2011 IEEE Computer Society Annual Symposium on VLSI

A NoC Traffic Suite Based on Real Applications

Weichen Liu, Jiang Xu, Xiaowen Wu, Yaoyao Ye, Xuan Wang, Wei Zhang†, Mahdi Nikdast, Zhehui Wang
The Hong Kong University of Science and Technology, Hong Kong, China. E-mail: {weichen,eexu}@ust.hk
†Nanyang Technological University, Singapore. E-mail: zhangwei@ntu.edu.sg

***Abstract*—As benchmark programs for microprocessor architectures, network-on-chip (NoC) traffic patterns are essential tools for NoC performance assessments and architecture explorations. The fidelity of NoC traffic patterns has profound influence on NoC studies. For the first time, this paper presents a realistic traffic benchmark suite, called MCSL, and the methodology used to generate it. The publicly released MCSL benchmark suite includes a set of realistic traffic patterns for 8 real applications and covers popular NoC architectures. It captures not only the communication behaviors in NoCs but also the temporal dependencies among them. MCSL benchmark suite can be easily incorporated into existing NoC simulators and significantly improve NoC simulation accuracy. We developed a systematic traffic generation methodology to create MCSL based on real applications. The methodology uses formal computational models to capture both communication and computation requirements of applications. It optimizes application mapping and scheduling to faithfully maximize overall system performance and utilization before extracting realistic traffic patterns through cycle-accurate simulations. Experiment results show that MCSL benchmark suite can be used to study NoC characteristics more accurately than traditional random traffic patterns.**

I. Introduction

By integrating multiple processing units on a single chip, multiprocessor systems-on-chip (MPSoCs) can provide higher performance per energy and lower cost per function to applications with burgeoning complexity. The performance of an MPSoC is determined not only by the performance of its processing units, but also by how efficiently they collaborate with one another. It is the MPSoC's communication architecture which determines the collaboration efficiency. The on-chip communication architectures of MPSoCs are gradually moving from traditional buses and ad-hoc interconnects to more sophisticated networks-on-chip (NoCs) [1], [2], and have become an active research area in both industry and academic communities [3], [4].

As benchmark programs for microprocessor architectures, NoC traffic patterns are essential tools for NoC performance evaluations and architecture explorations. Several works studied the properties of on-chip traffic and proposed various traffic models to capture their patterns. Grecu *et al.* provides an overview of the requirements for NoC benchmarks [5]. Varatkar *et al.* investigated the self-similarity property exhibited in burst traffic among on-chip functional units for MPEG-2 video applications [6]. Soteriou *et al.* proposed a 3-tuple traffic model to empirically capture on-chip traffic characteristics [7]. Bahn *et al.* presented a generic traffic model to describe on-chip traffic properties, including burstiness, injection rate, and the distribution of source-to-destination pairs [8]. Many NoC simulation environments currently provide traffic models to generate random traffic patterns for NoC studies [9], [10], [11], [12]. They often have the capabilities to accept new traffic models and even realistic traffic patterns.

While realistic traffic patterns are based on the behaviors of real applications, random traffic patterns use probability distributions to randomize on-chip communication traffic characteristics, such as packet destinations and transmission intervals. When configuring properly, their effects on NoC performance and power consumption can approximate to the long-term accumulated effects of realistic traffic patterns. However, configuring parameters properly for random traffic requires a comprehensive knowledge of the corresponding realistic traffic patterns. On the other hand, realistic traffic patterns can provide more accurate performance and power consumption results and more detailed information to improve NoC architectures.

In this paper, we present a realistic traffic benchmark suite, called MCSL, and the methodology used to generate it. MCSL benchmark suite is publicly released and can be downloaded from [13]. It currently includes a set of realistic traffic patterns for 8 typical MPSoC applications and covers popular NoC architectures in various scales. MCSL captures not only the communication behaviors in NoCs but also the temporal dependencies among them. Each traffic pattern in MCSL has two versions, a recorded traffic pattern and a statistical traffic pattern. The former provides detailed communication traces for comprehensive NoC studies, while the latter helps to accelerate NoC explorations at the cost of accuracy. We developed a systematic traffic generation methodology to create MCSL based on real applications. The methodology uses formal computational models to capture both communication and computation requirements of applications. It optimizes application mapping and scheduling to faithfully maximize overall system performance and utilization before extracting realistic traffic patterns through cycle-accurate simulations. Experiment results show that MCSL benchmark suite can be used to study NoC characteristics more accurately than traditional random traffic patterns.

This is the first time that a realistic NoC traffic benchmark suite is systematically developed and publicly released. The benchmark suite provides an essential tool for NoC architecture explorations and evaluations. It can be easily incorporated into existing NoC simulators and substantially improve NoC simulation accuracy. Application mapping and scheduling on MPSoCs are optimized in the traffic generation methodology to accurately reflect the true communication behaviors in

978-0-7695-4447-2/11 $26.00 © 2011 IEEE
DOI 10.1109/ISVLSI.2011.49

IEEE computer society

practical MPSoC designs. The performance of the traffic patterns are extensively evaluated and analyzed through cycle-accurate simulations and compared with traditional traffics.

The rest of the paper is organized as follows. Section II discusses the traffic generation methodology for MCSL benchmark suite in detail. Performance evaluation and analysis is conducted in Section III. Section IV concludes this work.

II. Traffic modeling methodology

An overview of the traffic generation methodology for real applications is showed in Fig. 1. The generation process starts with an application model and an architecture model. Traffic patterns are generated through five steps including application mapping, application scheduling, performance evaluation on application mapping and scheduling results, cycle-accurate simulation, and statistical traffic generation. Two types of traffic patterns will be obtained including recorded traffic patterns and statistical traffic patterns. The generation steps interact with each other closely. An unsatisfied performance at the evaluation step can cause iterative rollbacks to previous steps for better application mapping and scheduling decisions, until the performance requirement is fulfilled. Application mapping and scheduling steps are closely collaborated with each other. They are essential for the methodology since application mapping and scheduling decisions substantially affect the final traffic patterns. Optimized mapping and scheduling decisions can take full advantages of the parallel hardware resources in MPSoCs and improve overall resource utilization. We will discuss each component of the methodology in detail in the following subsections.

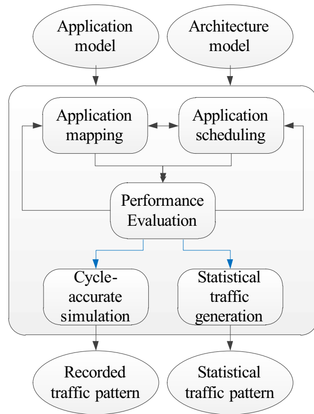


Fig. 1. An overview of the traffic generation methodology.

A. Application model

We use the task communication graph model as the input of the traffic generation methodology to faithfully capture the computation and communication requirements of real applications. A task communication graph is a directed graph $G_t = (V, E)$, where V is the set of computation tasks, and E is the set of communication channels between tasks. A task v has a normalized execution time t. A directed edge $e = (v_s, v_d, w)$ has a source task v_s, a destination task v_d and the amount of data w that sends from v_s to v_d. Figure 2 shows a part of the task communication graph for the H.264 decoder.

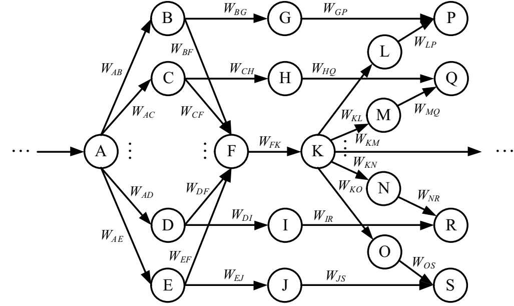


Fig. 2. Part of H.264 decoder's task communication graph.

Different schedules for a task communication graph have different performance when they are implemented on multiprocessor systems with tasks allocated to respective processing blocks for execution. Therefore, it is necessary to optimize the decisions of task mapping and scheduling on different hardware resources. This makes the generated traffic model more realistic and useful for practical implementations.

B. Architecture model

An architecture model captures the hardware resources in an MPSoC and includes processing blocks (PBs) and NoC. MPSoCs can have homogeneous as well as heterogeneous cores and use different NoCs. For the benchmark suite, we target regular NoC topologies, such as mesh, torus and fattree. MPSoC architectures with three different regular-topology NoCs are illustrated in Fig. 3. The selection tries to cover the most popular NoC architectures first and will be expanded in the future.

We define an architecture model as a graph $G_p = (P, N)$, where P is a set of heterogeneous PBs, and N is an on-chip communication architecture organized in a NoC paradigm. A PB p has an attribute, acceleration factor a, for tasks executed on it, and the actual execution time of the task on this PB is determined both by its normalized execution time and the acceleration factor of the PB. Specifically, the running time is the multiplication of the two values.

C. Application mapping and scheduling

The traffic generation methodology uses a centralized scheduling strategy to manage the entire chip resources and coordinate PBs. In this way, the scheduling and control decisions made are globally optimized for the whole system. Formally, given an application model $G_t(V, E)$, and an architecture model $G_p(P, N)$, the application mapping and scheduling problem is to find a mapping $M : V \rightarrow P$ for each task in V to a PB in P, as well as a static order schedule $S : V \rightarrow N$ for the set of tasks assigned to the same PB, where each task is assigned a unique number indicating its sequence of execution, such that the application performance is optimized. In static order scheduling, tasks assigned to the same PB execute following a pre-defined sequence strictly. It is proved to be more effective for multiprocessor systems [14].

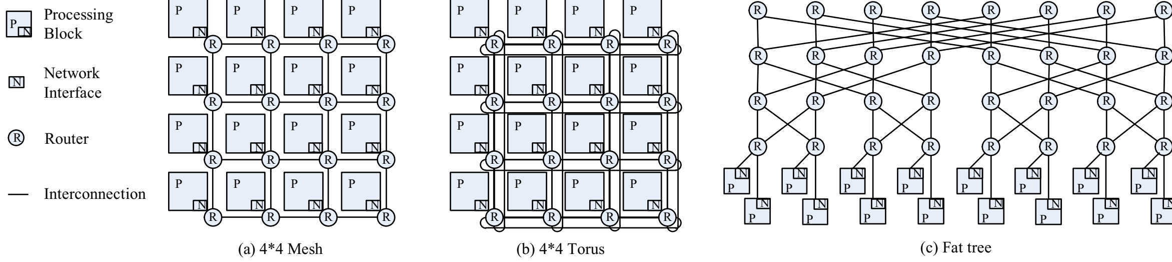


Fig. 3. 16-core MPSoCs with three different regular-topology NoCs.

TABLE I
DEFINITION OF THE SYMBOLS USED IN APPLICATION MAPPING AND SCHEDULING.

$w(v,p)$	The weight of mapping task v to PB p
$t(v,p)$	The required time for task v to finish on PB p
$f(p)$	the time for executing previously assigned tasks on PB p
$n(v,p)$	The total amount of network transmission generated by task v when it is assigned to PB p
$k(v,u)$	The number of packets that task v generates to edge (v,u)
$m(v)$	The mapping of task v
$l(p,q)$	The distance between PB p and q (predicted according to routing policies on different NoC topologies)

We develop a sophisticated load balanced mapping and static order scheduling approach for this problem. The basic idea is to distribute processing and network transmission workloads evenly and achieve high utilization to the hardware resources. The mapping strategy is to assign application tasks to PBs one by one in the order defined by the dependency relationships in the graph, and the schedule on each PB is determined by invoking the tasks mapped to the PB according to the dependency relations. The objective is to minimize the end-to-end delay of the application's execution with the consideration of network communication overhead. For a task $v \in V$, the weight of mapping it to a PB $p \in P$ is calculated by the following cost function:

$$w(v,p) = c_1 t(v,p) + qc_2 n(v,p), \tag{1}$$

in which $t(v,p)$ is the required time for task v to finish execution on p, defined by the time for executing previously assigned tasks on p plus the execution time of v on p, and $n(v,p)$ is the total amount of network transmission, defined by the number of packets generated by v and sent to other PBs (Table I lists the definitions of the symbols):

$$t(v,p) = f(p) + t(v) * a(p), \tag{2}$$

$$n(v,p) = \sum_{(v,u)\in E} k(v,u) \times l(p, m(u)), \tag{3}$$

where c_1, c_2 are the user-specified constant factors to tradeoff between the two concerns, and q is an architecture-specific scaling factor which balances the two terms in different units of measurement. For example, setting $c_1 = 1, c_2 = 0$ will obtain the mapping result that balances the PBs' workload only, and setting $c_1 = 2, c_2 = 1$ will balance PBs' workload as well as network traffic with bias towards PBs' workload. For performance evaluations in Section III, we set both factors c_1 and c_2 to 1, meaning that task execution and network transmission are considered to be of the same importance to the system performance. Factor q is set to 3 for the minimum number of clock cycles a packet used to cross a router.

Algorithm 1 The load balanced mapping and static order scheduling algorithm

```
Require: application model Gt(V,E), architecture model Gp(P,N)
 1: time = 0
 2: while there is unscheduled task in Gt do
 3:    readyQueue = UpdateReadyQueue(Gt)
 4:    for each task v in readyQueue do
 5:       minWeight = ∞
 6:       for each PB p ∈ P do
 7:          if w(v,p) < minWeight then
 8:             selectedProc = p
 9:             minWeight = w(v,p)
10:          end if
11:       end for
12:       m(v) = selectedProc
13:       s(v, selectedProc) = GetOrder(selectedProc)
14:       procAvailTimes = GetEarliestAvailableTimes(P)
15:    end for
16:    time = TimeAdvance(procAvailTimes)
17: end while
18: return mapping M, schedules S
```

The overall mapping and scheduling algorithm is shown in Alg. 1. The algorithm finds mapping and scheduling for each task until the application is finished (Lines 2–17). A ready queue is used to contain the tasks who get ready to execute (Line 3). In Lines 4–15, the tasks in the ready queue are all mapped to some PBs and scheduled in sequence. In Line 6–12, a task is measured by Eq. 1 and assigned to a PB with the minimum cost. After obtaining the mapping decision, Line 13 assigns a scheduling sequence number to the task. $m(v)$ and $s(v,p)$ are the mapping and scheduling result of task v, and both results are stored in the sets M and S for algorithm output. Each entry of the vector $procAvailTimes$ keeps the earliest available time of a PB, i.e., the earliest time instant that all the tasks already assigned to the PB are finished, and is used to compute the term $f(p)$ in Equation 2. In Line 16, a global variable $time$ is used to indicate the current time instant and is always advanced to the earliest future time with at least one available PB, i.e., the time to make new decisions. Relevant task executions and data transmissions are also conducted. When all the tasks are scheduled, we obtain a feasible mapping

and scheduling decision as well as its estimated performance.

D. Statistical traffic pattern generation

With all the results as described in the previous subsections, we can synthesize statistical traffic patterns for applications, and provide statistical distributions of task executions and packet transmissions on NoC-based MPSoCs. A *statistical traffic pattern* is given by $T_s = \{V_s(p) \mid p \in P\}$, where $V_s(p)$ represents the statistical behaviors of the set of tasks scheduled and executed on PB p. The task set $V_s = \{(s(v), D_t(v), IS(v), OS(v)) \mid v \in V\}$, where the schedule of task v is given by a unique sequence number $s(v) \geq 0$, and the execution time of the task in different instances follow the Gaussian distribution with mean μ_e and standard deviation σ_e, $D_t(v) = (\mu_t(v), \sigma_t(v)), \mu_t(v) \geq 0, \sigma_t(v) \geq 0$. Suppose task v is executed on p for l times. Let the execution time of the j-th ($j \in [1, l]$) execution be t_j. The mean and standard deviation for the Gaussian distribution of v's execution can be computed as follows:

$$\mu_t(v) = \frac{1}{l}\sum_{j=1}^{l} t_j, \qquad \sigma_t^2(v) = \frac{1}{l}\sum_{j=1}^{l}(t_j - \mu_t(v))^2 \tag{4}$$

The execution condition of task v is given by its input set of information $IS(v) = \{(v_i(e), n_i(e), m_i(e)) \mid e \in E_i(v), v_i(e) \in V\}$, where $E_i(v) \subseteq E$ is the set of incoming edges of v, the data on every incoming edge $n_i(e)$ must be ready for v, and the data are obtained from the corresponding predecessor task $v_i(e)$ and read from the memory space started at $m_i(e)$. The result of the task execution is given by the output set of information $OS(v) = \{(v_o(e), p_o(e), m_o(e), D_d(e), D_i(e)) \mid e \in E_o(v), v_o(e) \in V, p_o(e) \in P\}$, where $E_o(v) \subseteq E$ is the set of outgoing edges of v, and that is to generate some amount of data to each edge $e \in E_o(v)$, the destination is the successor task $v_o(e)$ on PB $p_o(e)$, and the data are written to the memory space started at $m_o(e)$, respectively. The data size generated on an edge can be described by the Gaussian distribution $D_d(e) = (\mu_d(e), \sigma_d(e)), \mu_d(e) \geq 0, \sigma_d(e) \geq 0$. Suppose the data sizes generated on edge e in the l times of v's executions are $d_1 \dots d_l$, respectively. The data size generated on each outgoing edge of task v can be calculated as follows:

$$\mu_d(e) = \frac{1}{l}\sum_{j=1}^{l} d_j, \qquad \sigma_d^2(e) = \frac{1}{l}\sum_{j=1}^{l}(d_j - \mu_d(e))^2 \tag{5}$$

If a successor task v_o is on the same PB with v, i.e., $p_o = p$, the data generated by v will be stored in the local memory and can be used directly by v_o. Otherwise, if the successor task v_o is on a different PB, i.e., $p_o \neq p$, the output data will be assembled into packets and sent to the target PB via the on-chip communication network. These packets are generated during the execution of the task. Our benchmark is compatible with any packet definition. As an example, we assume a fixed packet size is used. Thus, a number of packets are needed to assemble the data. The packet generation interval, which is the relative distance between two consecutive generated packets, follows the negative exponential distribution with rate parameter $\lambda_i(e)$, $D_i(e), \lambda_i(e) \geq 0$, for v's executions. Suppose the fixed packet size is z. The rate parameter for the negative exponential distribution of the packet generation intervals can be computed as follows:

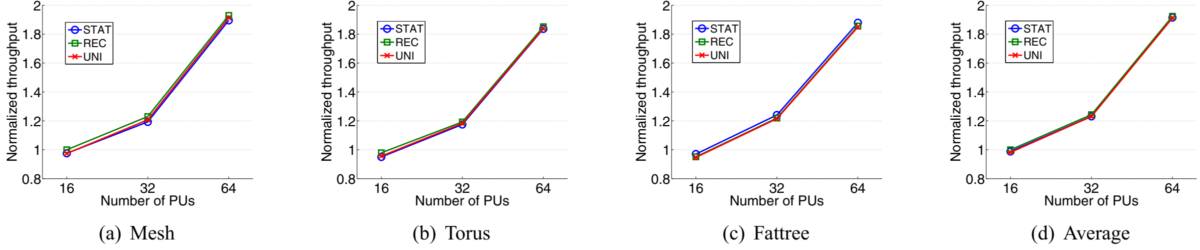
$$\lambda_i(e) = \frac{\mu_d(e)}{z \cdot \mu_t(v)} \tag{6}$$

The defined traffic pattern describes the statistical behaviors of the application running on the platform. We specify the deterministic task dependency relations in the generated traffic by the input and output sets $IS(v)$ and $OS(v)$, and include the task mapping and scheduling results. Three key components, the task execution times, the sizes of the data produced by the tasks and the relative time intervals that these data are assembled into packets, are described by statistical formulations. The generated pattern is useful for benchmarking NoCs with similar topologies, and other NoC metrics can be flexibly chosen.

E. Recorded traffic pattern generation

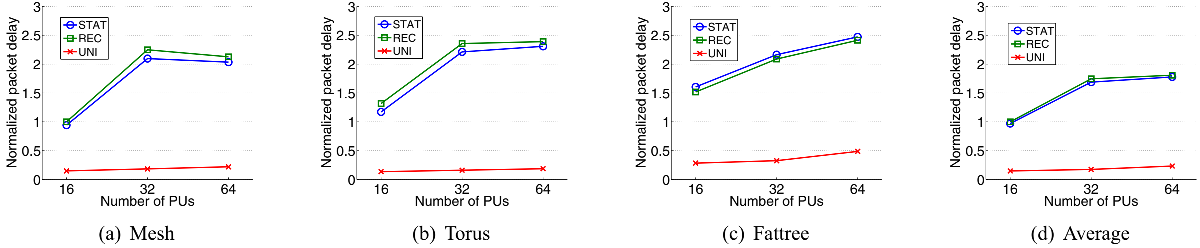
Besides statistical traffic patterns discussed in Section II-D, the methodology also generates recorded traffic patterns with detailed communication traces. A recorded traffic pattern is generated during cycle-accurate simulations for an application model on a NoC simulation platform with the mapping and scheduling result. It contains more accurate computation and communication traces, where all the task execution and packet generation events are recorded. The recorded traffic patterns are reusable on NoCs with different configurations but the same topology. Since the exact packet delays among processors are related with specific NoC configurations, the recorded traffic patterns keep the packet dependencies instead of exact timings. When the traffic patterns are applied to a different NoC configuration, all the temporal relations can be reconstructed correctly.

A *recorded traffic pattern* is given by $T_r = \{V_r(p) \mid p \in P\}$, where $V_r(p)$ represents the recorded behaviors of the set of tasks scheduled and executed on PB p. The tasks $V_r = \{(s(v), t(v), IS(v), OS(v)) \mid v \in V\}$, where task v has execution time t, and the unique sequence number for scheduling it on p is $s(v)$. The execution condition of task v is given by its input set of information $IS(v) = \{(v_i(e), n_i(e), m_i(e)) \mid e \in E_i(v), v_i(e) \in V\}$, where $E_i(v) \subseteq E$ is the set of incoming edges of v, the data on every incoming edge $n_i(e)$ must be ready for v, and the data are obtained from the corresponding predecessor task $v_i(e)$ and read from the memory space started at $m_i(e)$. The result of the task execution is given by the output set of information $OS(v) = \{(v_o(e), p_o(e), m_o(e), d_o(e)) \mid e \in E_o(v), v_o(e) \in V, p_o(e) \in P\}$, where $E_o(v) \subseteq E$ is the set of outgoing edges of v, and that is to generate data of size $d_o(e)$ to edge $e \in E_o(v)$, the destination is the successor task $v_o(e)$ on PB $p_o(e)$, and the data are written to the memory space started at $m_o(e)$, respectively. The data size $d_o(e)$ determines the number of packets that may be delivered through the network.



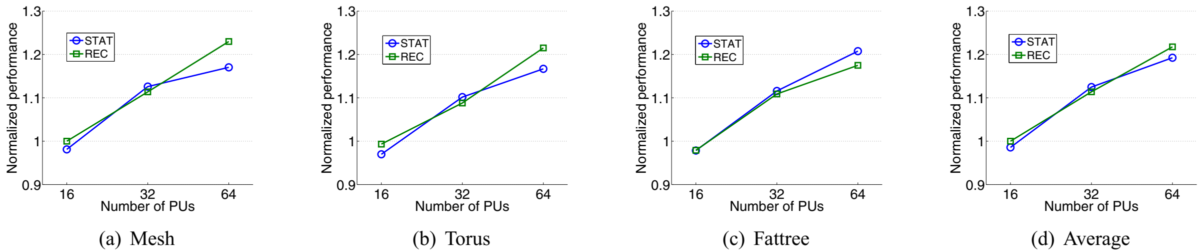
(a) Mesh (b) Torus (c) Fattree (d) Average

Fig. 4. Normalized throughput under different traffic patterns on mesh/torus/fattree-based 16-core, 32-core and 64-core MPSoCs.



(a) Mesh (b) Torus (c) Fattree (d) Average

Fig. 5. Normalized end-to-end packet delay under different traffic patterns on mesh/torus/fattree-based 16-core, 32-core and 64-core MPSoCs.



(a) Mesh (b) Torus (c) Fattree (d) Average

Fig. 6. Normalized application performance under different traffic patterns on mesh/torus/fattree-based 16-core, 32-core and 64-core MPSoCs.

Since the exact timing between task executions and communications is dependent on the simulation platform and can be affected by the topology, routing algorithm, etc., using it in new simulations with different setting will result in inaccuracy and make the result unrealistic. The recorded traffic pattern eliminates such temporal relations, while keeps the data dependencies among tasks in it, which is always true though the hardware environment changes. The timing will be generated according to the real dependencies in the applications and is guaranteed to be the correct semantics. This enhances the reusability of our traffic suite. NoC designs with similar topology to the given traffic pattern can use it to evaluate the performance of their platform.

III. Performance Evaluation and Analysis

We conduct extensive performance evaluations to verify the effectiveness of the proposed traffic generation methodology, and compare the NoC performance evaluation results based on MCSL benchmark suite with the traditional uniform traffic patterns. We develop a NoC simulation platform using SystemC for cycle-accurate simulations. We derive the application models from the real applications presented in [15], [16] for the traffic generation methodology, and list their details in Table II. Each application is mapped to mesh-based, torus-based, and fattree-based NoCs on homogenous MPSoCs with 16, 32, and 64 PBs (in 4*4, 4*8 and 8*8 for mesh and torus). Using MCSL and traditional uniform traffic patterns, we compare two important aspects of NoCs including network throughput and end-to-end packet delay under different topologies and network sizes. The injection rates of the uniform traffic patterns are set as the averages of the corresponding recorded traffic patterns to mimic the real conditions. In addition, MCSL can also be used to evaluate the overall applications performance in terms of application completion speed. Each application is run in pipeline for 10 iterations, and the average of all application in MCSL is used as the result for a particular network topology and size. In all the simulations, we use a fixed packet size of 8 flits and 32 bits per flit.

TABLE II
The realistic applications used for performance evaluation.

App.	Description	No. tasks
SAMPLE	Sample rate converter	612
H263E	H.263 encoder	201
H264DH	H.264 decoder with high resolution	6343
H264DL	H.264 decoder with low resolution	403
ROBOT	Robot control	88
FPPPP	SPEC fpppp	334
SATELL	Satellite receiver	4515
SPARSE	Sparse matrix solver	96

Figures 4–6 show the respective NoC performance results

based on the statistical and recorded traffic patterns of the realistic applications in the MCSL benchmark suite (marked by STAT and REC), and those results for the uniform traffic patterns (marked by UNI). The network throughput, packet delay and application's performance values based on the recorded traffic on the 16-core mesh-based NoCs are normalized to 1, and other results are given by relative values to these baselines. The average results in all the last subfigures are normalized to the 16-core recorded results.

There are several observations from the performance evaluation results. First, the MCSL benchmark suite shows significantly different NoC performance results comparing to the uniform traffic patterns. Since the injection rates of the uniform traffic patterns are set to the averages of the corresponding recorded traffic patterns to mimic the real conditions, their network throughput on the same platform should be close, as the result in Figure 4 shows on average 1.1% difference. Though the network workload is similar, the end-to-end packet delay shows significant difference, as shown in Figure 5. Compared to the uniform traffic, the realistic traffic patterns show on average 87.3% difference on the three NoCs. The difference is even larger in particular applications. A main reason for the huge difference is that the real traffic of an application often has local concentrations at a certain time and the concentrations move along time in the network, while uniform traffic patterns generate equally distributed traffic across the network. In comparison, the traffic patterns generated from real applications in the MCSL benchmark suite describe the real situations more accurately than the uniform traffic patterns, and thus will be more effective for evaluating NoC performances to obtain realistic results and conclusions.

Second, the network and applications' performances between the statistical and recorded traffic patterns are consistent in the experimental results. The overall throughput difference is 1.9%, and the average difference on packet delay is 5.5%. Figure 6 also shows that the performance of the applications in terms of running time only has an average of 2.1% difference for the two traffic patterns. The maximum difference in particular applications is 16.8%. Recorded traffic patterns offer sections of realistic NoC communications during application executions. They can be used to study detailed NoC behaviors in a short simulation time. Statistical traffic patterns include the runtime uncertainties in NoC communications. Although, they are not as accurate as the recorded traffic patterns for specific periods of application executions, statistical traffic patterns are useful to explore overall NoC characteristics through long-time simulations.

IV. Conclusion

For the first time, a realistic NoC traffic benchmark suite is systematically developed based on real applications and publicly released. The benchmark suite provides an essential tool for NoC studies. It captures not only the communication behaviors in NoCs but also the temporal dependencies among them. It can be easily incorporated into existing NoC simulators to substantially improve NoC simulation accuracy. MCSL provides two types of traffic patterns, recorded traffic patterns and statistical traffic patterns. The former offer detailed communication traces for comprehensive NoC studies, while the latter help to accelerate NoC explorations at the cost of accuracy. The traffic generation methodology creating MCSL uses the formal task communication graph model to capture both communication and computation requirements of applications. Application mapping and scheduling on MPSoCs are optimized by the methodology to truthfully reflect the communication behaviors in practical MPSoC designs. Possible future extensions include deploying the flexibility to support more architectural choices like topology and allow different mapping and scheduling strategies.

Acknowledgement

This work is supported by RGC, Hong Kong SAR.

References

[1] W. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," *Design Automation Conference, 2001. Proceedings*, pp. 684–689, 2001.

[2] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "H. 264 hdtv decoder using application-specific networks-on-chip," in *Multimedia and Expo, 2005. ICME 2005. IEEE International Conference on*, july 2005, pp. 1508 –1511.

[3] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Comput. Surv.*, vol. 38, no. 1, p. 1, 2006.

[4] STMicroelectronics, "Stmicroelectronics unveils innovative network-on-chip technology for new system-on-chip interconnect paradigm," 2005.

[5] C. Grecu, A. Ivanov, P. Pande, A. Jantsch, E. Salminen, and R. Marculescu, "An initiative towards open network-on-chip benchmarks," in *OCI-IP White Paper*, 2007.

[6] G. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for mpeg-2 video applications," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 12, no. 1, pp. 108 – 119, 2004.

[7] V. Soteriou, H. Wang, and L. Peh, "A statistical traffic model for on-chip interconnection networks," in *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*, 2006, pp. 104 – 116.

[8] J. H. Bahn and N. Bagherzadeh, "A generic traffic model for on-chip interconnection networks," in *NoCArc, First International Workshop on Network on Chip Architectures*, 2008.

[9] H. Hossain, M. Ahmed, A. Al-Nayeem, T. Islam, and M. Akbar, "Gpnocsim - a general purpose simulator for network-on-chip," in *Information and Communication Technology, 2007. ICICT '07. International Conference on*, mar. 2007, pp. 254 –257.

[10] J. Xi and P. Zhong, "A system-level network-on-chip simulation framework integrated with low-level analytical models," in *Computer Design, 2006. ICCD 2006. International Conference on*, oct. 2006, pp. 383 –388.

[11] C. Grecu, A. Ivanov, R. Saleh, C. Rusu, L. Anghel, P. Pande, and V. Nuca, "A flexible network-on-chip simulator for early design space exploration," in *Microsystems and Nanoelectronics Research Conference, 2008. MNRC 2008. 1st*, oct. 2008, pp. 33 –36.

[12] P. Wolkotte, P. Holzenspies, and G. Smit, "Fast, accurate and detailed noc simulations," in *Networks-on-Chip, 2007. NOCS 2007. First International Symposium on*, may. 2007, pp. 323 –332.

[13] www.ece.ust.hk/~eexu.

[14] W. Liu, Z. Gu, J. Xu, X. Wu, and Y. Ye, "Satisfiability modulo graph theory for task mapping and scheduling on multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems*, 2010.

[15] H. J. S. Kwon and S. Ha, "H.264 decoder algorithm specification and simulation in simulink and peace," in *International SoC Design Conference, pages 9-12*, 2004.

[16] T. Tobita and H. Kasahara, "A standard task graph set for fair evaluation of multiprocessor scheduling algorithms," *Journal of Scheduling*, vol. 5, no. 5, pp. 379–394, 2002.