Self-Related Traces: An Alternative to Full-System Simulation for NoCs

Francisco Triviño, Francisco J. Andujar, Francisco J. Alfaro, José L. Sánchez
Universidad de Castilla-La Mancha, Spain
{*ftrivino, jandujar, falfaro, jsanchez*}*@dsi.uclm.es*

Alberto Ros,
Universidad de Murcia, Spain
aros@ditec.um.es

POSTER PAPER

ABSTRACT

The network-on-chip (NoC) has become an integral part of multicore systems and multiprocessor systems-on-chip (MPSoCs). Detailed simulation models are one of the most common techniques to evaluate the performance of a NoC. Most of these models only include a subset of the complete architecture and use only synthetic traffic. However, there is usually a combined effect of other components of the architecture that can impact the obtained results. Thus, an alternative consists in modeling a full-system to obtain a complete architecture that allows us to simulate real workloads with high accuracy. In this paper, we first present the ***INetwork*** *interface which allows us to include any network simulator inside the Simics-GEMS system. For testing the simulator, we present a simple case of study for a baseline NoC model running real applications. We also present a trace-driven model based on* ***self-related*** *traces which allows using just the NoC simulator. With this model, we are able to obtain simulation results with high accuracy in a lower simulation time (reduction up to 75%).*

KEYWORDS: Network-on-Chip, Multicore, Simulation, Full-System, Trace-Driven.

1. INTRODUCTION

Simulation is one of the most important techniques used by computer architects to evaluate their innovations. Network simulators allow designers to quickly implement and test their own designs by allowing them to model the most important aspects involved in a NoC with high accuracy. Usually, most studies assume that the traffic of the applications can be modeled as synthetic traffic. Traffic patterns, uniform distributions and/or constant message generation rate are frequently used. In contrast, the workload of the simulator should be as realistic as the application that will run in the real system. For instance, a different order of message arrival caused by the interconnection network can impact the memory system behavior and may affect the performance of the CMP. This interaction is not simulated when synthetic traffic is used and might have negative effects on the accuracy of evaluation results of a given proposal.

A full-system simulator is a very accurate way of evaluating the behavior of a complete architecture. Full-system simulators consider both most components of the simulated architecture and the impact of the operating system on the execution of the applications. One of these platforms is the system composed by Simics and GEMS [1]. In order to obtain accurate results when designing new approaches for the interconnection network, we have developed an interface, named as *INetwork*, that allows to include any network simulator replacing the original network implemented in GEMS. Particularly, we have added Noxim [2], our target NoC simulator, to the Simics-GEMS infrastructure.

Unfortunately, a full-system simulator requires important computation requirements because of the high number of detailed components that are simulated. In order to deal with the lack of accuracy of synthetic traffic and the large execution times of a simulation, we propose a trace model called *self-related* traces that are independent of the timing

parameters of the network, since every message in the trace file contains some fields that indicate the dependency with respect to other previous messages. Thus, accurate simulations can be obtained with a reduced simulation time.

The structure of this paper is as follows: Section 2 presents the related work. Section 3 describes our full-system simulator and the INetwork interface. In Section 4 we present a case study to show the usefulness for researching architectures targeted to the use of NoCs. Section 5 presents the trace-driven model proposed in this work. Finally, Section 6 presents the conclusions.

2. RELATED WORK

Traditionally, researchers have used custom simulators that can simulate traffic for being used in NoC research. Nostrum [3] is a flexible NoC simulator focused on communication primitives along loss-less switches that implements protocol stack for link, network, and session layers.

Nirgam [4], Noxim [2], and Sicosys [5] are general-purpose interconnection network simulators for multiprocessor systems that allow the modeling of a wide variety of message routers in a precise way. These tools offer a lower computational cost having in mind modularity, versatility and connectivity with other systems.

Unfortunately, these simulators obtain results by using synthetic traffic or traditional trace files. In fact, the main shortcoming of all the simulators discussed above is that they do not simulate other components involved in a CMP architecture as the memory system, the processing elements, or their interactions with the NoC.

On the other hand, Garnet [6] is a NoC simulator which has been recently included in the Simics-GEMS infrastructure. It provides different operation modes that differ in micro-architectural details of the on-chip router that is modeled. Although Garnet is an excellent tool due to its accuracy and inter-operability with Simics-GEMS, it does not provide full end-to-end models of the NoC (e.g., core network interface, other IP infrastructure) within a single framework. Additionally, only 2D-mesh topologies can be modeled with the Garnet simulator.

Finally, Netrace [7] uses a communication library that captures dependencies at system memory level into a trace-file. However, in this case the NoC is not simulated when the trace-file is generated and, therefore, the dependencies do not include all interactions among the different components of the CMP system. Moreover, these trace-files are very limited to a particular processor and memory system configuration. In order to obtain a more precise evaluation study, we still need a system simulation and benchmarking platform equipped with all the architecture components. We can solve this drawback by using our INetwork interface.

These network simulators can be easily integrated with the Simics-GEMS system in order to obtain a full-system simulator. In this paper, we provide an example with the Noxim simulator.

3. FULL-SYSTEM SIMULATION TOOL

In this section we briefly describe each tool involved in the full-system simulator. Simics [8] is a platform for full-system simulation which attempts to strike a balance between accuracy and performance by modeling the complete final application and providing a unified framework for hardware and software design. The General Execution-driven Multiprocessor Simulator (GEMS) [1] is a simulation toolset to evaluate multiprocessor architectures using Simics that models the system memory hierarchy and the system interconnect. GEMS is composed of two main simulation modules: Ruby and Opal. Ruby is the module that implements the cache hierarchy and Opal allows simulating out of order processors.

Ruby uses an interconnection network model to simulate all communication in a simulated system. As such, all intra-chip and inter-chip communication is handled as part of the interconnection network, although each individual link can have different latency and bandwidth parameters. We have replaced this design by the Noxim NoC simulator by using the interface detailed in Section 3.1. The Noxim kernel is based on SystemC and simulates a NoC in detail up to the micro-architecture level. Noxim provides flexibility to specify many different properties of a NoC such as the routing algorithm (deterministic or adaptive), arbitration policies, buffer size, etc. Additionally, a wormhole switching router with virtual channel flow control is implemented.

On the other hand, power estimation is extremely important in this context in order to verify that power budgets are approximately met by the different components of a certain design, and evaluate the effect of the different proposals. For this reason, we have integrated the Orion 2.0 [9] power model inside the Noxim simulator to estimate the energy and area consumption of the simulated NoCs.

Figure 1 shows a global scheme of all the simulation tools running together. The top of the figure represents the Simics simulator. Simics is extended by the GEMS simulator to evaluate multiprocessor architectures. In GEMS, the most important module is Ruby where the memory system is modeled (in the middle). Finally, the interconnection network implemented in Ruby is extended through the INet-

work interface. This interface allows us to include any NoC simulator. In our case, we use the Noxim simulator where the Orion 2.0 tool has been integrated.

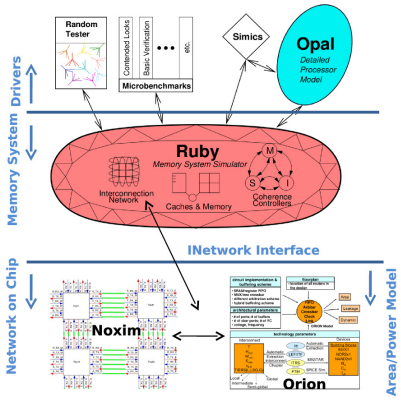

Figure 1. GEMS Simulator Extended by Noxim-Orion NoCs Simulator Through the INetwork Interface.

3.1. The INetwork Interface

We have developed the *INetwork* interface that allows us to include any network simulator in a simple way. This interface is highly independent of the network simulator. Ruby uses a queue-driven event model to simulate timing. Although many buffers are used in a strictly FIFO manner, the buffers are not only restricted to FIFO behavior. The simulation proceeds by invoking the *wakeup()* method for the next scheduled event on the event queue. The *INetwork* class produces events that are injected in the queue-driven event of the Ruby module, and these events implement the *wakeup()* method for our purpose.

In a global view, the transactions are produced as a consequence of the execution of an application in Simics. To satisfy a load/store that misses in the private cache of the processor that issues the request, a memory transaction begins. These transactions are composed by several network messages. When a transaction occurs, the driver sends a wait signal to Simics. The next step consists in simulating the messages crossing the Noxim network from a Ruby component to another one. After that, the obtained latency (due to memory and network) must be returned to Simics. The process continues with other transaction and so on.

The *wakeup()* routine is described in Algorithm 1. For each input port and for each $message$ that needs to be simulated in the network, this routine is responsible from extracting the $message$ (line 3), and transforming the Ruby $message$ to a Noxim $packet$ (line 4). Then, the $packet$ is injected in the Noxim network (line 6), previously storing the $packet$ id in the system (line 5). This $packet$ id is used to identify the packet while it is crossing the Noxim network.

Then, one cycle of the Noxim network is executed and, subsequently, the *wakeup()* method performs the inverse process described above. That is, for each node of the simulated system, and for each received $packet$ from Noxim, the *wakeup()* method extracts the id of the received $packet$ from the corresponding node (line 12 and line 13), and informs to Ruby of its reception in the simulated network. Finally, the noximNetwork event is injected in the global Ruby queue in order to be triggered again one cycle later.

Algorithm 1 wakeup(noximNetwork) pseudocode.

```
1:  for all input ports do
2:      for all message in the port do
3:          message ← get message from Ruby
4:          packet ← transform(message)
5:          stored packet id
6:          inject into noxim network (packet)
7:      end for
8:  end for
9:  put on noxim signal clock (1 cycle)
10: for all node of the CMP do
11:     for all received packet do
12:         packet ← get packet from noxim
13:         id ← get id (message)
14:         inject into Ruby (id)
15:     end for
16: end for
17: Ruby event queue ← noximNetwork event
```

Finally, we have extended the Noxim simulator with the Orion 2.0 model [9]. The integration consists in merging both simulators aided by the *INetwork* interface. On the one hand, Noxim simulator must account for all the actions involving energy consumption, for instance, to transmit a flit over the links, to forward a flit in one router, to store a flit in the input/output ports, etc. On the other hand, the Orion simulator estimates the energy and the area consumption depending of the configurable parameters such as the technology, operating frequency, operating voltage, etc.

4. CASE OF STUDY

Our full-system simulator consists of the Simics-GEMS system plus the Noxim-Orion simulator, and the interface that integrates both systems. Thus, we are able to simulate real world benchmark applications for evaluating the performance of NoCs.

To carry out the case of study, we model a homogeneous CMP with in-order cores. The CMP is structured in a number of nodes, each with a processing element (UltraSparc III), a private 32KBytes L1 cache, a portion of the shared L2 cache (1MByte), a directory cache bank, a mem-

ory controller, and a router. The L2 cache and the directory are physically distributed but logically shared at the chip level. Coherence between L1 and L2 caches is kept using a directory-based MOESI protocol. We assume 3D-stacking [11], and therefore, each node in the CMP has a memory controller. Regarding the on-chip network, we use a mesh topology which has all the links among nodes of the same size and width. The flit size is equal to the network link width, and we use a 8Bytes flit size. Finally, we use the XY routing algorithm in order to easily prevent cyclic dependencies in the network.

Regarding the Orion 2.0 tool we assume the configurable parameters most commonly used. For instance, we assume 32nm technology, the transistor type LVT for high performance networks, the operating voltage is $1.2v$, the wire layer type is intermediate and the layer length is 100μm according to the 32nm technology.

As workload we have used the PARSEC v2.1 [10] benchmark suite, which consists of 9 applications and 3 kernels where each one has been parallelized and focused on emerging workloads. Due to the lack of space, we have only selected the Blackscholes (BC), Bodytrack (BT), Streamcluster (ST), and Swaptions (SW) applications. In the next section, we analyze the impact on the number of cores as an example of what the platform can provide.

4.3. Impact on the Number of Cores

In this section we analyze the performance impact when the number of cores varies. For illustration purpose, we use the baseline NoC configuration and the workloads previously proposed. We have tested a CMP having 4, 8, and 16 nodes arranged in a 2x2, 2x4 and 4x4 2D meshes. Applications have been parallelized for the number of cores used in each test. We have collected statistics since the applications start until the end. Moreover, each value shown in the figures is the average of 30 different simulations varying the initial seed, and the confidence interval has been set to 95%. In order to illustrate the variation of performance, all the results in Figures 2 and 3 are shown in normalized terms compared with those obtained from the 2x2 mesh size case.

Figure 2.(a) represents the normalized execution time. As we can see, the applications obtain a good performance and scalability, with a reduction of 15% from 4 to 8 cores and of 30% from 4 to 16 cores. In general, these applications have a good scalability degree up to 16 cores.

Figure 2.(b) shows the network latency for the selected applications. For all the cases as the mesh size increases, the latency obtained also increases (approximately 150% from 2x2 to 4x4 mesh sizes). As expected, for the 2x2 mesh

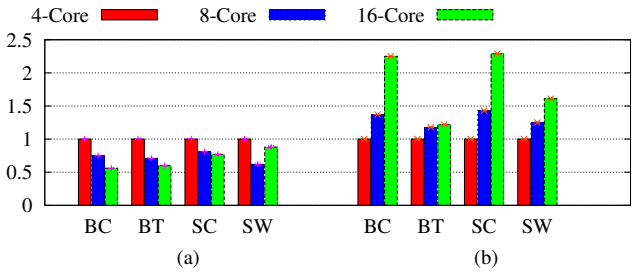

Figure 2. Normalized (a) Execution Time, and (b) Network Latency.

size the average message distance is significantly reduced compared to the 4x4 mesh size. Thus, the traffic of each application has a very low latency. When source and destination nodes are not placed adjacent to each other on the network, a packet needs to travel several intermediate nodes until reaching the destination. Although with a wormhole switching technique the message distance has not a direct influence on the latency, when the number of hops a message needs to arrive at its destination increases because the distance increases, the probability of interference with other packets grows, which turns in an increase of the latency.

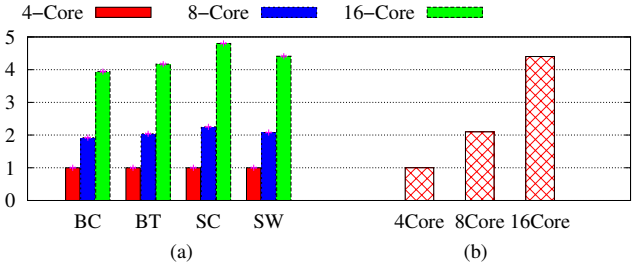

Figure 3. Normalized (a) Network Energy Consumption, and (b) Network Area Consumption.

Finally, we also show results of power and area consumption obtained from the Orion simulator. As expected in the Figure 3.(a), the 4x4 mesh size has a higher energy consumption because there are more network components running at the same time (routers and links). Finally, the network area is also higher as we increase the mesh size. In fact, the total amount of network resources increases proportionately over the mesh size, and its area consumption also increases (Figure 3.(b)).

5. SELF-RELATED TRACES

The full-system simulator has great computation requirements and one simulation could take days or even weeks. This is the main drawback of the full-system tool. In this section, we describe a trace-driven simulator that is able to obtain accurate results for the Noxim-Orion network by using *self-related* traces.

In contrast of traditional traces, self-related traces contain network messages which are independent of the timing parameter of the network. In contrast, each trace is only dependent on a previous message. To do this, every network message includes a field that indicates its dependency with the reception of another message. Additionally, each system component processing a network packet needs some time to perform certain operations (e.g., to write data in cache or main memory, executing other instructions, etc.). Our trace-based model should also account for this processing time. To this end, messages in the trace files are also extended with a new field that indicates the time elapsed from the reception of the previous message to the issue of the new dependent one.

Essentially, each message in the self-related trace has the following fields: the message *ID* (unique identifier), the source node memory component (*SC*), destination node memory component (*DC*), the message type of the coherence protocol (*T*), the size in flits (*S*), the computation time of the system component that receives a message in cycles (*CT*), and the id of the message that must be received before issuing the corresponding message (*D*). The two later fields are the key of the self-related traces. The *CT* field indicates the computation time that the simulated component needs for processing the received message and executing its corresponding actions before sending the next message. The *D* field indicates a message dependency. A message m_1 is dependent on another message m_2 when a system component c receives the message m_2, performs some computations during a certain *CT* time, and then sends message m_1. This dependency arises either because the message m_1 uses the information contained in the message m_2, or because the component c has no way to reach the instruction that sends the message m_1 until it receives the message m_2, even if m_1 does not use the information in m_2.

Table 1 shows the first messages of a self-related trace file. In this example, the message with *ID* 2 has a dependency with message 0. Also, message 4 has a dependency with message 2; message 5 depends on message 4, and so on. A value of -1 in the *D* field indicates no dependency. Notice that these messages only appear at the beginning of the trace and the remaining ones depend on previous messages.

We have modified the source code in Ruby for recording into a trace file the set of messages that are sent through the network. In this way, we can generate a trace file with the full-system simulator as a result of the simulation of a particular benchmark. Then, this trace can be used to feed the Noxim-Orion simulator as a real workload, and thus, this allows us to quickly run a set of simulations for testing a certain design. This mode offers a significant reduction in terms of simulation time while maintaining the accuracy of the obtained results, as we will see in Section 6.

Table 1. Example of a Self-Related Trace.

ID	SC	DC	T	S	CT	D
0	0-L1Cache	0-L2Cache	Gets	8	3	-1
1	1-L1Cache	1-L2Cache	Gets	8	3	-1
2	0-L2Cache	2-Directory	Gets	8	2	0
3	1-L2Cache	2-Directory	Gets	8	2	1
4	2-Directory	0-L2Cache	Data	72	158	2
5	0-L2Cache	0-L1Cache	Data	72	4	4

5.1. Efficiency and Accuracy of Self-Related Traces

In this section, we show the improvements in simulation time of using self-related traces, as well as its validation compared to the full-system simulation.

In Figure 4.(a) we can see the simulation time employed by each component of the full-system simulator. About 75% of the simulation time is employed by the Simics-GEMS component. Therefore, the self-related traces can help to reduce significantly the simulation time. Figure 4.(b) shows the improvements in simulation time of using self-related traces. We can see that the simulation time is reduced by 75% approximately. As can be seen, the benefits of using our trace model are significant.

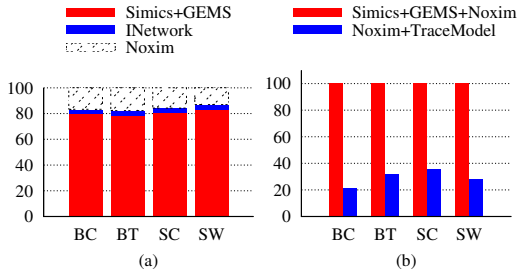

Figure 4. Simulation Time of (a) each Full-System Component, and (b) each Simulation Model.

Finally, we test the trace-driven model in order to ensure that the results are valid compared to the full-system model by using a statistical hypothesis test. The first step is to state the hypotheses to be tested. Our hypotheses are: H0 represents "trace-driven model is valid" and H1 represents "trace-driven model is invalid". The second step is to compute the relevant statistic test with the experimental results. In our case, we have selected the *HotellingsT2* test because is widely used to test the multivariate models equality [12]. We have run the BT application, varying the parameters of the on-chip network, with the trace-driven model and comparing the results with the full-system simulator results. The aim of the experiment is to conclude whether the full-system simulator and the trace-driven model show significant differences in the evaluation results under the same

configuration parameters. The test will obtain the probability of observing a result at least as extreme as the test statistic: the p-value. To understand this value in a correct way, the direct interpretation is that if the p-value is less than the required significance level, then we say the H0 hypothesis is rejected. We have selected a level of significance of 0.05, that is, there is a 5% of probability of rejecting it because of random variation.

As result of the test, the p-value is 0.1441 which is more than 0.05. Therefore, the H0 hypothesis is accepted at the given level of significance and we can conclude that the trace-driven model is valid under the given experimental frame. Therefore, the trace-driven model could be used to perform fast simulations and obtain experimental results. However, to get more accurate evaluation results the full-system simulator is needed.

6. CONCLUSION

In NoCs, new proposals are constantly appearing, but in the literature most of these proposals are tested with synthetic workloads. Consequently, not including real workloads can alter the final results. According to this fact, we have presented a simulation tool for evaluating the performance of NoCs that allows us to use workload of real applications. Thus, it is necessary to consider a simulation environment that allows to model a complete architecture.

In this paper, we have presented the *INetwork* interface that allows to include any network simulator inside the Simics-GEMS full-system platform. However, the main disadvantage is the high simulation time required to simulate a full-system with high level of accuracy. Therefore, we have also presented a trace-driven model that allows a great reduction of the computational time needed by the simulations (about 75%).

Finally, we have tested the equivalence of the results obtained with both models, and we have concluded that there are no significant differences between the trace-driven model and the full-system simulator. For testing the simulator, we have evaluated several mesh sizes running different applications from the PARSEC v2.1 benchmark suite.

ACKNOWLEDGEMENTS

This work was supported by the Spanish MEC and MICINN, as well as European Comission FEDER funds, under Grants CSD2006-00046 and TIN2009-14475-C04. It was also partly supported by JCCM under Grants PCC08-0078-9856 and POII10-0289-3724, and by the project NaNoC (project label 248972) which is funded by the European Commission within the FP7 Research Programme.

REFERENCES

[1] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, "Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset," *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[2] "Noxim Simulator," 2009. http://noxim.sourceforge.net/

[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*, Apr. 2005.

[4] "NIRGAM: A Simulator for NoC Interconnect Routing and Application Modeling," 2009. http://nirgam.ecs.soton.ac.uk/

[5] V. Puente, J. A. Gregorio, and R. Beivide, "SICOSYS: An integrated framework for studying interconnection network in multiprocessor systems," in *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing*, Jan. 2002, 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," in *ISPASS*, Apr. 2009, pp. 33–42.

[7] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: dependency-driven trace-based network-on-chip simulation," in *Proc. of the Third International Workshop on Network on Chip Architectures*, ser. NoCArc '10. New York, NY, USA: ACM, 2010, pp. 31–36. http://doi.acm.org/10.1145/1921249.1921258

[8] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A Full System Simulation Platform," *Computer*, vol. 35, no. 2, pp. 50–58, 2002.

[9] A. B. Kahng, B. Li, L.-S. Peh, and K. Samadi, "Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration," in *DATE*, 2009, pp. 423–428.

[10] C. Bienia and K. Li, "PARSEC 2.0: A New Benchmark Suite for Chip-Multiprocessors," in *MoBS*, 2009.

[11] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb, "Die Stacking (3D) Microarchitecture," in *Proc. of the 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO) '06*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 469–479.

[12] B. F. Manly, MULTIVARIATE STATISTICAL METHODS: A PRIMER. London, UK: Chapman & Hall, Ltd., 1986.