Requirements for Network-on-Chip Benchmarking

Erno Salminen[1], Tero Kangas[1], Jouni Riihimäki[2] and Timo D. Hämäläinen[1]
[1]Tampere University of Technology
P.O. Box 553, FI-33101, Tampere, Finland
erno.salminen@tut.fi
[2]Nokia Technology Platforms
P.O. Box 88, FI-33721, Tampere, Finland

Abstract

This work presents the motivation, basic concepts, and requirements for benchmarking a Network-on-Chip (NoC). Currently there is practically no benchmark sets for NoC or the presented tools do not meet the requirements. The presented benchmarking method utilizes traffic generator with a dataflow models of the applications. Combined with transaction-level NoC, the abstract application model allows approximately 200x speedup and on average 10% error in estimated runtime w.r.t. cycle-accurate HW/SW co-simulation without exposing the exact internal functionality of the application.

1. Introduction

Common benchmarks allow fair performance comparison. Traditionally, benchmarks have measured the performance of CPU and/or its compiler but nowadays they are used in many areas of system design, for example, CAD (computer-aided design) tools and Network-on-Chip (NoC). There are three basic benchmark categories: *synthetic*, *actual applications*, and *algorithm-based* (or derived) benchmarks. Synthetic benchmarks, e.g. Dhrystone [1] and Whetstone [2], abstract out the functional details. For example, they execute instructions according to an artificial distribution or the one measured from a real life program. They are intended to debugging and isolating certain functionality and hence they tend to be small in size. Application benchmarks, on the other hand, are functionally accurate tests that use system or user level applications for testing. An algorithm-based benchmark includes only the kernel of the real algorithm but not the full application [1].

Applications give the best accuracy but are harder to port to different systems, require standardized input data, and their simulation is usually slower than other types of benchmarks. In addition, many companies and institutions are not willing to contribute new test cases in the form of source codes. Small benchmarks, on the other hand, require only minimal resources so they can be widely adopted, but their expressiveness is limited and they tend to become obsolete as technology evolves. Furthermore, they may be easily tricked to achieve good results, for example, with some special compiler options. A scalable benchmark can obtain a superset of the information given by any particular fixed size benchmark and, hence, remains valid for longer time [3]. Pseudo-random graphs have been used for allocation and scheduling research. However, it is important to ensure that graphs are reproducible by other researchers to allow comparison [4]. Sometimes, however, it may be hard to prove that artificial cases represent properties of any known application.

So far, most of the NoC studies regard only implementation related issues, such as area and power, or anecdotal performance values. Furthermore, the researchers tend to use their own, proprietary test cases, which complicates comparison between research projects. Therefore, a common benchmark set is required also for NoC evaluation to allow fair and thorough comparison of different approaches.

In Section 2, we introduce the goals and methods for NoC benchmarking. Furthermore, we present some common requirements and an approach for meeting them. An exemplar application set is presented in Section 3. Finally, the concluding remarks are given.

2. Benchmarking Network-on-Chips

Network-on-Chips are often utilized in embedded systems with multiple processing elements (both CPUs and HW accelerators) and, hence, the traditional single-CPU benchmarks are not applicable. Unfortunately, also the multiprocessor benchmarks may be unsuitable. SPEC OMP2001 [5], for example, requires 8 GB memory and UNIX or Windows operating system. Therefore, new benchmarks specializing in Network-on-Chip domain are needed.

According to our experience in NoC design and analysis, we have identified the following requirements for NoC benchmark set:

- Open - to allow comparison and wide adoption.
- High accuracy both in timing and the amount of data.
- Several test cases and scalable workload - to generalize the results and to estimate future application requirements.
- Modularity - several applications can be combined to model heterogeneous behavior.

- Expandable - researchers can contribute new test cases easily to keep the set up-to-date.
- Standard interface - to allow wide portability.
- Fast to simulate - to allow design space exploration.
- Measures several performance factors - see next subsection.
- Detects corrupted, duplicated, and missing data - benchmark set is also a NoC testbench.
- Allows various components allocations and application mappings - optimal allocation-mapping pair depends heavily on topology and other NoC parameters. This measures also the performance of NoC design tools which have a profound impact on system performance.

A. Required results of benchmarking

Many different types of metrics must be defined in addition to application runtime which is the most obvious. Furthermore, the performance reports must explicitly state which test case was used and whether the performance values are given for the best, average, or worst case. The standard deviation of values is also of great concern.

There are three basic quantities: throughput, latency, and fault tolerance [6]. Many sources report the throughput (or accepted load) which can be defined as a fraction of total capacity of the network when the system saturates. It is determined by measuring the amount of received data when the transfer probability of each processing element (PE) is increased. Latency is defined as the time required for data to reach its destination. It consists of separate phases, such as contention, transfer initialization, transfer, and turn-around latencies. In a single bus, the contention latency easily dominates whereas in multi-hop network the transfer latency may have the biggest impact. Fault tolerance describes the ability of the network to perform in the presence of one or more faults. It has been recognized for long in wide area, such as telephone networks, but it is also emerging into NoC domain.

Unlike PCs, embedded systems are targeted for a certain performance level in a narrow application domain instead of maximum performance in general purpose computing. Therefore, *performance with given constraint* is desired. For example, define the smallest area for NoC that achieves at least 500 MB/s throughput for test case *X*. Such constraints are very important when designing real-time systems in which the violation of (hard) timing constraints may be hazardous. Therefore, also the quantities related to implementation must be documented. These include at least silicon area, power, and operating frequency. It is equally important to report how the values are achieved: with synthesis or full-custom design, is placement and routing information included, and the values of key parameters (e.g. buffer size and data width).

The basic metrics and estimated PE utilization can be obtained automatically by the generic benchmark tool. To obtain greater insight, they should be augmented with NoC-specific metrics with separate monitors, for example the fairness of flow control and the utilization of links and buffers. This way the designer can spot the bottlenecks and design errors in the NoC. The benchmarking should be complemented with theoretical studies, regarding factors such as bisection bandwidth.

B. Traffic generator types

Benchmarking a multiprocessor system using multiple instruction set simulators (ISSs) gives accurate results but is too slow with contemporary workstations even if the simulation is distributed to multiple computers [7]. Applications can be run on a host simulation computer (*native execution*) and the execution time is modeled by annotating wait statements [8][9]. The simulation is both accurate (3-10% error is reported) and fast (more than 1000x speedup over ISS). However, the application source codes must be distributed in the benchmark set. Error detection and detailed performance measuring may also be laborious to implement with a real application.

For pure NoC benchmarking, only the external behavior of each PE needs to be modeled, i.e. the amount and timing of data transfers regardless of the actual values of the internal variables or the transferred data. This higher abstraction level approach is utilized with *traffic generators*, which provide means to generate data transfers to network according to pre-defined communication profile. Hiding the internal functionality should allow researchers to more easily contribute data to test suite allowing it to be extended and updated with technology. Furthermore, the contents of data can be freely chosen to simplify error checking, NoC debugging, and performance measurements. Traffic generators have been widely utilized, for example, for characterizing Internet traffic in order to test and measure routers and servers [10][11].

Transfer-independent (also called stochastic) traffic generator does not take dependence of subsequent transfers into account but all PEs generate traffic according to a fixed probability and distribution [12][13][14][15][16]. For example, each PE transfers random number of data words to all other PEs with uniform probability regardless how it receives data from others [17]. In practice, the real traffic tends to be both localized and bursty and, therefore, the uniform model must be modified [18].

More realistic traffic may be generated by considering the dependencies between the transfers, in other words generating a *transfer-dependent* (i.e. reactive) communication profile [19][20][21][22][24][25]. Dependencies make the profile partially ordered, i.e. certain tasks are not executed before they have received their input data. Small runtime estimation errors of 2% [20] and 1.5% [22] have been reported. At the same time, the speedup over ISS can be in the range of 20x-50x [22] (accounting the effect of [23]).

In our approach, a transfer-dependent generator called Transaction Generator (TG) [25] is utilized. The basic principle is depicted in Fig. 1. Application model consists of communicating tasks which are mapped onto processing elements. The data transfers between tasks that are mapped to separate PEs are forwarded to the NoC. The task runtime and transfer sizes are defined according to the application and the characteristics of the PE models.

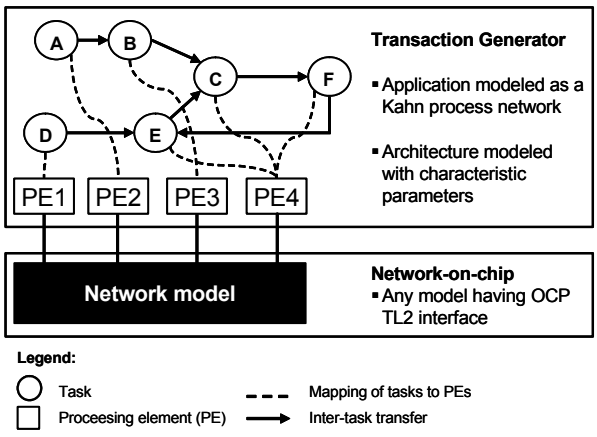

Figure 1. Transaction Generator is a tool for combining abstract application and architecture models for simulation with an arbitrary NoC

Both parameters may be varied uniformly within user-defined limits to mimic real behavior. Furthermore, TG allows both transfer-dependent and independent profiles. The exploration of various mapping possibilities and architectures is possible without modifying the application model. TG is written in SystemC and can be connected to different types of NoC models: VHDL at gate-level or RTL and SystemC at RTL or transaction level. Furthermore, TG automatically checks all the transfers and collects statistics about system performance such as execution counts for processes, execution and idle period lengths, and latency of data transfers. Hence, TG is likely to meet all the presented requirements for benchmarking tool.

C. *Traffic characterization*

Traffic characterization is required in order to utilize traffic generators. Static analysis prior to compilation is difficult and/or inaccurate since the execution (time) practically always depends on the input data set. For example, unbounded loops in SW are problematic. Therefore, it is often better to profile application by running it (either in simulator or in real HW) with multiple input data sets, collecting the communication (and potentially execution) traces, and generating the communication profile from the traces. Traditionally, the profiling is done after the application mapping by tracing the communication between PEs. This means that the architecture exploration is limited to the NoC itself. Every time the application, mapping, or PE allocation is changed, the profiling must be performed again. Clearly, this is contradictory to the requirements.

Another way is to profile the application execution in a reference PE or alternatively on each possible PE. The resulting database shows that, for example, application task *t* takes *n* clock cycles on *PE1*, *m* cycles on *PE2*, and so on. By utilizing this method, the same communication profile is applicable throughout the exploration for all application mappings and PE allocations [8]. Similarly as in architecture exploration, this facilitates the NoC benchmarking as the same communication profile remains valid for each NoC architecture.

3. Benchmark set for Network-on-Chip

To cover a wide spectrum of profiles, the aimed set consists of both application benchmarks and synthetic cases, both of which can be parameterized. Likewise, both transfer dependent and independent profiles are included. Models are preferably executed with a traffic generator since that allows easier mapping exploration, data checking, statistics collecting than actual or algorithm-based applications.

A. *Application benchmarks*

Different application types, i.e. communication and computation intensive cases, must be present. Regular and simple functions like FFT, DCT, IIR, and matrix multiplication are sometimes used [19][21][22] but they can hardly be considered as typical applications for SoC, i.e. they present only kernels. In contrast, video coding [14][16][21][26], 3GPP [24] and WLAN [27] baseband processing are more demanding and, hence, interesting.

In this work, a data-parallel video encoder [26] has been manually profiled to derive a parameterizable Transaction Generator model. The process network can be scaled by setting the picture size (QCIF, CIF etc.) and the number of slave CPUs. At the same, the number of tasks (e.g. *dct_8x8* and *vlc*) in the model varies between 20-220. The simulation speed was measured both cycle-accurate ISS simulation (10 ARM7 ISS connected to a co-simulation environment) and Transaction Generator simulation. ISS-based approach could be simulated at the speed of 235 cycles/second whereas TG provided over 10 000 cycles/second, i.e. over 40x speedup. Using transaction-level NoC model instead of RTL offers additional speedup, in the range of 5-7x. At the same time, the timing error remains at 10% on average which should be satisfactory for benchmarking purposes.

Manual profiling of video encoder application was tedious and time-consuming. Therefore, MAC protocol for proprietary WLAN described in UML 2.0 [27] was profiled by linking special functions to the automatically generated C code. The application was run on FPGA and the collected execution traces were transformed automatically into TG model (18 tasks) with scripts in few minutes which is comparable with results given in [22]. Other sources do not report time and effort needed for the profiling.

B. *Synthetic test cases*

Synthetic profiles are generated pseudo-randomly by setting few key parameters, for example, number of processes, minimum and maximum transfer sizes, and average number of targets per task. Random traffic patterns, both uniform [17][24] and localized [12][13][18], are the most commonly used. For example, Poisson or exponential distribution functions are used for localization. Simple synthetic, transfer-dependent, ring-shaped process

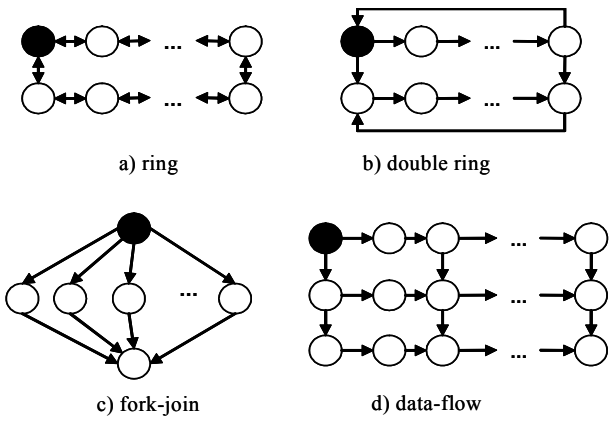

Figure 2. Examples of synthetic process networks

network were used for benchmarking in our earlier work [28].

Few examples of synthetic process networks for TG are shown in Fig. 2. The processes marked with black circle are start processes that are executed once after reset. In addition, they can be periodical, i.e. they are executed repeatedly irrespective of other transfers, or act as normal processes after their first execution. For simplicity, each example shows only one start process but their number can be freely parameterized. By using several periodical start processes, TG can also be used for generating transfer-independent, purely stochastic traffic. Furthermore, several synthetic test cases may be combined to generate larger and more heterogeneous traffic profiles.

4. Conclusions

This paper presents a background, classification, and requirements for NoC benchmarking. We shortly introduced Transaction Generator that uses communication profiles that are easy to generate, fast to simulate, and accurate enough in NoC benchmarking. Furthermore, we have generated a parameterizable set of synthetic and real application models. The following tasks are to generate new benchmarks, to evaluate and report the performance of existing Network-on-Chips, and prepare the on-line distribution of the benchmark set.

5. References

[1] A.R. Weiss, Dhrystone Benchmark - History, analysis, "scores" and recommendations, white paper, EEMBC Certification Laboratories, Austin, TX, Nov. 2002.

[2] R.P. Weicker, An overview of common benchmarks, Computer, Vol. 23, Iss. 12, Dec. 1990, pp. 65-75.

[3] J.L. Gustafson, R. Todi, Conventional benchmarks as a sample of the performance spectrum, HICSS, Kohala Coast, HI, Vol. 7, Jan. 1998, pp. 514-523.

[4] R.P. Dick, D.L. Rhodes, W. Wolf, TGFF: Task Graphs for Free, CODES/CASHE, Seattle, WA, Mar. 1998, pp. 97-101.

[5] The Standard Performance Evaluation Corporation, OpenMP Benchmark Suite, [online], http://www.spec.org/omp2001/, visited May 2005.

[6] W.J. Dally, B. Towles, Principles and practices of interconnection networks, Morgan Kaufmann Publishers, San Francisco, CA, 2004.

[7] J. Riihimäki *et al.*, Practical distributed simulation of a network of wireless terminals, Intl. Symposium on System-on-Chip, Tampere, Finland, Nov. 2004, pp. 49-52.

[8] A. Baghdadi, W.O. Cesario, A.A. Jerraya, N.-E. Zergainoh, Combining a performance estimation methodology with a hardware/software codesign flow supporting multiprocessor systems, IEEE TSE, Vol. 28, Iss. 9, Sep. 2002, pp. 822-831.

[9] A. Bouchhima, I. Bacivarow, W. Youssef, M. Bonaciu, A.A. Jerraya, using abstract CPU subsystem simulation model for high level HW/SW architecture exploration, ASP-DAC, Shanghai, China, Jan. 2005, pp. 969 - 972.

[10] D. Emma, A. Pescapè, G. Ventre, Analysis and experimentation of an open distributed platform for synthetic traffic generation, FTDCS, Suzhou, China, May 2004, pp. 277-283.

[11] S. Kalidindi, N. Huynh, B. Eklow, J. Goldstein, "Real life" system testing of networking equipment, ITC, Oct. 2004, pp. 1072 - 1077.

[12] K.Lahiri, A.Raghunathan, S.Dey, Evaluation of the traffic-performance characteristics of system-on-chip communication architectures, Intl.Conf. on VLSI Design, Bangalore, India, Jan. 2001, pp. 21-35.

[13] T. Salminen and J.-P. Soininen, Evaluating application mapping using network simulation, Intl. Symposium on System-on-Chip, Tampere, Finland, Nov. 2003, pp.27-30.

[14] S. Murali, G. de Micheli, Bandwidth-constrained mapping of cores onto NoC architectures, DATE, Paris, France, Feb. 2004, Vol 2, pp. 896-901.

[15] S. Santi *et al.*, On the Impact of traffic statistics on quality of service for networks on chip, ISCAS, May 2005, pp. 2349-2352.

[16] Jiang Xu, W. Wolf, J. Henkel, S. Chakradhar, Methodology for design, modeling, and analysis of networks-on-chip, ISCAS, May 2005, pp. 1778-1781.

[17] C. Neeb, M.J. Thul, N. Wehn, Network-on-Chip-Centric Approach to interleaving in high throughput channel decoders, ISCAS, May 2005, pp.1766-1769.

[18] P.P. Pande, C. Grecu, M. Jones, A. Ivanov, R. Saleh, Effect of traffic localization on energy dissipation in NoC-based interconnect, ISCAS, May 2005, pp. 1774-1777.

[19] M.E. Kreutz, L. Carro, C.A. Zeferino, A.A. Susin, Communication architectures for system-on-chip, SBCCI, Pirenópolis, Brazil, Sep. 2001, pp. 14 -19.

[20] K.Lahiri, A.Raghunathan, S.Dey, Design space exploration for optimizing on-chip communication architectures, IEEE TCAD, Vol. 23, No. 6, Jun. 2004, pp. 952-961.

[21] Jian Liang, A. Laffely, S. Srinivasan, R. Tessier, An architecture and compiler for scalable on-chip communication, IEEE TVLSI, Vol. 12 , Iss. 7, Jul. 2004, pp. 711-726.

[22] S. Mahadevan *et al.*, Network traffic generator model for fast network-on-chip simulation, DATE, 7-11 Mar. 2005, Vol. 2, pp. 780-785.

[23] L. Benini *et al.*, SystemC cosimulation and emulation of multiprocessor SoC designs, Computer, Vol. 36, Iss. 4, Apr. 2003, pp. 53-59.

[24] D. Wiklund, Development and performance evaluation of networks on chip, PhD thesis, No. 932, Linköpings universitet, Apr. 2005.

[25] T. Kangas, J. Riihimäki, E. Salminen, K. Kuusilinna, T. Hämäläinen, Using a communication generator in SoC architecture exploration, Intl. Symposium on System-on-Chip, Tampere, Finland, Nov. 2003, pp.105-108.

[26] T. Kangas, T. Hämäläinen, K. Kuusilinna, Scalable architecture for SoC video encoders, accepted to Journal of VLSI Signal Processing.

[27] P. Kukkala, V. Helminen, M. Hännikäinen, T. Hämäläinen, UML 2.0 implementation of an embedded WLAN protocol, PIMRC, Sep. 2004, Barcelona, Spain, pp. 1158-1162.

[28] E. Salminen *et al.*, Benchmarking mesh and hierarchical bus networks in system-on-chip context, SAMOS V, Samos, Greece, July 2005, pp. 354-363.