TOPAZ: An Open-Source Interconnection Network Simulator for Chip Multiprocessors and Supercomputers

Pablo Abad, Pablo Prieto, Lucía G. Menezo, Adrián Colaso, Valentin Puente, José-Ángel Gregorio
Departamento de Electrónica y Computadores
University of Cantabria
Santander, Spain
{abadp, prietop, gregoriol, colasoa, vpuente, monaster} @unican.es

Abstract—As in other computer architecture areas, interconnection networks research relies most of the times on simulation tools. This paper announces the release of an open-source tool suitable to be used for accurate modeling from small CMP to large supercomputer interconnection networks. The cycle-accurate modeling of TOPAZ can be used standalone through synthetic traffic patterns and application-traces or within full-system evaluation systems such as GEMS or GEM5 effortlessly. In fact, we provide an advanced interface that enables the replacement of the original lightweight but optimistic GEMS and GEM5 network simulator with limited performance impact on the simulation time. Our tests indicate that in this context, underestimating network modeling could induce up to 50% error in the performance estimation of the simulated system. To minimize the impact of detailed network modeling on simulation time, we incorporate mechanisms able to attenuate the higher computational effort, reducing in this way the slowdown of the full system simulation with accurate performance estimations. Additionally, in order to evaluate large-scale networks, we parallelize the simulator to be able to optimize memory resources with the growing number of cores available per chip in the simulation farms. This allows us to simulate node networks exceeding one million of routers with up to 70% efficiency in a multithreaded simulation running on twelve cores.

Keywords; simulator, interconnection networks, chip multiprocessor, supercomputer

I. INTRODUCTION

Retrospectively, the research in the field of interconnection networks has mostly been restricted to supercomputing. However, nowadays the outlook is very different, and the advent of on-chip interconnection networks [1] has significantly increased the relevance of the area. Today, the interconnection network is a key element everywhere from large supercomputers [2] to high-performance general purpose chip-multiprocessor [3][4], through systems-on-a-chip[5]. Different environments with particular technological constraints and system requirements have diversified this research area. Although the foundations of each kind of network are similar, in practice some of the design parameters are feasible in some fields but not in others. For example, in the on-chip context, wire availability is profuse, making it feasible to use ultra wide communication links. On the contrary, in the off-chip context, router implementation cost or network energy consumptions are not first-order design parameters, allowing the utilization of very large on-network storage. Many other characteristics such as link latency, network interfaces, node count, topology, etc. can also have a different impact depending on the environment, making network proposals very specific to the system constraints.

This diversification turns into an important handicap when trying to design "wide-spectrum" simulation tools, increasing the complexity and computational effort required to cover multiple environments. As a consequence of this, many simulation tools used in interconnection networks are limited to a single field. For chip-multiprocessor-oriented simulators, such as GARNET [6], which supports the integration with GEMS [7], it seems adequate to restrict simulator capabilities targeting only network on chip requeriments. On the other hand, off-chip system-oriented tools such as INSEE [8] are focused on simulating tens of thousands of nodes, which makes more sense in a supercomputing system. Finally, system-on-chip-oriented tools, such as NOXIM [9], allow the evaluation of heterogeneous networks, usually found in this kind of environment. Like in many computer architecture expertise areas, more accurate simulation tools imply higher computational effort. If this tradeoff is not carefully adjusted, simulations could present an error margin rendering meaningless the results or the computational effort could turn simulation time prohibitive. Systems such as SimOS [10] or GEM5 [11] are able to dynamically adjust this tradeoff when moving from the initial exploration of the design space to the final product. A different solution, employed by GEMS [7], relies on the simulation of different parts of the system with different levels of detail. Most of network simulation tools are not able to adapt their computational effort to different scenarios, relying only on one of these premises; speed or accuracy.

This work presents TOPAZ, a simulation tool with a broad spectrum of utilization scenarios and different tradeoffs between accuracy, simulation speed and field of application for interconnection network research. TOPAZ has been conceived to be embedded in other simulation tools with limited effort. In fact, the GEMS/GEM5-TOPAZ interface is also presented in this document. TOPAZ is designed to be easily extendable and deeply configurable on runtime. TOPAZ includes multithread capabilities, minimizing the impact of more accurate simulations on execution time. As well as a description of these characteristics, this paper provides a brief overview of TOPAZ construction, philosophy, capabilities, and examples of use along with pointers to sources of additional information. In order to facilitate access to the tool by other

researchers and simplify the adoption of their own modifications, a public source code repository and project management tools have been made available [12].

The rest of the paper is structured as follows: Section 2 describes the simulator, Section 3 explains the integration of the tool in a full-system simulation environment, Section 4 indicates how the simulator can be used as an effective tool in very large networks, and finally Section 5 draws the main conclusions of this paper.

II. SIMULATOR DESCRIPTION

TOPAZ is a general-purpose interconnection network simulator that enables the modeling of a wide variety of message routers, with different tradeoffs between speed and precision. TOPAZ originates from the SICOSYS [13] simulator, which was originally conceived to obtain results very close to those obtained by using HDL description of network components by hardware simulators but at lower computational cost. In order to make the tool easily comprehensible, extensible and reusable, the design of TOPAZ is object-oriented and has been implemented in C++. For the models provided, approximately 100 classes, distributed in about 50,000 lines of code have been necessary. TOPAZ provides router descriptions with different levels of detail. Simplified routers are described as a single component, which allows fast simulation times. On the other hand, detailed routers require the description of each router component separately, much more similar to RTL router descriptions. Detailed routers are much more precise, at the cost of simulation speed. Finally, the simulator has also support for parallel execution using standard POSIX threads. The portability is very high and it can be executed on any UNIX platform with a C++ standard compiler.

Simulations in TOPAZ can be divided into three different phases; building, running and printing. In the first phase, the network simulated is constructed at run-time according to a set of parameters selected by the user. TOPAZ network is constructed hierarchically. The simulator builds the network and the links (topology), then the network builds all the routers and finally the routers build its intrinsic components and interconnect them. Each simulated structure is associated with two C++ classes: *components* and *flows*. The components are descriptive, characterizing each structure and its relationship with the remaining components of the system. The flows establish how the stream of the information will move inside the component. As an example, for a buffer structure, the component will determine its size, number of ports or delay, while the flow will determine its behavior according to the flow control selected. During the running phase, all system components are iteratively visited and all dependent flows simulated for every cycle simulated. Simulation length can be configured both in terms of simulation cycles or messages injected. The flows of each router component are run in first place each cycle, and links interconnecting each component (both inter and intra router links) are run subsequently. While TOPAZ is a time-driven simulation tool, some flows can internally be constructed as FSMs, making these components event-driven. If chosen, results can be periodically printed during running phase. Finally, printing phase returns all the parameters derived from simulation process. As well as the main figures of merit of interconnection network (latency and throughput), this phase can be configured to produce additional details, such as execution time, per-VC results, per-router injection rate, per-router consumption rate, link utilization, event count (useful for power estimations), etc.

A. Structure

The implementation of the simulator has been oriented towards high portability and an easy learning curve. Files present into code directories (/inc and /src), can be divided into the following parts:

SGML constructors (*builder.cpp). Employed in the building phase. Responsible for interpreting the specification files and generating the objects required for the simulation. There is a constructor for each one of the three existing specification files described in next section.

Components & Flows. They represent the hardware that will be simulated. The functional components of the router, their characteristics and associated delays are modeled. In this way, a direct relationship between a specific hardware structure and the model that our simulator generates is established. The constructors simply generate a hierarchy of components related to each other.

Traffic patterns (TPZTrafficPattern.cpp). Module in charge of the injection of packets into the network. When the simulator is used in a stand-alone way, the main traffic patterns (random, matrix transpose, reversal bit, perfect shuffle, hot-spot, etc.) can be used. Additionally, trace-based simulations are also supported. For injecting traffic generated by the execution of real applications, an "empty" pattern has been defined with the aim of working in an integrated way with other tools.

Simulator: This module carries out the simulation process. It progressively performs the actions belonging to each simulation phase. This module is in charge of interpreting all the parameters provided through command-line, making feasible the utilization of scripting based simulation.

B. Specification files

The tool provides a solution to the generic problem of interconnection network simulation. The same compiled environment can be used to experiment with very different architectural proposals. The specification of the type of experiment is carried out through three description files written in SGML. These files are interpreted by the simulator at runtime, making code recompilation unnecessary if the user wants to use any of the modules included in the simulation. Static configuration only has been used to switch from sequential to multithread execution. Each file specifies the following aspects:

Simulation parameters (Simulation.sgm): Definition of general simulation parameters such as traffic pattern and distribution, applied load, message length, etc. In this file, the network to be used for the simulation is also referenced.

Interconnection network (Network.sgm): Specifications of the type of network through parameters such as topology (mesh, torus, etc.), dimensions and interconnection delays. The router employed at each network cross point is defined as an additional parameter of the network.

Router microarchitecture (Router.sgm): Description of the elements (memories, switches, multiplexers, etc.) which, connected to each other, form the router. The main parameters of each router component (delay, size, inputs, outputs) are also described in this file.

The simulation tool employs an additional file, named "*TPZSimul.ini*". This way, it is possible to easily select one among the multiple router, network and simulation description files. Through command line, we only need to specify the name of the simulation we want to run. In the file "*Simulation.sgm*" we find a simulation with the same name passed through command line, which will specify the dynamic conditions of the simulation, i.e. traffic pattern, packet or message length, etc. The network configuration we want to simulate is an additional parameter in this file. In "*Topology.sgm*" file we find a description of the selected network. Different topologies can be used, from bi-dimensional or tri-dimensional meshes, tori, to irregular networks. Finally, router is also selected at this point, and the file "*Routers.sgm*" contains a SGML description of all architectural characteristics. When components have been implemented, it is possible to instantiate them. In the example, fifo-buffers, routing units and a crossbar are used (detailed description). After instantiating the components, connections have to be made within the router and between the routers. In the instantiation, different parameters of the components can be set, overriding global router parameters, for example buffer capacity, which allows different components of the router to be customized. For example, escape channels may require different buffer size to adaptive channels in a fully adaptive router [2].

C. Out-of-the Box Models Available

Table I enumerates network models supplied in the initial release of the simulator. For most router models, both fast and accurate models are supplied. Some of the components target the CMP environment, such as low degree topologies, whereas others are more suitable for off-chip networks, such as 3D network interconnects. Although some combinations are fixed, for example the Adaptive Bubble Router requires *Bubble Flow Control*, it could be possible to combine different network topologies, router architectures and additional features. The models provided are selected in order to offer CMP researchers an out-of-the-box setup to work with, but are also intended to provide a framework to develop new components. The design style, once the programmer is familiar with it, simplifies the implementation of new components. This enables to implement new components in a short period of time. As was mentioned earlier, we provide an open platform to collaborate in the future development of the simulator, which could facilitate further advancement and new components. We hope that users of the tool will share their developments with other users.

TABLE I. COMPONENTS SUPPLIED WITH TOPAZ

Network Topologies				
Ring	Mesh (2D & 3D)	Torus (2D & 3D)	Midimew (2D)	Square Midimew (2D)
Flow controls				
Virtual Cut Through	Bubble Flow Control	Wormhole	Virtual Channel Flow control	
Multicast Support				
Source-decomposed	Path-based	DOR-Tree based	Adaptive (Path-Tree)	
Traffic Patterns				
Random	Bit-Reversal	Perfect-Shuffle	Transpose Matrix	
Tornado	Hot Spot	Local	Trace-Based	
Message Size / Packet Size				
1 to unlimited / 1 to unlimited				
Power consumption (Event count)				
Buffer Write	Buffer Read	VC Allocation	SW Allocation	
SW Traversal	Link Traversal			
Routers				
ID	**Reference**	**Year**	**Level of detail**	
Adaptive Bubble	[14]	2001	complex &simple	
Deterministic Bubble	[15]	1998	complex & simple	
Deterministic with VC (Dally)	[16][17]	2001	complex & simple	
VCTM (Dally + MC support)	[18]	2008	complex & simple	
Rotary Router	[19][20]	2009	complex & simple	
Bufferless Router	[21]	2010	simple	
Bidirectional Router	[22]	2009	simple	
Buffered Crossbar	[23]	1987	complex	
Pipeline Optimized	[24]	2008	complex & simple	
Configuration Parameters				
Buffer Size	Packet Size	Number of Virtual Channels	Number of message types	
Router Pipeline	Buffer Delay	Link Delay	Number of physical networks	

D. Multithread Implementation

In general, parallelization of computer architecture simulators is a non trivial task, and usually the performance benefit does not compensate the effort. The scalability of this development is greatly limited by the fine-grained parallelism: in a conservative implementation it is necessary to synchronize each thread every cycle. Although speculative approaches are possible, they introduce additional complexity in a usually very complex piece of software. If we consider the fact that the number of simulations required is very large, using throughput computing is more appealing. Nevertheless, the simulated system size can make this approach a costly solution. In particular, today's server farms devoted to running these simulations use CMPs with a

growing number of cores per chip. Under these circumstances, throughput computing demands massive quantities of DRAM per server, which increases acquisition and costs of ownership. From the TOPAZ perspective, this can happen if massive supercomputer networks have to be simulated, i.e. networks with hundreds of thousands of computing elements. In its design, TOPAZ has been conceived to run these necessarily large networks in parallel. For the sake of portability, Topaz parallelization is based on POSIX threads.

As the simulation engine is time-based, parallelization of the tool is greatly simplified. It is possible to split the network and assign their simulation to different slave execution threads. The boundary of the group of components assigned to each slave thread has to be synchronized periodically with neighboring threads. The synchronization between slaves is done each cycle using a barrier. The master thread is responsible for orchestrating the initialization and finalization of the simulation and facilitating the integration with full-system simulators. This approach abstracts the details of simulation implementation completely to the external full-system simulator.

III. Chip Multiprocessors: Integration with Full System Simulation Tool

A. Topaz-GEMS Integration

TOPAZ is fully modular and can be used in an abstracted way by full-system simulation tools. It provides the API necessary to connect external simulators without extensively modifying code. The developer should create an interface, according to external simulator peculiarities and system necessities. In order to simplify TOPAZ adoption, we provide a detailed implementation with GEMS [7] and as GEM5 [11]. This could be easily extended to other derived simulation tools such as FeS2 [25]. As Figure 1 shows, TOPAZ is connected to the memory subsystem of Ruby. As it is a common simulator with GEM5, FeS2 and GEMS, TOPAZ will work with all the tools seamlessly. We provide a patch for version 2.1 of GEMS that performs the integration. This patch could be merged with further development using standards tools such as *quilt* or equivalent ones. We also provide a clone of GEM5 development repository.

In contrast to GARNET, TOPAZ integration is fully isolated. In the process of compilation, the simple network simulator of Ruby is connected to the external simulator. A new event is introduced in the event-queue of Ruby in such a way that if TOPAZ has to be run, the engine is activated for the number of cycles that the network-memory clock ratio indicates. If multithreading is activated, the main thread does not wait for TOPAZ until the beginning of the next cycle. At this time Ruby takes the packets delivered by TOPAZ and injects the new ones.

Both, the modified GEMS and GEM5 are supplied as a Mercurial repository at the public TOPAZ page [12].

Undeniably, increasing accuracy decreases simulation performance. In full-system simulation, where thousands of millions of CPU cycles have to be simulated in order to obtain meaningful results, slightly decreasing simulation speed could have a significant cost. In order to minimize performance effects, TOPAZ provides three different mechanisms. First, its multithread implementation enables network simulations to be run in a separate thread from the main simulation. Although the usually reduced size of on-chip networks makes it unnecessary to subdivide them into separate threads, it makes sense to use master thread abstraction to isolate network simulation from the main thread. Second, we have developed an adaptive interface between both simulators. When network load is low, contention in the network is negligible and the original Ruby network is enough to model the network. Nevertheless, when contention is high, the optimistic latency provided by Ruby could be tens or even hundreds of cycles below the real value. Our adaptive interface works as follows: when the number of packets in the network is below a predefined threshold, TOPAZ is disabled and only Ruby is simulated. If this threshold is surpassed, TOPAZ is activated and will remain activated until a second threshold number of packets is surpassed. Finally, the third technique is based on the modification of the router complexity. For almost all routers provided, two different accuracy models can be used, making it possible to use simplified models in initial design phases and more detailed models in later stages.

No modification has been performed in GEMS's nor GEM5's original code besides the creation of the interface to connect Ruby to TOPAZ.

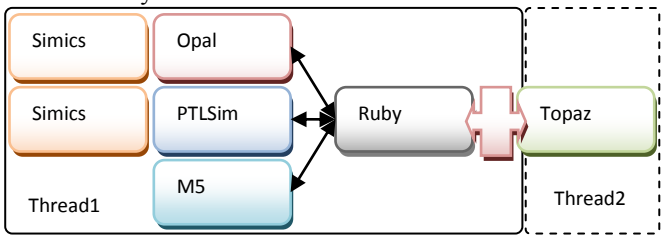

Figure 1 Topaz-Ruby Connection.

B. Effects of Network Simulation Accuracy

The original Ruby network simulator is quite simple: it models contention only at link level. Although this speeds up simulation, it introduces a non negligible error in network latency estimation when the load applied is high. Recent versions of GEMS/GEM5 have introduced the possibility of using GARNET [6] as a replacement for the original simulator. Although much more detailed than the original simulator, GARNET has limited flexibility with fixed router architecture. It only contemplates changing a few parameters in the network such as number of virtual channels, router pipeline stages, buffer size and flit size. TOPAZ is much more flexible, providing a full set of very different router architectures, routing algorithms, network topologies, flow-controls, etc. The TOPAZ-Ruby interface has been developed to support any CMP or SMP network configuration, using both file-defined topologies and regular

ones. TOPAZ does not replace the original Ruby network simulator but is added to it, in order to be able to dynamically activate or deactivate it to speed up the simulation during phases of low traffic. During those moments contention should be negligible and therefore results obtained by precise router modeling are usually close to those provided by a simpler router model.

In order to demonstrate how relevant accurate network simulation can be, we have carried out the simulation of very diverse applications and coherence protocols. We will restrict this study to GEMS. Twenty workloads are considered in this study, including both multi-programmed and multi-threaded applications (numerical and server) running on top of the OpenSolaris 10 OS. The numerical applications are the ***whole*** NAS Parallel Benchmarks suite (OpenMP implementation version 3.2 [26]) and four benchmarks of the PARSEC suite [27]. The server benchmarks correspond to the whole Wisconsin Commercial Workload suite [28], released by the authors of GEMS in version 2.1. The remaining class corresponds to multi-programmed workloads using part of the SPEC CPU2006 suite running in rate mode (where one core is reserved to run OS services) [29]. Each application is simulated multiple times with random perturbations in memory access time in order to reach 95% confidence intervals. The number of applications enables the sweeping of a broad spectrum of application types, with diverse network demands in order to know the margin of error caused by different network modeling.

TABLE II. EVALUATED WORKLOADS (PROBLEM SIZES).

Multithreaded-Workloads	Wisconsin Commercial Workload	Apache (1000 Surge dynamic)	Zeus (1000 Surge Static)
		Jbb (4000 SpecJbb)	OLTP (500 TPC-C alike)
	NAS Parallel Bench.	BT (Class A)	CG (Class A)
		FT (Class W)	IS (Class A)
		LU (Class A)	MG (Class W)
		SP (Class A)	UA (Class A)
	PARSEC	blackscholes (native)	
		canneal (native)	
		fluidanimate (native)	
		streamcluster (native)	
Multiprogrammed-Workloads	Spec 2006 (Rate Mode)	astar (reference)	hmmer (reference)
		lbm (reference)	ommetpp (reference)

We have used a 16-processor CMP, with out-of-order Nehalem [30] like cores, using a 4×4 mesh network which is provided in the version 2.1 of GEMS. We analyzed how the system behaves with the two coherence protocols provided: *MOESI_CMP_token* and *MOESI_CMP_directory*. No change has been made in either of the two coherence protocols. All the configuration parameters chosen in the comparison are provided in the GEMS Mercurial repository provided in [12].

We adjust all simulators to have similar router configurations. In particular we chose the fixed 5-cycle wormhole pipeline router provided by GARNET [6]. We will assume 1 cycle wires between routers. In the simple Ruby network simulator, the latency is adjusted to match GARNET routers. We use four virtual networks to avoid end-to-end deadlock and 10 flits of 16 bytes of buffer per virtual channel, having only one virtual channel per traffic class. TOPAZ is configured using the same router in its simple and complex implementation and matching all configuration parameters. The two routers are provided in out-of-the-box components.

C. MOESI_CMP_directory Coherence Protocol

This coherence protocol is characterized by a limited network load, therefore it can be considered as baseline where careful network simulation has lower impact on final system performance. Figure 2 shows the normalized execution time for the workloads considered. On the one hand, it is clear that contention modeling of the original simulator is too optimistic, inducing a substantial error in the estimation of the execution time of each application. Although in some cases the effect is lower, such as for *blacksholes* and *fluidanimate,* in others it is surprisingly high. If we compare TOPAZ's complex and simple implementations the differences are small but appreciable with respect to other simulators. Taking into account that the tool from which TOPAZ derives [13] is capable of achieving less than 3% error with respect to hardware simulators, it seems reasonable to use TOPAZ as a reference point.

On the other hand, GARNET network modeling seems to be too pessimistic. We used the publicly available tool, compiled and run using all the benchmarks considered. We prefer not to modify it in order to allow other researchers to repeat the evaluation. Like in the case of the original network simulator, with low load applications the error is small but with high contention applications the effect is substantially increased to almost three times the application execution time differences. This behavior can be easily reproduced with the configuration used, which is available in [12].

On average, the original network simulator introduces an optimistic error of 25% in execution time and GARNET a pessimistic one of 25%. Therefore, even in a not very demanding scenario, inaccurate modeling of the contention in the interconnection network could induce substantial errors in evaluation results, which could render the hypothetical comparison of two or more architectural solutions for the CMP unreliable. Note that original Ruby only models contention at link level, which is optimistic given the contention suffered at the crossbar by conventional routers, and GARNET seems to have some kind of error in pipeline implementation.

Figure 3 shows the performance of each simulator in terms of Ruby network normalized simulated cycles per second of CPU. We have used the same hardware platform to perform this measurement with a large number of samples in order to have reliable averages. As can be appreciated, the complex TOPAZ router has a non negligible impact on performance, increasing the simulation time by 20 % on average. In some applications, such as CG, performance could be degraded by up to 50%. As can be seen, using simple models could attenuate this slowdown on average,

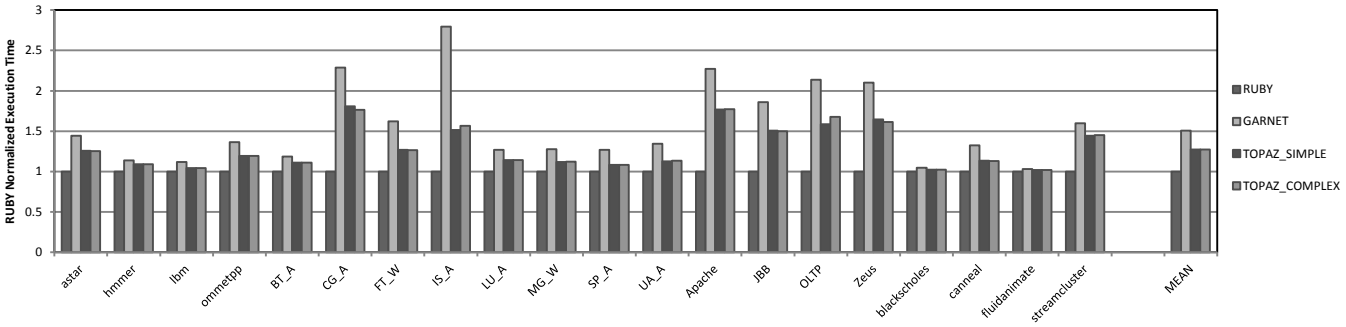

Figure 2. Directory Coherence Protocol: Execution time differences through different network simulators.

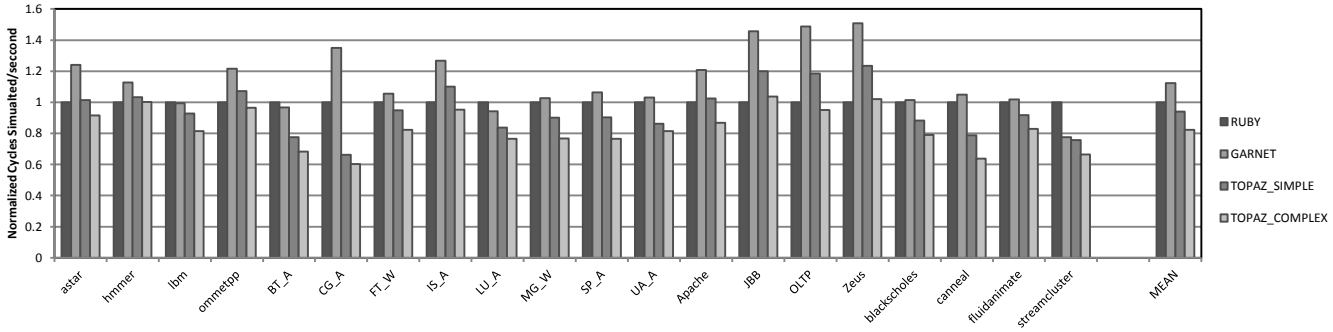

Figure 3 Directory Coherence Protocol: Simulator Performance differences through different network simulators

obtaining reasonable accuracy with less than 5% of performance degradation. Surprisingly, GARNET is the best performer, improving on the original gems simulator by an average of 10%, which is motivated by its pessimistic contention modeling. It increases the average latency perceived by the processors in such a way that the activity in the rest of the system falls, increasing the number of cycles simulated per clock cycle. In Topaz *simple*, this also happens in workloads with low parallelism at instruction level, in which processor activity is lower than in Ruby.

In contrast, Topaz *complex* compensates the lower computational cost of modeling the processors with the higher network activity, which makes the number of cycles simulated per second almost constant.

D. MOESI_CMP_token Coherence Protocol

In contrast to directory, this coherence protocol is characterized by intense network requirements due to multicast traffic. This example could be considered as a reference point where careful network simulation has a large impact on final system performance and simulation speed. Given the fact that GARNET has no native support for multicast traffic and this protocol uses it heavily, we decided to exclude GARNET in the comparison. Our simulator does have support for this kind of traffic. Note that broadcast-based coherence protocols heavily depend on network support for multicast [18].

Figure 4 shows how the CMPs perform for each application and as the coherence protocol is characterized by a larger bandwidth requirements [31], the contention in the network will be higher and consequently the optimistic modeling of GEMS induces a larger error in the execution time of the workloads. In some cases, such as *Apache* and *IS* the observed error is above 100%, which is much higher than that observed in the case of directory.

As mentioned before, more network load implies a higher computational workload for the network simulator and a slowdown in simulation time, as Figure 5 indicates. Now the performance reduction with the most detailed router falls on average by 40%. Simple implementation attenuates this fall by 5-10%. For some applications such as *CG*, the performance falls 80% i.e., simulation is five times slower. Under these conditions, in this protocol we explore additional performance optimizations such as an adaptive interface (denoted AI in the results). Even with such a small network and simple router, the unbalance between network simulation and the remaining components of the system keep multithreaded simulations (denoted P in the results) interesting, especially if it is required to run a particular application faster or maximum precision is required for network model. If a large batch of runs is required, in most cases to run sequential simulations will be more efficient because it requires two cores and in most cases speedup is lower than two. In contrast, adaptive interface is able to improve simulation performance significantly, reducing the gap with the original Ruby simulator by more than 10%, with an error below 2% for both simple and complex routers. The data provided have been obtained using a threshold of 25 packets in the network before turning on the Topaz simulator. Only with applications such as *streamcluster* is the error relevant. Even when using 25 in-flight messages as a threshold, the traffic pattern could create significant contention, for example, if most traffic is highly localized around some specific parts of the system.

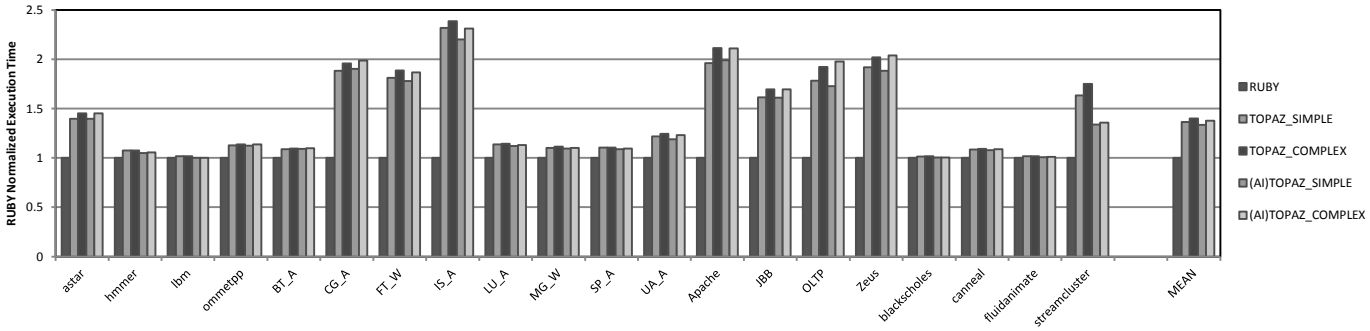

Figure 4 Token Broadcast Coherence Protocol: Execution time differences through different network simulators

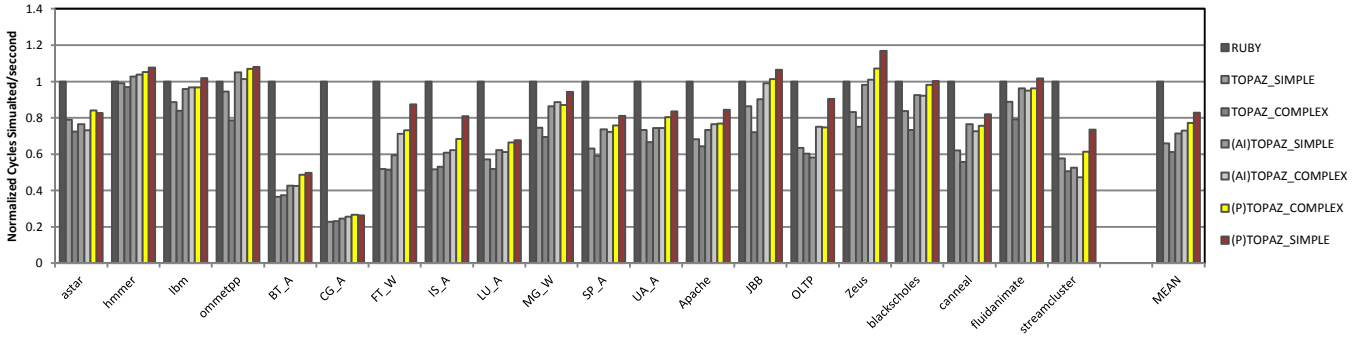

Figure 5 Token Broadcast Coherence Protocol: Performance differences through different TOPAZ optimizations.

IV. SUPERCOMPUTERS INTERCONNECTION NETWORK EVALUATION

This class of system has the singular characteristic of having a massive number of routers. The biggest challenge for a simulation tool for this kind of system is when the number of nodes grows significantly and accurate router modeling is paramount to discover potential instabilities in the system [32]. When thousands of nodes have to be simulated, the memory required could be significant. TOPAZ could use an advanced memory allocation approach which speeds up execution and limits further potential problems during the simulation. Additionally, multithread implementation enables taking advantage of current multicore server predominance.

To show the scalability limit of the tool, like in the case of IBM Blue Gene systems [2], we use a 3D torus with the simple model of the Bubble Router [14]. We evaluate the performance obtained with up to one million of routers in the network in order to determine the scalability of the parallelization. Figure 6 presents the results obtained with a server with 12 cores and with 54GBytes of main memory based on Intel Xeon E5645. As we can observe, the scalability with such large systems is adequate. With 32K routers the simulation uses approximately 1.5GB of memory, 5.5GB for 128K nodes, 12GB for 256K nodes, 24GB for 512K routers and for 1 million it uses 49GB. As the number of nodes grows the speedup decreases slightly. When the number of routers is higher, the data synchronization between threads is more demanding for the memory hierarchy of the server. With 12 threads there is performance degradation due to threads unbalance due to non divisible number of nodes.

TOPAZ has also demonstrated to be a suitable tool for dealing with such a large system, because the speedup achieved maximizes core utilization on the server. Running the simulation for such large systems using sequential simulations would be prohibitive due to the massive amount of memory needed to provide 12 or more cores with the

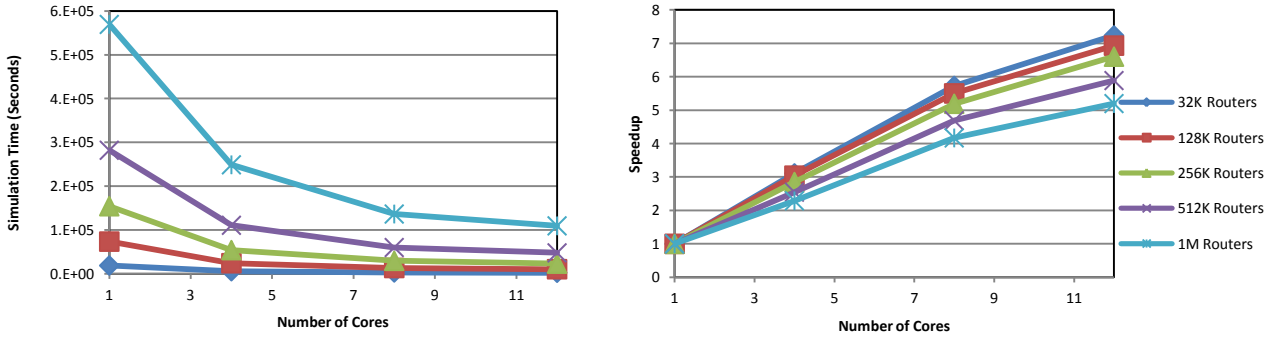

Figure 6 (a) Simulation Time for 50K Cycles (b) Speedup Observed.

memory requirements. According to the system size, scalability and the memory available in the server, the user could select the optimal number of threads per simulation. For example, in our case, if the number of routers is 256K, the optimal number of simulations to run in our server is three devoting for each one 4 cores. We would have 36GB memory utilization and almost a perfect speedup.

V. CONCLUSIONS

We have presented TOPAZ, which is a comprehensive and extensive tool conceived to facilitate interconnection network research. Its integration with one of the most common evaluation platforms in CMPs and its flexibility to simulate large-scale interconnection networks could make TOPAZ an attractive tool for a wide range of users. Given the current usage of GEMS, we hope the tool will attract many active users. In the long term, we will provide continued support for the tool, as it is one of our main resources for performing our research. The open source approach will also simplify how third party users can make their own contributions to the tool.

ACKNOWLEDGMENTS

The authors would like to thank José Ángel Herrero for his valuable assistance with computing environment, and the anonymous reviewers for many useful suggestions. This work has been supported by the Spanish Ministry of Science and Innovation, under contract TIN2010-18159, and by the HiPEAC European Network of Excellence.

REFERENCES

[1] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pp. 684-689.

[2] N. R. Adiga et al., "Blue Gene/L torus interconnection network," *IBM Journal of Research and Development*, vol. 49, no. 2.3, pp. 265–276, Mar. 2005.

[3] R. Kalla, B. Sinharoy, W. J. Starke, and M. Floyd, "Power7: IBM's Next-Generation Server Processor," *IEEE Micro*, vol. 30, no. 2, pp. 7-15, 2010.

[4] C. Park et al., "A 1.2 TB/s on-chip ring interconnect for 45nm 8-core enterprise Xeon® processor," in *2010 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2010, pp. 180-181.

[5] M. Coppola, R. Locatelli, G. Maruccia, L. Pieralisi, and A. Scandurra, "Spidergon: a novel on-chip communication network," in *International Symposium on System-on-Chip*, 2004, vol. 50, no. 2, pp. 15-22.

[6] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A detailed on-chip network model inside a full-system simulator," *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, pp. 33-42, Apr. 2009.

[7] M. M. K. Martin et al., "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 4, p. 99, 2005.

[8] J. Navaridas, J. Miguel-Alonso, J. a. Pascual, and F. J. Ridruejo, "Simulating and evaluating interconnection networks with INSEE," *Simulation Modelling Practice and Theory*, vol. 19, no. 1, pp. 494-515, Jan. 2011.

[9] F. Fazzino and M. Palesi, "Noxim: Network-on-chip simulator." [Online]. Available: http://sourceforge.net/projects/noxim.

[10] M. Rosenblum, S. A. Herrod, E. Witchel, and A. Gupta, "Complete computer system simulation: The SimOS approach," *Parallel & Distributed Technology: Systems & Applications, IEEE*, vol. 3, no. 4, pp. 34–43, 1995.

[11] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, p. 1, Aug. 2011.

[12] "Topaz Project page." [Online]. Available: http://code.google.com/p/tpzsimul/.

[13] V. Puente, J. A. Gregorio, and R. Beivide, *SICOSYS: an integrated framework for studying interconnection network performance in multiprocessor systems*. IEEE Comput. Soc, 2002, pp. 15-22.

[14] V. Puente, C. Izu, R. Beivide, J. A. Gregorio, F. Vallejo, and J. M. Prellezo, "The Adaptive Bubble Router," *Journal of Parallel and Distributed Computing*, vol. 61, no. 9, pp. 1180-1208, 2001.

[15] C. Carrion, R. Beivide, J. A. Gregorio, and F. Vallejo, "A flow control mechanism to avoid message deadlock in k-ary n-cube networks," in *Proceedings Fourth International Conference on High-Performance Computing*, 2001, pp. 322-329.

[16] W. J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, Mar. 1992.

[17] L.-S. Peh and W. J. Dally, "A delay model and speculative architecture for pipelined routers," in *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, 2001, pp. 255-266.

[18] N. E. Jerger, L. S. Peh, and M. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in *Computer Architecture, 2008. ISCA'08. 35th International Symposium on*, 2008, pp. 229-240.

[19] P. Abad, J. A. Gregorio, V. Puente, and P. Prieto, "Rotary router: an efficient architecture for CMP interconnection networks," in *International Symposium on Computer Architecture*, 2007, vol. 35, no. 2.

[20] P. Abad, V. Puente, and J. A. Gregorio, "MRR: Enabling fully adaptive multicast routing for CMP interconnection networks," in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, 2009, pp. 355-366.

[21] T. Moscibroda and O. Mutlu, "A case for bufferless routing in on-chip networks," *ACM SIGARCH Computer Architecture News*, vol. 37, no. 3, p. 196, Jun. 2009.

[22] Y.-C. Lan, H.-A. Lin, S.-H. Lo, Y. H. Hu, and S.-J. Chen, "A Bidirectional NoC (BiNoC) architecture with dynamic self-reconfigurable channel," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 3, pp. 427-440, Mar. 2011.

[23] M. Katevenis, "Fast switching and fair control of congested flow in broadband networks," *IEEE Journal on Selected Areas in Communications*, vol. 5, no. 8, pp. 1315-1326, Oct. 1987.

[24] A. Kumary, P. Kunduz, A. P. Singhx, L.-S. Pehy, and N. K. Jhay, "A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS," *2007 25th International Conference on Computer Design*, pp. 63-70, Oct. 2007.

[25] N. Neelakantam, C. Blundell, J. Devietti, M. M. K. Martin, and C. Zilles, "FeS2: A Full-system execution-driven simulator for x86 Simulating an Application," in *Poster ASPLOS*, 2007.

[26] H. Jin, M. Frumkin, and J. Yan, "The OpenMP implementation of NAS parallel benchmarks and its performance," *NASA Ames Research Center," Technical Report NAS-99-011*, Citeseer, 1999.

[27] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite," in *Proceedings of the 17th international conference on Parallel architectures and compilation techniques - PACT '08*, 2008, pp. 72-80.

[28] A. R. Alameldeen et al., "Simulating a $2 M Commercial Server on a $2 K PC," *Computer*, vol. 36, no. 2, pp. 50-57, 2003.

[29] V. Standard Performance Evaluation Corporation, SPEC*, http://www.spec.org, Warrenton, "SPEC 2006." .

[30] N. Kurd, J. Douglas, P. Mosalikanti, and R. Kumar, "Next generation Intel® micro-architecture (Nehalem) clocking architecture," in *2008 IEEE Symposium on VLSI Circuits*, 2008, pp. 62-63.

[31] M. Martin and M. Hill, "Token Coherence: a New Framework for Shared-Memory Multiprocessors," *Micro, IEEE*, pp. 108-116, 2004.

[32] J. Miguel-Alonso, J. Gregorio, V. Puente, F. Vallejo, and R. Beivide, "Load Unbalance in k-ary n-cube Networks," in *Euro-Par 2004 Parallel Processing*, 2004, pp. 900–907.