Noc Book

Andrew K. Chan and Cheng Peng

May 21, 2008

Contents

Chapter 1

On-chip Processor Traffic Modeling for NoC Design

Antoine Scherrer†, Antoine Fraboulet‡, Tanguy Risset‡
†Laboratoire de Physique - Université de Lyon
F-69364, Lyon Cedex 07
antoine.scherrer@ens-lyon.fr
‡CITI - INSA de Lyon
Bat 502, 20 avenue Albert Einstein
F-69621, Villeurbanne, France
antoine.fraboulet@insa-lyon.fr, tanguy.risset@insa-lyon.fr

1.1 Introduction

Next generation system on chip (SoC) architectures will include many processors on a single chip, performing altogether the computation that used to be done by hardware accelerators. They are referred to as MPSoC for multi-processor SoC. When multi-processors were not *on chip* as in parallel machines twenty years ago, communication latency, synchronization and network contention were the most important fences for performance. This was mainly due to the cost of communication compared to computation. For simple SoC architectures the communication latency is kept low and the communication scheme is simple: most of the transactions occur between the processor and the main memory. For multi-processor SoC, a network on chip (NoC), or at least a hierarchy of buses, is needed and communication has a major influence on performance and on power consumption of the global system. Predicting communication performance at design time is essential because it might influence physical design parameters, such as the location of various IPs on the chip.

MPSoC are highly programmable, and target possibly any application. However they are currently mostly designed for signal processing and multimedia applications with real-time constraints, which are not as harsh as for avionic. In order to meet these real time constraints, MPSoC are composed of many master IPs (processors) and few slave IPs (memories and peripherals).

In this chapter we investigate on-chip processor traffic for NoC performance evaluation. Traffic modeling and generation of dedicated IPs (e.g. MPEG-2, FFT, etc.) use predictable communication schemes, such that it is possible to generate a traffic that *looks like* the one these IPs would produce. Such a traffic generator is usually designed together with (or even before) the IP itself. This is very different for processors. Processor traffic is much more difficult to model for two main reasons: *i*) cache behavior is difficult to predict (very program and data dependent) and *ii*) operating system interrupts lead to non deterministic behavior in terms of communication and contention. In order to build an efficient tool for predicting communication performance for a given application it is therefore essential to have a very precise modeling of the communications induced by applications running on processors.

Predicting communication performance can be done by a precise (cycle accurate) simulation of the complete application or by using a *traffic generator* instead of real IPs. Simulation is usually impossible at early stages of the design because IPs and programs are not available yet. Note also that SoC cycle accurate simulations are very time consuming, unless they are performed on very expensive hardware emulators (based on hundreds of FPGA). Traffic generators are preferred because they are parameterizable, faster to simulate, and simpler to use. However they are less precise because they do not execute the real program.

Traffic generators can produce communications in many ways, ranging from the replay of a previously recorded trace to the generation of sample paths of stochastic processes, or by writing a very simple code emulating the communications of a dedicated IP. Note that random sources can have parameters fitted to the statistical properties of the observed traffic, or parameters fixed by hand. Which communication parameter is emulated (latency, throughput, etc.) and which statistical property is emulated are important issues that must be addressed when designing a NoC traffic modeling environment. This is the main topic of this chapter.

One of the main difficulties in modeling processor traffic is that processor activity is not *stationary* (its behavior is not stable in time). It rather corresponds to a sequence of traffic phases (corresponding to program phase [10]). In each stationary phases, data can be fitted to well known stochastic processes with prescribed first (marginal distribution) and second (covariance) statistical orders.

The chapter is divided in two main parts: Section 1.2 gives background on stochastic processes as well as on-chip processor traffic. In Section 1.3 we detail the various steps involved in the design of a traffic generation environment and illustrate it with the MPTG environment [28]. Conclusion and related works are reported in Section 1.4.

1.2 Statistical Traffic Modeling

In this section, we introduce the specificities of on-chip processor traffic and how it can be modeled with stochastic processes. We then present some statistical background which is useful in describing and in simulating NoC traffic. This theoretical background includes basic statistical modeling methods, decomposition of the traffic into stationary phases and also an introduction to long range dependence.

1.2.1 On-chip Processor Traffic

On-chip processor communications are mostly issued by caches (instruction cache, data cache or both), and not by the processor itself. The presence and the behavior of this component imply that processor traffic is the aggregation of several types of communication described hereafter. Note that a processor without cache can be seen as a processor with a cache with one line of one single word. Transactions initiated by the cache to the NoC can be separated in three categories:

- **Reads**. Read transactions have the size of a cache line. The time between two reads corresponds to a time during which the processor computes only on cached data. Two flows must be distinguished, the instruction flow (binary from instruction memory) and the data flow (operands from data memory).

- **Writes**. Write transactions can have various sizes depending on the cache writing policy: write through (one word at a time), write back (one line at a time). If a write buffer is present then the size is variable, as the buffer is periodically emptied.

- **Other requests**. Requests to non-cached memory parts have a size of one word, as for atomic reads/writes. If a cache coherency algorithm is implemented then additional messages are also sent among processors.

Cache performance is a very popular research field [3, 14]. In the scope of embedded systems however, low cost solutions (low area, low consumption and low delay costs) are usually preferred. For instance in DSP architectures, no data cache is needed because the temporal locality of data accesses is likely to be low (data flow). For the same reason, each processor has a single communication interface, meaning that the type of communication mentioned above are interleaved. In other words, the traffic generated by the processor cannot be split into data and instruction streams, these two streams are merged. Based on this assumption we can now define more precisely how an on-chip processor communication can be modeled.

1.2.2 On Chip Traffic Formalism

The traffic produced by a processor is modeled as a sequence of transactions composed of *flits* (flow transfer units) corresponding to one bus-word. The k^{th} transaction is a 5-uple $T(k) = \big(A(k),\ C(k),\ S(k),\ D(k),\ I(k)\big)$ meaning respectively, target address, command (read or write), size of transaction, delay and inter-request time. This is illustrated on Figure 1.1. We also define the latency of the k^{th} transaction $L(k)$ as the number of cycles between the start of a k^{th} request and the arrival of the associated response. This is basically the round-trip time in the network and is used to evaluate the contention. We further define the aggregated throughput $W_\delta(i)$ as the number of transaction sent in consecutive and non-overlapping time windows of size δ. Note that this formalism only holds for communication protocols for which each request is expecting a response (even for write requests), which is the case for most IP communication interfaces such as VCI (Virtual Component Interface [21]).

One can distinguish two main communication schemes used by IPs: the *non-split transactions* scheme where the IP is not able to send a request until the response to the previous one has been received, and the *split transactions* scheme in which new requests can be sent without waiting for the responses. The non-split transaction scheme is widely used by processors and caches (although for cache, it might depend on the cache parameters), whereas the split transaction scheme is used by dedicated IPs performing computation on streams of data which are transmitted via *direct memory access* (DMA) modules.

1.2.3 Statistical Traffic Modeling

With the formalism introduced in the previous section, the traffic of processors consists in five time-series composing a 5-dimensional vector sequence $(T(k))_{k\in\mathbb{N}}$. Our goal is to emulate real traffic by means of traffic generators which will produce $T(k)$ for each k. This can be done in many ways and we present hereafter a non-exhaustive list of possibilities.

1. **Replay**. One can choose to record the complete transaction sequence $T(k)$ and to simply replay it as it is. This provides very accurate simulations, however it is limited

by the input simulation length. Furthermore, the size of the recorded simulation trace might be very large, and thus hard to load and to store.

2. **Independent random vector**. One can also consider the elements of the vector as sample paths of independent stochastic processes. The statistical behavior of each element will then be described, but the correlations between them will not be considered.
3. **Random vector**. In this case, the vector is modeled by doing a statistical analysis of each element as well as correlations between each pair of elements.
4. **Hybrid approach**. One can also, from the knowledge of the processor's behavior, introduce some constraints on top of the stochastic modeling. For instance, if an instruction cache is present, read requests targeted to instruction memory always have the size of the cache line.

One can also distinguish two ways of modeling the times at which each transaction occurs, leading to different accuracy levels:

- **Delay**. Use the delay sequence $D(k)$ representing the time (in cycles) between the reception of the k^{th} response and the start of the $(k+1)^{th}$ request.
- **Aggregated throughput**. Use the sequence of aggregated throughput of the processor $W_\delta(k)$, and transactions can be placed in various ways within the aggregation window δ.

The statistical modeling in itself relies on signal processing tools and methods that benefit from a huge amount of references [5, 13]. Let us recall that a stochastic process X is a sequence of random variables $X[i]$ (we use brackets to denote random variables). We will consider two statistical characteristics of stochastic processes: the marginal law (or probability distribution function), which represents how the values taken by the process are distributed, and the covariance function, which gives an information on the correlations between the random variables of the process as a function of the time lag between them. For instance, the sequence of delays $D(k)$ can be generated as the sample path of some stochastic process $\{D[i]\}_{i\in\mathbb{N}}$, with prescribed first and second statistical orders. Typical models for probability distribution functions (PDF) and covariances are reported in Table 1.1 and 1.2.

For each probability distribution and covariance, we use state of the art parameter estimation techniques (using mainly maximum likelihood expectation) [5, 13].

1.2.4 Statistical stationarity and traffic phases

When we want to model one element of the transaction vector with stochastic processes we should take stationarity into account. Indeed most stochastic process models are stationary and non-stationary models are much harder to use in terms of parameter estimation as well as model selection.

Let us first recall some background. The covariance function γ_X of a stochastic process $\{X[i]\}_{i\in\mathbb{N}}$ describes how the random variables of a process are correlated with each-other as a function of the time lag between these random variables. It is defined as follows ($\mathbb{E}$ is the expectation):

$$\gamma_X(i,j) = \mathbb{E}(X[i]X[j]) - \mathbb{E}(X[i])\mathbb{E}(X[j])$$

A process X is wide-sense stationary if its mean is constant ($\forall(i,j) \in \mathbb{N}^2, \mathbb{E}(X[i]) = \mathbb{E}(X[j]) \triangleq \mathbb{E}(X)$) and its covariance reduces to a one variable function as :

$$\forall(i,j) \in \mathbb{N}^2, \quad \gamma_X(i,j) = \gamma_X(0, |i-j|) \triangleq \gamma_X(|i-j|)$$

So, when modeling a time series, one should carefully check that stationarity is a reasonable assumption. For on-chip processor traffic, algorithms which are executed on the processor have many different phases resulting in different communication patterns, most of the time the traffic will not be globaly stationary. If signs of non-stationarity are present, one should consider building a piecewise stationary model. This implies the estimation of model parameters on several stationary phases of the data. At simulation time the generator will change the model parameters when it switches between phases.

A *traffic phase* is a part of the transaction sequence $T(k), i \leq k \leq j$. Since most multimedia algorithms are repetitive it is likely that similar phases appear several times in the trace. For instance, in the MP3 decoding algorithm each MP3 frame is decoded in a loop leading to similar treatments.

Phase decomposition The question is therefore to determine stationary traffic phases automatically. In general, decomposing a non-stationary process into stationary parts is very difficult. Calder *et al.* have developed a technique for the identification of program phases in SimPoint [32] for advanced processor architecture performance evaluation. This is a powerful technique that can dramatically accelerate simulations by simulating only one *simulation point* per phase and replicating that behavior during all the corresponding phases. In NoC traffic simulation, we do not pursue the same goal 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 communication flows, picking simulation points becomes a difficult task.

From Calder's work [32] we have developed a traffic phases discovery algorithm [29]. It uses the k-means algorithm [18] which is a classical technique to group multi-dimensional values in similar sets. The worst case complexity of this algorithm is exponential but in practice it is very fast. The automatic phase determination algorithm is the following:

1. Firstly, we select a list of M *elements* of the transaction sequence (delay, size, command, address, etc. see section 1.2.2).
2. The transaction sequence is then split into non-overlapping *intervals* of L transactions. Mean and variance are computed on each interval and for each of the M selected elements. Thus we build a $2M$-dimensional representative vector used for the clustering.
3. We perform clustering in k phases using the k-means algorithm with different values of k (2 to 7 in practice). The algorithm finds k centroids in the space of representative vectors. Each interval will finally be assigned the number of its closest center (in the sense of the quadratic distance) and therefore each interval will get a phase number.
4. To evaluate different clusterings we compute the Bayesian Information Criterion (BIC) [25]. The BIC gives a score of the clustering and 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 [31] for details). Examples of phases discovered by this algorithm are illustrated in Figure 1.8.

1.2.5 Long Range Dependence

Long-range dependence (LRD) is an ubiquitous property of Internet traffic [16, 24], and it has also been demonstrated on an on-chip multimedia (MPEG-2) application by Varatkar and Marculescu [35]. They have indeed found LRD in the communications between different components of an MPEG-2 hardware decoder at the macro-block level. The main interest in LRD resides in its strong impact on network performance [23]. In particular, the needed memorization in the buffers is higher when the input traffic has this property [9]. As a consequence, for macro-networks as well as for on-chip networks, LRD should be taken into account if it is found in the traffic that the network will have to handle.

Long-range dependence is a property of a stochastic process that is defined as a slow decrease of its covariance function [23]. We expect this function to be decreasing, because correlated data are more likely to be close (in time) to each other. However, if the process is long-range dependent, then the covariance decays very slowly and is not summable:

$$\sum_{k\in\mathbb{N}} \gamma_X(k) = \infty$$

Therefore, long-range dependence reflects the ability of the process to be highly correlated with its past, because even at large lags, the covariance function is not negligible. This property is also linked to self-similarity, which is more general, and it can be shown that asymptotic second order self-similarity implies long-range dependence [1].

A long-range dependent process is usually modeled with a power-law decay of the covariance function as follows:

$$\gamma_X(k) \underset{k\to+\infty}{\sim} ck^{-\alpha}, \quad 0 < \alpha \leq 1$$

The exponent α (also called *scaling index*) provides a parameter to tell how much a process is long-range dependent ($0 < \alpha \leq 1$). The Hurst exponent, noted H, is the classical parameter for describing self-similarity [23]. Because of the analogy between LRD and self-similarity, it can be shown that a simple relation exists between H and α: $H = (2-\alpha)/2$. As a consequence, H ($1/2 < H < 1$) is the commonly used parameter for long-range dependence. Note that when $H = 0.5$, then there is no long-range dependence (this is also referred to as short-range dependence).

Estimation of the Hurst parameter A standard wavelet-based methodology can be used for the estimation of the Hurst parameter [1]. Let $\psi_{j,k}(t) = 2^{-j/2}\psi_0(2^{-j}t - k)$ denote an orthonormal wavelet basis, derived from the mother wavelet ψ_0. The j index represent the *scale*: the larger j is, the more the wavelet is dilated. The k index is a shift in time.

For any (j,k), $d_X(j,k) = \langle \psi_{j,k}, X\rangle$ are called the *wavelet coefficients* of the stochastic process X ($\langle ., .\rangle$ is the inner product in the L^2 functional space). These wavelets coefficients enable a study of the process X at various times (values of k) and various scales (values of j). In particular, when X is a long-range dependent process with parameter H, the following limit behavior for the expectation of wavelet coefficients can be shown [1]:

$$\forall j, \quad \mathbb{E}(d_X(j,k)^2) \underset{j\to+\infty}{\sim} c2^{j(2H-1)} \tag{1.1}$$

Moreover, it can also be shown that the time averages S_j for each scale j (n_j is the number of wavelet coefficient available at scale j):

$$S_j = (1/n_j)\sum_{k=1}^{n_j} |d_X(j,k)|^2 \tag{1.2}$$

can be used as relevant, efficient and robust estimators for $\mathbb{E}(d_X(j,k)^2)$ [1]. From Eq. (1.1) and (1.2), the estimation of H is as follows: *i)* plot $\log_2 S_j$ versus $\log_2 2^j = j$ and *ii)* perform a weighted linear regression of $Log_2 S_j$ in the coarsest scales (see for instance Figure 1.2). These plots are commonly referred to as log-scale diagrams (LD). In such diagrams, LRD is evidenced by a straight line behavior in the limit of large scales. In particular, if the line is horizontal, then $H = 0.5$ and there is no long-range dependence.

To illustrate how we use this tool to evaluate the Hurst parameter, we provide in Figure 1.2 a typical log-scale diagram extracted from an Internet trace. Along the x axis are the different values of the scale j at which the process is observed. For each scale, $\log_2 S_j$ is plotted together with its confidence interval (vertical bars). The Hurst parameter can be estimated if the different points plotted are aligned on a straight line for large scales.

Synthesis of long-range dependent processes The synthesis (generation of sample paths) of long-range dependent processes is easy if the marginal law is Gaussian [4]. The so-called *Fractional Gaussian Noise* (FGN) is commonly used for that. However, if one wants to generate a long-range dependent process whose marginal law is non-Gaussian, the problem is more complex. The inverse method [35] only guarantees an asymptotic behavior of the covariance function. We have developed, for several common laws (exponential, gamma, χ^2, etc.), an exact method of synthesis described in [31]. We can thus produce synthetic long-range dependent sample paths that can be used in traffic generation. It is important to note that most elements of the transaction sequence of on-chip processor communications have non-Gaussian distributions. For instance, delay sequences rather exhibit an exponential distribution as we expect many small delays and few big ones. With our synthesis method, we can produce a synthetic exponential process with long-range dependence. Such non-gaussian and LRD models have been used for Internet traffic modelling [31] as well.

We have introduced the major theoretical notions useful for a precise modeling of the on-chip traffic. We will now adopt a more practical vision and explain how these statistical modeling notions can be used in a SoC simulation environment.

1.3 Traffic Modeling in Practice

This section provides our practical experience of on-chip processor traffic analysis and generation. We first provide some generic guidelines that can be useful for any project leader in charge of designing an MPSoC architecture. Then we present a particular experimental framework called MPTG for Multi-Phase Traffic Generator.

1.3.1 Guidelines for designing a Traffic Modeling Environment

Even though our experimental framework is closely related to the SystemC environment, it is possible to postulate several practical issues that any NoC prototyping environment will have to be confronted with. These issues should be taken into account before designing the environment itself because their absence will, at some point, break the efficiency of the traffic simulation process.

Simulation precision A first and important point concerns the precision of the simulation. Depending on the goal of the simulation several precision levels can be targeted. There is a great emphasis today on *transaction level modeling* (TLM) (sometimes called *programmers view*) which basically consists in a functional simulation of the MPSoC without any time information. This level is primarily used by application developers who seek mainly a very fast simulation. It should definitively be useless in a NoC prototyping environment: no precise network performance evaluation can be made at this level. Another possibility is to maintain a notion of global time in the simulation by stamping messages as proposed by Chandry and Misra for instance [6]. This level is sometimes referred to as TLM-T. Even if this level permits some performance indication, the NoC protocol is usually not precisely simulated, hence contention behavior cannot be detected.

If the simulation environment is intended to detect contention in the network, it should be *cycle accurate* at IP boundaries. In other words, a bit accurate and cycle accurate simulation of the computation itself is not necessary, but the low level protocol of the IP input/output must be emulated very precisely to model burst and cache behavior. The designer must be aware that this level of simulation implies a very slow simulation, mainly because each NoC transaction should be precisely simulated.

Note also that the behavior of caches must be simulated very precisely as small details of the cache protocol might have an important influence on the global on-chip traffic.

Trace analysis Another important issue is the power of the trace generation and trace analysis tool. All the results will be obtained via simulation traces. These traces can be huge, therefore an efficient and parameterizable trace generation tool is needed. Traces must be compressed dynamically. Parsing and analysis is also a critical treatment. If parser generator tools are used, the grammar expressing the trace syntax must be carefully written so as to generate an efficient parser [8].

As a trace can be generated with many different parameters (size of FIFOs, latency of communication, address of communication, etc), it should be easy to instrument the simulation platform to record any requested information. Using an existing trace format such as VCD (Value Change Dump) should be preferred.

Platform generation Describing the hardware of a MPSoC platform usually consists in connecting wires between existing IPs and deciding the memory mapping. It becomes very quickly intractable to do these connections by hand, wire by wires, whereas the platform designer thinks at a coarser grain: how many processors are connected to which router. The *top* system simulated *must* be generated by some in-house script adapted to the simulation environment. Scripting should be used to generate families of platforms in order to prototype different system architectures or different numbers of processors. Some kind of source language should be designed for high level platform description. Enabling the connection with the Spirit [7] IP description format should be realized somewhere in the environment.

Traffic analysis and synthesis flow Obviously, the heart of the prototyping environment are the traffic analysis and synthesis tools which are discussed in section 1.2. These tools should be as generic as possible. They are basically signal processing tools that should be independent of the NoC simulation environment.

The global NoC prototyping flow should be clearly stated as soon as possible. Experiments should also be carefully classified according to a clear experimental protocol so as to be able to recover a previous result that could possibly be incoherent with a future experiment. Finally, as in any experimental framework, reproducibility is mandatory.

1.3.2 Multi Phase Traffic Generation Environment

In this section we describe our multi-phase on-chip traffic generation environment called MPTG, its integration in the SocLib simulation environment and its key features.

Most recent traffic generation methods use stochastic models coupled with static traces or deterministic replay functionalities. Statistical analysis and synthesis of on-chip processor traffic is difficult because this traffic usually presents complex statistical behaviors that are difficult to reproduce.

Using traffic generators in a simulation platform involves several steps among which realistic traffic and integration are key points. We have developed our environment in the SocLib simulation environment [22]. SocLib is a library of open-source SystemC simulation models of IPs that can be interconnected through the VCI (Virtual Component Interface [21]) interface standard [2]. VCI is a point to point communication protocol depicted in Figure 1.4. The simulation models available in SocLib are described at cycle-accurate level or at transaction level. All our experiments have been done at cycle accurate level as precise information were needed for NoC contention prediction. To each simulation model corresponds a synthesizable model (not necessarily open source) that can be used for designing a chip. Examples of simulation models available in SocLib are: a MIPS R3000 processor (with its associated data and instruction cache), standard on-chip memories, DMA controller and several kinds of networks on chip.

Within the SocLib framework all components are connected via VCI ports to a NoC interconnection. A simple SocLib platform is represented in Figure 1.6. We used the DSPIN network on chip (an evolution of SPIN [20]) which uses wormhole and credits based contention control mechanisms. DSPIN uses a set of 4-port routers that can be interconnected in a mesh topology in order to provide the desired packet switched network architecture. The software running on the processors used in SocLib is compiled with the GNU GCC tool suite. A tiny open source operating system called *mutek* [27] is used when several processors run in parallel. This OS can handle multi-threading on each processor.

The global MPTG flow is depicted in Figure 1.3. It is composed of three main parts, described hereafter.

- **Reference trace collection**. This is the entry point of our MPTG flow. Since we follow a trace-based approach, we perform a simulation and we get a reference trace. As it will be explained afterwards our traffic analysis is independent of network latency, therefore the initial simulation can take place very rapidly on a platform with an ideal interconnect (fixed latency). It is important to understand that the same reference trace can be used for many platform simulations (various interconnections, IP placement, memory mapping, etc.) because with such an ideal interconnect we gather the intrinsic communication patterns of IPs. We therefore make the assumption that the behavior of IPs is not influenced by the latency of the network. For instance, the delay between transactions corresponds, up to a constant, to computation time in the IP which is not influenced by communication latency. We have carefully checked the validity of this assumption in our simulations as explained in section 1.3.3. In the last phase (design space exploration) the real latency of the network will be precisely simulated taking into account contention and IP placement in the SoC.
- **MPTG configuration**. The trace is then analyzed with a semi-automatic configuration flow which starts by parsing the trace file in order to extract the transaction sequence. Then we run our phase segmentation algorithm described in section 1.2.4 on each identified phase. According to the designer's choice for the models, each phase is then described and all parameters are estimated. At the end we obtain a MPTG configuration file such as the one reported in Figure 1.5. Note that if the designer fails to find an adequate stochastic model for the traffic, they can choose to replay the reference trace. In this case the trace can be compressed to save disk space.
- **Design space exploration**. The traffic generator component with the configuration file can now be used in place of real processors in order to evaluate the performance of various interconnect, IP placement, memory mapping, etc. This can be done faster because the simulation of the traffic generator requires much less resources than the simulation of a processor. One important point here is that designers can also evaluate the stability of a given architecture by investigating if small changes in the configuration files imply small or large changes in the performance.

A generic traffic generator has been written once for all for the SocLib environment. This traffic generator is used as a standard IP during simulations and provides a master VCI interface.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 in a replay of a recorded trace, or in a stochastic model with parameters adjusted by the fitting procedure. These traffic patterns can be described in sequence. These sequences will be used during next runs of the simulation. Figure 1.5 shows an illustration of such configuration. The entry point of a configuration is the sequencer part that will schedule the different phases of the traffic sequences. Each phase is then described in the file using its traffic shape and the associated packet size and address (destination among the IPs on the NoC).

Designer's choices made at this stage for the MPTG configuration can be categorized using the following points, also illustrated in the configuration file presented in Figure 1.5.

1. **Timing modeling**. We distinguish two types of placement of transactions in time as already mentioned in section 1.2.3. On one hand the designer can choose to model the delay $D(k)$ and on the other hand they can model the aggregated throughput time series $W_\delta(i)$. This choice depends on the context and purpose of the traffic generation. Using the aggregated throughput, one loses specific information concerning the time lag between transactions but the traffic load (on a scale of time exceeding the size of the window δ) is respected. If we choose accuracy over aggregated throughput then two sub-groups will be considered independent: addresses, orders and size ($A(k)$, $C(k)$, $S(k)$) on one hand and aggregated throughput ($W_\delta(i)$) on the other.

2. **Content modeling**. Once the time modeling for transactions has been decided, the designer must model the content of transactions (address, command and size). We have defined different types of modeling to handle different situations:

   - **Random**. In this mode each element (address, control, time and size) is random, hence independent of the others. This can be used for generating customizable random load on the network.

   - **Cache**. In this mode the size of the read requests is constant (equal to the size of a line cache). There is a mode for instruction cache mixed with data cache and a mode for data cache only.

- **Instruction cache**. This method is specific to an instruction cache. It contains instruction specificities, meaning that accesses are only read requests of the size of a cache line.

3. **Phase duration modeling**. A phase may appear several times in a trace, therefore it is necessary to characterize the size and number of transactions for each phase.

4. **Order of phases appearance**. This stage involves the configuration of the sequencer to choose the sequence of phases. It can basically play a given sequence of phases, or randomly shuffle them.

On top of the traffic content the MPTG must also define modes for memory access. Let us recall that one of the objectives of our traffic generation environment is to be able, from a reference trace collected with a simple interconnection, to generate traffic for a platform exhibiting an arbitrary interconnect. To do so, we have to prove that the communication scheme is not affected by the communication latency. From the point of view of the component it means that communications will be the same regardless of the latency of the interconnect. In the general case of a CPU with a cache, we cannot guarantee that, because the content of transactions ($A(k)$, $C(k)$ and $S(k)$ series) may be affected by the latency of the network. This is especially due to the presence of the write buffer. The behavior of such buffer may, in some cases, cause modifications in the size of transactions sent in the network, depending on the latency, especially in the case of large sequences of consecutive writes (zero initialization of a portion of the memory for instance).

This is why we use the time $D(k)$, which relates to the receipt of the request (so that it is independent of the network latency), instead of the time between two successive transactions. However, the problem of the recovery of calculations and communications remains. We must be able to determine if the delay $D(k)$ is a time during which the component is awaiting a reply (the component is blocked waiting for the response), or if it is a time during which the component keeps running, and thus may produce new communications. This led us to define different operating modes for the traffic generator, described hereafter.

- **Blocking requests**. In this mode, regardless of the order, the traffic generator emits a burst of type $C(k)$ to address $A(k)$, and of size $S(k)$ bus-words. Once the response is received, then the traffic generator waits $D(k)$ cycles before issuing the next transaction ($T(k+1)$) on the network. This characterizes a component which is blocked (pending) when making a request.

- **Non-blocking requests**. In this mode, regardless of the order, the traffic generator emits a burst of type $C(k)$ to address $A(k)$ with a size of $S(k)$ words. Once the $S(k)$ words have been sent, the traffic generator restarts after $D(k)$. Upon receipt of the answer, if $D(k)$ is in the past, then the next request is sent immediately, otherwise we wait until $D(k)$ is reached. This allows to model a data-flow component (hardware accelerators, for example) that is not blocked by communications. It is likely to be used for processor traffic. We included it for sake of generality.

- **Blocking / non-blocking read and write**. We can also specify more precisely if read transactions and/or write transactions are blocking or not. For example, a write-through cache is not blocked by writes (the processor keep on running). However, a processor reading blocks must wait before continuing its execution. The mode "non-blocking writes, blocking reads" acts as a good approximation of the behavior of a cache.

- **Full data-flow mode**. In order to emulate the traffic of data flow components we have finally established a communication mode in which only the requests are considered (the arrival of the answer is not taken into account). In this mode, the traffic generator issues a request, waits $D(k)$ cycles, and makes the following request, without concern for the arrival of the answer.

The definition of these operating modes allows us, without loss of generality, to be able to deal with different types of SoC platforms.

Key features of the MPTG environment As a summary, we recall the key features of the MPTG environment hereafter.

- MPTG is a fully integrated, fast and flexible NoC performance evaluation environment.
- It includes many traffic generation capacities, from trace replay to the use of advanced stochastic processes models.
- From an initial trace obtained with the simplest interconnection, a configuration file can be used extensively in order to evaluate the performance of any interconnection under realistic traffic patterns.

1.3.3 Experimental analysis of NoC traffic

In this section, we present some experimental results on processor traffic analysis and generation. We study simulation speedup, phase decomposition and the presence of long-range dependence.

Speed up In order to evaluate the speed up of using a traffic generator instead of real IPs, we built several platforms with different number of processors and different network sizes. The results are reported in Table 1.3. We also compare our speed-up factor with a traffic generation environment [19] that performs smart replay of a recorded trace.

The reference simulation time used for speedup ("S" columns) computation is the "Mips without VCD" (processor simulation without recording the VCD trace file) configuration. The speedup factor for "Mips with VCD" is less than one because recording the trace takes a fair amount of time. The simulation speedup is never greater than 2.27, which is obtained with no interconnection ("0x0" mesh). However, the speedup increases with the number of processors meaning, as expected, that large platforms will benefit more from traffic generators speedup than small ones. One can further note that the VCD recording impact decreases for large platforms and even becomes negligible for a "4x4" mesh. Generation of stochastic processes ("sto." and "lrd" lines) does not have a big impact on simulation speedup, meaning that reading values from a file and generating random numbers (even LRD processes) is almost equally costly in terms of computation time. The impact of the number of phases is also very small.

Speedup factors obtained are of the same order of magnitude as the ones obtained by Mahadevan in [19], and are quite small. The fact is that most of the simulation time is spent in the core simulation engine and in the simulation of the interconnection system, which cannot be reduced. Note that our conclusion is opposite to [19] which claims a noteworthy speedup factor. On the contrary, we found the speedup is too small to be useful for designers and we believe that the real interest of a traffic generation environment relies in its flexibility (various generation modes, easy to configure, etc.). This will be illustrated in the following paragraphs.

Simulation set-up For the initial trace collection, we use a simple platform (see Figure 1.6) in order to truly characterize the traffic of the triplet (implementation / processor / cache). If we study communications on a more complex platform, the traffic of the processor is influenced by other IPs and network-on-chip configuration (topology, routing protocol, etc.) as already discussed in section 1.3. This simple platform includes a MIPS r3000 processor (associated with instruction and data cache), directly connected to a memory holding all necessary data. Applications with input stimuli are reported in Table 1.5.

Next, in order to validate our environment, we run simulations on a more realistic platform shown on Figure 1.7. This platform includes five memories, a terminal type (TTY) as output peripheral, a MIPS processor executing the application and a background traffic generator (BACK TG) used to introduce contention in the network. During design space exploration, the MIPS processor is replaced by a traffic generator producing traffic fitted to the reference trace. The simulation of the platform of Figure 1.7 (with the MIPS processor) is only used to check that the MPTG traffic corresponds precisely to the MIPS traffic.

Multi-phase We have processed each traffic trace with the segmentation algorithm described in section 1.2.4, using delay as the representative element and for different number of phases (k). The size of intervals is set to $L = 5000$ transactions. The choice of k is a trade-off between statistical accuracy (we need a large interval for statistical estimators to converge) and phase grain (we need many intervals in order to properly identify traffic phases). Figures 1.8b, 1.8c and 1.8d show the results for various numbers of phases. One can see that the algorithm finds the analogy between the two frame processings, and identifies phases inside each one of them. The segmentation appears to be valid and pertinent. The segmentation is done with mean and variance as representative vectors, so we expect that each identified phase is stationary, likely to be processed by a stochastic analysis.

Long range dependence We illustrate here how the presence of LRD in embedded software can be evidenced. To this end, we compute aggregated throughput time series as the number of flits sent in consecutive time-windows of size 100 cycles. This time scale allows for a fine grain analysis of the traffic. For each application, we comment hereafter the log-scale diagrams presented in Figures 1.9, 1.10 and 1.11.

- **mpeg-2** (see Figure 1.9). The shape of the LD does not exhibit evidence for LRD. Indeed the estimated value for the Hurst parameter, 0.56 indicates LRD is not present in the trace ($H = 0.5$ means no LRD). In this case, an IID (Independent Identically Distributed) process would be a good approximation of the traffic. One can note a peak around scale 2^5, meaning that a recurrent operation with this periodicity is present in the algorithm which might have an impact on network contention. Such behavior could be captured by an ARMA process for instance [5]. It is interesting to note that this software implementation of MPEG-2 does not exhibit long range dependence whereas the hardware implementation does [35].

- **mp3** (see Figure 1.10). Similar to the MPEG-2 implementation, no trace of LRD can be found in this case: the estimated Hurst parameter is close to 0.5. The other parts of the communication trace, do not manifest any long-range dependence either.

- **jpeg2000** (see Figure 1.11). For this application, the traffic trace exhibits a strong non-stationarity, so that the trace must be split in rather short parts for the analysis. In some of these parts, corresponding specifically to the *Tier-1* entropic decoder of the JPEG2000 algorithm, LRD is present with an estimated Hurst parameter value between 0.85 and 0.92 (depending on the parts). In the other parts of the algorithm, no long-range dependence could be evidenced.

We can conclude from these experiments that long-range dependence is not an ubiquitous property of the traffic produced by a processor associated with a cache executing a multimedia application. In some parts of JPEG2000 LRD is present, however it is combined with periodicity effects which may have an equivalent impact on the network-on-chip performance. In this case, short-range dependent models such as ARMA [5] could be used instead of LRD ones.

1.3.4 Traffic modeling accuracy

In this section we investigate ways of evaluating the accuracy of a traffic generator. It essentially consists in defining indicators of the difference between the traffic trace obtained from the simulation with processors and the traffic trace obtained with traffic generators. It is clear that one should not look at global metrics such as the average delay or the average throughput. This would not highlight the interest of the multi-phase approach. As such, we define an accuracy measure by computing the mean evolution of each transaction's element (delay, size, command and throughput). The mean evolution is defined as the average value of the series, computed in consecutive time windows of size L.

To summarize those results we define 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 element for the reference simulation. Further, let $M(i)$ be the evolution of the same element for another simulation, and finally let n be the number of points of both functions. The error (in percent) is: $Err = \frac{1}{n} \sum_i |M_{ref}(i) - M(i)|/M_{ref}(i) * 100$. Note that this is a classical signal processing technique to evaluate the distance between two signals. Furthermore, we define the cycle error as the relative difference between numbers of simulated cycles.

In order to illustrate those metrics, Table 1.4 shows accuracy results on the NOC platform and the execution of the MP3 application.

As expected, the higher the phase number is, the more accurate the simulations are. In particular, the error on latency becomes very low when the number of phase is greater than one. This is of major importance since the latency of communications reflects the network state. It means that the traffic generation from a network performance point of view is satisfactory with multi-phase traffic generation. Multi-phase traffic generation provides an interesting trade-off between deterministic replay and random traffic.

1.4 Related work and conclusion

SoC design companies usually have in-house NoC prototyping environments but it is very difficult to have information about this tools. Concerning academic research, there are several works on NoC performance analysis and design that use deterministic traffic generated (trace replay) [11, 17, 19]. For instance, the TG proposed in [19] uses a trace compiler that can generate a program for a reduced instruction set processor that will replay the recorded transactions in a cycle-accurate simulation without having to simulate the complete processor. This TG is sensitive to the network latency: changing network latency will produce a similar effect on the TG as on the original IP. This is a very important point which is also taken into account in our environment.

An alternative solution for NoC performance analysis is to use stochastic traffic generators, this technique is used in many environments [15, 26, 34, 36]. None of these works proposes a *fitting procedure* to determine the adequate statistical parameters that should be used to simulate traffic. Recently, the work presented in [33] studies a LRD on-chip traffic model. To our knowledge, no NoC traffic study has introduced multi-phase modeling. A complete traffic generation environment should integrate both deterministic and stochastic traffic generation techniques. From the seminal work of Marculescu et al. [35], long range dependence is used in on-chip traffic generators [12]. Marculescu et al. have isolated a long-range-dependent behavior in the communications between different parts of a hardware MPEG-2 decoder at the macro-block level.

Rapid NoC design is a major concern for next generation MPSoC design. In this field, processor traffic emulation is a real bottleneck. This chapter has investigated many issues related to the sizing of NoC. In particular it insists on the fact that a serious statistical toolbox must be used to generate realistic traffic patterns on the network.

References

[1] P. Abry and D. Veitch. Wavelet analysis of long-range dependent traffic. *IEEE Trans. on Info. Theory*, 44(1):2–15, January 1998.

[2] VSI Alliance. Virtual component interface standard. On line: `http://www.vsi.org/library/specs/summary.html`, April 2001.

[3] J. Archibald and J. L. Baer. Cache coherence protocols: Evaluation using a multiprocessor simulation model. *ACM Transactions on Computer Systems*, 4:273–298, November 1996.

[4] J. M. Bardet, G. Lang, G. Oppenheim, A. Philippe, and M. S. Taqqu. *Long-Range Dependence: Theory and Applications*, chapter Generators of long-range dependent processes: A survey, pages 579–623. Birkhäuser, 2003.

[5] P. J. Brockwell and R. A. Davis. *Time Series: Theory and Methods, 2nd edn.* Springer Series in Statistics, New York, 1991.

[6] K. M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Trans. Software Eng.*, 5(5):440–452, 1979.

[7] The SPIRIT consortium. Enabling innovative ip re-use and design automation. On line: `http://www.spiritconsortium.org/`, 2008.

[8] K. D. Cooper and L. Torczon. *Engineering a Compiler.* Morgan Kaufmann, 2004.

[9] A. Erramilli, O. Narayan, and W. Willinger. Experimental queueing analysis with long-range dependent packet traffic. *ACM/IEEE transactions on Networking*, 4(2):209–223, 1996.

[10] B. Calder et al. Simpoint. On line: `http://www.cse.ucsd.edu/~calder/simpoint/`, April 2001.

[11] N. Genko, D. Atienza, G. De Micheli, J. M. Mendias, R. Hermida, and F. Catthoor. A complete network-on-chip emulation framework. In *DATE 05*, pages 246–251, 2005.

[12] A. Hegedus, G.M. Maggio, and L. Kocarev. A ns-2 simulator utilizing chaotic maps for network-on-chip traffic analysis. In *ISCAS*, pages 3375– 3378, May 2005.

[13] R. Jain. *The Art of Computer Systems Performance Analysis.* John Wiley & Sons, 1991.

[14] R. H. Katz, S. J. Eggers, D. A. Wood, C. L. Perkins, and R. G. Sheldon. Implementing a cache consistency protocol. In *Proceedings of the 12th annual international symposium*

on Computer architecture, pages 276–283. IEEE Computer Society Press, 1985.

[15] K. Lahiri, A. Raghunathan, and G. Lakshminarayana. LOTTERYBUS: A new high-performance communication architecture for system-on-chip designs. In *Design Automation Conference*, pages 15–20, 2001.

[16] W. E. Leland, M. S. Taqqu, W. Willinger, and D. V. Wilson. On the self-similar nature of ethernet traffic (extended version). *ACM/IEEE transactions on Networking*, 2(1):1–15, February 1994.

[17] M. Loghi, F. Angiolini, D. Bertozzi, L. Benini, and R. Zafalon. Analyzing on-chip communication in a mpsoc environment. In *DATE 04*, page 20752, 2004.

[18] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[19] S. Mahadevan, F. Angiolini, M. Storgaard, R. Grøndahl Olsen, Jens Sparsø, and Jan Madsen. A network traffic generator model for fast network-on-chip simulation. In *DATE 05*, pages 780–785, 2005.

[20] MEDEA+. *SPIN: A scalable Network on Chip*, November 2003.

[21] OCP-IP. On line: `http://www.ocpip.org/socket/ocpspec/`, 2001.

[22] Computer Science Laboratory of Paris IV. Soclib simulation environment. On line: `http://soclib.lip6.fr/`, 2006.

[23] K. Park and W. Willinger, editors. *Self-Similar Network Traffic and Performance Evaluation*. John Wiley & Sons, 2000.

[24] V. Paxon and S. Floyd. Wide-area traffic: The failure of poisson modeling. *ACM/IEEE transactions on Networking*, 3(3):226–244, June 1995.

[25] 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.

[26] Santiago Gonzalez Pestana, Edwin Rijpkema, Andrei Rădulescu, Kees Goossens, and Om Prakash Gangwal. Cost-performance trade-offs in networks on chip: A simulation-based approach. In *DATE 04*, page 20764, 2004.

[27] F. Pétrot and P. Gomez. Lightweight implementation of the posix threads api for an on-chip mips multiprocessor with vci interconnect. In *DATE 03 Embedded Software Forum*, pages 51–56, 2003.

[28] A. Scherrer. *Analyses statistiques des communications sur puces.* PhD thesis, ENS Lyon, LIP, France, December 2006.

[29] A. Scherrer, A. Fraboulet, and T. Risset. Automatic phase detection for stochastic on-chip traffic generation. In *CODES+ISSS*, pages 88–93, Séoul, South Korea, October 2006.

[30] A. Scherrer, A. Fraboulet, and T. Risset. Generic multi-phase on-chip traffic generator. In *ASAP*, pages 23–27, Steamboat Springs, USA, September 2006.

[31] A. Scherrer, N. Larrieu, P. Borgnat, P. Owezarski, and P. Abry. Non gaussian and long memory statistical characterisations for internet traffic with anomalies. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 4(1):56–70, 2007.

[32] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, and B. Calder. Discovering and exploiting program phases. *IEEE Micro*, 23(6):84–93, 2003.

[33] V. Soteriou, H. Wang, and L. S. Peh. A statistical traffic model for on-chip interconnection networks. In *International Conference on Measurement and Simulation of Computer and Telecommunication Systems (MASCOTS '06)*, September 2006.

[34] R. Thid, M. Millberg, and A. Jantsch. Evaluating NoC communication backbones with simulation. In *21th IEEE Norchip Conference*, Riga, November 2003.

[35] 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.

[36] D. Wiklund, S. Sathe, and D. Liu. Network on chip simulations for benchmarking. In *IWSOC*, pages 269–274, 2004.

PDF	Description
Gaussian	The most widely used PDF, used for aggregated throughput for instance
Exponential	Fast decay PDF, used for delay sequence for instance
Gamma	Gives intermediate PDF between exponential and Gaussian
Lognormal	Gives asymmetric PDF
Pareto	Provides *heavy-tailed* PDF (slow decay)

Table 1.1: Some classical probability distribution functions (PDF)

Covariance	Description
IID (independent identically distributed)	No memory
ARMA (auto-regressive moving average)	Short range dependence
FGN (fractional Gaussian noise)	Long range dependence
FARIMA (fractional integrated ARMA)	Both short and long range dependence

Table 1.2: Some classical covariance functions

Number of processors	1		2		3		4	
Mesh size	0x0		2x2		3x3		4x4	
	Time	S	Time	S	Time	S	Time	S
Mips without VCD	36.1	1	249.5	1	477.5	1	1261.3	1
Mips with VCD	59.4	0.61	279.9	0.89	559.8	0.85	1263.8	0.95
MPTG replay	15.9	2.27	177.2	1.41	344.5	1.39	804.0	1.57
MPTG 1 phase (sto.)	19.9	1.82	177.4	1.41	337.1	1.42	790.8	1.59
MPTG 10 phase (sto.)	19.9	1.81	177.2	1.41	337.6	1.41	790.0	1.60
MPTG 1 phase (lrd)	28.8	1.25	180.2	1.39	364.1	1.31	806.5	1.56
MPTG 10 phase (lrd)	29.1	1.24	184.0	1.36	341.5	1.40	826.4	1.53
Mahadevan [19]	-	2.15	-	2.64	-	2.60	-	3.05

Table 1.3: Speed up of the simulation: simulation time in seconds and speedup factor (S columns) for various platforms and traffic generation schemes.

Config.	Delay	Size	Cmd	Throughput	Latency
replay	1.153	0	0	0.197	0.117
random	41.278	75.242	7.709	102.316	27.825
1 phase	18.604	14.759	6.256	12.696	10.086
3 phases	17.194	8.169	3.255	6.212	0.767
5 phases	14.772	3.239	1.210	5.651	0.626

Table 1.4: Accuracy of MPTG: error (in percent) on various metrics with respect to the reference MIPS simulation (NOC platform).

App.	Input
MPEG-2	2 images from a clip (176×144 color pixels)
MP3	2 frames from a sound (44,1 kHz, 128 kbps)
JPEG2000	"Lena" picture (256x256)

Table 1.5: Inputs used in the simulations

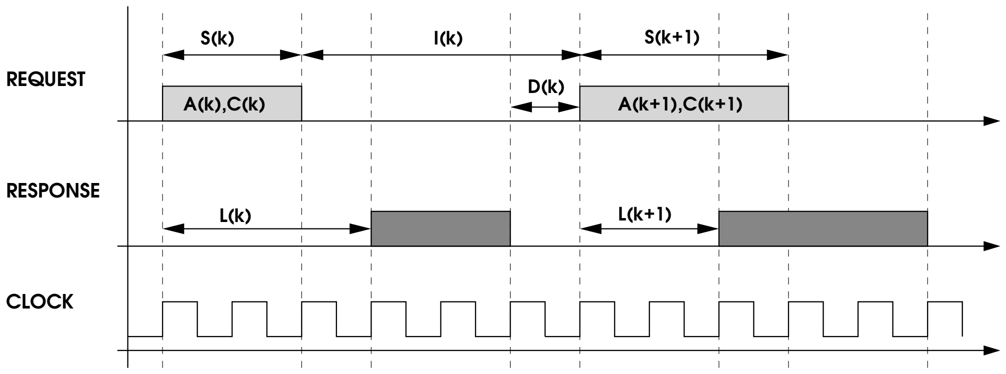


Figure 1.1: Traffic modeling formalism: $A(k)$ is the target address, $C(k)$ the command (read or write), $S(k)$ is size of transaction, $D(k)$ is the delay between the completion of one transaction and the beginning of the following one, and $I(k)$ is the inter-request time.

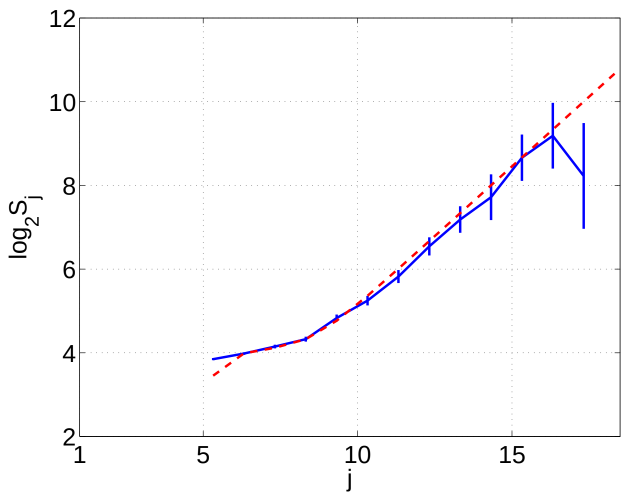


Figure 1.2: Example of log-scale diagram (LD), the Hurst parameter is estimated with the slope of the dashed line (here, $H = 0.83$)

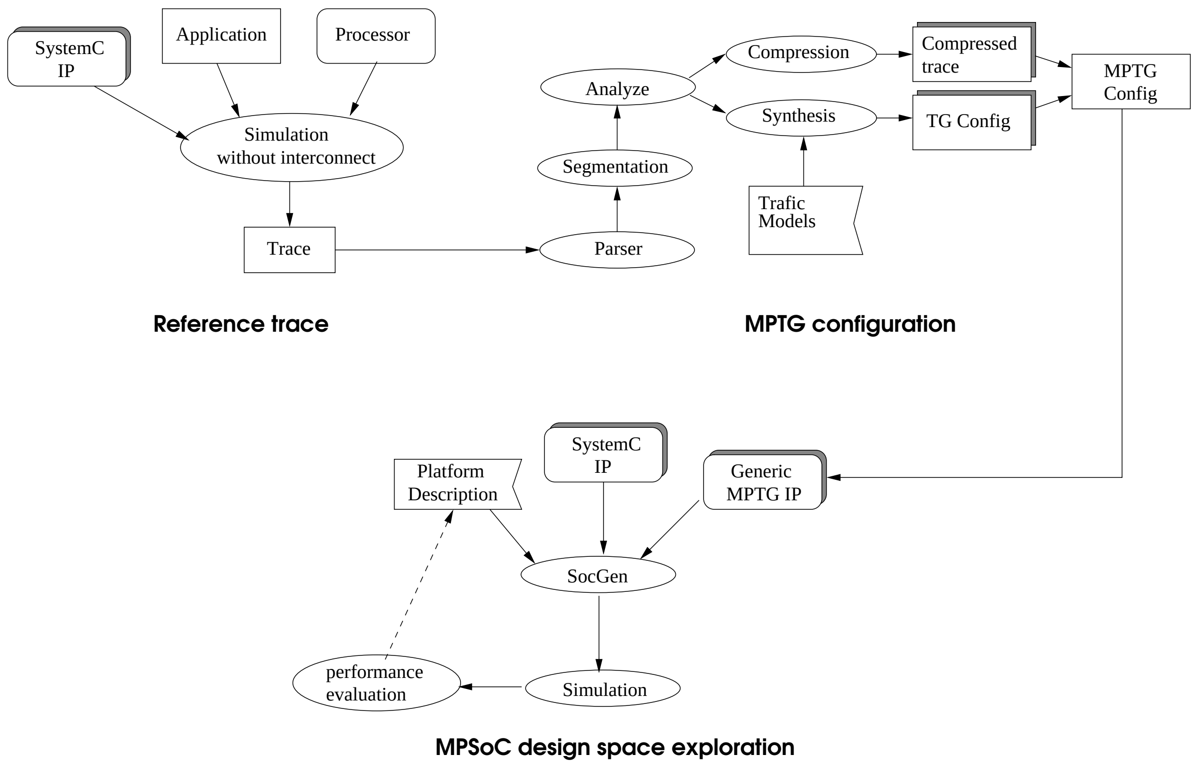


Figure 1.3: MultiPhase traffic generation flow: An initial trace is collected by simulation with ideal interconnection, The trace is then analyzed and segmented to generate a configuration of the MPTG, then the real simulation can take place.

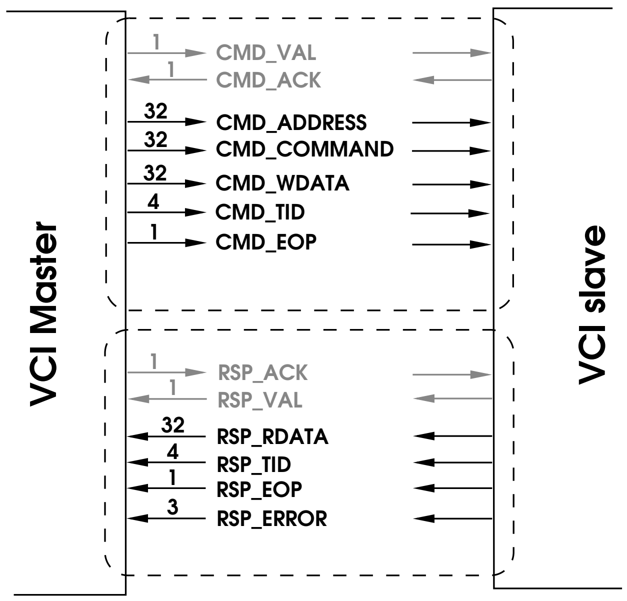


Figure 1.4: Example of NoC interconnect interface: advanced VCI, defined by the OCP consortium

```
phase0{
 time:         // transaction temporal caracteristics
     mode=IA;  // selected precision is  Delay
     exponential(15);
               // IID process following an exponential law of mean 15 flits
 content:      // transaction content caracteristics
     random("ptable",exponential(10));
               // random generation,
               // the ptable file contains destination addresses probability table
               // size is modeled with an exponential law of mean 10 flits
 duration:     // phase duration
     constant(10000); // 10000 transactions
}

phase1{
 time:         // transaction temporal caracteristics
     mode=TP;  // selected precision is throughput
     deterministic("fic");
               // trace is replayed from a previously recorded one.
 content:      // transaction content caracteristics
     cache("ptable",exponential(16));
               // cache type generation (non-blocking writes)
               // size is modeled with an exponential law of parameter 16
 duration:     // phase duration
     deterministic("fic"));
               // phases duration are store in a file
}

sequencer{     // phase switch behavior
  round(10);   // round robin, repeated 10 times
}
```

Figure 1.5: MPTG configuration file example

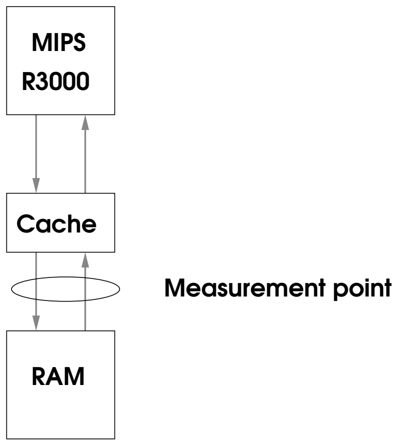


Figure 1.6: Simulation platforms for initial trace collection.

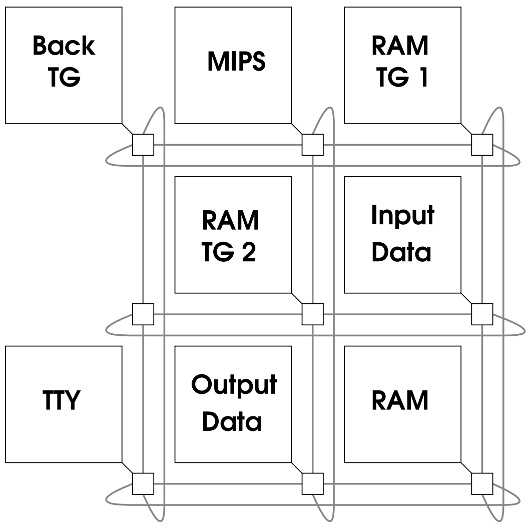
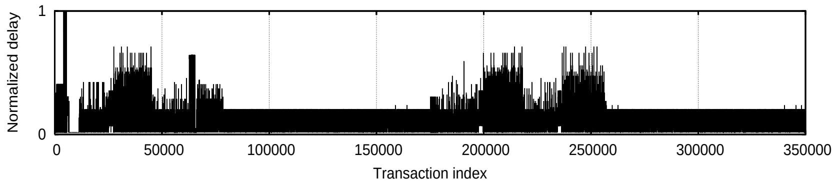
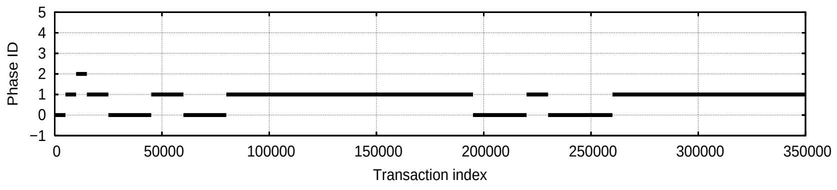


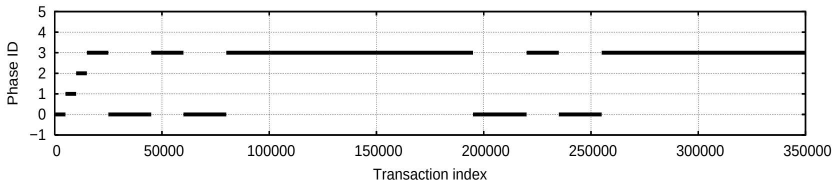
Figure 1.7: Simulation platforms for MPTG validation including five memories, a terminal type (TTY), a MIPS processor and a background traffic generator (BACK TG) used to introduce contention in the network



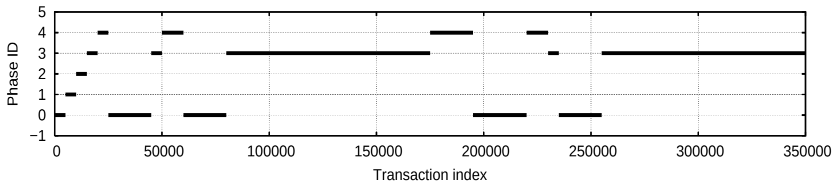
(a) original trace



(b) 3-phases clustering



(c) 4-phases clustering



(d) 5-phases clustering

Figure 1.8: Phases discovered by our algorithm on the MP3 traffic trace using the delay, for different phase numbers.

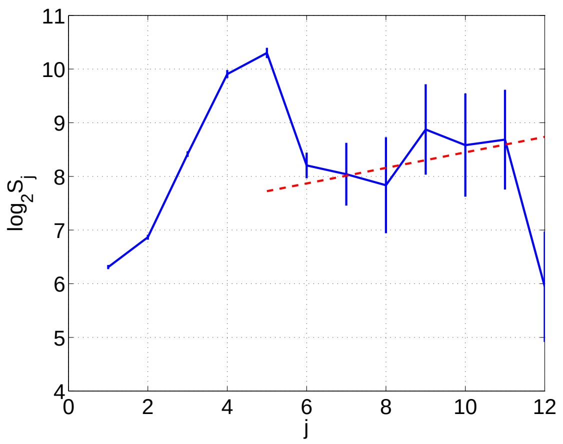


Figure 1.9: LD of the traffic trace corresponding to the MPEG-2 and MP3 implementations. $\hat{H} = 0.56$.

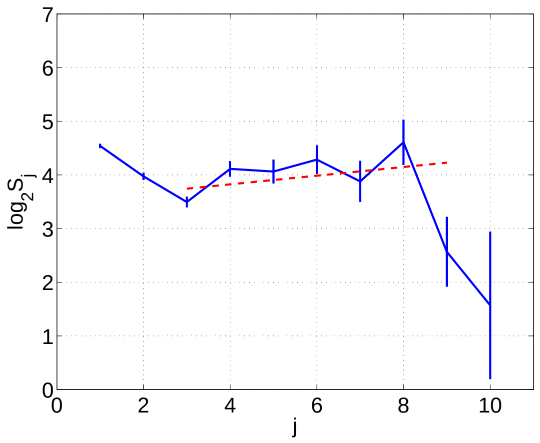


Figure 1.10: LD of the traffic trace corresponding to the MP3 implementation. $\hat{H} = 0.58$

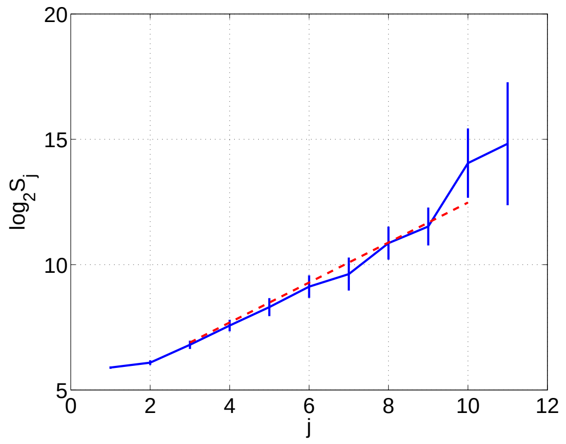


Figure 1.11: LD of the traffic trace corresponding to the JPEG2000 implementation. $\hat{H} = 0.89$