Optimizing Application Mapping Algorithms for NoCs through a Unified Framework

Ciprian Radu, Lucian Vinţan
Advanced Computer Architecture & Processing Systems Research Lab - http://acaps.ulbsibiu.ro/research.php
"Lucian Blaga" University of Sibiu,
Romania
ciprian.radu@ulbsibiu.ro, lucian.vintan@ulbsibiu.ro

***Abstract*—This paper presents a preliminary PhD research towards developing a framework to evaluate and optimize application mapping algorithms for Network-on-Chip architectures. Several such algorithms have been proposed for mapping the threads of a parallel application on a NoC architecture. However, the performance of those algorithms is evaluated only on some specific NoC designs. A unified approach for evaluating such algorithms allows a better comparison of their performance and can potentially lead to some optimizations. The proposed framework is intended to be flexible so that the algorithms can be tested on different NoC designs. To this end, a scalable and flexible Network-on-Chip simulator is proposed. Some preliminary results obtained with this simulator are presented, too. They show the flexibility of this simulator and that it is feasible for addressing the application mapping problem in a unified manner.**

***Index Terms*—Network-on-Chip (NoC), parallel application NoC mapping, evaluation, optimization, software framework, simulation.**

I. Introduction

The application mapping problem for Networks-on-Chip (NoCs) was introduced in [1], where it was formulated as the *topological placement of the IPs onto the on-chip tiles*. This is an NP-hard problem because the search space increases factorially with the system size. For example, a NoC with 8x8 tiles theoretically allows 64! mappings. Obviously, before mapping each IP to a NoC tile, the application must be divided into a graph of threads (Communication Task Graph - CTG) and then, the threads have to be assigned and scheduled to the available IPs.

Several algorithms [1], [2], [3], [4], [5] were proposed for addressing the application mapping problem for NoC architectures. The purpose of such an algorithm is to determine the best topological placement of cores onto network nodes. The optimality of a certain mapping is given by the best trade-off between some network performance metrics like bandwidth, latency, energy consumption etc. A CTG provides the input for such an algorithm because it contains the communication patterns between the application's threads. Based on those patterns, the application mapping algorithm has to search for the best possible mapping such that the performance of the network is optimized.

The application mapping problem is tightly connected to the routing problem [2]. While a good mapping of cores onto network nodes can lead to energy savings, the routes used by the cores to communicate can have a great impact on the NoC's performance. The "best" topological placement of the cores onto nodes is not enough to account for the network's performance. When the algorithm performs the optimization of the application mapping, the load of the network nodes and the communication between them must also be considered. An application mapping algorithm can find better solutions if it takes into account the network routes as well. For example, the network congestion can be decreased by performing multi-path routing instead of single-path routing [3].

The existing application mapping algorithms are evaluated on some specific Network-on-Chip architectures. For example, frequently only 2D mesh NoC topologies are considered. Also, not all comparisons are reported on the same NoC design. Therefore, evaluations of such algorithms, from different researches, cannot be directly compared because a common evaluation methodology is still missing.

This paper proposes a unified approach for the application mapping algorithms evaluation. A common framework can also lead to optimizations of such algorithms. The envisioned framework will also be flexible. This will allow the study of a certain application mapping algorithm on a larger number of NoC designs. More precisely, an evaluation of those algorithms on different scalable NoC topologies is desired. In order to achieve these aims, a scalable and flexible NoC simulator is proposed.

Since the relevance of application mapping algorithms is still mainly researched on a small class of NoCs (e.g.: 2D meshes), this unified framework will contribute to determine the most suitable mapping algorithm, for a certain applications set and NoC design. Due to the huge design space of NoCs, an application mapping algorithm can potentially be optimized for a specific NoC architecture.

The rest of this paper is organized as follows. Section II briefly presents some related work. The next section describes the design of the framework. Section IV presents some preliminary results. The last section summarizes the contributions of this paper and outlines some directions for future work.

II. Related Work

Network simulation is proposed for evaluating different kind of parallel application mappings in [6]. A network simulator

based on SystemC [7] is used to determine the NoC's performance when different application mappings algorithms were investigated. The simulator uses a 2D mesh network. Besides the fact that this simulator only works with 2D mesh topology, it must also be noted that no actual mapping algorithm is used to determine the placement of IP cores onto network nodes. Rather than this, four communication patterns are used in order to mimic the network traffic generated by application mappings. Unfortunately this methodology is weakly related to the application mapping.

The application-based selection of a NoC topology is addressed in [8]. A general mapping algorithm extends NMAP [3] so that it can be applied on other topologies too, not just on a 2D mesh. Thus, topologies like torus, hypercube, 3-stage clos and butterfly are also considered. A tool called SUNMAP is designed with the purpose of automatically selecting the best NoC topology for a given application. The general mapping algorithm is used to produce a mapping of cores onto the researched topologies. The tool uses multiple routing protocols: dimension ordered routing, minimum-path and traffic splitting. The best topology is selected based on floorplanning information. Also, the following objectives are considered: the minimization of the average packet latency, by satisfying bandwidth constraints, and the minimization of power consumption, by satisfying area constraints. SUNMAP is a complex tool for automatically evaluating different topologies for Networks-on-Chip, in an application-aware context. However, only a single application mapping algorithm is considered. Taking into account other mapping algorithms too, should provide a more comprehensive view on the performance of different NoC architectures.

The work done in [6], [8] represents a starting point for building a common NoC simulation framework for evaluating different application mapping algorithms. Like in [6], we propose using a Network-on-Chip simulator for the evaluation of application mapping algorithms. As it is done in [8], the simulator allows using different NoC topologies. In contrast with the frameworks proposed in [6], [8], our framework will contain a NoC simulator that uses communication patterns from applications, and that is also scalable and more flexible. Additionally, multiple application mapping algorithms will be implemented. This will allow a useful unified approach for their evaluation and optimization.

III. The Framework Design

This section describes the framework architecture, which allows a unified evaluation/optimization of the NoCs application mapping algorithms.

The proposed framework is composed of three major components:

- A module that contains the implementation of different application mapping algorithms;
- A network traffic generator;
- A Network-on-Chip simulator.

An application mapping algorithm determines the topological placement of the IP cores onto NoC nodes. The representation of this mapping process will then have to be translated into communication messages between the network's nodes. This involves generating the application's traffic pattern. The network traffic generator has the responsibility of injecting messages into the network such that the communication between the application's threads is emulated. A Network-on-Chip simulator models the behavior of the NoC based on the generated traffic.

The purpose is to evaluate the global performance of mapping algorithms on different Network-on-Chip architectures. For example, it can be evaluated how a mapping algorithm behaves when the network topology changes.

The following subsections provide details regarding the design of the framework.

A. Obtaining the communication patterns from applications

Application benchmarks are accurate but they do not scale well with system size. In [9] synthetic benchmarks are considered the suitable kind of benchmarks for NoCs. The main reason is that synthetic benchmarks scale well with the system size while still keeping the properties of some particular fixed size application benchmarks. A synthetic benchmark represents an abstraction for a task graph with known computation times and communication loads. Thus, such a benchmark does not actually contain application code but rather it tries to capture the communicational behavior of the application.

A network simulator is not capable of executing binary code because it is communication oriented. A complex simulator, which simulates IP cores that execute real parallel applications and communicate over an interconnection network, would be better suited. However, while currently there are scalable network simulators, multicore simulators do not provide the same scalability (they can usually run no more than tens of cores).

In [10] it is shown why it is difficult to capture the communication patterns of real applications. The authors propose a *reactive* network traffic generator. The traffic generator has to be reactive because the network latency can vary from one NoC architecture to another. If only the timestamps of the communication would get collected, whenever a message is delayed (due to network congestion for example), this delay should propagate to subsequent messages as well.

Usually, an application mapping algorithm needs to have the application described by a Communication Task Graph (CTG) [1]. The CTG keeps the data regarding the communication that occurs between the network nodes. To simplify the implementation model, it is not considered what kind of data is communicated, but rather the volume of communication. This information can be used to determine the amount of traffic which must be generated between any two network nodes.

CTGs can be automatically generated with the TGFF tool [11]. The embedded systems synthesis benchmarks suite [12], based on the EEMBC [13], contains task graphs for five application suites: automotive/industrial, consumer, networking, office automation and telecommunications. There is a version

of each task graph for three kinds of systems: distributed, wireless client-server and system-on-chip.

The application mapping algorithm uses the information provided by the CTG to determine the best mapping: communication volume, execution time, power consumption. Mapping constraints like bandwidth can be specified manually.

B. Modeling the Processing Element (PE) as a Finite State Machine (FSM)

In [9] it is considered that the communication model is enough for the communication architecture's evaluation. The details of the computations performed are not necessary and thus, Finite State Machines that emulate the communication between the tasks of the real applications are proposed.

Each Processing Element (PE) from the NoC is modeled as a Finite State Machine (FSM). The FSM is generated automatically from the CTG and mapping, and it contains the following information:

- **Task list**: what tasks are mapped to the PE (this says to what PEs data can be sent and from what PEs data can be received);
- **Control information**: the data dependencies (e.g.: a communication with a certain PE will be initiated only after enough data was received from another PE);
- **Processing time (P)**: how much time a PE requires for executing a task. When enough data was received, the PE will execute P operations (i.e. will wait for a certain amount of time) before generating a response;
- **Transaction data amount (D)**: the size of the communication, which will be generated after processing.

The following figure describes the FSM associated to a PE.

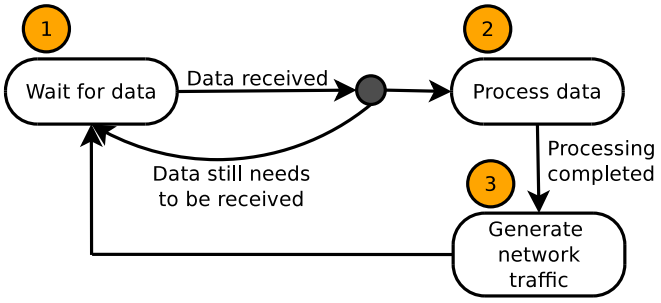

Fig. 1. The FSM associated to a PE

The PE that contains the root task of the CTG starts injecting packets into the network with a frequency specified by the CTG. While the PE with the root task is initially in the processing state (2) of the FSM, the rest of the PEs are in the waiting state (1). The PEs enter in the processing state after all required data are received. The duration of the processing is given by the task type and by the processing element type, too. After the processing is done, any PE enters in state 3. In this state, the generated data is sent to the required PEs. After all the generated data has been sent, the PE reenters in the waiting state.

The presented framework models the Processing Elements from the NoC as Finite State Machines, automatically generated based on the information from the CTG.

C. The ns-3 NoC simulator

A flexible Network-on-Chip simulator, called *ns-3 NoC*, was developed for the framework presented in this paper. It is based on the ns-3 network simulator [14], which is an event-driven simulator for Internet systems. ns-3 NoC provides a framework for researching different NoC designs. This simulator aims to provide a good trade-off between simulation accuracy and speedup, by making use of ns-3, which is one of the fastest and most memory efficient network simulators currently available [15]. Scalability is another important characteristic of ns-3, which is inherited by ns-3 NoC. This simulator allows the user to customize the packet size, packet injection rate, buffer size, network size, switching mechanism, routing protocol, network topology and traffic patterns. It can evaluate the NoC architecture in terms of network latency and throughput.

1) The ns-3 NoC architecture: The design of the ns-3 NoC simulator is derived from the design of ns-3. There is a set of fundamental components (objects) which are used by ns-3:

- **Node**: the representation of a network entity such as a personal computer, a router, etc. A Node can aggregate other components as protocol stacks and therefore, it has the capability to process packets;
- **Application**: a packet generator and consumer which can run on a Node and interact with a set of network stacks;
- **Topology**: represents the network's topology and it is composed of two elements:
  - **NetDevice**: the Network Interface (the link between a Node and a Channel);
  - **Channel**: the medium used to communicate and interconnect NetDevice objects.

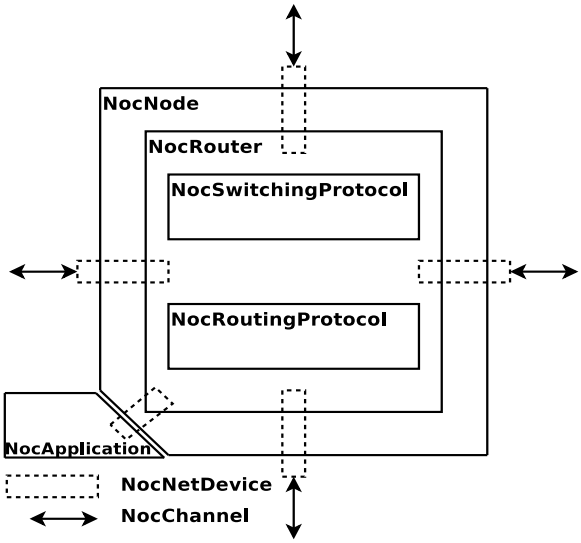

Fig. 2. The basic architecture of ns-3 NoC

The ns-3 NoC simulator was designed by implementing these components. The basic architecture of the ns-3 NoC

simulator is illustrated in Figure 2, which presents the major components of a network node.

The components of the ns-3 NoC simulator are the NoC application, node, router, net device, and channel. They are presented during the next paragraphs.

a) NoC application: The **NocApplication** models an ns-3 application. It represents the Processing Element of a Network on Chip, being responsible with injecting packets into the network and with receiving packets from the network. Packets are injected into the network with a frequency specified by the parameter called *data rate*. The data rate is expressed in bits per second, and, based on the user configurable size of packets, the *packet injection frequency* is determined. An ns-3 application runs for a given amount of time. Therefore, packets are injected into the network, until the running time of the application ends. Additionally, the user may specify a maximum amount of bits that an application can inject into the network. This kind of ns-3 application allows for packets to be injected into the network asynchronously. However, the simulator also models a synchronous ns-3 application (like in [16]). The synchronous ns-3 application allows packets to be injected into the network with a certain *injection probability*. This application can inject one packet per clock cycle. Each network router can route one packet per network cycle. Packets are injected for a certain number of cycles (specified by the user).

The NocApplication uses the following traffic patterns for determining the destination node of the injected packets: bit-complement, bit-reverse, matrix-transpose, uniform random. They are representing a set of communication patterns which consider the permutations that are usually performed in parallel numerical algorithms [17].

b) NoC node: The **NocNode** models the network node. It allows ns-3 applications to connect to it, and it also aggregates a router.

c) NoC router: The **NocRouter** has the responsibility to route the packets through the network. It uses a switching mechanism for deciding when packets are forwarded, and a routing protocol which determines the next network node where the packet will go.

The **NocSwitchingProtocol** represents a common specification for all the switching techniques.

NocRoutingProtocol is an interface for all of the simulator's routing protocols. An abstraction called **NocRouter** specifies a generic router for the simulator. This abstraction allows any concrete implementation of a router to make use of the network's load. However, this is not mandatory: a router is able to work with or without network load. The **LoadRouterComponent** is an interface that requires its implementations to specify how a router is supposed to compute its local load, and also what load value must be propagated to the neighboring node (from a certain direction). Note that the load values' computation is the responsibility of the router, whereas a routing protocol that uses load information only uses the load values to perform route evaluations. The advantage of such decoupling is that any routing protocol can make use of the load information, as it is computed by a certain load router component.

d) NoC net device: A **NocNetDevice** represents a network interface. It connects a network node to a network channel, and it is responsible with sending and receiving packets to and from adjacent nodes. The router has direct access to the net devices. Based on what routing path is established, a certain net device will be chosen for sending a packet. The packet will travel through the channel associated to the selected net device and it will arrive at the neighboring node via the corresponding net device. Each net device has one input queue, which allows for *input channel buffering*. Each net device continuously monitors its input queue and as long as packets are available, it requires the router to provide a route for the packet from the head of the queue. The switching mechanism will first have to decide when the packet can be routed. After the packet is ready to be sent, if the respective channel is available, the transmission will begin.

e) NoC channel: The **NocChannel** implements a bidirectional communication channel used for the network. It is characterized by two parameters: data rate and delay. The *data rate* is practically the bandwidth of the channel. The *delay* parameter can be used to specify how much time takes for a data transmission through the channel.

2) Network topologies: The ns-3 NoC simulator currently works with a 2D mesh topology. Additionally, the network topology from the Irvine architecture [18] is implemented as well. This topology is a variation of a 2D mesh topology. It has two channels for each two nodes vertically interconnected.

3) Network routers and routing protocols: For the Irvine topology the router consists of: an internal router, a right router and a left router. The internal router is only used as an interface to the processing element. The right router is capable of routing packets horizontally only from West to East, while the left router can route horizontally from East to West. For the North and South directions, both left and right routers have their own channels. The Irvine router uses a Dimension Order Routing protocol. This routing protocol is also used for a generic router implemented for the 2D mesh topology. Both XY and YX routing mechanisms are available. Additionally, the routing protocols Static-Load-Bound (SLB) and Self-Optimized (SO) [16] were implemented as well. These two routing protocols use information about the network load, being therefore adaptive.

4) Switching mechanisms: The ns-3 NoC simulator currently works with the packet-switching policy. This is more common than circuit-switching is, as it is used in about 80% of the current NoC research [19]. In order to have a flexible NoC simulator, three of the most used switching techniques were implemented: Store-and-Forward, Virtual Cut-Through and Wormhole.

5) The packet format: A packet contains one head flit. The head flit keeps the header of the packet. The payload data is kept in data flits. The size of the packet is parameterizable.

The format of the packet header is based on the one used in [18]. Compared to the header format from [18], the header

used by ns-3 NoC for 2D mesh topologies uses 32 bits to encode the X and Y coordinates of the network nodes. This allows for a scalable 2D mesh network topology. The fields *subdataId* and *peGroupAddress* are not currently used but, they were kept for further developments. The field *load* was added to the header. It occupies 8 bits, and it is used at propagating the information about the network load (the SLB and SO routing protocols are using this field).

IV. Preliminary Results

This section presents some preliminary results obtained with the developed ns-3 NoC simulator. The purpose is to illustrate the flexibility of the implemented simulator, which is important for the unified framework that evaluates application mapping algorithms.

The first simulations evaluate the Irvine architecture. As already mentioned, this architecture uses a variation of the 2D mesh topology. Supporting different topologies is a goal for the framework. The main characteristic of the Irvine architecture is that compared to head flits, a faster clock can be used to advance data flits. A data flit follows the route determined for the head flit of the packet it belongs to. The conventional wormhole switching mechanism uses the same clock to advance head and data flits. The advantage of the Irvine architecture is given by a reduction of the average latency of a packet. The results from Figures 3 show a significant decrease of the average packet latency as data flits are advanced through the network using a clock frequency which is two or four times higher than the clock frequency used for advancing the head flits. Compared with the original wormhole switching approach (i.e. the data flits advance at the same speed like head flits do), when the data flits advance twice or four times faster, the average packet latency is significantly lower. The simulations were run on multiple traffic patterns, for 10000 clock cycles, with 1000 warm-up cycles. At each cycle, a flit can be injected in any node of the network, with a certain injection probability. XY routing with wormhole switching has been used. The packets contain 9 flits, and the size of the input channel buffers was 8 flits. The size of the network was set to 8x8. With the matrix-transpose traffic pattern and using a 4 times higher clock frequency for the data flits, the packet's average latency remains close to the zero-load latency, as long as the injection probability is lower or equal than 0.9. The Irvine architecture helps at decreasing the network congestion. This is also visible for the other three traffic patterns. The network is significantly less congested when data flits are transmitted faster than head flits. For the bit-complement traffic pattern, the average packet latency is fairly higher because each node injects packets. This is not true with the other traffic patterns because they can create traffic from a certain node to exactly the same node, which is not injected into the network. Therefore, we believe that this behavior might contribute to the bit-complement's higher packet latency.

The second simulation results presented in this paper show how the ns-3 NoC simulator can be used at evaluating the impact of the available buffering resources. As shown in Figure 4, the more buffering resources are available, the better the performance of the NoC architecture gets. As expected, the size of the input channel buffers becomes more important as the number of packets injected into the network increases. The simulations were run using the uniform random traffic pattern, for 10000 clock cycles, with 1000 warm-up cycles. At each cycle, a flit can be injected in any node of the network, with a certain probability of injection. XY routing with wormhole switching has been used on a 4x4 Irvine NoC architecture. No speedup has been used for the data flits. Each packet has 9 flits, and the size of the input channels was varied uniformly, from 2 up to 8 flits.

These preliminary results show how differently the performance of the network varies based on the traffic pattern used and some NoC parameters, too (like the amount of buffering resources, faster data clock, etc.). The performance of a NoC architecture is directly influenced by the application mapping algorithm. By comparing the results of different mapping algorithms, it can be determined which algorithm is most suited for mapping a specific application on a certain NoC design.

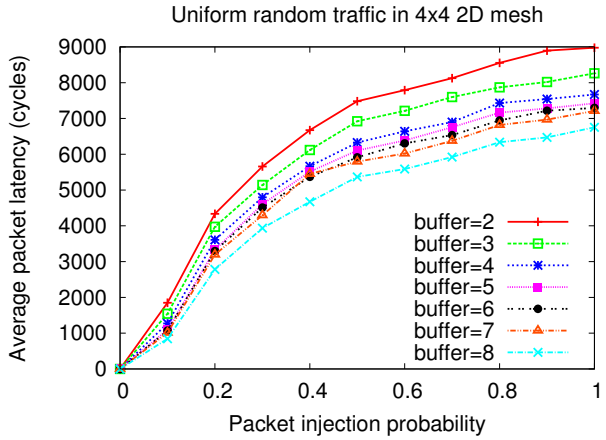

Fig. 4. The average packet latency on a 4x4 Irvine NoC architecture, while the size of the input buffers varies uniformly

V. Conclusions and Further Work

This paper presents an initial PhD research towards developing a unified framework for the evaluation and optimization of application mapping algorithms for Network-on-Chip architectures. The major problems that must be addressed are outlined. A scalable and flexible ns-3 NoC simulator was developed as the backbone of the framework. Some preliminary simulation results show the current capabilities of this simulator.

Future work involves the implementation of several known application mapping algorithms. Also a network traffic generator, capable of injecting packets into the network based on Communication Task Graphs, will be developed. The ns-3 NoC simulator will also be extended so that it will allow other network topologies too, not just 2D mesh and variations of this one.

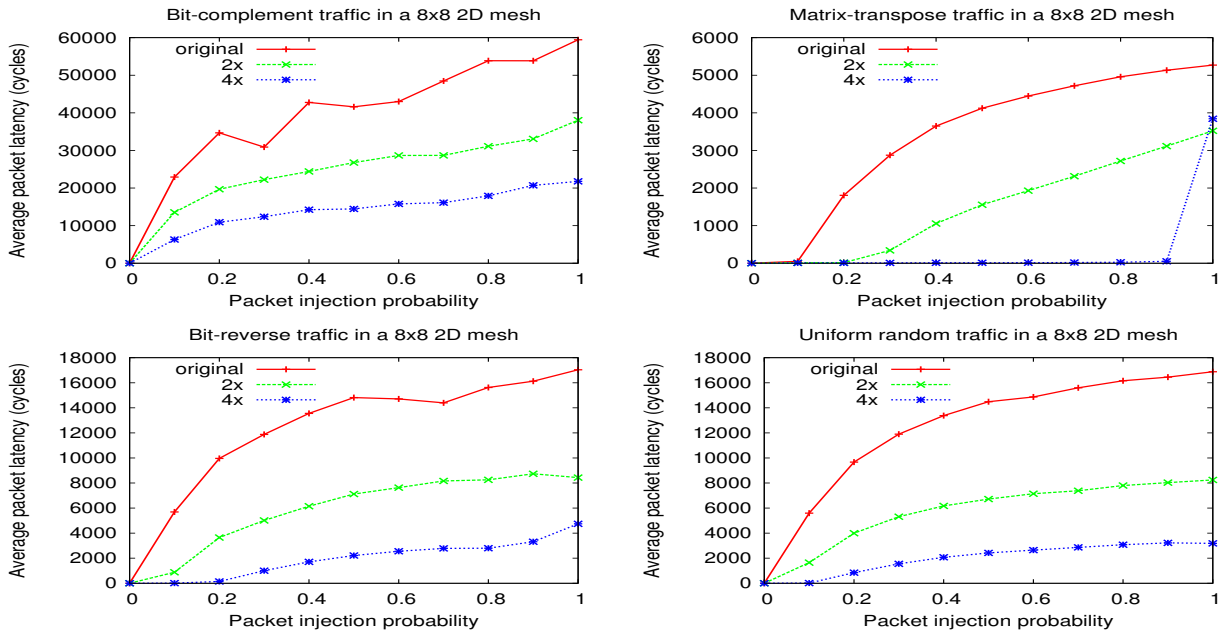

Fig. 3. The average packet latency on a 8x8 Irvine NoC architecture, while the speed with which data flits advance in the network varies for 4 different communication patterns

ACKNOWLEDGMENT

The ns-3 NoC simulator was developed during the first author's external PhD stage, at the University of Augsburg, Germany. We would like to thank Prof. Dr. Theo Ungerer, chair of the Systems and Networking department, and to his research group for their advices and help during this stage (2009/2010, 5 months period). This work was partially supported by CNCSIS no. 485/2008 research grant offered by the Romanian National Council for Academic Research.

REFERENCES

[1] J. Hu and R. Marculescu, "Energy-aware mapping for tile-based NoC architectures under performance constraints," in *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*. Kitakyushu, Japan: ACM, 2003, pp. 233–239.

[2] R. Marculescu and J. Hu, "Energy- and performance-aware mapping for regular NoC architectures," *IEEE Transactions on ComputerAided Design of Integrated Circuits and Systems*, vol. 24, no. 4, pp. 551–562, 2005.

[3] S. Murali and G. D. Micheli, "Bandwidth-Constrained mapping of cores onto NoC architectures," in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 2*. IEEE Computer Society, 2004, p. 20896.

[4] K. Srinivasan and K. S. Chatha, "A technique for low energy mapping and routing in network-on-chip architectures," in *Proceedings of the 2005 international symposium on Low power electronics and design*. San Diego, CA, USA: ACM, 2005, pp. 387–392.

[5] G. Ascia, V. Catania, and M. Palesi, "Multi-objective mapping for mesh-based NoC architectures," in *Proceedings of the 2nd IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*. Stockholm, Sweden: ACM, 2004, pp. 182–187.

[6] J. P. Soininen and T. Salminen, "Evaluating application mapping using network simulation," *Proc of the Inter Symp on SystemonChip*, vol. 1100, no. Kaitovyl 1, p. 2730, 2003.

[7] (2010) The SystemC website. [Online]. Available: http://www.systemc.org

[8] S. Murali and G. D. Micheli, "SUNMAP: a tool for automatic topology selection and generation for NoCs," in *Proceedings of the 41st annual Design Automation Conference*. San Diego, CA, USA: ACM, 2004, pp. 914–919.

[9] C. Grecu, A. Ivanov, P. Pande, A. Jantsch, E. Salminen, U. Ogras, and R. Marculescu, "Towards open Network-on-Chip benchmarks," in *Proceedings of the First International Symposium on Networks-on-Chip*. IEEE Computer Society, 2007, p. 205.

[10] S. Mahadevan, F. Angiolini, M. Storgaard, R. G. Olsen, J. Sparso, and J. Madsen, "A network traffic generator model for fast Network-on-Chip simulation," in *Proceedings of the conference on Design, Automation and Test in Europe - Volume 2*. IEEE Computer Society, 2005, pp. 780–785.

[11] R. P. Dick, D. L. Rhodes, and W. Wolf, "TGFF: task graphs for free," in *Proceedings of the 6th international workshop on Hardware/software codesign*. Seattle, Washington, United States: IEEE Computer Society, 1998, pp. 97–101.

[12] (2010) The Embedded System Synthesis Benchmarks Suite (E3S) website. [Online]. Available: http://ziyang.eecs.umich.edu/~dickrp/e3s/

[13] (2010) The Embedded Microprocessor Benchmark Consortium (EEMBC) website. [Online]. Available: http://www.eembc.org

[14] (2010) The ns-3 network simulator website. [Online]. Available: http://www.nsnam.org/

[15] H. vom Lehn, K. Wehrle, and E. Weingärtner, "A performance comparison of recent network simulators," *2009 IEEE International Conference on Communications*, pp. 1–5, 2009.

[16] S. Schlingmann, "Selbstoptimierendes routing in einem network-on-a-chip," Master's thesis, University of Augsburg, 2007.

[17] J. Duato, S. Yalamanchili, and L. M. Ni, *Interconnection Networks: An Engineering Approach*, 1st ed. Institute of Electrical & Electronics Enginee, 1997.

[18] S. E. Lee and N. Bagherzadeh, "Increasing the throughput of an adaptive router in network-on-chip (NoC)," in *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis*. Seoul, Korea: ACM, 2006, pp. 82–87.

[19] E. Salmien, A. Kulmala, and T. D. Hamalainen, "Survey of network-on-chip proposals," White paper, © OCP-IP, Tampere University of Technology, March 2008. [Online]. Available: http://ocpip.biz/uploads/documents/OCP-IP_Survey_of_NoC_Proposals_White_Paper_April_2008.pdf