Speed and Accuracy Dilemma in NoC Simulation: What about Memory Impact?

Manuel Selva, Abdoulaye Gamatié, David Novo, Gilles Sassatelli

Speed and Accuracy Dilemma in NoC Simulation: What about Memory Impact?

Manuel Selva, Abdoulaye Gamatié, David Novo, Gilles Sassatelli
LIRMM (CNRS / Université de Montpellier)
161 Rue Ada, 34090 Montpellier
France
Email: firstname.lastname@lirmm.fr

***Abstract*—Network on Chip (NoC) communication infrastructures are increasingly being used in modern manycore architectures. Many industrial and research NoC simulators have been proposed in the last years in order to facilitate the design of such communication infrastructures. As any simulator, all of them have to trade off speed and accuracy. Simulation time directly depends on the simulation accuracy. It also directly depends on the complexity of the system to be simulated, e.g., the number of cores and their unit complexity. In this work, we show that the memory footprint of NoC simulators can be a serious factor limiting the simulation of manycore architectures with a large number of cores. We first quantitatively compare the memory footprint of a transactional level modeling NoC simulator and its cycle-accurate counterpart to show that memory footprint is a concern. Then, we show that memory footprint is also largely impacted by the choice of the programming abstraction by comparing two cycle-accurate simulators written using different application programming interfaces, i.e., plain C++ and SystemC.**

I. Introduction

Simulation software is a key component in system design process. During this process, it helps handling the increasing complexity of modern processors at different levels. Simulation tools first allow to perform large design space explorations purely in software. They also help reducing the time to market by allowing software development to start before the first version of the hardware is available.

Processors with 256 cores [1] are already on the market. According to the ITRS road-map, this core count in a single chip will continue to increase in the near future. To help designing these future processors, simulation tools must be able to handle this huge core count. One main concern when simulating such large systems is simulation time, which heavily depends on simulation accuracy and on the number of components to be simulated.

Among the existing studies on simulator design, most of the addressed issues concern the speed and accuracy dilemma. In other words, how to make the simulation faster while providing a relevant accuracy in the results. Nevertheless, none of the existing studies focus directly on the memory footprint, which can be a strong limitation to the practical simulation of large systems. As shown intuitively in Figure 1, for a given level of accuracy, simulation time becomes very long when the simulated manycore exceeds a particular number of cores. The point where performances drastically decrease is reached when the memory requirement of a simulation exceeds the physical memory size of the simulation host. In such a situation, the host system needs to use swapping mechanisms in order to handle the simulation. This drastically increases simulation time.

Even if the memory requirements of the simulation are below the capacity of the host, the swapping limit can be reached depending on the load of the system induced by other running applications. This is particularly true for desktop computers being more and more used to run simulations. Memory footprint is thus a concern that must be taken into account when choosing or building a simulation tool.

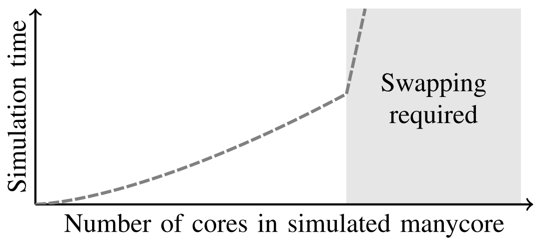

Fig. 1. Simulation time as a function of the number of cores in the simulated manycore. Because of swapping, simulation time increases drastically when the memory requirements of a simulation exceed the physical memory capacity of the host machine.

In this work, we focus on the simulation of Network on Chip (NoC) communication infrastructures that are increasingly being used in manycore architectures. We measure the memory footprint of different NoC simulators to evaluate when the swapping limit is reached in practice. We compare simulators classified according to two criteria:

- *Simulation abstraction* defines the level of details taken into account by the simulation. It can be either Transactional Level Modeling (TLM) simulation or cycle-accurate simulation;
- *Programming abstraction* concerns the level of abstraction (and thus the programming language) used to develop a simulator.

Our first contribution concerns simulators of NoCs for real-time embedded systems. We empirically compare the memory requirements of two simulators for a real-time NoC using

978-1-5090-2520-6/16/$31.00 ©2016 IEEE

different simulation abstractions. The first simulator, named McSim-CA, is cycle-accurate while the second one, named McSim-TLM is based on TLM concepts. The second contribution of this work compares the memory footprint of two cycle-accurate NoC simulators built upon different programming abstractions. We compare McSim-CA based on SystemC with BookSim [2], a cycle-accurate NoC simulator written in plain C++. In this paper we focus on the comparison of memory footprint depending on simulation and programming abstractions for NoC simulation. However, we believe that the conclusion drawn from the current study apply more generally to full system simulation.

The rest of the paper is organized as follows. Section II first introduces NoCs and simulation and programming abstractions commonly used to build NoC simulators. It then presents related work. Section III compares the memory footprint of McSim-CA and McSim-TLM both simulating the same NoC architecture but using different simulation abstractions. To show the impact of the programming abstraction on memory footprint, we compare in Section IV McSim-CA based on SystemC with BookSim, which is written in C++ only. Finally, Section V concludes and presents perspectives.

II. Background and Related Work

This section first introduces NoCs and the different ways of designing and implementing a simulator for such communication medium. Then, related work is reviewed.

A. Network on Chip (NoC)

NoCs alleviate the bottleneck of usual buses and the prohibitive cost of crossbars in the context of a large number of cores by allowing parallel communications. Figure 2 shows an example of a NoC-based manycore architectures with distributed memory. It is made of tiles communicating with each other by exchanging *packets* through the NoC. Each tile comprises a router dedicated to receive and forward packets over the NoC. Each tile also comprises a core and a private memory. Among existing NoC topologies, the 2D-mesh illustrated in Figure 2 is one of the most used.

As for distributed systems, routing, control flow and resource arbitration are required in NoCs. Routing consists in deciding which links of the NoC must be used to carry a message between different tiles. Control flow, also known as packet switching, decides the packet allocation to the internal router resources and to the NoC links. Resource arbitration decides in case of conflict between packets, which packet should be given priority. For performance concerns, this control flow and arbitration are often done at the *flit* level. The flits are the basic elements composing a packet. Many routing algorithms and control flow mechanisms have been proposed in literature [3].

Figure 3 shows the router architecture supporting the packet switching and arbitration policies considered in this work. For routing, we consider the *XY* strategy that first routes packets according to the X axis and then according to Y axis. Each router comprises five ports (some ports may not exist for routers at the boundaries of the mesh). Four ports are used to communicate with neighboring tiles. The extra port, called local port, allows communication with the core of the tile. Each port contains several buffers, called virtual channels, which temporarily store the flits of exchanged packets. These buffers provide a better utilization of the NoC links by storing blocked flits and still making usable the resources of the router and the links of the NoC. The arbitration block handles resource allocation in case of conflicts between packets. Different arbitration policies such as oldest-first or priority-based exist. In this work, we focus on priority-based arbitration [4]. With such a policy, each virtual channel is associated with a priority level, and the arbiter always gives priority to virtual channels with the highest priority. The priority of a flit is assigned according to the priority of the packet containing the flit. This packet priority is assigned by packet producers (cores) before sending them into the NoC. A credit-based mechanism, not shown in Figure 3 for the sake of simplicity, is used to ensure that routers do not forward flits to neighbors without enough space in their virtual channels.

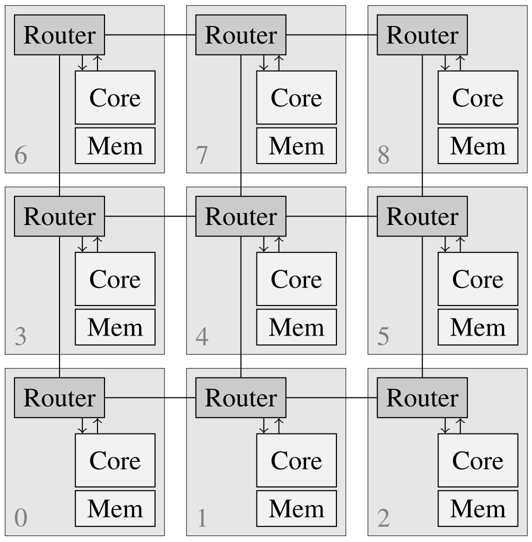

Fig. 2. Example of a 2D-mesh NoC-based manycore architecture with distributed memory. The architecture is made of tiles, each comprising a core, a private memory and a router allowing communication with other tiles.

B. Simulation Abstraction

In order to simulate a NoC in an accurate way regarding timing, many micro-architectural details should be taken into account. Indeed, most of them have an impact on the timing behavior of the NoC. As a consequence, a *cycle-accurate* simulator, i.e., accurate at the clock cycle granularity, needs to simulate all these details impacting timing. The simulation of all the details affecting accuracy comes at the price of simulation time; the more detailed the system, the longer the simulation. Cycle-accurate simulation also has a cost on the development of the simulator, again because all design details need to be taken into account.

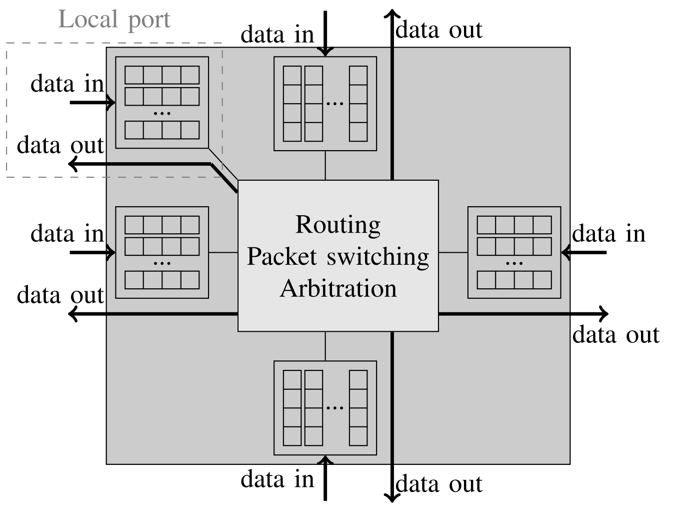

Fig. 3. Router with virtual channels. Each input port has several virtual channels used to store flits. The packet switching block is responsible to allocate output resources to input flits stored in virtual channels. Different arbitration policies such as oldest-first and priority-based exist.

To speed up the simulation and to reduce the complexity of building a simulator, the TLM abstraction has been proposed. In TLM, communications between components are represented as high level transactions abstracting away some details impacting timing. In other words, the accuracy of the results is decreased to speedup the simulation. TLM simulators are also often simpler to implement than their cycle-accurate counterpart because less details are taken into account.

The choice of the simulation abstraction depends on the expected result accuracy. In practice, TLM simulators are often built in first stages of hardware design to perform large design space explorations. They are also widely used to start software development before the hardware is available. Later on, cycle-accurate simulators are used in the design process for a more accurate validation of design choices.

C. Programming Abstraction

From the perspective of a simulator developer, beyond the simulation abstraction described above, the programming abstraction should be considered carefully. When building a simulator, at least, three levels of abstraction can be considered as shown in Figure 4.

1) Low level programming languages: At a low level, the developer of a simulator can decide to use a programming language such as C. In this case, all the simulator logic devoted to the management of time and concurrent processes, should be implemented from scratch. This task can result in high simulation speed if it is carefully handled, but it is time consuming and tedious.

2) Object-oriented languages: The next level consists in using higher level programming languages such as object oriented ones. As for low level programming languages, the developer has to manually implement the logic related to time and concurrent processes. Nevertheless, the high level concepts of the language such as classes and inheritance can make this task easier compared to using a language such as C.

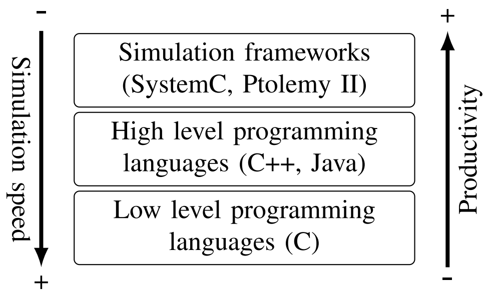
-
Simulation speed
+
Simulation frameworks (SystemC, Ptolemy II)
High level programming languages (C++, Java)
Low level programming languages (C)
+
Productivity
-

Fig. 4. Programming abstractions for hardware simulation. As in any domain, abstraction has a cost regarding performance.

3) Simulation frameworks: A higher programming abstraction is provided by simulation frameworks such as Ptolemy or SystemC. The latter, considered in this work, is a C++ based framework widely adopted in both industry and academia. It allows to implement simulators very easily. Indeed, all the management of timing and parallel activities is handled by the framework. To that end, the developer uses a high level Application Programming Interface (API) providing access to concepts such as hardware modules and hardware communication mechanisms such as signals and first-in-first-out buffers.

In addition to the above programming abstractions, further alternatives are not shown in Figure 4. Among them, hardware description languages provide a high accuracy, but they often induce low productivity and simulation speed. On the opposite, analytic models are very fast to simulate but provide low accuracy.

As in any other programming domains not related to simulation, abstractions have a cost regarding performances. In this work we evaluate this cost regarding memory footprint, impacting directly simulation time when swapping is required.

D. Related Work

Among all the cycle-accurate NoC simulators proposed in the past, BookSim [2] is a state-of-the-art simulator written in C++ only. It is able to simulate NoCs under different usage conditions. It provides support for configuring the NoC topology, the routing algorithm and the router architecture. NoCTweak [5] is a SystemC-based NoC simulator focusing on 2D-mesh topologies. It provides results for common performance metrics such as throughput and latency. Noxim [6] is also written in SystemC and allows to configure different parameters regarding the internal micro-architecture of the routers to be simulated. TOPAZ [7] is another simulator that enables the modeling of a wide variety of message routers. It was originally conceived to obtain results very close to those obtained by using hardware description languages.

To speedup simulations of NoC-based systems, TLM concepts can be applied. In this work we restrict the NoCs to be simulated to NoCs with priority preemptive virtual channels. It allows to simulate the NoC 1000 times faster with more than

90% of accuracy [8], [9]. Horsinka et al. [10] also proposed a TLM NoC simulator integrated in a multi-level simulation framework.

Most of the simulators described above, have been subject to analysis regarding simulation speed. Standalone analyses regarding the simulation speed and accuracy dilemma have also been proposed [11], [12]. To the best of our knowledge, none of these studies focused on the memory footprint criteria.

III. Impact of Simulation Abstraction on Memory Footprint

We compare the memory footprint of two simulators built in the context of the DreamCloud[1] European project. Both simulators are included in the Manycore platform Simulation (McSim) tool suite. They simulate the same NoC-based manycore architecture, but they use different simulation abstractions. The fastest simulator, called McSim-TLM[2], represents the NoC at a transactional level. The second one, called McSim-CA[3], is cycle-accurate. It handles all the internals of the NoC architecture impacting timing.

Both McSim-TLM and McSim-CA simulate the 2D-mesh NoC architecture depicted by Figures 2 and 3. From the cores perspective, both simulators simulate application models described at a high level. This application model is used only as a packet injector for simulating the NoC traffic and has been described in a previous work [13]. McSim-TLM and McSim-CA simulators only differ in the accuracy of the NoC simulator as described in the next two subsections.

A. McSim-TLM

The McSim-TLM simulator is based on existing work [9] and has been implemented in SystemC. To speedup simulations of such NoC architectures we rely on the fact that a packet with high priority will always preempt a packet with lowest priority. Based on this preemption property and considering static routing only, McSim-TLM simulates only the instants at which a packet enters and at which a packet leaves the NoC. Internally, McSim-TLM maintains a map containing the expected leaving time for all the packets into the NoC. Each time a packet enters the NoC, the expected leaving time of all the packets impacted by the new packet is delayed. Impacted packets are the ones with a lower priority and having a route interfering with the one of the new packet. Compared to the simulation of all the events happening inside all the routers of the NoC, McSim-TLM drastically reduces the number of simulated events.

Abstracting the NoC in this way has nevertheless some impact on the accuracy of results. McSim-TLM may consider conflicts that do not occur in practice. These false conflicts come from the fact that packets may not use all their route all the time in practice. The number of these false conflicts mainly depends on the NoC size and the packet length. In the opposite way, the assumption about using the entire route all the time can also lead to scenarios where the simulator ignores some conflicts occurring in practice [9]. Figure 5 illustrates these inaccuracies by showing the comparison of the execution time of the same application simulated with McSim-TLM and McSim-CA. In all the different NoC sizes except for 2x2, the McSim-TLM simulator considers conflicts that do not occur in practice leading to a higher execution time than the one reported by McSim-CA. In the 2x2 configuration, we face the opposite situation where the TLM abstractions lead to ignoring conflicts occurring in practice.

[1]http://www.dreamcloud-project.org/
[2]https://github.com/DreamCloud-Project/McSim-TLM-NoC
[3]https://github.com/DreamCloud-Project/McSim-CA-NoC

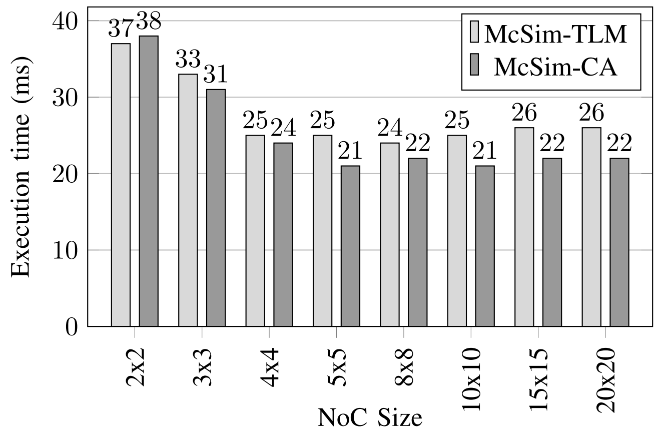

Fig. 5. Accuracy comparison between McSim-TLM and McSim-CA. McSim-TLM can under and over estimate the real contention occurring in the NoC.

B. McSim-CA

For McSim-CA, because of the complexity of writing a cycle-accurate NoC simulator from scratch, we reuse an existing simulator. We opted for NoCTweak [5] for two main reasons. First, because the NoCTweak simulator is based on SystemC, the integration of the cores simulation engine already developed for McSim-TLM was straightforward. Second, thanks to its modular design, NoCTweak was easily extended to support arbitration based on virtual channels. In McSim-CA, all the internal components of each router, e.g., virtual channels, arbitration logic, are simulated. McSim-CA extends NoCTweak with a new virtual channel allocator and a new switch allocator both based on priority.

C. Experimental Setup

All the experiments described in the following have been performed on a system with an Intel i5-4670 processor, 4 gigabytes of main memory and running Linux. We use the last 2.3.1 version of the Accellera SystemC implementation[4]. In order to monitor memory footprint during each simulation, we periodically read the `/proc/PID/smaps` virtual file reflecting kernel memory data structures for the process identified by `PID`. We simulate the same application model using different NoC sizes for both McSim-TLM and McSim-CA using 8 virtual channels with 16 flit entries. The application model has a uniform randomly distributed injection rate of 0.0069.

[4]http://www.accellera.org/downloads/standards/systemc

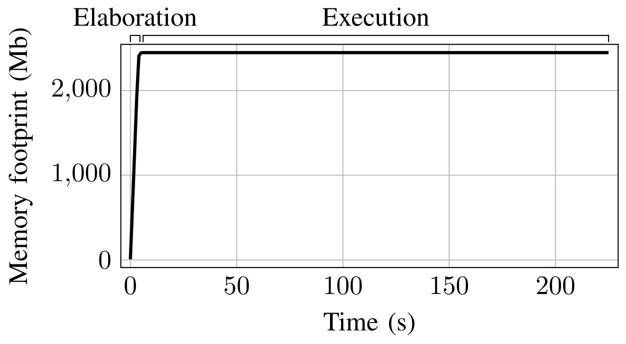

Fig. 6. Memory footprint of McSim-CA over time for during the simulation of a 16x16 manycore architecture. All the memory allocations are performed during the elaboration phase and never freed until the end of the simulation.

It is worth mentioning that the simulated application and thus the injected NoC traffic, do not impact the memory footprint. Indeed, a SystemC simulation is made of two phases. The first one, usually called elaboration, instantiates all the objects and their connections, i.e., modules and signals in SystemC terminology. The second phase, called execution, consists in effectively running the simulation. Most of the memory allocations take place in the elaboration phase. Thus, considered as static, even if they are dynamic memory allocation from the operating system point of view. As a consequence, the memory footprint measured at the end of the elaboration phase is present until the end of the execution phase. To support this claim, Figure 6 reports the evolution of the memory footprint for the cycle-accurate simulator simulating a 16x16 manycore architecture. From this plot, we see that the memory footprint is growing during elaboration phase and then stays constant up to the end of the simulation. Because memory footprint is almost constant from the end of the elaboration phase to the end of the simulation, we report only the average memory footprint in the following sections.

D. Results and Analysis

Figure 7 reports average memory footprint for both simulators according to the NoC size. The vertical axis reports average memory footprint in megabytes on a logarithmic scale while the horizontal axis shows different NoC sizes. As expected, this plot shows that McSim-TLM uses far less memory than McSim-CA. Nevertheless, the memory footprint of McSim-CA is surprisingly high. It uses around 3.8 gigabytes of memory to simulate a 20x20 platform while McSim-TLM only uses 31 megabytes which is 121 times lower. With McSim-TLM, we are able to simulate a platform made of 128x128 cores using one gigabyte of memory while the swapping limit of four gigabytes of our host machine is reached for a manycore with 20x20 cores with McSim-CA. Above this NoC size limit, the simulation time grows exponentially.

In McSim-TLM, a single object is created to simulate the NoC. The increase of the total memory footprint mainly comes from the creation of additional core objects. On the other hand,

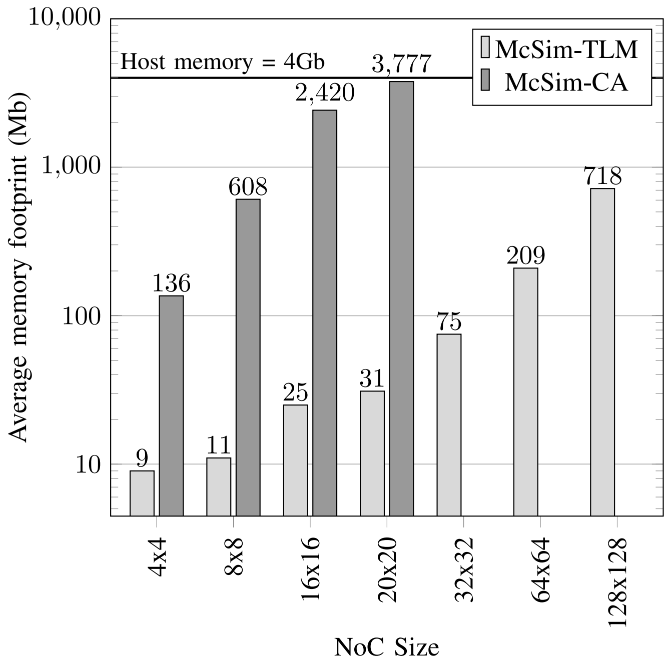

Fig. 7. Average memory footprint of McSim-TLM and McSim-CA. The memory footprint is expressed in megabytes on a logarithmic scale. Memory footprint of McSim-TLM is really small compared to the one of McSim-CA which is almost 4 gigabytes for a 20x20 NoC.

in McSim-CA, all internal components of the NoC have an associated object in memory. The simulator creates objects for each router, each virtual channel and each internal component of the routing and packet switching block. So, the total number of these objects and their corresponding memory footprint, is proportional to the size of the 2D-mesh.

From this experiment we see that simulating a 20x20 NoC based manycore with McSim-CA consumes all the four gigabytes of memory of our simulation host. Simulating bigger manycores will require swapping and thus will lead to impractically long simulations. More generally, this study gives additional information that can help the choice of going from TLM to cycle-accurate simulation. In addition to the well-known simulation speed degradation because of the number of events simulated, cycle-accurate simulation can lead to impractical simulation times if the swapping limit is reached.

IV. Impact of Programming Abstraction on Memory Footprint

In this section, we compare McSim-CA with BookSim [2], a state-of-the-art cycle accurate NoC simulator written purely in C++, i.e., not using SystemC.

A. BookSim Configuration

In order to be able to compare McSim-CA with BookSim, we calibrate the latter to simulate as close as possible the same NoC architecture as McSim-CA. We use the following configuration parameters:

- 2D-mesh topology,

- 8 virtual channels with 16 flit entries,
- default virtual channel and switch allocators,
- uniform injection rate.

Regarding virtual channel and switch allocators, BookSim does not support the notion of per channel priority compared to McSim-CA. We decided to use the default switch and virtual channel allocator provided in BookSim. Even if this difference could have a strong impact on the accuracy of the results, it does not influence significantly the memory footprint because in both McSim-CA and BookSim we use the same 2D-mesh topology, the same number of virtual channels and the same number of entries in each virtual channel.

Regarding the injection of packets, we can also fairly compare BookSim with McSim-CA even if the latter uses an application model to inject packets into the NoC while the former uses synthetic injection patterns. The memory footprint contributed by the objects corresponding to the NoC architecture is very high compared to the footprint of the objects corresponding to packets. To assess this claim, we performed memory evaluation of BookSim for a 20x20 and a 128x128 manycore architectures under different injection rates. Figure 8 shows the average memory usage of BookSim for injection rates varying from 0.0001 to 1 (for example, an injection rate of 0.20 means each core injects one packet every 5 clock cycles). From this figure, we see that most of the memory used by BookSim is not related to the number of injected packets. Over 1 gigabyte of memory is required to run the simulation of the 128x128 architecture with an injection rate as small as 0.0001. The memory footprint then increases slightly up to an injection rate of 1. This slight increase is due to the memory required to store packets into temporary queues at cores level before they are injected into the network. These queues grow infinitely for injection rates above the saturation point of the NoC. Nevertheless, the graph shows that the memory footprint induced by these packets is much lower than that of the objects corresponding to the NoC architecture. Moreover, in the following we report results for an injection rate of 0.0069 because it is the injection rate of the application used with McSim-CA. As shown in Figure 8, the additional memory footprint caused by packet objects is negligible for such a low injection rate.

We report average memory footprint for each simulation. Figure 9 shows that BookSim also performs most of its memory allocations before starting the effective simulation, i.e., most of the memory is allocated at the end of the initialization, which takes about two seconds. Then the memory footprint slightly increases, but not significantly, during the simulation phase up to the end of the simulation (time nine).

B. *Results and Analysis*

Figure 10 shows the results of the average memory footprint of BookSim compared to McSim-CA. For McSim-CA the results are the same as the ones presented in Section III. From this plot it is clear that BookSim memory footprint is much lower than the one of McSim-CA. BookSim can simulate manycore architecture with 128x128 (i.e., 16384) cores using

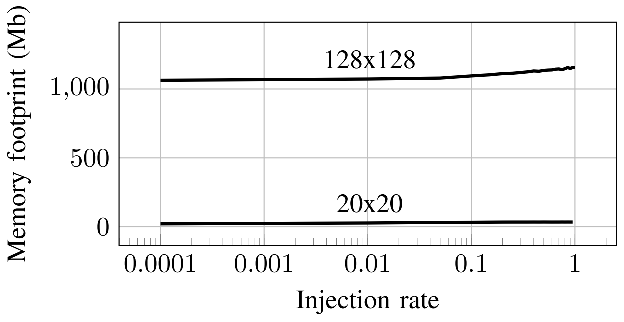

Fig. 8. BookSim memory footprint according to injection rate for a 20x20 and a 128x128 manycore architectures. Varying injection rate from 0.0001 to 1 does not impact memory footprint significantly.

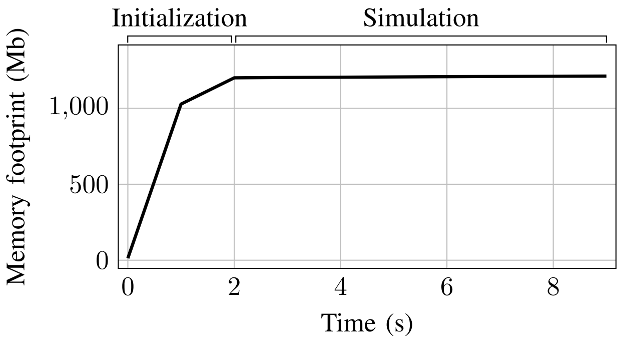

Fig. 9. Booksim memory footprint over time for a 128x128 manycore platform with injection rate of 0.0069. As for McSim-CA, memory footprint increase in the first phase of the simulation before reaching a plateau.

1 gigabyte of memory while McSim-CA requires 3.8 gigabytes for 20x20 (i.e., 400) cores.

Because both BookSim and McSim-CA simulators create an internal memory representation for each object in the NoC, and because they simulate the same NoC architecture, the main explanation for this large difference is the individual size of each object type among router, virtual channel, buffer, link, and all the other objects required for the cycle-accurate simulation. We performed a memory analysis of McSim-CA by instrumenting the code to measure the size of all the allocated objects. No memory killer objects were identified even if we were able to save few megabytes by removing some object fields that were not strictly required. Further investigations on BookSim implementation are required to have a better understanding of this big gap, nevertheless our study leads to the conclusion that this big memory footprint is the result of a sub-optimized usage of SystemC. Indeed, to easily write a cycle-accurate NoC simulator, one is tempted to use extensively all the abstractions provided by the API, such as signals and buffers.

The main conclusion we draw from this comparison is that the programming abstraction has a strong impact on the memory requirement for a cycle-accurate simulator. Large memory footprint has an impact on the simulation when swapping is required. Depending on the size of the simulated

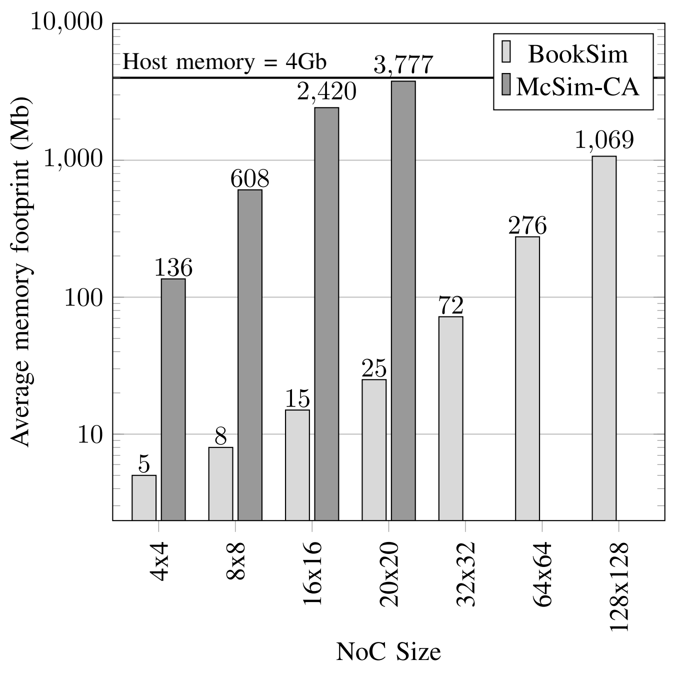

Fig. 10. Average memory footprint of McSim-CA and BookSim. Footprint is expressed in megabytes on a logarithmic scale. BookSim, written in plain C++, uses far less memory than McSim-CA which is based on SystemC.

architecture, memory footprint must be an important criterion for the choice of the programming abstraction. Moreover, it is worth mentioning that the memory limitation presented for NoC simulator in this work will be even more important for a full system simulator including cores. The swapping limit reached for a 20x20 manycore architecture with McSim-CA on our simulation host machine will be reached for smaller architectures in the case where McSim-CA also simulates cores accurately. The 4 gigabytes footprint of 20x20 simulations is also a problem when they run on server machines. On such machines, the memory available per core is few gigabytes by default. Running one simulation on each core of the system at the same time may also lead to the swapping limit.

V. Conclusion And Perspectives

This work presents an evaluation of the memory footprint of three NoC simulators that differ from the accuracy point of view and from the programming abstraction used to implement them. The study provides quantitative results to support the natural intuition that TLM NoC simulators also save memory in addition to time. By comparing the memory footprint of two cycle-accurate simulator based on different programming abstractions, we also evaluated the cost of using programming abstraction provided by framework such as SystemC. This study concludes that depending on the size of the NoC to be simulated, great care must be taken when building or choosing a NoC cycle-accurate simulator. Even if this study focuses on NoC simulators, we believe that the tendencies of the results presented would be similar for the simulation of other hardware components. In the case of a full system cycle-accurate simulator, the swapping limit would be reach earlier because of all the memory used to simulate the cores behavior accurately.

From this work, we think that it will be very interesting to study how existing SystemC model could benefit or not from lazy memory allocations. For large systems, instantiating a lot of components that are not used from the beginning of the simulation, such lazy allocation mechanisms could help for saving memory.

Acknowledgment

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under the DreamCloud Project: http://www.dreamcloud-project.org, grant agreement no. 611411.

References

[1] B. D. de Dinechin, Y. Durand, D. van Amstel, and A. Ghiti, "Guaranteed Services of the NoC of a Manycore Processor," in *Proceedings of the 2014 International Workshop on Network on Chip Architectures*, ser. NoCArc '14. New York, NY, USA: ACM, 2014, pp. 11–16.

[2] N. Jiang, D. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. Shaw, J. Kim, and W. Dally, "A detailed and flexible cycle-accurate network-on-chip simulator," in *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE International Symposium on*, April 2013, pp. 86–96.

[3] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[4] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny, "QNoC: QoS architecture and design process for network on chip," *Journal of Sytems Architecture*, vol. 50, pp. 105–128, 2004.

[5] A. T. Tran and B. Baas, "NoCTweak: A highly parameterizable simulator for early exploration of performance and energy of networks on-chip," VLSI Computation Lab, ECE Department, University of California, Davis, Tech. Rep. ECE-VCL-2012-2, 2012.

[6] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open, extensible and cycle-accurate network on chip simulator," in *Application-specific Systems, Architectures and Processors (ASAP), 2015 IEEE 26th International Conference on*, July 2015, pp. 162–163.

[7] P. Abad, P. Prieto, L. Menezo, A. Colaso, V. Puente, and J.-A. Gregorio, "Topaz: An open-source interconnection network simulator for chip multiprocessors and supercomputers," in *Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on*, May 2012, pp. 99–106.

[8] L. S. Indrusiak and O. M. dos Santos, "Fast and accurate transaction-level model of a wormhole network-on-chip with priority preemptive virtual channel arbitration," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[9] L. S. Indrusiak, J. Harbin, and O. M. Dos Santos, "Fast simulation of networks-on-chip with priority-preemptive arbitration," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 20, no. 4, pp. 56:1–56:22, Sep. 2015.

[10] S. A. Horsinka, R. Meyer, J. Wagner, R. Buchty, and M. Berekovic, "On RTL to TLM abstraction to benefit simulation performance and modeling productivity in NoC design exploration," in *Proceedings of the 2014 International Workshop on Network on Chip Architectures*, ser. NoCArc '14. New York, NY, USA: ACM, 2014, pp. 39–44.

[11] G. Schirner and R. Dömer, "Quantitative analysis of the speed/accuracy trade-off in transaction level modeling," *ACM Trans. Embed. Comput. Syst.*, vol. 8, no. 1, pp. 4:1–4:29, Jan. 2009.

[12] L. Lehtonen, E. Salminen, and T. D. Hmlinen, "Analysis of modeling styles on network-on-chip simulation," in *NORCHIP, 2010*, Nov 2010, pp. 1–4.

[13] K. Latif, M. Selva, C. Effiong, R. Ursu, A. Gamatie, G. Sassatelli, L. Zordan, L. Ost, P. Dziurzanski, and L. S. Indrusiak, "Design space exploration for complex automotive applications: An engine control system case study," in *Proceedings of the 2016 Workshop on Rapid Simulation and Performance Evaluation: Methods and Tools*, ser. RAPIDO '16. New York, NY, USA: ACM, 2016, pp. 2:1–2:7.