Design Automation for Embedded Systems (2019) 23:41–55
https://doi.org/10.1007/s10617-018-09218-7

Trace-driven and processing time extensions for Noxim simulator

Ivan Luiz Pedroso Pires[1] · **Marco Antonio Zanata Alves**[1] · **Luiz Carlos Pessoa Albini**[1]

Received: 22 May 2018 / Accepted: 13 December 2018 / Published online: 5 January 2019

Abstract
Simulation is one of the main tools used to analyze and test new proposals in the Network-on-Chip field. Several simulators can be found in the literature, among them the Noxim simulator stands out. It is being used by many researchers due to the wireless support and open-source availability. An important issue at the simulation phase is the choice of workload, as it may affect testing the system and its features. The correct workload can lead to rapid and efficient system development, while the wrong one may compromise the entire system evaluation. To ensure a more realistic simulation, simulators usually relies on real workloads by using a trace-driven approach. Although Noxim provides a simple support for input traces, it is very limited to a general behavior of the system, accepting only a generic injection rate parameter over time. Another important part of the simulator is the ability to consider the Processing Elements processing time. We propose in this paper an extension of the Noxim simulator to address these issues. Consequently, results are more realistic and may be possible to predict the total execution time very accurately. This extension is demonstrated and evaluated using the NAS-NPB workload.

Keywords Network-on-Chip · Simulation · Processing time · Trace

1 Introduction

Network-on-Chip (NoC) [7] is the current interconnection paradigm to design all large-scale chips. It is scalable and can be adapted to several computational paradigms and applicable to

This work was partially supported by CNPq and CAPES.

✉ Luiz Carlos Pessoa Albini
albini@inf.ufpr.br

Ivan Luiz Pedroso Pires
ilppires@inf.ufpr.br

Marco Antonio Zanata Alves
mazalves@inf.ufpr.br

[1] Department of Informatics, Federal University of Paraná (UFPR), Curitiba, Brazil

various areas, like high performance multiprocessing computers [25]. NoC communication is performed through packets, divided into small information units called flits, transmitted from source to destination by routers, hubs and network interfaces over wired or wireless links [11].

When designers want to evaluate new interconnection architectures and organizations, simulation is one of the main tool used to analyze traffic and measure the performance, power and area of NoC. It is a suitable tool for testing and analyzing results from new concepts and ideas without hardware prototyping. To guarantee the accuracy of the results, the simulator must comply with the state-of-the-art proposals and technologies.

Nowadays, there are several specific simulators for NoC interconnections such as: Noxim [9]; Booksim [20]; Naxim [26]; Hornet [29]; Topaz [1]; HNOCS [6]; NoC for OMNeT++ [5]; WNoC Simulator [35]; Darsim [23]; Netrace [15]; Garnet [2]; gpNoCsim [16]; Nirgam [18]; NNSE [24]. Other simulators non-specific for NoC could be adapted in some way: Gem5 [8]; CACTI [37]; NS-3 [30]; etc. In this work, only the specific-NoC simulator with focus on performance and execution time that were published in scientific papers were considered.

To ensure a more realistic simulation, any simulator must provide ways to receive input from realistic executions, which can be made using a trace-driven approach to run real data in testing system. A trace is a log of events and usually includes: time, type, size and other important parameters associated with each event, varying depending on the type of simulation is being performed. Trace-driven simulations can be used by resource management algorithms; deadlock prevention; CPU scheduling; cache analysis; etc. Among all advantages, trace-driven simulation allows realistic workload very similar to the actual implementation. However, also has some disadvantages as the complexity and single point of validation [19].

Table 1 shows a brief summary of the NoC simulators and depicts their ability to handle traffic input tracing, to support wireless communication and the source code availability.

A first version of this article considering only the trace-drive extension was previously published in [28]. It extended the Noxim Simulator to accept external input traces based on

Table 1 Summary of NoC simulators

Simulator	Trace-driven traffic	Wireless communication	Source availability
Noxim'15 [9]		•	•
Booksim'13 [20]	•		•
Naxim'13 [26]			
Hornet'12 [29]	•		•
Topaz'12 [1]	•		•
HNOCS'12 [6]	•		•
NoC OMNeT++'11 [5]	•		•
WNoC Simulator'11 [35]		•	
Darsim'10 [23]	•		•
Netrace'10 [15]	•		•
Garnet'09 [2]	•		•
gpNoCsim'07 [16]			•
Nirgam'07 [18]	•		•
NNSE'05 [24]			

messages. This extension enabled Noxim to receive realistic and controlled traffic traces as input, providing ways to compare real workloads in different NoC scenarios.

In this paper we introduce the ability to consider the Processing Elements (PEs) processing time as well as the support to Message Passing Interface (MPI) communication on Noxim for the first time, making it a huge improvement on the results presented in [28]. Now, results are more realistic and may be possible to predict the total execution time very accurately considering the MPI primitives for synchronization barrier, blocking and non-blocking communication. It can be observed that none of the previous proposed NoC simulators provides support to all these features. The combination of all these features results in a powerful simulator tool for realistic workloads, including using message passing, to design new concepts for inter and intra-chip communication.

The rest of this paper is organized as follows: Sect. 2 contains an overview of Noxim Simulator and related proposals for Noxim extensions; Sect. 3 present the proposed modifications in the simulator to support external traces and processing time; Sect. 4 shows the methodology and discussion of the evaluation for this work; Sect. 5 brings the final considerations and future work ideas.

2 Noxim simulator

Noxim [9] is an open-source cycle-accurate simulator developed in C++ and integrated with System-C for heterogeneous wired and wireless NoC architectures which estimates performance and energy consumption. The simulator works with two main conceptual elements: tile nodes and communication infrastructure. Tile nodes are computational or storage nodes. The communication infrastructure consists of router(s) for each tile interconnected by wired links with their neighbors and possibly the wireless hub element. The wireless hub is wired connected with one or more tiles and wireless connected with other hubs. Therefore, the simulator offers three communication patterns: tile-to-tile, tile-to-hub and hub-to-hub. Communication is performed through packet split in small parts called flits.

The modeling of the NoC system is made in a human-readable configuration file in the YAML format. The configuration parameters are organized in three groups: the wired NoC configuration, the wireless setup and the simulation parameters. The wired NoC configuration sets the dimension of the mesh, internal buffers, routing table, routing algorithm and strategy for selection between multiple output directions. The wireless configuration sets the wireless hub and their buffers, the channels with its data rate, policy of medium control access and bit error rate. The simulation parameters set the clock period, time for reset, warm up time, flag to wireless usage, flag to debug mode, output trace mode, packet sizes, packet injection rate, probability of re-transmission and traffic distribution.

Even though simulate real applications on the original Noxim is possible, it is very hard and not accurate. It is achieved through mapping tasks communication graphs into customized table-based format, with mandatory fields such as packet injection rate and probability of re-transmission for each link. On the other hand, several critical parameters are not considered, for example, message sizes.

Noxim simulator was upgraded and enhanced by several authors. Originally, it supported only mesh topology, in [33], authors present an enhanced Noxim Simulator for performance evaluation of other NoC topologies. They implemented the mesh, torus and twisted torus topologies. These authors also proposed different routing algorithm to test the improvements.

In [21], the Noxim was used together with QEMU to form a hardware/software co-simulator for NoC. Each CPU core was simulated by QEMU and connected by a TCP connection with the Noxim simulator. The Noxim simulator interacted with QEMU as the processing elements were formed by CPU cores.

This Noxim extension is able to consider the PE processing time, support to MPI communication and accept external traces as simple as possible. Thus, it allows researches to use real communication traces providing more realistic results. In next section this add-ons are presented in details.

3 Noxim extensions

This section presents our proposal to extend the Noxim simulator to support processing time and external input traces.

3.1 Simulator modifications

Modifications focus on the YAML configuration file and inside the source code from the simulator. The YAML configuration file modifications are the following:
traffic_distribution in the original Noxim, the traffic distribution can be set to random, transpose, hot spot, table based, bit reversal, shuffle and butterfly. We extended this parameter to add the *traffic trace* option to support external input traces.
traffic_trace_filename this parameter informs the name of input trace without the Processing Element (PE) identification. For each PE in the system a respective trace file must exist even if empty (in the case the PE has zero packages). For instance, a system with 4 PEs must have the following traces *000_trace.txt*, *001_trace.txt*, *002_trace.txt* and *003_trace.txt*.
traffic_trace_flit_headtail_size this parameter informs the size of packages head and tail (summed) for the modeled network technology. This feature helps the simulator to estimate the data throughput considering the overhead of each communication technology.

To support the new parameters and the package generation described in the trace files, multiple source code files were modified. Following we point each source code file and a brief description of the modifications.
ConfigurationManager.cpp the *loadConfiguration()* method was altered to load the new configuration parameters of the traffic trace from YAML file, and these new parameters are validated in the *checkConfiguration()* method.
GlobalTrafficTrace.cpp new code file added to the simulation. It is responsible to receive the input traces and carry them into the simulator.
Hub.cpp altered to fix memory leakages (present on the current version of Noxim). The *antennaToTileProcess()* and the *tileToAntennaProcess()* methods contained memory allocation flaws which lead to system crash with huge trace files.
Noc.cpp a controller of input trace was added in the *buildMesh()* method, in such way the simulator recognizes when all the input traces are complete so it can stop the simulation.
ProcessingElement.cpp this file received most of our modifications. In the *rxProcess()* method the traffic trace controller implements the stop trace criteria, which permits to stop the simulation after all the traces from all PEs are sent. The traffic controller uses a queue to control the buffered and transmitted flits, and it is used by *rxProcess()* and *txProcess()* methods. In *canShot()* method, the loaded traces from each PE are executed, one by one, in the following way: (*a*) verify the control flow if the flit can be shot; (*b*) if the control flow allows the shot, a

new packet will be created, formed by source and destination addresses, timestamp, payload (message size plus head and tail size) and an eventual padding if the packet size is smaller than the minimum packet size. Considering that the trace file can contain millions of lines, we only generate the packages whenever the simulator can send it, reducing the memory overhead of our proposal.

Furthermore, we remove the Noxim limitation allowing it to run simulation bigger than 2147 milliseconds. In order to achieve this we had to increase statistical and control variables inside Noxim.

3.2 The input trace file format

In order to be as generic as possible, we choose to use a trace based on pure text format. The new input trace mode and the trace file name must be informed in the YAML configuration file. This file format is quite simple, each line should contain the name of the MPI primitive, the initial and finish timestamp of primitive, the destination PE and the message size in bytes. The message size considers only the useful payload without any protocol encapsulation and MPI overhead. Note that the input trace does not contain the communication source as each PE has its own trace file. The trace may have multiple lines to inform the full workload (multiple messages to be sent).

The input trace format must follows:

```
<MPI_Primitive> <Start_Time> <End_Time> <Destination> <Message_Size>
```

For example:

```
MPI_Alltoall 192605257 256628513 15 4
```

The example shows the input format as follows: the Message Passing Interface (MPI) primitive used is *MPI_Alltoall*, start time is 192605257 nanoseconds, finish time is 256628513 nanoseconds, PE 15 is the destination and the message size is 4 bytes.

The name of the MPI primitive indicates the kind of primitive it is simulating. In this extension of Noxim we implement the most used MPI primitives, presented in NPB benchmark, and a MPI synchronization barrier. The primitive type and its associated timestamp are essential to simulate the processing time of the PEs.

3.3 MPI

The following MPI primitives are supported in this version:

- MPI_Allgather
- MPI_Allgatherv
- MPI_Allreduce
- MPI_Alltoall
- MPI_Alltoallv
- MPI_Bcast
- MPI_Gather
- MPI_Gatherv
- MPI_Reduce
- MPI_Scatter
- MPI_Scateerv

– MPI_Send
– MPI_Isend

The MPI instrumentation manipulates the primitives registering the start time, the finish time, origin, destination and size of the message. Only the thread which invokes the MPI primitive uses the instrumentation. At the end of the primitive the registered information is saved.

Primitives must be manipulated according to its characteristics, all-to-all, all-to-one or one-to-all communication. All-to-all communication represents the following primitive: Allgather, Allgatherv, Allreduce, Alltoall and Alltoallv. All-to-one communication represents the MPI_Gather, MPI_Gatherv and MPI_Reduce primitives. One-to-all communication represents the MPI_Bcast, MPI_Scatter and MPI_Scatterv primitives.

All-to-one communications are implemented through a loop during the MPI communication delivering the message to all PEs. All-to-one communication does not need additional coding. As all MPI communications are thread separated, the thread itself registers the transmissions and saves the log in the file. One-to-all communications follows the same implementation used in the all-to-all one, except that the loop is restricted to sender PE.

The MPI_Send and MPI_Isend primitives are blocking ones. They indicate that the sender gets blocked until it receives an ACK from the destination. Our implementation considers that each PE has one incoming buffer, messages are maintained on it until sending the acknowledgement. The MPI_Send blocks the processing and all communications of the PE, while the MPI_Isend only blocks the communications. As ACKs can be piggybacked in other messages or followed through control dedicated links, we chose to implement it virtually, using a global simulator structure.

The MPI Barrier is used to synchronize the communications in a given moment. It was included in the trace file to indicate the barrier timestamp for each PE. Upon detecting the MPI_Barrier in the trace file, each PE raises a local flag indicating the barrier and increases a global counter. The counter is shared by all threads, and indicates the number of blocked PEs. When the counter indicates that all PEs are blocked all flags are reset and the PEs can proceed with the communications.

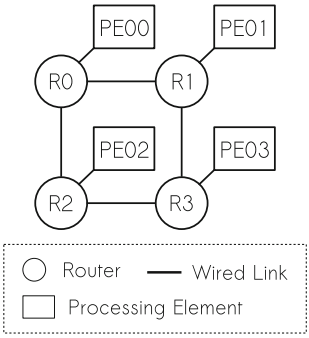
3.4 Usage example

In order to fully understand the extension in the Noxim, we present a usage example to follow step-by-step a synthetic trace simulation. In this example we model a NoC composed by a 2×2 mesh two-dimensional topology (i.e. four processing elements). Each PE is connected to one router, and the routers are interconnected to each other by a 32-bit wide wired link. Figure 1 illustrates this NoC topology. This topology is built declared inside the YAML configuration file, creating the PE, routers, channels and hubs. The filename containing the routing table is also defined and the table is loaded, here we consider that XY routing algorithm is defined.

Consider a trace file formed by 3 messages, one for each other PE present on the system (similar to a broadcast behavior). Inside the configuration file, the base filename *trace.txt* for the trace was also set for this example, enabling the PE-00 to load the trace file *00_trace.txt*. Each line of this trace is interpreted as a communication packet, formed by head, payload and tail.

Whenever the simulation starts, the simulator will behave the following way: PE-00 will load its trace file and retrieve the first packet information. The packet is prepared by adding the head, payload and tail, it may also have padding to attend the minimal packet size (for tiny payloads). Once the packet is ready to be sent, it is inserted inside the packet queue split

Fig. 1 NoC mesh example



in flits with 32 bits. Once the router becomes available, it will transmit the packet over the NoC.

After sending the packet, the PE-00 will not send a new packet until the previous one was received. This condition is part of what we call control flow in our proposal and depends on acknowledge signal from the receiver so the sender can follow the trace. We believe that this control creates a more realistic scenario, as it can be used to simulate the TCP behavior.

Upon receiving a packet, destination will check if the message refers to a blocking or non-blocking primitive. Blocking communication refers to messages which require a strict acknowledgement. While, non-blocking ones are messages which doe not required any acknowledgement.

Using a very similar mechanism, this extension also implements the MPI_Barrier to synchronize all PEs. Although several efforts to consider non-traditional MPI communication can be found in the literature, as streams for HPC application [27], this extension considers only the tradition one, that is, the send-receive and collectives approaches.

During the packet transmission, the flits follow the routing present in the routing table or the routing algorithm. Considering that we used an XY routing algorithm, it would mean that first the flits would flow in the horizontal direction and then on vertical direction until achieve the final destination. When modeling a wireless NoC where each router can be connected or not to a wireless hub, this routing algorithm would be valid between the source and the wireless hub, and from the wireless hub (from a different NoC) until the destination. Moreover, depending on the configuration parameters, the routers may have input and output buffers to store the incoming and outgoing flits.

Following the example, when the destination (PE-01) receives the packet coming from PE-00 internally the simulator unlock the PE-00 to perform the next packet transmission (if any left). Notice that any other PE (01, 02 and 03) can perform communications in parallel to the PE-00. Whenever all the PEs fully performs the transmission of their trace files, the simulator recognizes it and finishes the simulation.

3.5 Processing time

Another major advance in this version of Noxim is the simulation of the PEs processing time. This eliminates the send-and-wait primitives used in our previous version.

The processing time of each packet is calculated as the difference between the final timestamp of the packet and start timestamp of the next packet. In the following example the MPI_Send primitive start at 2625852823 nanoseconds and end at 2625853538 nanoseconds, while the MPI_Reduce start at 2704602479 and finish at 2704607248. This means that

between 2625852823 and 2704602479 nanoseconds there is a processing time of 78749656 nanoseconds.

Upon scheduling a packet for transmission, the simulator verifies if there are more packets to be sent. Then it calculates the processing time as previously mentioned. During this time interval, the PE is busy and no new transmission schedule is allowed.

```
MPI_Send    2625852823 2625853538 2 4
MPI_Reduce  2704602479 2704607248 0 4
```

However, including the processing time in the simulation might imply in huge simulation time for large workloads with high inter-message processing time. To overcome this issue, we include a new feature to Noxim, Step Forward. Step Forward consist in fast forward the simulation during PEs processing time to reduce the total simulation time cost. In each simulation step two conditions are verified, *i* PE is in processing time; *ii* there are no flits on the reception queue. If both conditions are satisfied the simulation is fast forwarded until the time in which the first PE finishes the processing time. The fast forward time is accounted in all scenarios.

4 Evaluation

This section brings a validation of the proposed extensions. In order to show the potential of our proposal, we created a Message Passing Interface (MPI) wrapper inside the MPE2-2.4.7 from MPICH implementation version 3.2[1]. MPICH is a high-performance and widely portable implementation of the Message Passing Interface (MPI) standard. During the application execution, this wrapper can trace information regarding message size, origin and destination split per thread basis. This wrapper writes in different trace files all the messages sent by the different MPI processes which used the MPI primitives (i.e. send, isend, bsend, bcast, gather, reduce, scatter, etc). Each timestamp was generated using the MPI_WTime with nanosecond precision according to the MPI_WTick.

Thus we are able to simulation this MPI application as it was executing in a system interconnected by different interconnections, such as 10 Gbps Ethernet [22], Wireless Gigabit [14], InfiniBand [17], Wireless Interconnection with Code Division Multiple Access [32] and Wireless Interconnection Token-based [31]. We modeled each of these interconnection standards as if they were the external link, which connects four NoC systems together. A very brief explanation for each interconnection technology used in our evaluations is present bellow:

Ethernet The Ethernet is a known wired connection widely used in local area networks. The 10 Gbps Ethernet standard differs from earlier Ethernet standards because it operates only over fiber and in full-duplex mode [22]. The MAC parameters were maintained unaltered, with the maximum and minimum frame size being 1518 B and 64 B, respectively. Considering the head and tail, the minimum and maximum packet size is 72 B and 1526 B, respectively.

Wireless Gigabit The new Wireless Gigabit (WiGig) standard [36] is a wireless connection that works at 60 GHz with four channels and each channel transmission rate varies between 7 Gb/s and 10 Gb/s. Although the specification of 60 GHz [36] reports that the information may have 8 B at minimum size, considering the Protocol Adaptation Layer for wireless bus extension [3], the packet payload field has variable size. The maximum packet contains the

[1] The file /mpe2-2.4.7/src/wrappers/src/trace_mpi_core.c was instrumented to trace all the communication messages sent by the MPI.

maximum payload size (128 B), plus overhead encapsulation (up to 16 B) and the Protocol Adaptation Layer header is 4 B. Thus, the maximal packet size of the is 148 Bytes.

InfiniBand The InfiniBand is an architecture which supports a range of applications on the backplane wired interconnection, such as the clustered hosts and I/O components. It works at full duplex transmission between any two fabrics elements. For the data rate, this work considers 50 Gb/s as predicted by authors for the year of 2017 [17]. The minimum payload in is 256 B and the maximum is 4096 B, and their head and tail packet has 126 B. In this way, the minimum packet size is 382 B and the maximum is 4222 B.

Wireless Network-on-Chip Wireless Interconnection with Code Division Multiple Access (WI-CDMA) and Wireless Interconnection Token-based (WI-Token) are designed for a seamless hybrid wired and wireless interconnection network for multi-chip systems. Nevertheless, [32] proposes the use of CDMA achieving only 6 Gb/s in the wireless link, [31] uses a token based collision avoidance method reaching 16 Gb/s. In both, the packet size is fixed in 256 B and the payload, head and tail size was not informed.

4.1 NAS-NPB workload

After creating the MPI wrapper proper to trace all the communication from MPI workloads. We used the NAS Parallel Benchmark (NPB) 3.3-MPI suite as workload to generate our traces. This benchmark suite contains MPI parallel applications that solves numerical methods for aerodynamic simulation problems. The following NPB applications run double-precision floating-point and were written in FORTRAN: Block Tri-diagonal (BT), Conjugate Gradient (CG), Fast Fourier Transform (FT), Lower and Upper triangular system (LU), Multigrid (MG) and Scalar Pentadiagonal (SP); while application Data Traffic (DT) and Integer Sort (IS) were written in C and uses mainly integer and logical operations. The more details for each application can be found in [34], the impact of NPB workload in NoC architectures are presented in [13] and their pattern communication in [10].

4.2 Metrics and parameters

The main metric used in this experimentation is the total execution time in nanoseconds. A secondary metric is the communication overhead in bytes. These metrics are the best ones to depict the contribution of this work.

The applications from NPB suite have different problem sizes (benchmark classes) named in ascending order as S, W, A, B, C and D [34]. Our evaluations used the problem size *A*, which is the most used in real machine tests, due to its medium size and reasonable execution time to be evaluated in a simulated environment. Each NPB application have a different communication pattern, already published in different papers [4,12].

The first test considers only the MPI support to obtain the communication performance with blocking and non-blocking primitives and barrier of synchronization. This test also varies the number of PEs in each system using four mesh size 2×2, 3×3, 4×4 and 5×5. Note that, mesh size was chosen accordingly to NPB applications which require square numbers of threads.

The second one aims to test the impact of processing time on systems performance. For this test, the MPI support was maintained and included the processing time. To reduce the number of simulated parameters, the number of threads was set to sixteen consisting of four systems with 2×2 mesh size each, with a total of 16 PEs executing all workload.

Table 2 Network parameters

Parameter	Ethernet	Wireless Gigabit	InfiniBand	WiNoc CDMA-based	WiNoC Token-based
Head and tail	26	4	126	–	–
Minimum payload	46	4	256	–	–
Maximum payload	1500	144	4096	–	–
Smallest packet	72	8	382	256	256
Largest packet	1526	148	4222	256	256
Data rate (Gb/s)	10	8	50	6	16

All units are expressed in bytes except Data Rate

The Table 2 shows the parameters for each network interconnection evaluated in this paper. These parameters are the input values to setup the YAML configuration file for Noxim simulator (extended version). The simulation of each system was performed with these different parameters to allow a network technology comparison on NoC architecture.

The parameters of Table 2 were chosen because of their impact on communication overhead. Head and tail are always included in each packet. If message size is smaller than the minimum payload padding must be included. If message size is larger than the maximum payload, message must be split in more than one packet with additional head and tail fields. Lastly, data rate directly affects the network performance.

4.3 Results and discussion

Table 3 shows the communication overhead for each interconnection network in order to process the messages from the NPB applications. This overhead were obtained as the difference between the payload to send and received bytes for each application. The received bytes were obtained as part of the simulation results. In this table, the columns WI-CDMA and WI-Token are exactly the same as these connections have the same minimum and maximum package size. It is possible to observe that WiGig has a tiny overhead compared with others for all NPB applications while the InfiniBand has the highest one on average due to the minimum packet size restriction.

Table 3 Overhead for NPB applications

Apps.	Total Payload (B)	Ethernet (%)	Wireless Gigabit (%)	InfiniBand (%)	Wi-CDMA (%)	Wi-Token (%)
BT	1.766.806.620	1.79	2.79	3.39	1.59	1.59
CG	256.366.712	2.58	2.84	7.60	1.64	1.64
FT	251.658.900	1.75	2.78	3.10	1.64	1.64
IS	360.334.860	1.80	2.78	3.69	1.59	1.59
LU	730.982.856	3.89	3.13	15.11	1.91	1.91
MG	103.673.604	3.19	2.87	12.01	1.69	1.69
SP	2.961.299.748	1.82	2.79	3.47	1.60	1.60

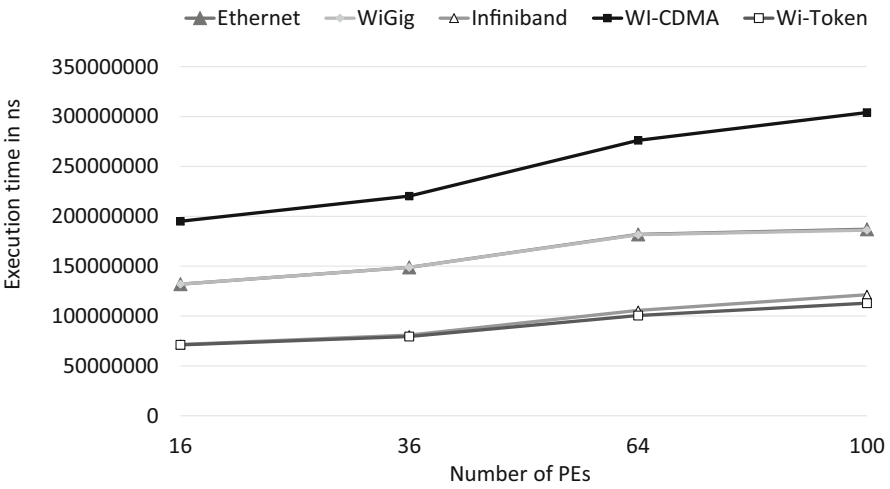


Fig. 2 Simulation of increasing the mesh size executing bt application

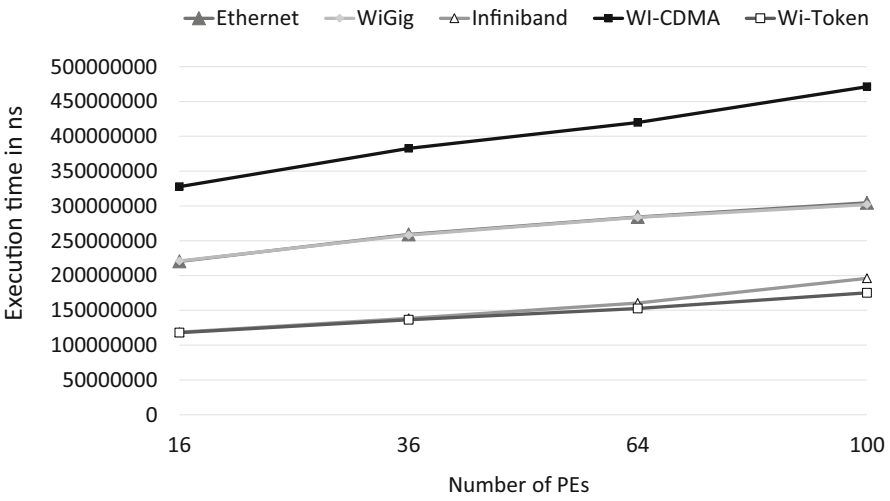


Fig. 3 Simulation of increasing the mesh size executing SP application

Figures 2, 3 and 4 show the performance while increasing number of PEs to execute BT, SP and LU application. It is possible notice that there is a linear growth of execution time. BT and SP applications have similar performance behavior, in both the Wi-Token and Infiniband have better results. Considering LU application it changes, as WiGig and Ethernet present better results. This is due to the traffic shape of the applications. Application with few larger messages have better performance on Infiniband. On the other hand, applications with a high number of small messages present better results with Ethernet and WiGig.

Figures 5 and 6 show the total execution time considering the processing time of the PEs and the different transmission of all messages through the different interconnection for the workload applications. The different workloads are separated in two Figures due to the scale difference. Results are expressed in nanoseconds.

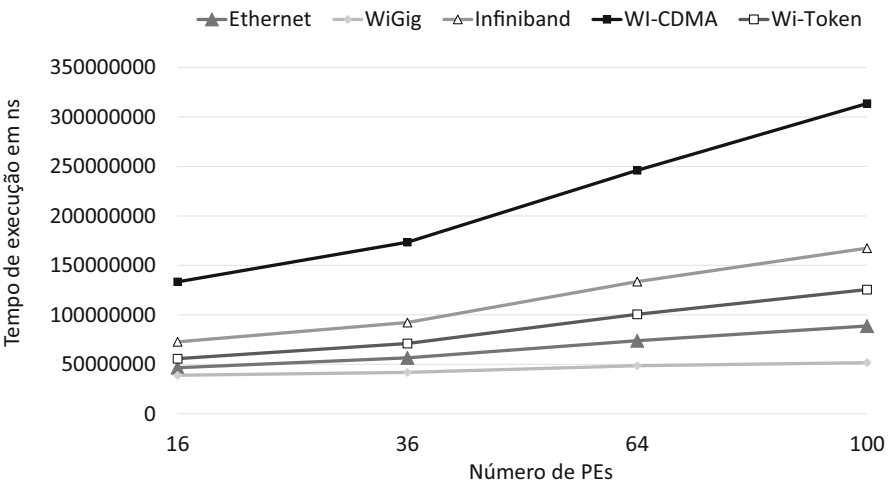


Fig. 4 Simulation of increasing the mesh size executing LU application

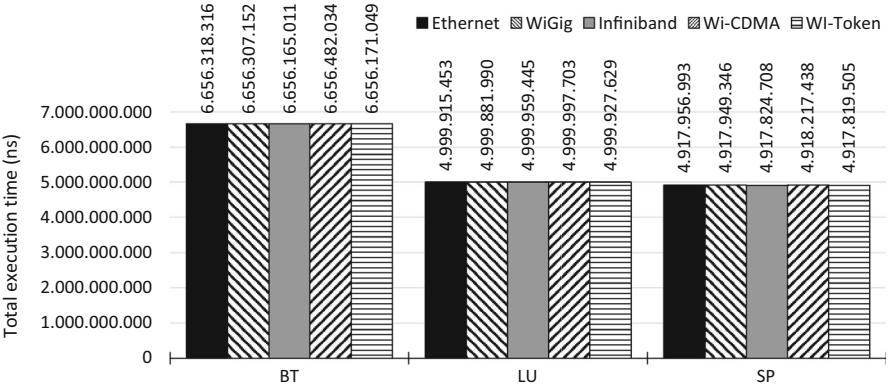


Fig. 5 Total simulation time for BT, LU and SP workloads

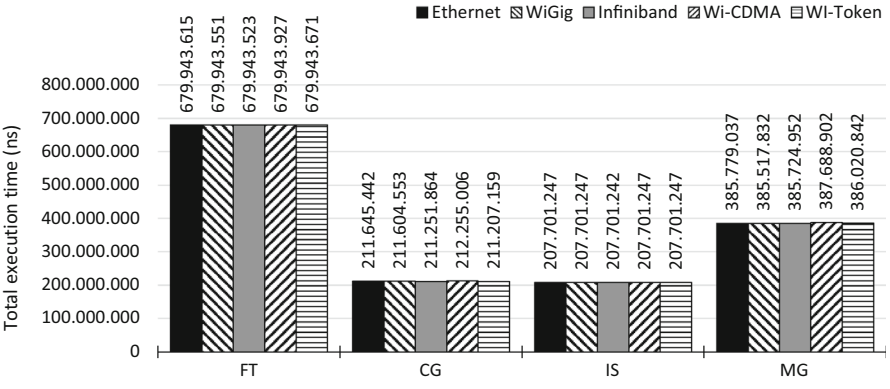


Fig. 6 Total simulation time for FT, CG, IS and MG workloads

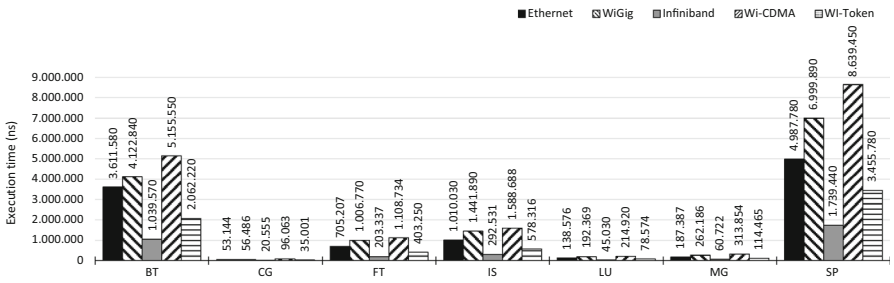


Fig. 7 Previous simulation results without processing time and MPI support

Figure 7 depicts the results without the extensions proposed in this version. It is possible to compare the actual results (Figs. 5 and 6) and the previous ones (Fig. 7) and see a clear accuracy difference. This is the processing time effect, which make the actual extensions much closer to the real world. Results are expressed in nanoseconds.

As can be seen in the presented results, the processing time of the workload is much higher than the communication time. This time was hidden in all Noxim simulations. Obviously, if the processing time is constant for the different scenarios, it is possible to simply ignore it and compare the other results. However, if the processing time is one of the key facts that must considered in the simulation, it is essential to consider it. With this version of the Noxim, we are allowing researchers to use the correct tool, with or without the processing time. Nevertheless, researches must have in mind that using the processing time may hide other results, as shown before. In any case, this work brings the simulation results closer to the real world, by allowing users to choose additional parameters on their simulations.

5 Conclusions and future work

Simulation is one of the main tool used to analyze new proposals in the Network-on-Chip field. Among all proposed simulators, the Noxim simulator stands out due to its accuracy near to a real system. However, traces of real applications were not easily supported by the trace table method (present on Noxim), which requires the researcher to inform the general behavior of the system (traffic injection rate) not even considering the packet size parameter. However, such method presents clear limitations to model more close to application behavior.

In this work, we extend the Noxim simulator to support MPI primitives and consider the PE processing time. We performed this extension by modifying the Noxim source code files. A list of the files and required modification was presented together with a validation of these changes.

Our validation was based on the simulation of messages from real application modelled over different interconnection systems. The performance results fits to the overhead estimates using analytical formulas that consider the minimum packet size, head and tail overheads. In this way, the presented Noxim extension will help researchers during performance tests of new proposals through real application traces. Future work includes the support of wireless broadcast, virtual channels for communication and improvement in the deadlock avoidance.

References

1. Abad P, Prieto P, Menezo LG, Colaso A, Puente V, Gregorio JA (2012) Topaz: an open-source interconnection network simulator for chip multiprocessors and supercomputers. In: 2012 IEEE/ACM sixth international symposium on networks-on-chip, pp 99–106. https://doi.org/10.1109/NOCS.2012.19
2. Agarwal N, Krishna T, Peh LS, Jha NK (2009) Garnet: A detailed on-chip network model inside a full-system simulator. In: 2009 IEEE international symposium on performance analysis of systems and software, pp 33–42. https://doi.org/10.1109/ISPASS.2009.4919636
3. Alliance W (2014) Wi-fi alliance wigig wireless bus extension technical specification. www.wi-fi.org. Accessed 10 Sept 2016
4. Bailey D, Harris T, Saphir W, van der Wijngaart R, Woo A, Yarrow M (1995) The NAS parallel benchmarks 2.0. Tech. rep., NAS Technical Report, NAS-95-020
5. Ben-Itzhak Y, Zahavi E, Cidon I, Kolodny A (2011) Nocs simulation framework for omnet++. In: Proceedings of the fifth ACM/IEEE international symposium, pp 265–266. https://doi.org/10.1145/1999946.1999993
6. Ben-Itzhak Y, Zahavi E, Cidon I, Kolodny A (2012) Hnocs: modular open-source simulator for heterogeneous nocs. In: 2012 international conference on embedded computer systems (SAMOS), pp 51–57. https://doi.org/10.1109/SAMOS.2012.6404157
7. Benini L, Micheli GD (2002) Networks on chips: a new soc paradigm. Computer 35(1):70–78. https://doi.org/10.1109/2.976921
8. Binkert N, Beckmann B, Black G, Reinhardt SK, Saidi A, Basu A, Hestness J, Hower DR, Krishna T, Sardashti S, Sen R, Sewell K, Shoaib M, Vaish N, Hill MD, Wood DA (2011) The gem5 simulator. SIGARCH Comput Archit N 39(2):1–7. https://doi.org/10.1145/2024716.2024718
9. Catania V, Mineo A, Monteleone S, Palesi M, Patti D (2015) Noxim: An open, extensible and cycle-accurate network on chip simulator. In: 2015 IEEE 26th international conference on application-specific systems, architectures and processors (ASAP), pp 162–163. https://doi.org/10.1109/ASAP.2015.7245728
10. da Cruz EHM, Alves MAZ, Carissimi A, Navaux POA, Ribeiro CP, Mehaut JF (2011) Using memory access traces to map threads and data on hierarchical multi-core platforms. In: 2011 IEEE international symposium on parallel and distributed processing workshops and Ph.d. Forum, pp 551–558. https://doi.org/10.1109/IPDPS.2011.197
11. Deb S, Ganguly A, Pande PP, Belzer B, Heo D (2012) Wireless noc as interconnection backbone for multicore chips: promises and challenges. IEEE J Emerg Sel Top Circuits Syst 2(2):228–239. https://doi.org/10.1109/JETCAS.2012.2193835
12. Diener M, Cruz EH, Pilla LL, Dupros F, Navaux PO (2015) Characterizing communication and page usage of parallel applications for thread and data mapping. Perform Eval 88–89:18–36. https://doi.org/10.1016/j.peva.2015.03.001
13. de Freitas HC, Schnorr LM, Alves MAZ, Navaux POA (2010) Impact of parallel workloads on noc architecture design. In: 2010 18th euromicro conference on parallel, distributed and network-based processing, pp 551–555. https://doi.org/10.1109/PDP.2010.53
14. Hansen CJ (2011) WiGiG: multi-gigabit wireless communications in the 60 GHz band. IEEE Wirel Commun 18(6):6–7. https://doi.org/10.1109/MWC.2011.6108325
15. Hestness J, Grot B, Keckler SW (2010) Netrace: Dependency-driven trace-based network-on-chip simulation. In: Proceedings of the third international workshop on network on chip architectures, NoCArc '10, ACM, New York, NY, pp 31–36. https://doi.org/10.1145/1921249.1921258
16. Hossain H, Ahmed M, Al-Nayeem A, Islam TZ, Akbar MM (2007) Gnocsim–a general purpose simulator for network-on-chip. In: International conference on information and communication technology
17. InfiniBand Trade Association and others: InfiniBand Architecture Specification, release 1.0 (2000). www.infinibandta.org. Accessed 23 Oct 2016
18. Jain Lavina, Al-Hashimi BM, Gaur MS, Laxmi V, Narayanan A (2007) NIRGAM: a simulator for NoC interconnect routing and application modeling. In: Design, automation and test in Europe conference
19. Jain Raj (1990) The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling. Wiley, London
20. Jiang N, Balfour J, Becker DU, Towles B, Dally WJ, Michelogiannakis G, Kim J (2013) A detailed and flexible cycle-accurate network-on-chip simulator. In: 2013 IEEE international symposium on performance analysis of systems and software (ISPASS), pp 86–96. https://doi.org/10.1109/ISPASS.2013.6557149
21. Kurimoto Y, Fukutsuka Y, Taniguchi I, Tomiyama H (2013) A hardware/software cosimulator for network-on-chip. In: 2013 international SoC design conference (ISOCC), pp 172–175. https://doi.org/10.1109/ISOCC.2013.6863964
22. LAN/MAN Standards Committee: IEEE Standard for Ethernet. IEEE Std 802.3-2015 (2016)

23. Lis M, Shim KS, Cho MH, Ren P, Khan O, Devadas S (2010) DARSIM: a parallel cycle-level NoC simulator. In: Workshop on modeling, benchmarking and simulation
24. Lu Z, Thid R, Millberg M, Jantsch A (2005) NNSE: nostrum network-on-chip simulation environment. In: Swedish system-on-chip conference
25. Micheli GD, Benini L (2017) Networks on chips: 15 years later. Computer 50(5):10–11. https://doi.org/10.1109/MC.2017.140
26. Nakajima K, Kurebayashi S, Fukutsuka Y, Hieda T, Taniguchi I, Tomiyama H, Takada H (2013) Naxim: a fast and retargetable network-on-chip simulator with qemu and systemc. Int J Netw Comput 3(2):217–227. https://doi.org/10.15803/ijnc.3.2_217
27. Peng IB, Markidis S, Gioiosa R, Kestor G, Laure E (2017) Mpi streams for hpc applications. N Front High Perform Comput Big Data 30:75
28. Pires ILP, Alves MAZ, Albini LCP (2017) Trace-driven extension for noxim simulator. In: 2017 VII Brazilian symposium on computing systems engineering (SBESC), pp 102–108. https://doi.org/10.1109/SBESC.2017.20
29. Ren P, Lis M, Cho MH, Shim KS, Fletcher CW, Khan O, Zheng N, Devadas S (2012) Hornet: a cycle-level multicore simulator. IEEE Trans Comput-Aided Des Integr Circuit Syst 31(6):890–903. https://doi.org/10.1109/TCAD.2012.2184760
30. Riley George F, Henderson Thomas R (2010) The ns-3 network simulator. Springer, Berlin
31. Shamim MS, Mansoor N, Narde RS, Kothandapani V, Ganguly A, Venkataraman J (2017) A wireless interconnection framework for seamless inter and intra-chip communication in multichip systems. IEEE Trans Comput 66(3):389–402. https://doi.org/10.1109/TC.2016.2605093
32. Shamim MS, Muralidharan J, Ganguly A (2015) An interconnection architecture for seamless inter and intra-chip communication using wireless links. In: Proceedings of the 9th international symposium on networks-on-chip, NOCS '15, ACM, New York, NY, USA, pp 2:1–2:8. https://doi.org/10.1145/2786572.2786581
33. Swaminathan K, Thakyal D, Nambiar SG, Lakshminarayanan G, Ko SB (2014) Enhanced noxim simulator for performance evaluation of network on chip topologies. In: 2014 recent advances in engineering and computational sciences (RAECS), pp 1–5. https://doi.org/10.1109/RAECS.2014.6799570
34. VanderWijngaart RF, Haopiang J (2003) NAS Parallel benchmarks, multi-zone versions, NAS Technical Report
35. Wang C, Hu WH, Bagherzadeh N (2011) A wireless network-on-chip design for multicore platforms. In: 2011 19th international euromicro conference on parallel, distributed and network-based processing, pp 409–416. https://doi.org/10.1109/PDP.2011.37
36. WiFi Alliance: 60 GHz Technical Specification v1.0 (2017). www.wi-fi.org. Accessed 10 Sept 2016
37. Wilton SJE, Jouppi NP (1996) Cacti: an enhanced cache access and cycle time model. IEEE J Solid-State Circuits 31(5):677–688. https://doi.org/10.1109/4.509850