

# Automatic Phase detection for Stochastic On-Chip Traffic Generation

## ABSTRACT

During System on Chip (SoC) design, Network on Chip (NoC) prototyping is used for adapting NoC parameters to the application running on the chip. This prototyping is currently done using traffic generators which emulate the SoC components (IPs) behaviour: processors, hardware accelerators, etc. Traffic generated by processor-like IPs is highly non regular, it must be decomposed into program phases. We propose an original feature for NoC prototyping, inspired by techniques used in processor architecture performance evaluation: the automatic detection of traffic phases. Integrated in our NoC prototyping environment, this feature permits to have a completely automatic toolchain for the generation of stochastic traffic generators. We show that our traffic generators emulate precisely the behavior of processors and that our environment is a versatile tool for networks-on-chip prototyping. Simulations are performed in a SystemC-based simulation environment with a mesh network-on-chip (DSPIN) and a processor running MP3 decoding applications.

## Keywords

Traffic generation, Network-on-chip, Phase behavior, Stochastic traffic modeling, Performance evaluation

## 1. INTRODUCTION

Systems on chip (SoC) are now commonly used in embedded systems for multimedia and telecommunication applications. Most of these SoC are composed of a single processor controlling various components (Intellectual Property: IP) all connected together. The computing power required by emerging applications running on mobile terminals, such as video on mobile phone for instance, has induced the development of a more complex SoC infrastructure, the so called multi-processor SoC (MPSoC) typically composed of a number of master components (processors or DMA for hardware accelerators) connected to a network-on-chip (NoC) or a hierarchy of busses.

The advent of networks-on-chip (NoC) has significantly increased the design complexity of such systems with some hard problems related to parallelism: non-determinism, memory and cache coherency, efficient workload distribution and network contention. Solving these problems during the short time available for design requires fundamental improvements in design methodologies. The most important shift is the setting of a refinement methodology allowing designers to explore design space at various levels of precision. These levels, called *transaction*, *bus-accurate*, *synthesizable*, allow the designer to check quickly that performances related to various metrics are achieved before writing the complete description of the system.

During design space exploration, simulation time is a major problem. There are two run-time behaviors very difficult to model at a high level: cache behavior and network contention. Precise simulation of these two behaviors can only be done with a low-level description of the components. This means hours (sometimes days) of simulation for a single execution or, as it is usually preferred, the use of extremely expensive hardware emulators. Reducing simulation time can be achieved by a clever analysis of the behavior of the system during execution. We are interested in the simulation of on-chip network behavior and performance evaluation. Traffic generators (TG) are more and more used during SoC design for platform prototyping or performance evaluation. When using TGs, simulation time is decreased because the IP is not fully simulated. Simulation is also more flexible: *i*) the same generic TG can be used for a class of similar applications, *ii*) the designer does not need to integrate the complete IP in its simulation environment, which is useful in early stage of design if the IP is not fully specified.

Most recent traffic generation methods use stochastic models. Statistical analysis and synthesis of on-chip traffic is difficult because this traffic usually presents complex statistical behavior. As pointed out by [13], the behavior of application code is decomposed into phases which have very different characteristics. Each phases can appear several times during the complete execution of the program. The precision of the simulation and the possibility of integrating the whole process in an automatic (or at least semi-automatic) framework are important parameters for evaluating the usefulness of a traffic generation environment.

In this paper, we present the automatic detection of traffic phases by analyzing simulation traces and show that these phases are necessary to emulate the traffic generated by multi-media applications running on SoC. By gathering different features presented individually in various recent

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

works, our traffic generation environment provides a very flexible tool for networks-on-chip prototyping. It can run a deterministic traffic replay (as in [5]) or generate a stochastic traffic with first order (as in [9]) and second order (as in [15]) statistics fitted to a particular trace. The feature described in this paper is the ability to run a traffic divided into separate *phases*, each phase having different characteristics. This makes our TG able to capture the inherent non-stationarity present in the traffic generated by the processors. We validate the precision of our traffic simulation in SystemC. We show that the network latency, transaction delay and aggregated throughput of a complete SoC platform are very close when we use TGs replacing processors. With our environment, the designer can explore the design space in a very flexible manner by, for instance, exploring other network architectures on a single phase in which network contention occurs.

The paper is organized as follows. In Section 2 we review the different existing techniques of traffic generation. Section 3 presents our traffic generator and the flow that we propose for analyzing and synthesizing on-chip traffic. Section 4 presents our experimental results that highlight the points mentioned above.

## 2. RELATED WORK

Using traffic generator in a simulation platform involves the following steps: *i*) the IP designer collects simulation traces by observing the behavior at the interface of each master component (if the IP is available), *ii*) he builds traffic models as close as possible to these traces, *iii*) the platform designer instantiates a traffic generator for each master component based on these models, and *iv*) inserts them in the simulation platform in place of the original components. Traffic generators can be separated into two main categories: the *deterministic approach*, in which traffic is produced using a finite state machine (FSM) configured by the IP designer or using a previous simulation trace, and the *stochastic approach*, in which the traffic is produced by a parameterized non-deterministic process.

Deterministic traffic generator (TG) [6, 5, 10] are derived from real simulation traces or written from scratch by IP designers. Such TGs can generate accurate transactions in time, size, and idle time that match the behavior of an IP. The advantages of this approach are the precision and the speedup factor it can achieve compared to the complete IP simulation. However, one limitation of the deterministic approach is that the length of the simulation is limited by the length of input traces used. Furthermore, such TGs cannot handle behaviors that are dependent on input data sets.

An alternative solution is to use stochastic traffic generators. These TGs build a model of the traffic. This modeling permits to investigate how small variations in the model parameters impact performance. This is an interesting way of testing a NoC robustness with reasonably accurate traffic (much more accurate than completely random traffic). Such a model can also be useful when the IP is not fully available or when the behavior is likely to change slightly from one execution to the other. However some traffics are very difficult to model and the traffic generation environment should include advanced statistical analysis tools such as multi-phase statistical analysis and second order statistics fit. For instance Marculescu et al. [15] have isolated a long-range-dependent behavior (i.e. second order statistical prop-

erties) at the coarse-grain level. Our simulation environment is currently able to capture and generate traffic with second order statistic fitted to a particular covariance [1]. However, our experiments do not exhibit long-range dependence. This result has to be confirmed by other simulations; our conjecture is that this is due to the presence of caches that smooth the communications.

The major part of NoC performance evaluation is currently done using random sources [16, 14, 9, 4]. These works mainly focus on the evaluation of the NoC in its early stage of development, and on its performance under random traffic. However none of these works propose a *fitting procedure* to determine the adequate statistical models that should be used to simulate the traffic: most of them choose arbitrarily the statistical behavior of each IP. To our knowledge, none of these approaches have introduced multi-phase modeling. A complete traffic generation environment should integrate both deterministic and stochastic traffic generation techniques.

A processor associated with a cache generates a non-stationary traffic, which can be divided into phases corresponding to different parts of the executed program. Each phase is stationary in the sense that the stochastic characteristics of the process observed during this phase are almost constant. This point has been thoroughly investigated in the domain of processor architecture performance evaluation. A very good summary of these works is presented by Calder et al. in [13]. Calder et al. isolate program phases by analyzing basic blocs repartition in successive *intervals* (an interval can represent 10 millions of instructions). Then, these phases are compared and grouped using a *k*-mean algorithm [11]. We have adopted a similar approach to decompose the traffic generated by a processor in phases. Our model is simpler and the interval is approximately composed of a thousand of transactions. The data used to represent the activity of the processor are the traffic's statistical characteristics.

Calder et al. use this phase decomposition in SimPoint [8] for architecture performance evaluation. This is a powerful technique that can provide huge improvement in simulation by simulating only one *simulation point* per phase and replicating the behavior during all the corresponding phase. This is not our goal here because we target precise traffic simulation of a given IP for NoC prototyping. Network contention needs to be precisely simulated, and as it is the result of the superposition of several traffics, picking simulation points becomes a difficult task. However a study in the spirit of [3] may be applicable of NoC prototyping, this is part of our futur works.

## 3. MULTIPHASE TRAFFIC GENERATORS

We now present our analysis and synthesis flow for building multi-phase traffic generators that can be used to replace an IP in cycle-accurate NoC performance evaluation.

### 3.1 On-chip traffic modelling

The traffic produced by a component is modelled as a sequence of transactions. The  $i^{th}$  transaction is a 4-uple  $(A(i), C(i), S(i), D(i))$  meaning in this order, target address, command (read or write), size of transaction, and delay (number of cycles between two successive requests). This is illustrated in Figure 2. From this transaction sequence, we define the *aggregated throughput*  $W_{\Delta}(j)$ , which corresponds to the amount of bus-words transferred in the

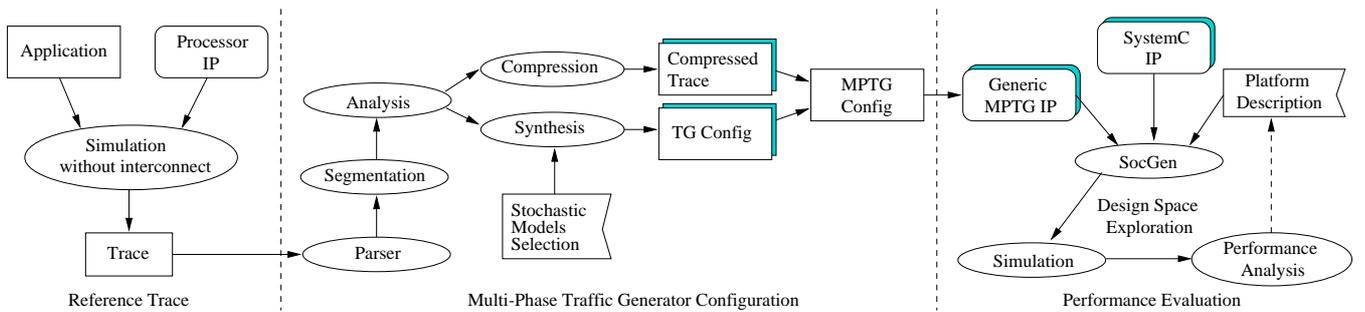


Figure 1: MPTG Framework: Traffic analysis and synthesis flow

time interval  $[j\Delta, (j+1)\Delta]$ . We also define the latency of the  $i^{\text{th}}$  transaction  $L(i)$  as the number of cycles between the start of a  $i^{\text{th}}$  request and the start of the associated response. This is basically the round-trip time in the network.

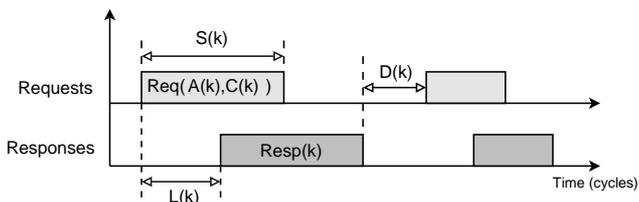


Figure 2: Traffic modelling formalism

Traffic is generated according to the 4-uple describing each transaction, and this 4-uple can be either read from a previously recorded trace (replay), or generated as the realization of a stochastic process.

### 3.2 Global methodology

The global simulation flow is depicted on Figure 1. First, we generate a reference trace by simulating the processor IP to be emulated. This trace is obtained with an ideal network environment (no network contention), which makes the simulation very fast. Then, we process the trace in our traffic analysis and synthesis tool [1] and we obtain configuration files for our traffic generators. A parametric generic traffic generator has been written once for all, it is referred to further in the text as MPTG. Transactions are generated by MPTG according to a phase description file and a sequencer is in charge of switching between phases. Each phase consists either of a replay of a recorded trace, or of a stochastic model with parameters adjusted by the fitting procedure described in [1]. Finally, the platform designer describes the desired platform architecture (such as the one presented in Figure 3) and uses a PERL script (referred to as SocGen) that generates all files needed for simulation. Thus the simulation takes place and performance analysis indicates whether some parameters of the platform have to be changed or not.

Two important features of our MPTG are the following: *i*) it is aware of the network latency (requests are sent only if the network is ready), and *ii*) it can be configured to emulate the communication scheme of the target IP. For example, as we target processor/cache traffic, the MPTG is configured with blocking reads and non-blocking writes in order to emulate the write buffer of the cache. These properties ensure

that *the same MPTG configuration files* can be used on various on-chip interconnects, thus allowing fast design space exploration of the NoC.

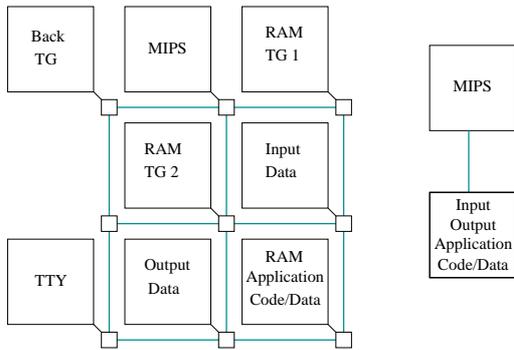
### 3.3 Automatic phase determination

The contribution of the paper lies in the adaptation of the work of [13] to NoC prototyping. In general, decomposing a non-stationary process into stationary parts is very difficult. Nevertheless, it appears that our programs are *piece-wise* stationary. We use the  $k$ -mean algorithm [11] which is a classical technique to group multi-dimensional values in similar sets, and we end up with a good clustering as demonstrated in section 4. Our automatic phase determination algorithm is the following:

1. First, we select a list of  $M$  metrics that will be used to perform the clustering. These metrics are chosen in the ones introduced in Section 3.1 (delay, size, command, etc.). Each metric is a sequence of values.
2. Each one of these sequences is then splitted into *intervals* of length  $L = 5000$  transactions. Mean and variance are computed on each interval, and this makes a  $2M$ -dimensional representative vector used for the clustering.
3. We perform clustering in  $k$  phases using the  $k$ -mean algorithm with different values of  $k$  (2 to 7 in practice). This algorithm finds  $k$  centroids in the space of representative vectors. Each interval will be assigned the number of its *closest* center (in the sens of the quadratic distance).
4. To evaluate different clustering, we compute the Bayesian Information Criterion (BIC) [12]. This information gives a score of the clustering evaluating the log of the probability of the data, given the model. A higher BIC means better clustering.

Once the phases are identified, statistical analysis is performed on each extracted phase by an automatic fitting procedure that adjusts the first and second statistical orders (see [1]) for details). The designer has to choose which model he wants to use before analyzing the trace.

We developed an independent random number generator that can produce realizations of a wide variety of processes [1]. This generator is integrated in the MPTG and the analysis produces the adequate MPTG configuration file.



**Figure 3: Simulation platforms: DIRECT (right) and MESH1 (left)**

## 4. EXPERIMENTATIONS

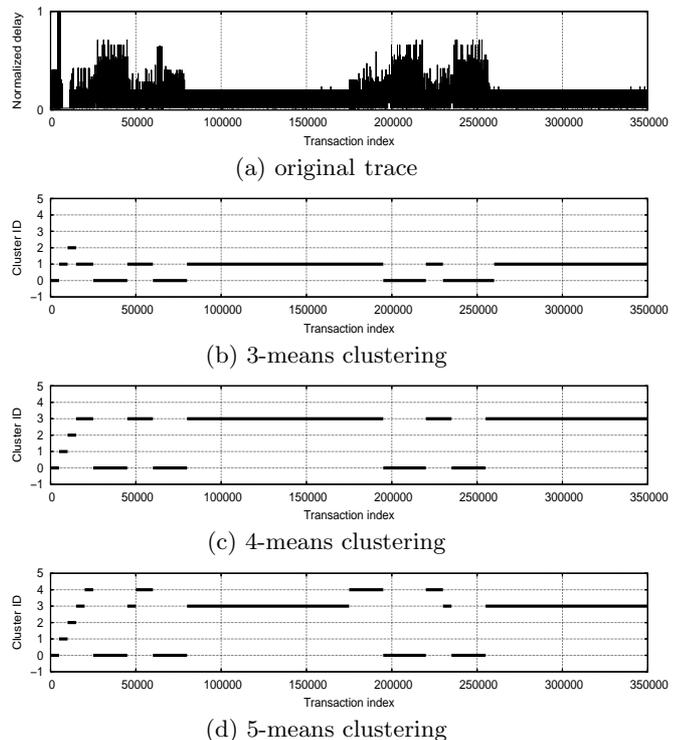
In this section, we present experimental results. With these results, we want to demonstrate that *i*) the automatic segmentation of traffic traces is efficient, and *ii*) the accuracy of stochastic multi-phase traffic generation is good.

### 4.1 Experimental setup

We use an open source, SystemC-based, cycle-accurate and bit-accurate simulation environment: SOCLIB [2]. We use a tiny operating system for multiprocessor management (Mutek). We present here the results on an implementation of the MPEG-layer 3 audio decoding software, further referred to as MP3. 2 frames are decoded in the results presented here, representing 350000 memory transactions when executed on the processor (MIPS r3000). Similar results have been obtained for MPEG2 and JPEG2000 applications, but are not presented here because of space limitation. The NoC used is DSPIN inherited from the research of the LIP6 laboratory (evolution of SPIN [7]). It uses wormhole memorization strategy and XY routing. The processor caches includes both data and instructions. It is composed of 32 lines of 8 words. Aggregated throughput has been computed as the number of flits transferred in consecutive time window of size 100 cycles. This is further referred to as throughput.

We intend by *platform* a particular physical interconnection of various IP. We used two platforms:

- The DIRECT platform does not use any interconnect, the processor is directly connected to a memory holding all necessary data. The latency is thus constantly equal to 1 cycle. This platform is used for basic validation of the MPTG and for reference trace collection as shown in Figure 1.
- The MESH1 platform is shown in Figure 3. The components are interconnected with the DSPIN NoC. The MIPS processor of Figure 3 is running the application. The BACK TG of Figure 3 is used for introducing contention over the network. It sends requests to both memories RAM TG1 and RAM TG2, whereas the MIPS communicates with the three other memories used for code, data, input and output streams. In order to test the MPTG in a more realistic way, the BACK TG has alternates between two phases, one with a high communication load and another with a low one. This in-



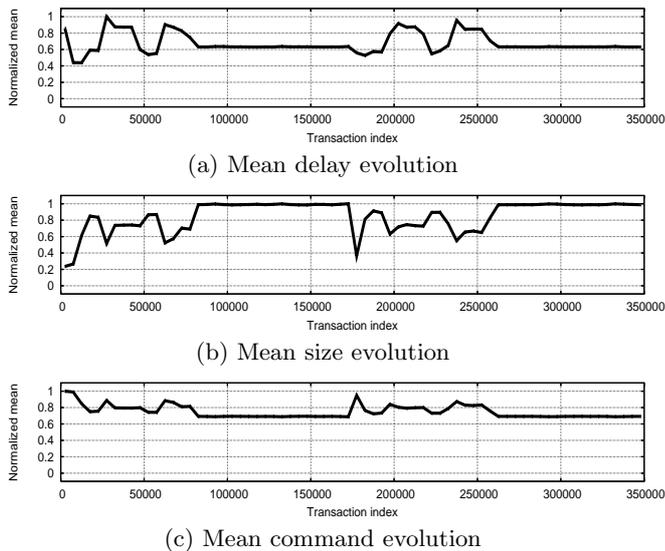
**Figure 4: Phases discovered by our algorithm on the MP3 traffic trace using delay as metric, for different phase numbers.**

roduce a time-varying contention and approximately multiply the number of cycles of the execution by 3.

### 4.2 Segmentation of the MP3 application

Figure 4a shows the delays of transactions  $D(i)$  as a function of the transaction index  $i$ . One can distinguish the boot at the start, and then the two frames being decoded. For each frame, several phases can be identified (for instance a long one at the end). This shows two important points. Firstly, the time evolution of the traffic is not stationary, so a stochastic fit on the whole trace would be meaningless. Secondly, from the moment similar behaviors appear, a segmentation should be done. This was already noticed in the high performance computing community [13], however this is, to our knowledge, the first time a traffic trace is being analyzed in this way.

We have run the phase segmentation algorithm described in Section 3.3 for different values of  $k$ , using delays of transactions as the metric. Figures 4b, 4c and 4d show the results for various number of phases. One can see that the algorithm finds the analogy between the two frames, and identifies phases inside each one of them. The segmentation seems valid and pertinent. As the segmentation is done with mean and variance as representative vectors, one expects that each identified phase exhibits a stationary behavior, likely to be processed by a stochastic analysis. In order to investigate on which metric (delay, size, command) the segmentation should be based, we have performed the segmentation with different representative vectors. Address sequence is not considered in these results because the dynamics of address values is such that the clustering fails.



**Figure 5: Evolution of the mean of different metrics (delay, size, command) of the mp3 application, computed in intervals of size  $L = 5000$  transactions.**

Figures 5a, 5b and 5c show respectively the evolution of the mean for three different metrics. As mentioned in Section 3.3, variance is also taken into account but it is not reported here because it shows a similar evolution. One can clearly see that all metrics are highly correlated. This was expected because program phases have an impact on all these metrics. This confirms that the segmentation done on delays is meaningful for all other metrics. Segmentations based on other metrics, or on a mixture of them, exhibit similar results. Table 1 presents the Bayesian Information Criterion (BIC). It should be read row by row, for instance the first row gives the BIC of the clustering done with the delay metric, giving the other metrics (delay, size, command). The highest value per row is always situated on the diagonal which is expected (the clustering score is always higher on the metric it was built with). However, on each row, the BIC decay remains reasonably low, so that we can trust the fact that clustering on any metric should provide the same kind of results. Further-down in the experimentations, we used a segmentation based on the delay metric.

### 4.3 Accuracy of the traffic generation

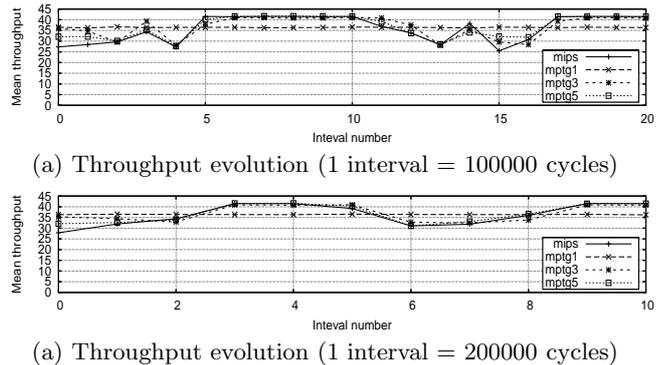
Let us first detail the *statistical analysis* introduced in Section 3.2. In order to build MPTG configuration files, we automatically compute for each identified phase: the probability distribution function of the delay, the access probability of each memory segment, and for each segment, the read/write probability. As we are emulating a processor and its cache, we fixed the read transaction size to the cache line size, and we also computed the probability distribution function of the write transaction’s sizes.

We have performed simulations with different configurations: “MIPS”, which this is the reference simulation of the MP3 application running on the MIPS, “MPTGN” for which traffic is generated with a MPTG with  $N$  phases, “DR” which

| -       | Delay   | Size    | Command |
|---------|---------|---------|---------|
| Delay   | 610.111 | 562.737 | 550.671 |
| Size    | 231.061 | 439.377 | 427.349 |
| Command | 913.042 | 1124.68 | 1112.66 |

**Table 1: Bayesian Information Criterion (BIC) for each metric (columns), given the metric used for clustering (rows).**

corresponds to a deterministic replay (the trace has been recorded and is replayed), and “RANDOM” which is a constant rate traffic with uniformly distributed target selection (the rate is fixed to the mean observed rate in order to do a fair comparison). Each configuration is run on both DIRECT and MESH1 platforms.

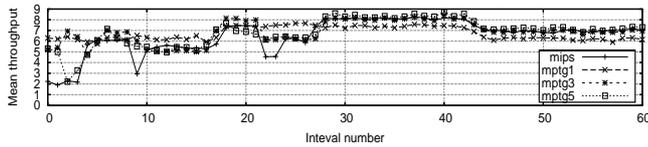


**Figure 6: Evolution of the mean throughput of the mp3 application on the DIRECT platform for various scales and configurations.**

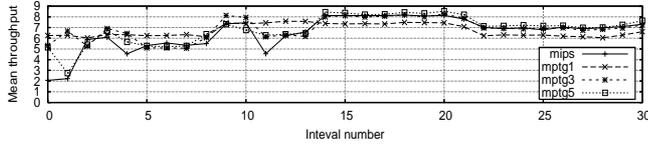
In order to compare a given configuration with the reference MIPS one, we should not look at global metrics such as the average delay or the average throughput. This would not point out the interest of the multi-phase approach. So, we have defined an accuracy measure that can be computed on each metric (delay, size, command and throughput). We compare the mean evolution of the metrics, just as represented in figure 5, for both simulations (MIPS and the one under study). This can be done graphically as depicted in figures 6 and 7. But, to summarize the results we defined the *error* as the mean of absolute values of relative differences between two mean evolutions. Let  $M_{ref}(i)$  be the mean evolution of some metric for the reference simulation, let further  $M(i)$  be the evolution of the same metric for another simulation, and let finally  $n$  be the number of points of both functions. The error (in percent) reads:  $Err = \frac{1}{n} \sum_i |M_{ref}(i) - M(i)| / M_{ref}(i) * 100$ .

This error is reported in tables 2 and 3. As expected, the higher the phase number is, the more accurate the results are. This highlights the importance of multi-phase traffic generation. Accuracy is lower on the MESH1 platform because the stochastic nature of traffic generation has a stronger impact. Still the multi-phase stochastic traffic generation lies in between the very accurate deterministic replay and the very inaccurate random traffic.

Figures 6 and 7 show the evolution of throughput (being a combination of size and delay) at different scales. The MPTG1 is naturally a straight line (one phase only) on DI-



(a) Throughput evolution (1 interval = 100000 cycles)



(b) Throughput evolution (1 interval = 200000 cycles)

**Figure 7: Evolution of the mean throughput of the mp3 application on the MESH1 platform for various scales and configurations.**

| Config. | Delay  | Size   | Cmd   | Throughput | Latency |
|---------|--------|--------|-------|------------|---------|
| dr      | 0      | 0      | 0     | 0          | 0       |
| random  | 31.784 | 34.315 | 6.911 | 13.640     | 0       |
| mptg1   | 7.833  | 15.184 | 6.431 | 11.449     | 0       |
| mptg3   | 1.972  | 8.336  | 3.279 | 5.577      | 0       |
| mptg5   | 1.334  | 3.270  | 1.154 | 2.600      | 0       |

**Table 2: Error (in percent) on various metrics with respect to the reference Mips simulation (DIRECT platform).**

RECT. On the MESH1 platform, the sort of wave oscillation in Figure 7 is a consequence of the two traffic phases of the BACK TG. One can see that MPTG3 and MPTG5 configurations follow the evolution of the reference simulation.

| Config. | Delay  | Size   | Cmd   | Throughput | Latency |
|---------|--------|--------|-------|------------|---------|
| dr      | 1.153  | 0      | 0     | 0.197      | 0.117   |
| random  | 41.278 | 75.242 | 7.709 | 102.316    | 27.825  |
| mptg1   | 18.604 | 14.759 | 6.256 | 12.696     | 10.086  |
| mptg3   | 17.194 | 8.169  | 3.255 | 6.212      | 0.767   |
| mptg5   | 14.772 | 3.239  | 1.210 | 5.651      | 0.626   |

**Table 3: Error (in percent) on various metrics with respect to the reference Mips simulation (MESH1 platform).**

These results show that multi-phase stochastic traffic generation is worth a try for NoC prototyping. Even though it is not as precise as deterministic replay, the phase behavior of the IP is preserved, which is in our opinion a key point of emulating the true contention on the network. The choice between stochastic and deterministic traffic generation depends on the purpose of the study. For instance random traffic generation is a good way to evaluate and compare routing strategies and other *large scale* design choices, whereas deterministic trace replay can provide very good accuracy for tuning the implementation details in the routers. We believe that the multi-phase stochastic traffic generation is interesting as a compromise between random and deterministic approaches. It combines a good accuracy and overcomes deterministic limitations. It especially provides the designer a phase description of traffic, and a stochastic model for each identified phase. This allows more flexibility in the traffic generation. For instance, the parameters of the models can

be slightly changed in order to evaluate the robustness of the NoC.

## 5. CONCLUSION AND FUTURE WORKS

In this paper, we have explained how traffic phases are automatically identified and synthesized in our traffic generation environment. Experimental results show that this automatic clustering is meaningful and that it can be performed on various metrics. This feature, coupled with the advanced stochastic analysis, fitting and synthesis procedure already available, makes our traffic generation environment an efficient NoC prototyping tool. Experimental results show the accuracy and the versatility of our MPTG, highlighting some of its key features: accurate replay over various interconnections, multi-phase traffic generation, stochastic traffic analysis and generation. Future works include the study of other applications, and the investigation of simulation time reduction using the phase behavior of each IP of the SoC.

## 6. REFERENCES

- [1] Reference omitted.
- [2] Soclib simulation environment. On-line, available at <http://soclib.lip6.fr/>, 2005.
- [3] M. V. Biesbrouck, T. Sherwood, and B. Calder. A co-phase matrix to guide simultaneous multithreading simulation. In *SPASS*, 2004.
- [4] S. G. P. et al. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *DATE*, pages 764–769, 2004.
- [5] S. M. et al. A network traffic generator model for fast network-on-chip simulation. In *DATE*, pages 780–785, 2005.
- [6] N. Genko, D. Atienza, G. D. Micheli, J. M. Mendias, R. Hermida, and F. Catthoor. A complete network-on-chip emulation framework. In *DATE*, pages 246–251, 2005.
- [7] A. Greiner and P. Guerrier. A generic architecture for on-chip packets-switched interconnections. In *DATE*, 2000.
- [8] G. Hamerly, E. Perelman, and B. Calder. How to use *eval* to pick simulation points. *Sigmetrics Perform. Eval. Rev.*, 31(4):25–30, 2004.
- [9] K. Lahiri, S. Dey, and A. Raghunathan. Evaluation of the traffic-performance characteristics of system-on-chip communication architectures. In *VLSI '01*, page 29, 2001.
- [10] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a mpsoc environment. In *DATE*, page 20752, 2004.
- [11] J. MacQueen. Some methods for classification and analysis of multivariate observations. pages 281–297, 1967.
- [12] D. Pelleg and A. Moore. X-means: Extending k-means with efficient estimation of the number of clusters. In *International Conference on Machine Learning*, pages 727–734, San Francisco, 2000.
- [13] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6):84–93, 2003.
- [14] R. Thid, M. Millberg, and A. Jantsch. Evaluating NoC communication backbones with simulation. In *21th IEEE Norchip Conference*, Riga, Nov. 2003.
- [15] G. Varatkar and R. Marculescu. On-chip traffic modeling and synthesis for mpeg-2 video applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(1):108–119, 2004.
- [16] D. Wiklund, S. Sathe, and D. Liu. Network on chip simulations for benchmarking. In *IWSOC*, pages 269–274, 2004.