the packet`s flits are assigned the same out-port and VC of their head-flit. The scheduler sends grant messages to the in-port whenever it can transmit the packet`s flits (8); in return, the in-port sends the next package`s flit or NACK if there are no ready flits to transmit.

The scheduler serves the request messages from the in-ports (per VC) in a FIFO manner. It employs a round-robin (each VC with outstanding flits sends a single flit) or a winner-takes-all arbitration (the winner VC sends all its outstanding flits) [13]. The grant messages are only sent when there are credits for the proper VC.

HNOCS contains three types of hierarchical routers: synchronous, synchronous virtual output queue (VoQ) and asynchronous. The main difference between the synchronous and virtual output queue to the asynchronous routers is at which time grant messages are sent. The synchronous router attempts to send a grant message in every clock cycle; while, the asynchronous router attempts to send a grant message when it gets a request, a flit or a credit message, or after it finishes transmitting a flit. The synchronous and asynchronous NoC routers implement FIFO buffers per VC. The synchronous virtual output queuing (VoQ) router allows packets to bypass packets destined to other output queues. This enables higher utilization since it limits the head-of-line blocking scenario only between VCs going to the same output port. The number of concurrent packets being sent to different output ports is an HNOCS parameter, which can be configured.

B. *Traffic Configuration*

Traffic is configured by setting the destination and packet-arrival time parameters for each source. The destination can be either deterministic or randomly distributed. The distributions are set using the built-in OMNeT++ generation functions (e.g. *intuniform*: uniform distribution OMNeT++ function). The packet inter-arrival times can be either randomly distributed (e.g. exponential, constant) or driven by a trace file. The trace file includes specific times when a packet should be generated. Hence, HNOCS can cover a variety of traffic patterns. We plan to extend HNOCS so that it will simultaneously support different packets generation schemes for each source.

C. *Statistics Collection*

HNOCS provides a rich set of statistical measurements collected by the sink, the source and the in-port modules. The sinks collects throughput and different latency statistics at the flit and package levels. The source collects several source queue indicators in order to identify the point of saturation. The in-port collects VC acquisition latencies (i.e. latency to acquire an out-port VC) and transfer latencies. The user can easily add more statistical measurements and export them to his favorite workspace. In particular, one can easily integrate a power estimation model (e.g. ORION) into HNOCS in order to evaluate NoC power consumption.

IV. PERFORMANCE EVALUATION

This section demonstrates the capabilities, features and run-time of HNOCS. To that end, we present a comparison between three routers implemented using HNOCS (i.e. synchronous, synchronous virtual output queue (VoQ) and asynchronous) for a uniform traffic pattern and for a synthetic non-uniform traffic example. We also present an evaluation for a heterogeneous NoC with variable link capacities and number of VCs per port, which none of the existing NoC simulators is capable of evaluating (see TABLE I).

A. *Uniform Traffic Pattern*

Fig. 5 presents the average throughput and end-to-end latency for a uniform traffic pattern over a 4x4 NoC with two GBps and two VCs for all links. We compare three types of NoC routers: synchronous, synchronous VoQ and asynchronous employ winner-takes-all arbitration. HNOCS

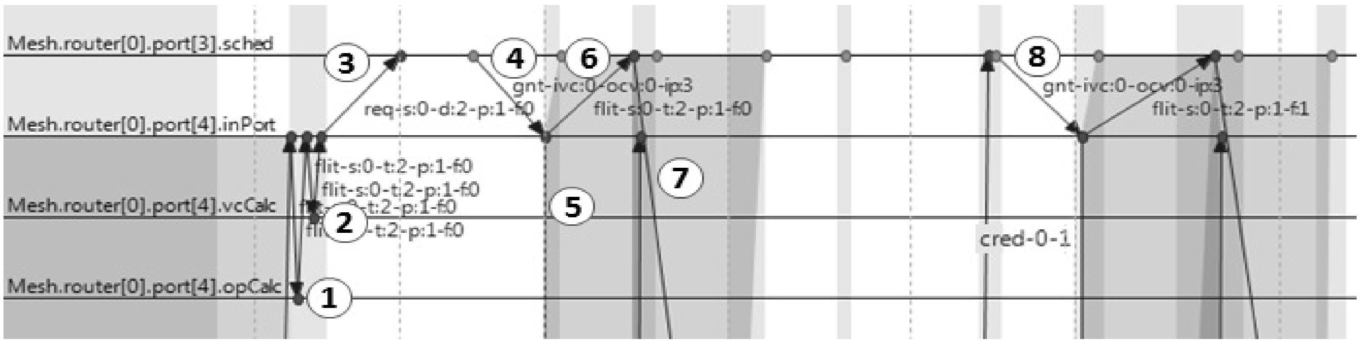

Figure 4. Router transmission process example (OMNeT++ event log viewer snapshot (non linear time scale) *some of the events are overlapping in real time.

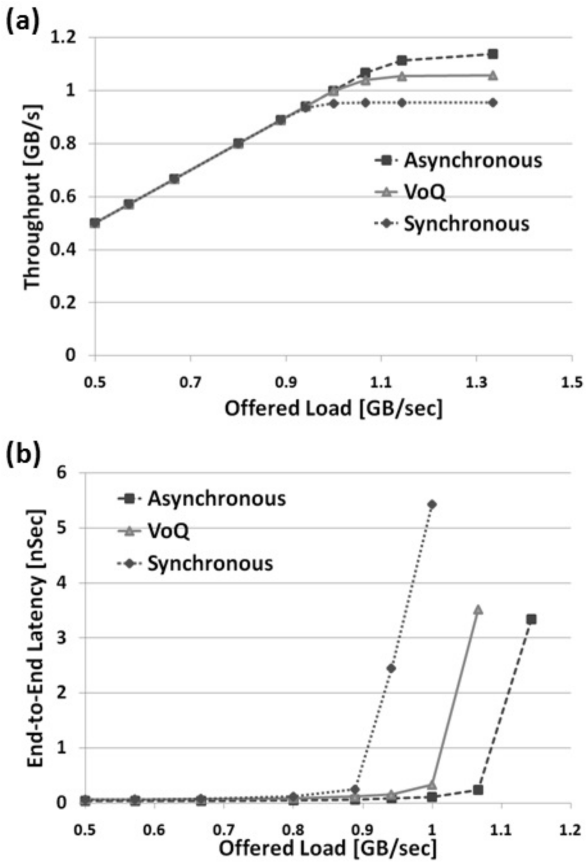

Figure 5. Uniform traffic pattern. (a) Average throughput versus offered load ; (b) Average end-to-end latency versus offered load

modules type parameterization feature enables the use of the same simulation setup for these three different NoC implementations. The results show that under high loads the asynchronous router has the highest throughput, followed by the synchronous VoQ and synchronous routers. This behavior is expected since the asynchronous router incur in lower latency (no clocking or pipeline) as compared to the synchronous VoQ and synchronous routers. The VoQ router (which uses a separate FIFO for each input port, output port and input VC) is expected to outperform the synchronous router which utilizes a single FIFO per each input port VC.

B. Synthetic Non-Uniform Traffic Example

In this section, we present an example for performance evaluation of an asynchronous NoC router, which employs round-robin arbitration, for non-uniform traffic. Fig. 6(a) presents the synthetic example, where all links are 16 Gbps and have single VC. Fig. 6(b) presents the end-to-end latency versus offered load for each flow. It can be seen that flows 1 and 2 incur higher latency compared to the other flows. Furthermore, flow 3 incurs higher latency compared to flow 4 and so on. This phenomenon can be explained by looking at the path acquisition latency (i.e. the time it takes the head-flit to reach the destination) of each flow (Fig. 6(c)). For instance, the path acquisition latency of flow 2 depends on the time that flow 1 occupies the first link. However, the time during which flow 1 occupies the first link

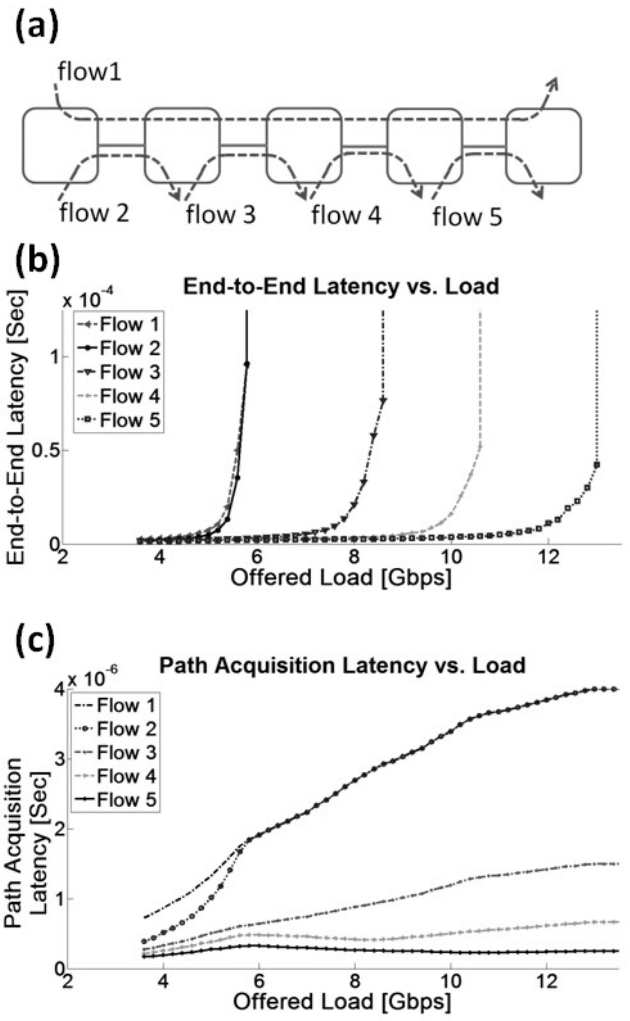

Figure 6. (a) Synthetic Example for Non-Uniform Traffic ; (b) End-to-end latency versus offered load per flow ; (c) Path acquisition Latency versus offered load per flow

depends on its own path acquisition latency which depends on the time that flows 3 to 5 occupy their links [10]. Hence, understanding different scenarios is possible due to a rich set of statistical measurements which can be provided by HNOCS.

C. Synthetic Example for Heterogeneous NoC Evaluation

In this section, we present a synthetic example of a heterogeneous NoC (see Fig. 7) which employs round-robin arbitration. All flows have the same packet generation rate. Unless noted, all links are 16Gbps and have sufficient number of VCs (i.e. at least the number of flows transmitted over it). Fig. 8 presents the end-to-end latency versus the offered load for each flow. It can be seen that flow 1 has the lowest latency. Flows 2-4 incur higher latencies since they are transmitted through link 12 which is only 12 Gbps and through link 7 which has only single VC. Therefore, these flows are transmitted through `bottleneck` links which cause increase of the VC acquiring and transfer latencies.

HNOCS is the only NoC simulator which can evaluate heterogeneous NoCs, such this scenario (see TABLE I).

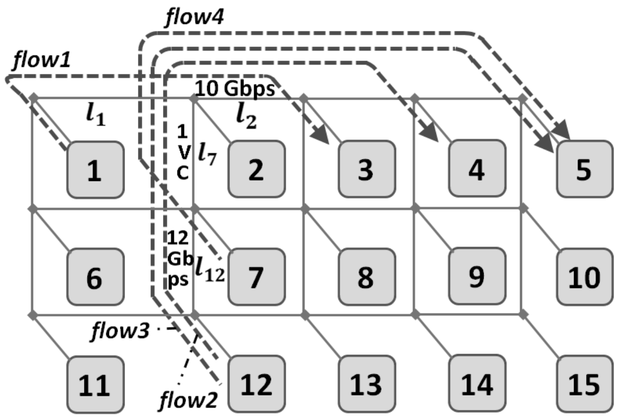

Figure 7. Synthetic example of NoC with non-uniform numbers of VCs and non-uniform capacities of links (Unless noted links are 16Gbps and have sufficient number of VCs). $C_{l_2} = 10\ Gbps; C_{l_{12}} = 12Gbps; V_{l_7} = 1$.

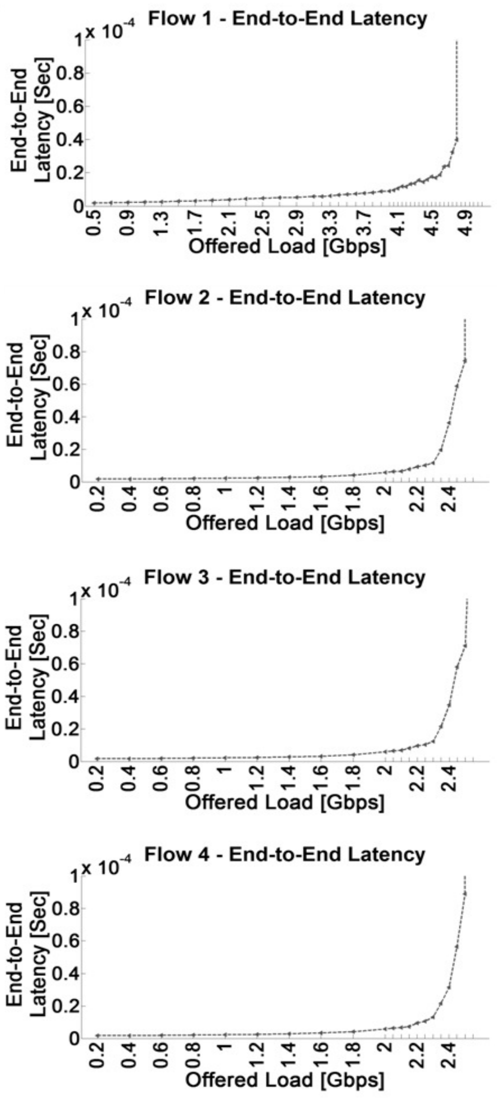

Figure 8. The end-to-end latency of the flows in Fig. 7.

D. *HNOCS Run-Time*

TABLE III presents the HNOCS run-time simulation of 2 ms for both low load and saturated NoCs. The run-time is measured for the uniform traffic pattern evaluation presented in section IV.A , the synthetic traffic examples presented in sections IV.B and IV.C, and for the synthetic example presented in Fig. 9. The latter synthetic example consists of link capacity of 16 Gbps and single VC for all links.

TABLE III. HNOCS SIMULATION RUN-TIME

	Simulation Run-Time [Sec]	
	Low Load	**Saturation**
Uniform Traffic Pattern (Section IV.A) [*Sync / SyncVoQ / Async*]	295 /401 / 272	588 / 872 / 587
Synthetic Non-Uniform Traffic Example (Section IV.B)	57	96
Synthetic Example for Heterogeneous NoC Evaluation (Section IV.C)	45	69
Synthetic Example (Fig. 9)	74	101

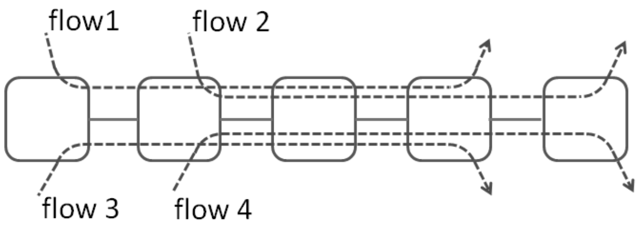

Figure 9. The syntethic example used in TABLE III. All links are 16 Gbps with single VC.

We run HNOCS over a standard desktop computer with Q6600 Intel processor and 2GB memory. Our experience shows that one should repeat the simulation (An OMNeT++ parameter) for five to ten times in order to get low variance and sufficient accuracy.

V. SUMMARY AND FUTURE WORK

A modular NoC simulator based on OMNeT++ framework has been presented and implemented. Several advantages of HNOCS compared to previous NoC simulators have been discussed. It has been shown that HNOCS is the only simulator capable of supporting heterogeneous NoC routers with variable link capacities and number of VCs per unidirectional port. The detailed structure of HNOCS architecture and its model descriptions have been explained. It has been demonstrated that HNOCS offers an open-source, scalable, extendible and fully parameterizable framework for modeling NoCs.

HNOCS is planned for easy extensions in several directions. For instance, supporting different routing protocols; employing different arbitration schemes; implementing various QoS mechanisms; and supporting power/energy estimation.

REFERENCES

[1] V. Puente, J. Gregorio, and R. Beivide, “SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems,” in *Parallel, Distributed and Network-based Processing, 2002. Proceedings. 10th Euromicro Workshop on*. IEEE, 2002, pp. 15–22.

[2] F. Fazzino, M. Palesi, and D. Patti, “Noxim: Network-on-chip simulator,” 2008.

[3] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, “NNSE: Nostrum network-on-chip simulation environment,” in *Swedish System-on-Chip Conference (SSoCC’03)*. Citeseer, 2005, pp. 1–4.

[4] L. Jain, B. Al-Hashimi, M. Gaur, V. Laxmi, and A. Narayanan, “NIRGAM: a simulator for NoC interconnect routing and application modeling,” in *Workshop on Diagnostic Services in Network-on-Chips, Design, Automation and Test in Europe Conference (DATE’07)*, 2007, pp. 16–20.

[5] H. Hossain, M. Ahmed, A. Al-Nayeem, T. Islam, and M. Akbar, “Gpnocsim-A General Purpose Simulator for Network-On-Chip,” in *Information and Communication Technology, 2007. ICICT’07. International Conference on*. IEEE, 2007, pp. 254–257.

[6] R. Al-Badi, M. Al-Riyami, and N. Alzeidi, “A parameterized NoC simulator using OMNet++,” in *Ultra Modern Telecommunications & Workshops, 2009. ICUMT’09. International Conference on*. IEEE, 2009, pp. 1–7.

[7] A. Mishra, N. Vijaykrishnan, and C. Das, “A case for heterogeneous on-chip interconnects for CMPs,” in *Proceeding of the 38th annual international symposium on Computer architecture*. ACM, 2011, pp. 389–400.

[8] A. Bakhoda, J. Kim, and T. Aamodt, “Throughput-effective on-chip networks for manycore accelerators,” in *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2010, pp. 421–432.

[9] M. Kreutz, C. Marcon, L. Carro, F. Wagner, and A. Susin, “Design space exploration comparing homogeneous and heterogeneous network-on-chip architectures,” in *Proceedings of the 18th annual symposium on Integrated circuits and system design*. ACM, 2005, pp. 190–195.

[10] Y. Ben-Itzhak, I. Cidon, and A. Kolodny, “Delay analysis of wormhole based heterogeneous NoC,” in *Proceedings of the fifth ACM/IEEE international symposium on Networks-on-Chip (NOCS 2011)*, 2011.

[11] A. Varga *et al.*, “The OMNeT++ discrete event simulation system,” in *Proceedings of the European Simulation Multiconference (ESM’2001)*, 2001, pp. 319–324.

[12] [Online]. Available: http://www.omnetpp.org/models/catalog

[13] W. Dally and B. Towles, *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.