MPSoCSim extension: An OVP Simulator for the Evaluation of Cluster-based Multicore and Many-core architectures

Maria Méndez Real, Vincent Migliore, Vianney Lapotre, Guy Gogniat, Philipp Wehner, Jens Rettkowski, Diana Göhringer

MPSoCSim extension: An OVP Simulator for the Evaluation of Cluster-based Multicore and Many-core architectures

Maria Méndez Real, Vincent Migliore, Vianney Lapotre, Guy Gogniat
Univ. Bretagne-Sud, UMR 6285, Lab-STICC,
F-56100 Lorient, France
maria.mendez@univ-ubs.fr

Philipp Wehner, Jens Rettkowski, Diana Göhringer
Ruhr-University Bochum, Germany
{philipp.wehner, jens.rettkowski, diana.goehringer}@rub.de

***Abstract*—In this paper, an extension of the OVP based MPSoC simulator MPSoCSim is presented. This latter is an extension of the OVP simulator with a SystemC Network-on-Chip (NoC) allowing the modeling and evaluation of NoC based Multiprocessor Systems-on-Chip (MPSoCs). In the proposed version, this extended simulator enables the modeling and evaluation of complex clustered MPSoCs and many-cores. The clusters are compound of several independent subgroups. Each subgroup includes an OVP processor connected by a local bus to its own local memory for code, stack and heap. The subgroups being independent, the attached OVP processor model can be different from the other subgroups (ARM, MicroBlaze, MIPS,...) allowing the simulation of heterogeneous platforms. Also, each processor executes its own code. Subgroups are connected to each other through a shared bus allowing all the subgroups in the cluster to access to a shared memory. Finally, clusters are connected through a SystemC NoC supporting mesh topology with wormhole switching and different routing algorithms. The NoC is scalable and the number of subgroups in each cluster is parameterizable. For a dynamic execution, the OVP processor models support different Operating Systems (OS). Also, some mechanisms are available in order to control the dynamic execution of applications on the platform. Different platforms and applications have been evaluated in terms of simulated execution time, simulation time on the host machine and number of simulated instructions.**

I. Introduction

Networks-on-Chip (NoC) have recently emerged as a promising solution for on-chip scalable communication in highly integrated multi and many-core systems. Such an approach is very attractive in order to offer users with high performance. The design space of these sophisticated systems is very large. A simulator is thus mandatory for an early evaluation of such complex systems.

The OVP technology allows the connection between the OVP simulator and some platforms prototyped in SystemC [9]. MPSoCSim is an OVP based instruction accurate simulator for MPSoCs [1]. It extends the OVP simulator with a SystemC NoC in which each router is connected to a node. Each node is compound of an OVP processor model, a local memory for heap and stack and a local RAM. This latter is used to store the code but also to communicate with distant processors. MPSoCSim allows the modeling of heterogeneous MPSoCs with distributed memory. The NoC is scalable and currently supports 2-D mesh topology with wormhole switching and different routing algorithms (XY, minimal West-First and adaptive West-First algorithms). Finally, MPSoCSim supports traffic generators additionaly to OVP processor models in order to perform NoC design exploration. MPSoCSim has been validated through the comparison with an HW implementation [1].

The main contribution of this paper is the extension of MPSoCSim in order to model more complex multi and many-core systems. Thus it allows the modeling and validation of clustered NoC based platforms. Each cluster encompasses several independent subgroups and a RAM shared between the subgroups. Each subgroup is compound of an OVP processor model directly connected to its local memory. The processors in the subgroups can be heterogeneous (ARM, MicroBlaze, MIPS,...). The clusters are connected to each other through a SystemC NoC. The number of subgroups is parameterizable.

The remainder of this paper is organized as follows. Section II presents the related work on NoC based multicore simulators. In Section III, the structure of the MPSoC simulator MPSoCSim and its comparison to the HW implementation are presented. In Section IV the extension of MPSoCSim is explained, as well as some possible mechanisms in order to simulate a dynamic execution on the platform. Section V shows an evaluation of the extended simulator. Finally, Section VI draws some conclusions and presents future work.

II. Related work

Several simulators for NoC based MPSoCs are already available, each one addressing some specific issues. In this section, some of the existing simulators are described and compared with the extended version of MPSoCSim presented in this paper.

Full-system simulators, such as GEM5 [2] (full-system mode) and SimFlex [3] run applications and system software

(mostly OSes) modeling both micro architecture details and OSes. GEM5 for instance, chooses a complex execute-in-execute approach that integrates functional and performance simulations to model micro architecture details with a very high level of accuracy. However, this kind of simulator is very slow and is not dedicated to simulate many-core systems. In contrast, there are also available application-level simulators, such as SimpleScalar [4] or GEM5 [2] (system-call emulation mode). In contrast with [3] and [4], MPSoCSim benefits from OVP features and thus supports both, application-level simulations as well as different OSes such as Linux or FreeRTOS. Moreover, in contrast with MPSoCSim, GEM5 does not currently support NoC communication infrastructure. Finally, compared with GEM5, which is cycle accurate, MPSoCSim is instruction accurate with allows much shorter simulation time while being sufficiently accurate [1].

Several existing simulators for multi and many-core systems have been designed and focus on one subsystem of the multi or many-core system. Some of the cycle accurate simulators exploring the NoC design space are highlighted in the following. These simulators enable the configuration of several NoC parameters. Nirgam [5] is a cycle accurate SystemC simulator supporting several NoC topologies and enabling the configuration of the clock frequency, the buffer depth, flit size and virtual channels. In Noxim [6], a SystemC NoC simulator, parameters such as the network size, buffer size, routing algorithm, packet size distribution and selection strategy are customizable to evaluate the communication system. The evaluation is enabled by traffic generators. In addition, these generators support different traffic patterns. Booksim 2.0 [7] is another cycle-accurate NoC simulator. A traffic manager generates packets that are sent through the NoC. As all the aforementioned simulators, Booksim 2.0 has also parameterizable components. The topology, routing algorithm, flow control and additionally the router micro architecture are configurable. This enables the evaluation of different performance metrics.

All the aforementioned simulators analyze the NoC with traffic patterns generated by traffic generators. This evaluation is essential since it allows the study of the NoC properties. Compared to these previous works, MPSoCSim is also convenient for NoC simulation and evaluation since it currently supports a parameterizable mesh NoC. Moreover, MPSoCSim supports traffic generators in order to evaluate the NoC properties. However, in contrast with the work presented above, MPSoCSim also supports OVP processor models. Each processor is programmable and run its own code. This feature enables the HW/SW co-design of MPSoCs.

Similar to MPSoCSim, the work presented in [8] is also based on OVPSim. However, Rosa et al. focus on power estimations including an energy model in the OVPSim simulator. The simulation results are compared with a gate-level implementation of the simulated platform. On the contraty, power estimation is not the purpose of MPSoCSim.

Compared to previous efforts, MPSoCSim in its extended version is a flexible NoC based multi and many-core simulator. It enables the evaluation of the NoC properties since it supports traffic generators as well as configurable NoC parameters. Finally, MPSoCSim benefits from the OVP processor models features and thus supports OSes such as Linux and FreeRTOS models.

III. MPSoCSim in its original version

The OVP technology allows the connection of the simulator to existing SystemC platforms [9]. Within the SystemC environment, the OVP simulator executes open source processors. An OVP model is provided with a TLM2.0 interface in the form of a C++ header file [10]. To use OVP models, SystemC instantiates one tlmPlatform object. This object keeps the quantum period which sets how long each processor model instance waits before running again. This quantum is adjustable.

A. NoC simulator

MPSoCSim proposes an environment for the simulation of distributed multicore systems, based on a NoC [1]. Figure 1 shows an overall overview of the simulator. It relies on a NoC, modeled in SystemC, where each router is connected to a SystemC TLM Network Interface (NI) connected to a local group called node.

Each node is compound of an OVP processor model, a local memory (for the stack and heap) and a local RAM. Moreover, a SystemC timer has been attached to each node in order to provide local execution time results.

The local RAM is used to store the code but also to communicate with distant processors. In fact an API has been developed in order to send packets through the NoC. These packets are stored on the local RAM of the targeted node at an address offset chosen by the sender. Each processor can thus read into the local RAM within its own node and write to any other distant RAM.

The NoC currently supports 2D-Mesh topology, a communication through packets wormhole and several routing algorithms. The routers provide 5 connectors (Fig. 2(a)). Each connector consists of one target and one initiator sockets to enable TLM data transmission. A local router's port connects a node to the NoC through the NI.

B. HW implementation

In [1], MPSoCSim has been evaluated and compared to the HW implementation on a Xilinx Zynq device [11] which provides an ARM processor besides an FPGA. The simulated system which has been implemented is shown in Fig. 2(b). The FPGA contains a NoC and 3 MicroBlazes. The processors are connected by RAR-NoC [12]. The ARM processor communicates via the high performance port to the network. Only one core of the ARM processor is used. The MicroBlaze are linked via the FSL interface to the NoC.

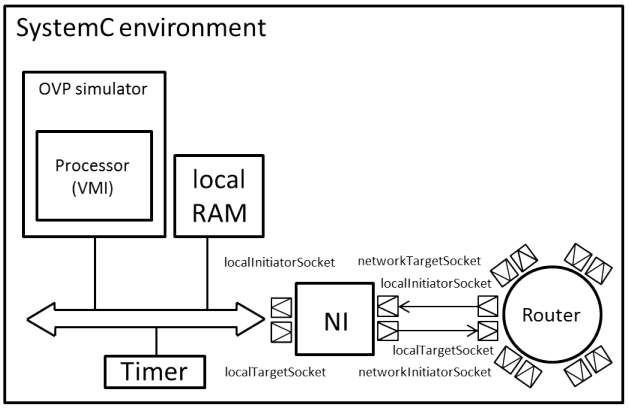
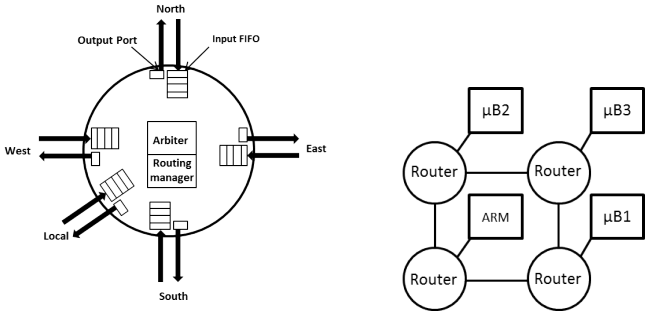


Fig. 1. MPSoCSim simulator



(a) Router structure (b) 2D-Mesh NoC

Fig. 2. Router structure and 2D-Mesh NoC [1]

Parameters	Chosen value
Quantum period	10us
ARM Frequency	667MHz
MicroBlaze (B) Frequency	100MHz
Nominal MIPS	100
Real flit time (ARM)	850ns
Real flit time (B)	40ns
Clock delay routing	0
Clock delay pass through	1
Network frequency	100MHz
Network size	2×2 clusters

TABLE I
SYSTEM PARAMETERS USED FOR EVALUATIONS

C. MPSoCSim evaluation

Table I sums up the system parameters used to compare the simulator with the HW implementation.

For the comparison with the HW platform, a tiled matrix multiplication has been used. The ARM creates the matrices to be multiplied and sends the necessary rows and columns to the MicroBlazes by dividing the matrices into equal parts. Each MicroBlaze does the corresponding computation and sends the results back to the ARM which collects all the results. Figure 3 shows the comparison between the simulator and the HW implementation for the multiplication of 4×4, 8×8 and 16×16 matrices. Results show a deviation of 17% for a

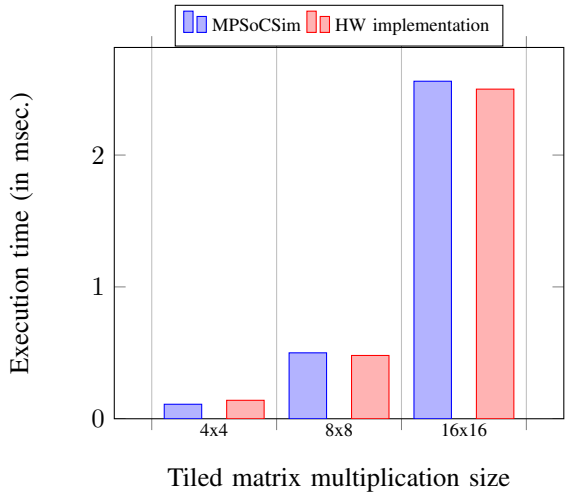


Fig. 3. Comparison between MPSoCSim and the HW implementation [1]

4×4 matrix. This deviation decreases for higher matrices sizes (down to 2.5% for 16×16 matrices multiplication). The results show an accurate simulation. The extension of MPSoCSim presented in this paper relies on this validation of MPSoCSim through the comparison with the real HW implementation.

IV. MPSoCSim extension

In its first version, MPSoCSim was able to simulate simple multiprocessor architectures, with one single processor per node with distributed memory. It has been extended, in order to allow the modeling of more complex clustered multi and many-core systems supporting distributed memory between clusters and shared memory within a cluster.

A. Clustering the architecture

The simulator has been extended at the node level in order to form a more complex cluster. Each cluster can encompass up to 4 subgroups and shared resources between subgroups such as a shared memory and a NI to connect the cluster to the NoC. Figure 4 shows the structure of each cluster.

Each subgroup is compound of one processor directly connected as a master to its local memory through a local bus (TLM decoder). The local memory stores heap, stack and the processor code. This choice is made in order to improve performance and avoid congestion on the shared bus. Indeed, this choice allows better performance since each processor locally access its code by its local bus instead of accessing to the shared RAM. In this way, potential congestion on the shared bus is avoided. Finally as subgroups are independent, processors can not access to other processor's local memory, thus unauthorized accesses to other processor's code is avoided.

A processor can read and write to its own local memory and to the shared RAM within its cluster. The shared RAM is used to exchange data between processors within the same

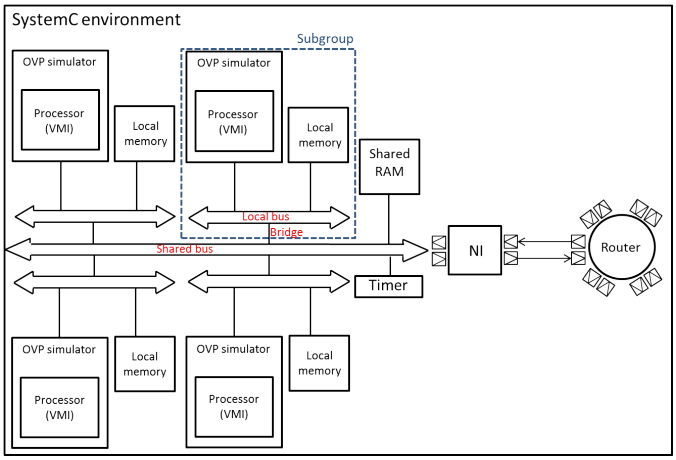


Fig. 4. Extension of the MPSoCSim simulator

cluster as well as with distant clusters by message passing through the NoC. Moreover, each subgroup is independent and processor models can be different in each subgroup (ARM, MicroBlaze, MIPS,...) allowing the modeling of heterogeneous architectures.

A shared bus (TLM decoder) connects the shared resources (shared RAM, NI, peripherals,...) to the NoC through the corresponding router. Subgroups are connected as initiators to the shared bus through a bridge, thus, no additional arbiter is needed.

The number of subgroups in each cluster as well as the number of clusters are parameterizable. This approach allows a flexible modeling and evaluation of clustered heterogeneous multi and many-core systems where the memory is distributed between clusters and cluster resources (e.g. shared RAM, NI, peripherals,...) are shared between processors within the cluster. This solution is flexible and scalable since each cluster can encompass a different number of independent and heterogeneous processors, each one executing its own application code.

B. Dynamic execution

The simulator benefits from the OVP processor models which support different OSes such as Linux or FreeRTOS. However, some other OVP and SystemC features can be used in order to simulate a dynamic scheduling of applications. Indeed, the SystemC environment enables to run the simulator for a given period of time. This feature enables to run, stop and resume the execution. Moreover the OVP environment allows the simulator to reset any processor through its reset pin. In this way, the processor's program counter is reseted and the application on this processor can be restarted from the beginning or another application code can be executed. It is worth noting that during the time taken to reset a single processor, the other processors are still running independently. These 2 mechanisms can be used together in order to control the dynamic execution on the platform. In this study, one cluster encompasses an ARM processor acting as a master of the platform. The other clusters are compound of 4 processors sharing local resources. The master of the platform run different deployment strategies (scheduling, mapping, memory allocation) in order to dynamically control the execution of several applications running in parallel. The extended version of MPSoCSim allows, in addition to the evaluation of the NoC, the evaluation of different dynamic deployment strategies on a multi or many-core architecture without the implementation cost of modifying an OS.

V. Evaluation

In this section, the evaluation of the extended simulator is presented.

A. Experimental protocol

The presented extension of MPSoCSim relies on the validation of MPSoCSim in its first version [1]. For the evaluation of the MPSoCSim extension, we use the same system parameters as for the validation of MPSoCSim (Table I).

The extension of MPSoCSim has been validated and evaluated through synthetic applications such as a matrix multiplication of several sizes on different architectures. For these experimentations, one cluster contains a single ARM and the other clusters encompass 4 subgroups (meaning 4 MicroBlazes in each cluster). Matrix multiplications of several matrices sizes have been evaluated on a 2×2 cluster (12 MicroBlazes and 1 ARM) and 4×4 cluster (60 MicroBlazes and 1 ARM) architectures.

As in Section III the ARM generates the matrices, divides the computation into equal parts, sends the corresponding rows to the MicroBlazes and collects the results. Each MicroBlaze executes its own code, does the corresponding computation and sends back to the ARM the generated results.

In Section V-B, the results for both architectures, for several sizes of matrix multiplications, in terms of *elapsed time*, *simulated execution time* and *number of instructions simulated* are discussed. The *elapsed time* is the host machine time necessary to simulate the execution scenario on the OVP environment. The *simulated execution time* is the time needed by the simulated system to complete the execution of the simulated scenario. The simulated execution time is also computed for each processor. Results in terms of *number of instructions simulated* in total as well as per processor are also available. Finally, results are compared with the execution on a homogeneous architecture.

B. Experimental results

Figure 5 shows the evaluation on the *elapsed time* in milliseconds necessary on the host machine to simulate the execution of several sizes of matrix multiplications on 2 different platforms; a 2×2 cluster and a 4×4 cluster architectures (12

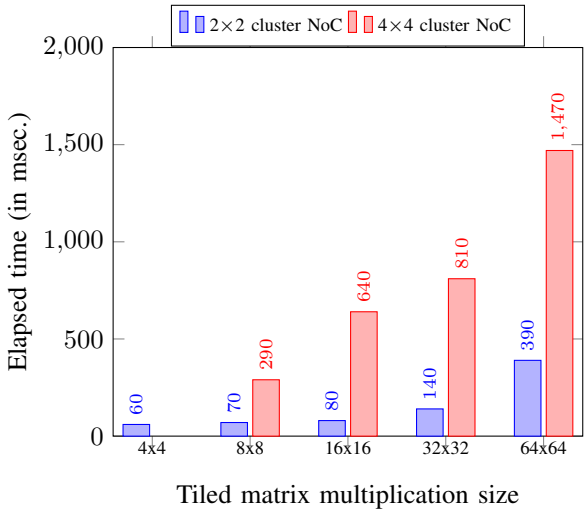


Fig. 5. **Elapsed time** (in msec.) on the host machine for different sizes of matrix multiplications on a 2×2 and 4×4 cluster architectures (12 MicroBlazes and 1 ARM, and 60 MicroBlazes and 1 ARM respectively)

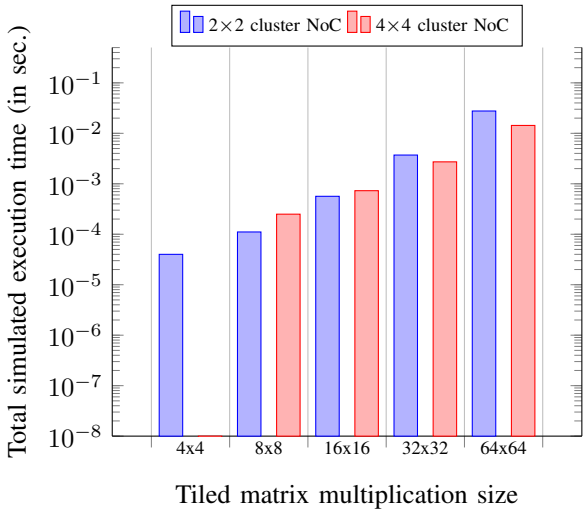


Fig. 6. **Total simulated execution time** (in sec.) for different sizes of matrix multiplications on a 2×2 and 4×4 cluster architectures (12 MicroBlazes and 1 ARM, and 60 MicroBlazes and 1 ARM respectively)

MicroBlazes and 1 ARM, and 60 MicroBlazes and 1 ARM, respectively). Notice that the 4×4 matrix multiplication has not been executed on the 4×4 architecture (60 MicroBlazes and 1 ARM). This is because each MicroBlaze may execute at least one computation. The results presented in this paper have been generated on an Intel Core 2 Quad Q9400, 2.66GHz frequency PC with 3.87 GB RAM (usable). The elapsed time on the host machine increases with the size of the simulated platform, however it can be noticed that the simulator enables very fast simulation time on the host machine even for the most complex architecture (around 1.5 seconds for a 60 MicroBlazes and 1 ARM architecture).

The *total simulated execution time* on the simulated system (Fig. 6), for several sizes of matrix multiplications on the 2 different platforms described above, has also been evaluated. These results are useful in order to choose the best architecture size for a given execution scenario by comparing the trade off of having more computational resources and the communication complexity overhead induced for larger and more complex platform. In this case, for instance, a naive remark is that, according to results in Fig. 6, for small matrix multiplications (below 32×32 matrix multiplication), it is not worth having a larger platform since the computation is negligible compared to the communication cost. In fact in larger platforms, the packets, always sent by the ARM, need to cross a greater number of routers to get to their destination. Furthermore, the MicroBlazes are polling the memory until the packets arrive. On the other hand, for greater computation, it is worth executing the application on larger platforms. However, if we look deeper into results, it can be noticed that executing a 16×16 matrix multiplication on a 4×4 cluster architecture provides a gain of 22% of the execution time compared to the execution on a 2×2 cluster architecture. However, to obtain this gain, 60 MicroBlazes are needed instead of only 12. Thus, the results obtained with this simulator can be used in order to choose the best size of the platform for a given execution scenario.

Finally, results in terms of *simulated instructions* for a given execution scenario, are also available in Fig. 7. These results allow the evaluation of the communication cost for different platforms for a given execution scenario. In fact, it can be seen that the communication overhead for a larger platform decreases when the matrix multiplication size increases for the same computation load (down to 22% more instructions for a 64×64 matrix multiplication on a 4×4 cluster compared to the execution on a 2×2 cluster architectures). The simulated instructions results are also provided per processor (Fig. 8) allowing the evaluation and validation of an execution scenario. For better understanding, only the results of a 2×2 cluster architecture (12 MicroBlazes and 1 ARM) are presented. It can be noticed for example that the ARM executes a greater number of instructions than the MicroBlazes in every matrix multiplication scenario. This is explained by the fact that the ARM is responsible for creating and splitting the computation between the number of MicroBlazes. The ARM sends as well the necessary rows to each MicroBlaze. Moreover, it can be seen from the results that any MicroBlaze is blocked waiting for data for instance which means that there is no congestion on the network. Finally, in the evaluated scenario, the processors within the cluster 3 execute more instructions than the other ones. This is because the cluster 3 is the furthest cluster from the ARM, thus, they wait longer for the data sent by the ARM (memory polling instructions). The provided approach could also be used in order to evaluate load balancing algorithms on such systems. It will be studied on future work.

MPSoCSim allowing the evaluation on homogeneous architectures as well, the same experimentations have been evaluated on a 2×2 cluster homogeneous architecture encom-

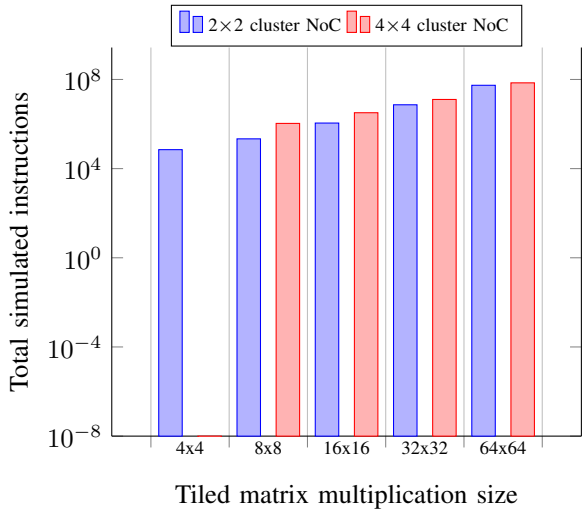


Fig. 7. **Total simulated instructions** for different sizes of matrix multiplications on a 2×2 and 4×4 cluster architectures (12 MicroBlazes and 1 ARM, and 60 MicroBlazes and 1 ARM respectively)

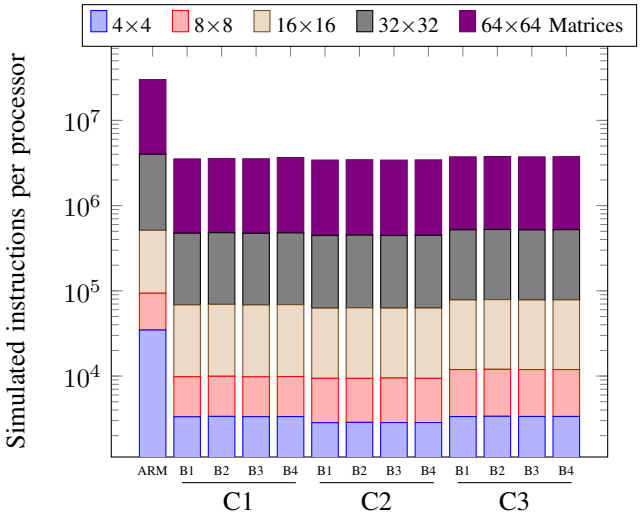


Fig. 8. **Simulated instructions per processor** for different sizes of matrix multiplications on a 2×2 architecture (12 MicroBlazes and 1 ARM)

passing ARM processors only. Table II compares the results on both architecures. As the frequency of the ARM processor is higher that the Microblaze one, the exectution time is lower.

VI. Conclusion

In this paper, an extension of the OVP-based simulator MPSoCSim is presented. MPSoCSim has been evaluated and validated through the comparison with the HW implementation. In its extended version, the simulator allows the modeling and evaluation of clustered NoC based multi and many-core architectures with distributed memory between clusters and shared memory within a cluster. Each cluster comprises up to 4 processors that might be heterogeneous (ARM, MicroBlaze, MIPS,...). The simulator is flexible and scalable and can encompass different number of clusters, each comprising

Matrix multiplication size	Het. architecture	Hom. architecture
4×4 Mat Mult	0.04	0.001
8×8 Mat Mult	0.111	0.0033
16×16 Mat Mult	0.56	0.138
32×32 Mat Mult	3.7	0.6093
64×64 Mat Mult	27	2.9

TABLE II
Execution time (in sec.) on a heterogeneous and a homogeneous architecure

different number of processors that might be heterogeneous. The simulator presents some features that can also be used in order to execute a dynamic scenario. Finally, results in terms of elapsed time in order to simulate such architectures, simulated execution time on the simulated system as well as the number of simulated instructions per processor are available. The simulation results on different architectures encompassing up to 60 MicroBlazes and 1 ARM, show that the simulator enables the flexible simulation of a great number of instructions in a very short time. Mechanisms enabling the control of the platform in a dynamic scenario, as well as features in order to load applications dynamically, will be further explored in future work.

References

[1] P. Wehner, J. Rettkowski, T. Kleinschmidt, D. Göhringer, "MPSoCSim: An extended OVP Simulator for Modeling and Evaluation of Network-on-Chip based heterogeneous MPSoCs", in Proc. of International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS), 2015

[2] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, The GEM5 Simulator", Computer Architecture News, vol. 39, no. 2, 2011

[3] T. F. Wenisch, R. E. Wunderlich, M. Ferdman, A. Ailamaki, B. Falsafi, J. C. Hoe, "SimFlex: Statistical Sampling of Computer System Simulation", IEEE Micro, vol. 26, no. 4, 2006

[4] T. Austin, E. Larson, D. Ernst, SimpleScalar: An Infrastructure for Computer System Modeling, Computer, vol. 35, no. 2, 2002

[5] L. Jain, B.M. Al-Hashimi, M.S. Gaur, V. Laxmi, A. Narayanan, NIRGAM: a simulator for NoC interconnect routing and application modeling, Workshop on Diagnostic Services in Network-on-Chips, Design, Automation and Test in Europe Conference (DATE), France, 2007, http://nirgam.ecs.soton.ac.uk

[6] F. Fazzino, M. Palesi, D. Patti, Noxim: Network-on-Chip Simulator, http://sourceforge.net/projects/noxim/

[7] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, J. Kim, W. J. Dally, A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator, in Proc. of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software, 2013, http://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/BookSim

[8] F. Rosa, L. Ost, T. Raupp, F. Moraes, R. Reis, Fast Energy Evaluation of Embedded Applications for Many core Systems, in Proc. of the 24th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS), 2014

[9] Open Virtual Platforms Imperas Software Limited, OVP & SystemC, http://www.ovpworld.org/technology_systemc

[10] Imperas Software Limited, Using OVP Models in SystemC TLM2.0 Platforms, http://www.ovpworld.org/documents/OVPsim_Using_OVP_Models_in_SystemC_TLM2.0_Platforms.pdf

[11] Xilinx Inc.,Zynq-7000 All Programmable SoC, http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/

[12] J. Rettkowski, D. Göhringer, RAR-NoC: A Reconfigurable and Adaptive Routable Network-on-Chip for FPGA-based Multiprocessor Systems, in Proc. of the 2014 International Conference on ReConFigurable Computing and FPGAs (ReConFig), 2014