On the Development of a Qt-based Multithread NoC Simulator

Eduardo Alves da Silva, Luiz Gustavo Metzger, Cesar Albenes Zeferino
Laboratory of Embedded and Distributed Systems
University of Vale do Itajaí
Itajaí, Brazil
{eduardoalves, metzger, zeferino}@univali.br

***Abstract*— Communication networks for embedded systems with multiple cores, called Networks-on-Chip (NoC), have several characteristics that impact on cost and performance and should be considered in the system design. One way to measure the performance of a NoC is through simulation. This work is inserted in this context and aimed at the development of a new version of a simulator tool for evaluating the performance of a Network-on-Chip. This new version overcomes drawbacks of the original simulator by improving the modularity and maintainability of the simulator, as well increasing its performance while supporting its use in multiple platforms. It provides a Qt-based graphical interface and supports the execution of multiple simultaneous simulations, reducing the time for design space exploration.**

Keywords—Network-on-Chip; Performance Evaluation; Simulation; Multithreading.

I. INTRODUCTION

The design space of Networks-on-Chip – NoCs [1] is large and several architectural issues need to be considered to meet costs and performance requirements. The exploration of this design space can be facilitated with the use of tools that allows to determine the performance of the network for configurations with different costs.

In this context, [2] presented BrownPepper, a simulator for performance evaluation of a Network-on-Chip. This NoC simulator has a GTK+ graphical user interface (GUI) and is based on SystemC models of routers and traffic generators. It integrates a set of tools for modeling of traffic, experiment configuration and execution, and analysis of results.

The original version of the NoC simulator [2] has some limitations, including:

- low modularity (it was developed using structured C);
- lack of detailed software documentation;
- single platform support (it runs only in Linux platform); and
- single thread execution.

These features reduce the software maintainability, restrict its scope of use and limit simulator performance because it does not take advantage of the multiple cores available in current computers.

In order to remove these drawbacks, this work aims at improving the modularity and maintainability of the simulator, as well as increasing its performance, while supporting its use in multiple platforms.

A new version of the simulator was implemented and named *RedScarf*. The integrated tools were redesigned and documented using UML (Unified Modeling Language) and reprogrammed using object-oriented C++ language. The GUI was migrated from GTK+ to Qt, a platform-independent framework for GUI development that facilitated the implementation of the multiplatform and multithread support. In this paper, we describe the implementation of *RedScarf* and present performance results.

This paper is organized in five sections. Section II discusses related work. Section III describes the simulator architecture and features. Section IV discusses some results. Section V presents final remarks and future works.

II. RELATED WORK

Table I summarizes the basic features of some NoC simulators described in the literature. It identifies the language used to implement the simulation models, if the simulators have a GUI to facilitate its use, the platforms they can be executed, and if they support multithread execution.

As one can see, most of the simulators were based on C++ or SystemC and can only run on more than one platform if the target platform is compliant with the language used to implement the simulator. However, few simulators have a GUI to facilitate its use and only one (Atlas) is multiplatform because it is based on Java.

Considering the multithread support, this feature was only identified in GSNOC UI. However, it is limited to two threads: one for the front-end (the GUI) and another for the back-end (the simulator engine). *RedScarf* stands out mainly because it supports the execution of multiple threads by the back-end, allowing the execution of more than one simultaneous simulation. Not only that, but the GUI is heavily used for the configuration of the experiments and for the analysis of the results.

TABLE I. CHARACTERIZATION OF NOC SIMULATORS

Simulator	Ref.	Language	GUI	Supported Platforms	Multithread
Booksim	[3]	C++		Unix, Windows (Cygwin)	
NoC Simulator	[4]	C++		STL-compliant	
Noxim	[5]	SystemC		SystemC-compliant	
NNSE	[6]	SystemC		*n.a.*	
Nirgam	[7]	SystemC		SystemC-compliant	
OCCN	[8]	SystemC		Unix, Sun/Solaris	
Atlas	[9]	SystemC/VHDL	Java	Linux, Windows, Sun OS, OS X	
FlexNoC	[10]	SystemC		*n.a.*	
NoCGEN	[11]	SytemC/VHDL		*n.a.*	
GSNOC UI	[12]	SystemC	Qt	Linux [a]	Yes
BrownPepper	[2]	SystemC	GTK+	Linux	
RedScarf	This work	SystemC	Qt	Linux, Windows, OS X	Yes

[a] Not tested in other platforms.

III. REDSCARF ARCHITECTURE

A. Basic Features

The simulation environment is composed of a set of tools that automatize the NoC configuration process, the traffic scenarios, the router parameters, the simulation and the analysis of the results.

The parameters defined for the creation of the network are used to generate the RTL (Register Transfer level) model of the NoC in SystemC. The traffic scenarios are used by the traffic generators described at a mixed of RTL and TL (Transaction Level) model. Traffic meters are used to collect data from packets transferred through the NoC in order to obtain the performance metrics.

The simulation is automatized, being executed for several experiments. The analysis of the results is also automatized through the data collection, generation of simulation reports and inference by means of plotting graphs, exhibition of waveforms and reports regarding the throughput and latency of the NoC.

B. Design Process

The original version of BrownPepper was used as the reference to survey the functional and non-functional requirements, to create the UML artifacts (such as class diagrams, use cases and activity diagram), to redesign the GUI using Qt and to specify of the functionalities that would compose *RedScarf*.

After that, the software project document was created in order to facilitate the comprehension of the system for the designers, showing the characteristics that will be affected with each desired modification.

Therefore, the software artifacts are: (*i*) the aforementioned UML diagrams; (*ii*) the redesign of the graphical interface; (*iii*) prototypes of the new screens created to support new features and a functioning version of the tool developed with Qt, which supports the multithread execution to speed up the simulation of experiments; and (*iv*) better modularity with the use of the MVC (Model-View-Control) paradigm.

As a result, with the same functionalities of BrownPepper plus the multithread implementation, RedScarf has around 23,000 lines of code, while BrownPepper has 30,500. This reduction is possible due to the application of object oriented programming, which allows for a greater reuse of code compared to the functional approach used in BrownPepper.

C. Integrated Tools

The modularized simulator has tools for the generation of the NoC and SoC models, description of the traffic flows, as well as visualization of results. All of the tools are integrated into the environment and parameterized through the graphical interface, facilitating the organization of the input data.

For the visualization of the results, third party tools are used to display the waveform graph and to plot the other graphs (*e.g.* GTK Wave and GNU Plot in Linux, respectively).

The multithread approach is used to execute more than one simulation in parallel by taking advantage of the multiple processing cores of the host computer. By default, all of the available cores are used, but it is possible to change the number of threads through the environment parameters. Also, the environment is multiplatform and is available for Windows, Linux and Mac OS X.

IV. IMPLEMENTATION AND RESULTS

A. Qt Framework

Qt is a framework that abstracts high- (*e.g.* GUI functionality) and low-level (*e.g.* thread handling) APIs of the operating system. It uses the "write once, compile anywhere" approach. It is supported by several platforms, such as Windows, Linux, Solaris, Mac OS X and others. It is distributed with private and commercial licenses [13].

The GUI was built with the designer tool available with Qt, which counts with the use of common widgets for GUI applications and QML (Qt Meta Language) for the creation of new functionalities, such as keeping track of the state of the network during simulations.

B. *Graphical User Interface and Multithread*

RedScarf GUI has three main tabs (shown in Fig. 1):

- *System Configuration*: where the network and traffic parameters are configured;
- *System Simulation*: where the routers and experiments are configured; and
- *Performance analysis*: where the inference over the results is carried out.

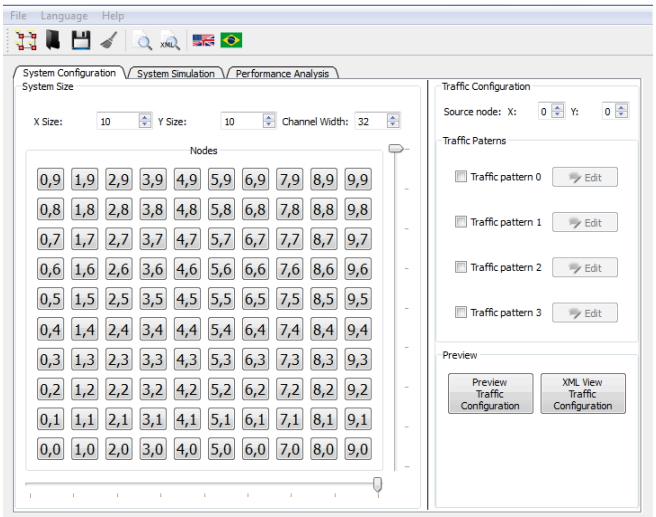
The use of threads was implemented using the resources available with the Qt framework. The application itself is the main thread. Each experiment (or simulation) runs in a separate, synchronous thread, and sends control signals to the main thread.

C. *Experiments*

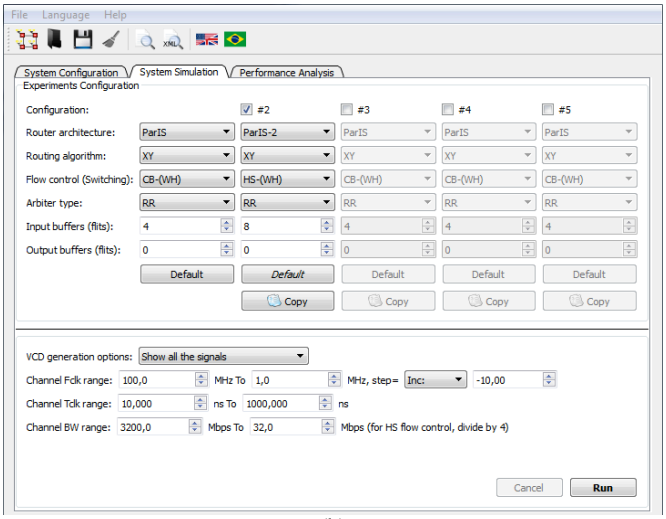
A set of experiments was run in order to evaluate the effectiveness of the multithread support in *RedScarf*. These experiments were based on the performance evaluation of a system model composed of 16 traffic generators interconnected by a 4x4 2-D mesh NoC. The traffic generators were configured to inject traffic into the NoC under an uniform distribution, sending 1000 9-flit packets to each one of the other nodes in the system at a rate of 3.2 Gbps. In the experiment, we evaluated two different NoC configurations and varied the operating frequency in a range of 11 different clock frequencies. This resulted in a total of 22 simulations.

The 22 simulations were run in an 8-core server (Intel$^{(R)}$ i7 4850HQ - 2.3 GHz) by varying the number of simultaneous threads from 1 to 22.

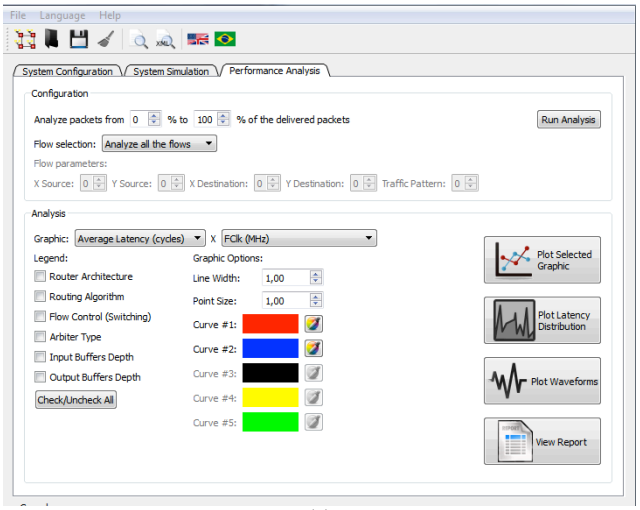
Fig. 2 presents the obtained results. As one can see, by increasing the number of simultaneous threads (*i.e.* simulations) from 1 to 8 threads, the execution time is reduced from 50.3 to 9.3 minutes. The speedup increases as more cores are used. However, after all the cores in the system are occupied, the speedup varies, decreasing and increasing as the number of simulatenaus threads is increased. This behaviour is discussed as follows.



(a)



(b)



(c)

Fig. 1. Graphical interface of *RedScarf*: (a) system configuration tab; (b) system simulation tab; and (c) performance analysis tab

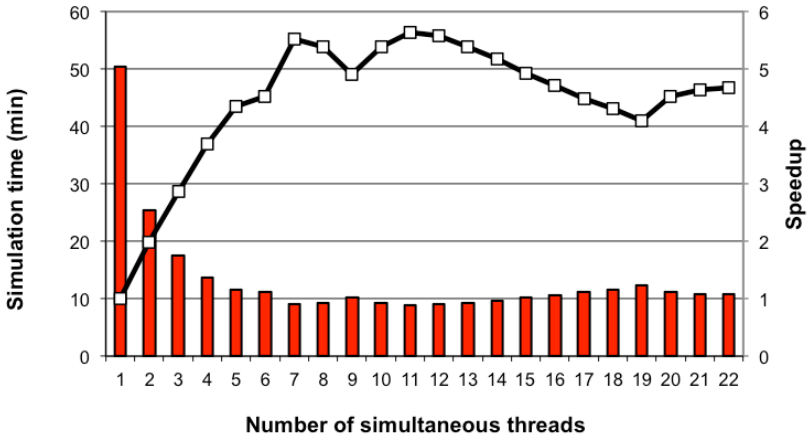


Fig. 2. Experimental results: simulation time (columns) and speedup (line)

To better understand the variation of the speed-up presented in Figure 1, two aspects must be taken into consideration. The first are the input and output operations and the second is the task scheduler of the operating system. The simulator works by reading the traffic models, simulating the scenario and writing the simulation logs. As these read and write operations are much slower than the simulation part and also block the scheduler process, context switches are done between the tasks. This allows the other tasks in the scheduler's queue to obtain the processing resource while an input-output operation is done in one or several simulators. As several simulators are executed simultaneously, in separate threads, many input-output operations and a high workload of the task scheduler along with the SystemC scheduler cause the variation of the speed-up. Because the scheduling of the operating system is not predictable, finding the optimal number of threads is very difficult.

V. CONCLUSIONS

As the design space of NoCs presents a large quantity of parameters and characteristics, simulation tools are one of the best ways to evaluate different configurations. In general, as the complexity of the NoC grows, it is expected that these tools are able to keep up.

In this context, we used new software features present in the Qt framework to improve an existing NoC simulation environment. For the end user, the most important result was the significant decrease in the simulation time for experiments. As for developers, it becomes much easier to integrate new features because of the application of the MVC paradigm.

As future works, we intend to integrate features such as the simulation of attack scenarios for researches about Security in NoCs. Also, we intend to implement a graphical tool to visualize the state of the network components (buffers and links) during simulation. As soon as a stable version is reached, the tool, along with its documentation, will be available for download for the general public.

REFERENCES

[1] A. Hemani, A. Jantsch, S. Kumar, A. Postula, J. Öberg, M. Millberg, D. Lindqvist, "Network on Chip: an Architecture for Billion Transistor Era," in *18th IEEE NorChip Conf.* (NORCHIP 2000), Turku, Finland, Nov. 2000.

[2] J. V. Bruch, M. R. Pizzoni, C. A. Zeferino, "BrownPepper: A SystemC-based simulator for performance evaluation of Networks-on-Chip," in *17th Int. Conf. on Very Large Scale Integration* (VLSI-SoC 2009), Florianopolis, Brasil, 2009.

[3] *BookSim 2.0 User's Guide*, 2013. Available in: http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim.

[4] *NoC Simulator*, 2007. Available in: http://nocsim.blogspot.com.br/

[5] *Noxim: the NoC Simulator.* Available in: http://noxim.sourceforge.net/

[6] *NNSE: The Nostrum NoC Simulation Environment*. Available in: http://www.ict.kth.se/nostrum/NNSE/

[7] *NIRGAM: A Simulator for NoC Interconnect Routing and Application Modeling,* 2007. Available in: http://nirgam.ecs.soton.ac.uk/

[8] *OCCN: On-Chip Communication Architecture*, 2011. Available in: http://occn.sourceforge.net/

[9] *Atlas: An Environment for NoC Generation and Evaluation.* Available in: https://corfu.pucrs.br/redmine/projects/atlas

[10] *Arteris FlexNoC Interconnect IP*. Available in: http://www.arteris.com/flexnoc

[11] J. Chan, S. Parameswaran, "NoCGEN: A Template Based Reuse Methodology for Networks on Chip Architecture," In *Very Large Scale Integration Design (VLSID 2004),* 2004, p. 717-720.

[12] P. Gottschling, H. Ying, K. Hofmann, "GSNOC UI – A Comfortable Graphical User Interface for Advanced Design and Evaluation of 3-Dimensional Scalable Networks-on-Chip," in *High Performance Computing and Simulation (HPCS 2012)*, Madrid, 2012, p. 261-267.

[13] J. Blanchette, M. Summerfield, Mark. *C++ GUI programming with Qt4*. Stoughton, Prentice Hall, 2006.