Flexible Bus and NoC Performance Analysis with Configurable Synthetic Workloads

Rikard Thid, Ingo Sander, Axel Jantsch
Royal Institute of Technoogy
Stockholm, Sweden
{thid, ingo, and axel}@imit.kth.se *

Abstract

We present a flexible method for bus and network on chip performance analysis, which is based on the adaptation of workload models to resemble various applications. Our analysis method assists in the selection of a communication infrastructure early in the design process. The method uses (1) synthetic workload models which are similar to timed Petri nets and (2) the b-model for self-similar workloads. This allows the exploration of larger portions of the design space than possible with traditional stochastic models. The method is illustrated with tutorial examples where both a NoC and a bus based platform are analyzed.

Keywords - NoC, Bus, Performance Analysis, Simulation, Benchmark, Synthetic Workloads

1 Introduction

Benchmarks are important tools to estimate the performance of various systems and components. Accurate architectural performance analysis of Networks on Chip (NoC) requires the right combination of relevant workload models, the right metrics, and model of interfering traffic. In this context workload models define the communication that is simulated and measured directly during performance analysis. Workload models can be defined at different abstraction levels. CPU benchmarks are often very exact while many network workloads use abstract stochastic processes. No matter what abstraction level is used, a benchmark will only reflect the performance under a particular workload. This is very important to keep in mind since benchmarks are often used when a platform is selected for a specific application. The more the application differs from the benchmark workload, the more unsound such a method is. Instead, we propose that high-level models which can be configured to resemble the target application are used in such cases. Since we are interested in the performance of NoCs, which are likely to run multiple parallel tasks, it is very important to be able to model the interfering traffic in a flexible way as well.

Metrics are the characteristics that a benchmark measures. Preferably, the metric is expressed in a unit that is in itself an important property of the system. For instance, benchmark runtime is the best way to measure how fast a microprocessor system is. However, in some benchmarks properties that are of secondary importance are measured. In microprocessor benchmarks, measuring the memory latency does not necessarily give a comprehensive view of how the system acts since it is not the only contributing factor to microprocessor performance. In the same way, network characteristics such as bandwidth and latency are not the only factors that influence a NoC applications performance.

Recent research emphasizes the impact that *variability* of workloads has on evaluation of microprocessor performance [1]. While microprocessors are deterministic, small changes in the initial state of the surrounding system (such as interfering peripheral devices, task schedule) may have a large impact on execution time. Variability is an emerging problem in multi-threaded applications where different tasks share resources. Comparisons between simulations with different settings may lead to incorrect conclusions if the variability of the results is high and dominates differences. The authors argue that instead of comparing single simulations, several simulations (with slightly different boundary conditions) should be used and the average runtime should be the performance metric while confidence intervals indicate the magnitude of variability.

One major research challenge is that we do not know exactly what the future applications that will use NoCs are, how they will be decomposed into subcomponents, and how the communication infrastructure will be used. In spite of

*The authors would like to thank S. González Pestana and K. Gossens at Philips Research for many discussions and cooperation during an earlier phase of this work.

this, many NoC communication infrastructures have been developed and now there is a need to evaluate these and find out how they perform under various scenarios. The major benefit with our approach is that we do not rely on traces of real applications. Instead, we use synthetic workloads which are easily modifiable in terms of burstiness and topology. This allows us to try a wide range of configurations and learn what the best design patterns are for different NoCs.

We present a method for performance analysis that is:

- Configurable in topology
- Investigating a wide rage of bursty behaviour
- Measuring simulated runtime
- Addressing the variability

The remainder of this paper is organized as follows. Section 2 presents previous research in NoC performance estimation and benchmarks in adjacent fields. Section 3 explains how our performance analysis method works. In Section 4, our method is used in order to compare a bus with a NoC infrastructure and Section 5 draws final conclusions.

2 Related work

This section summarizes related work regarding performance analysis from NoCs and adjacent areas from which much can be learned.

Analyzing performance on the Internet is an interesting but difficult topic. The difficulty lies in the ever changing topology and set of applications on the Internet. The Internet Engineering Task Force (IETF) is a joint effort that has defined metrics and a methodology that helps to standardize the performance measurement process for the Internet [9]. Performance is normally measured in terms of throughput and latency since they have direct impact on most Internet applications.

When networks of more limited sizes such as local area networks and multiprocessor interconnection networks are benchmarked *Traffic patterns* may be used. A Traffic Pattern [3] is a graph that describes the spatial distribution of traffic in the network. In general, they focus on the amount of traffic that is to be transmitted and not on the dependencies between them.

Microprocessors are benchmarked with the runtime for different applications. MiBench [5] and Mediabench [2] contain applications that are representative for media applications. Dhrystone [15] is a synthetic benchmark that represents not a specific but an average of applications. Multiprocessor computers are used for more computation-heavy applications which for instance the SPLASH-2 benchmarks [17] reflect.

Skadron et al. [12] argue that benchmarks that are used to evaluate computer architectures do not give an accurate view of performance in emerging application areas such as embedded systems, mobile computing, and real-time systems. The solution to the performance analysis problem for an ever increasing design space is not to throw even more benchmarks at it. Instead the authors propose synthetic benchmarks that are parametrized to model a range of different behaviors. We have allowed these insights to influence our work this paper describes.

A Petri net [10] is a graphical and mathematical representation of distributed systems. Petri nets consist of *places*, *transitions*, and *arcs* that interconnect them. Places store *tokens* that express the state that the model is in. It is possible to include time in the simulation model with a static delay or a stochastic delay before the transition fires. The main benefit with timed Petri nets is that they model both *qualitative* behaviour of timing (dependence, throttling) and *quantitative* properties (delay in single subcomponents).

It is well known that self-similar burstiness occur in many areas such as LAN traffic [7], disk I/O, and CPU-to-memory communication [13]. Self-similar workloads are not only interesting because they occur in many real world applications, they also behave very different in comparison to simpler models such as Poisson-processes. Therefore, incorporation of self-similarity in workload models is essential for NoC performance analysis. An effective method to create self-similar workload models is described in [14].

Lahiri et al. [6] present an efficient performance analysis technique for bus-based SoC architectures. Low-level resource models are abstracted away with a graph that expresses the behaviour of the application. Mahadevan et al. [8] use a similar approach. A graph representation of the data-dependencies and computational delays is extracted from benchmarks and this graph is used as a basis for simulation of the very same benchmark on different communication infrastructures with little loss of accuracy but at high speed. While both these approaches speed up the actual run time for simulations, the novelty of our method is the configurability of the workloads spatial distribution and self-similar burstiness.

The work that has been done on NoC performance analysis has focused on either measuring plain latency and throughput of statistical processes [4, 16], or using benchmarks from other areas such as multicomputers [18]. To our knowledge, no other NoC benchmarks that allow modeling of self-similar burstiness have been presented until now.

3 Method

This section describes our approach that has the advantages previously mentioned. We describe how the different subelements in a model are interconnected through a *com-*

munication graph, how *tokens* are sent in the model and how *burstiness* is modeled using the b-model. Finally, two workload patterns are introduced.

3.1 Communication graph

We describe the *spatial* composition of the workload-model with a graph where the edges represent *workload elements* and the arches define channels over which all inter-process communication occurs. Each channel is simplex so bidirectional communication is modeled with two channels. Either a resource element is *active*, which means that it emits tokens so that they represent the generated b-model, or it is *reactive*, responding to incoming tokens after a delay decided by the b-model. It is possible to model sinks with reactive elements that are configured to receive but not send tokens.

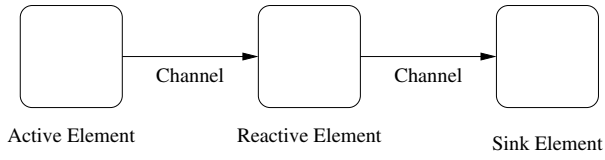


Figure 1. A simple communication graph.

3.2 Tokens

The benchmarks work together with a NoC or a bus simulator (hereafter called "interconnect simulator") as depicted in Figure 2. Tokens are sent between workload elements through a interconnect simulator. In the boundaries between workload and interconnect simulator, tokens are converted to transactions and vice versa. In our simulation setup, a token corresponds to a 64-bit transaction but may be refined differently in other interconnects. For example, if a 32-bit bus is benchmarked one token would be represented by two bus transactions.

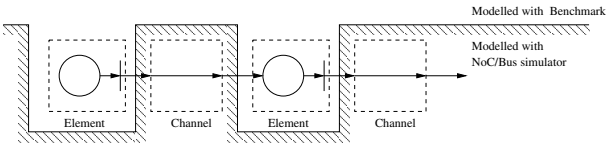


Figure 2. The channels in the workload model are executed in an interconnect simulator.

3.3 Timing

Our approach is inspired by timed Petri nets [10]. In order to model bursty workloads with self-similarity, each token is delayed with a time that is calculated with the b-model [14]. This separates our approach from conventional timed petri nets. The b-model is very simple and can efficiently generate realistic traces of self-similar and bursty data. It requires only one parameter to describe the level of burstiness.

The b-model initially assumes that the level of activity is evenly distributed over time. The total time is divided in two equally long blocks which are given an amount of the total activity specified by b and $(1-b)$ respectively. The part that is given the larger portion is decided by a random number generator for which each resource element has its own unique seed. This procedure is repeated on the blocks recursively until the block-size equals the time resolution. Figure 3 depicts how the level of activity over time is changed for every recursion step.

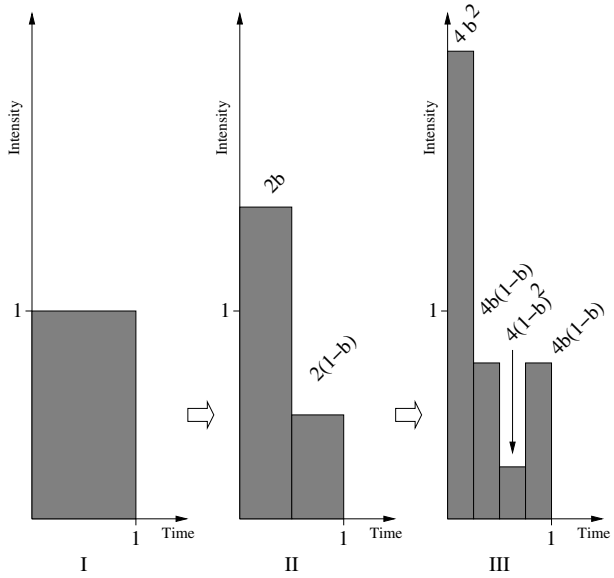


Figure 3. The b-model is created by gradually dividing a block (I) into subblocks(II), which are also divided into subblocks(III).

A trace of an active element is shown in Figure 4. The b parameter has been selected to three different values. A b parameter of exactly $1/2$ generates uniform traffic. If the b parameter is slightly more than $1/2$, the traffic looks like it has been generated with a Poisson process with burstiness on smaller timescales primarily.

One parameter that quantifies the burstiness of a process is the *Hurst parameter* which varies between $1/2$ and 1. In

the b-model the b parameter has the following relation with the Hurst parameter: $H \approx \frac{1}{2} - \frac{1}{2} log_2(b^2 + (1-b)^2)$. A small selection of typical b parameters and their corresponding Hurst parameters are shown in the following table:

b parameter	Hurst parameter
0.5	1
0.7	0.89
0.9	0.64
1	0.5

When larger timescales are studied, the process looks smoother. This is typical for simple stochastic processes and they are called short range dependent. A higher parameter means that burstiness occurs on more than one timescale. Many processes found in real world applications have a Hurst-value near 0.7. This is called the *Hurst effect* and it can be modeled with a b parameter near 0.86. A detailed explanation of this phenomenon and its consequences for network modeling is given in [7].

It is only the duration of tokens in the resource elements that is modeled with the b model, note that the communication is modeled with a cycle-level network simulator.

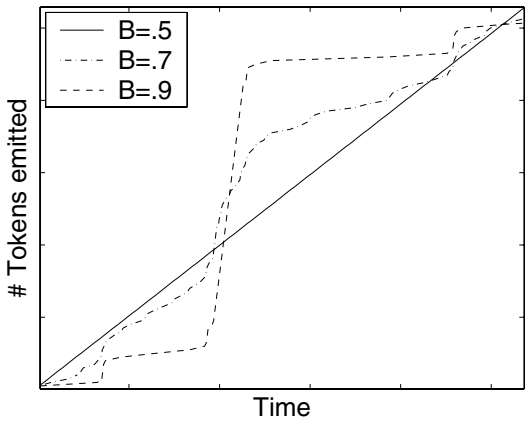


Figure 4. The number of tokens that are emitted by an active element over time. The three different graphs have different b parameters.

3.4 Measurement

Performance is defined as the time it takes for a number of tokens to be processed by a specific workload element. It is sufficient to base performance measurement on only one element if the workload elements are reactive and depending on each other for propagation. We call this the "measurement element".

Before measurements start it is necessary to warm the simulator up by running it until a predefined number of tokens (N_{wu}) have been processed by the measurement element. This is done to make sure that measurements do not start with an empty NoC or bus which would give an impression of an unrealistically fast execution. N_{wu} may differ for different combinations of workloads and interconnects. It is important that N_{wu} remains constant when experiments are repeated with different parameters or random seeds. It is common to let the simulator automatically detect when the load of the interconnect is stabilized and then decide that the warm-up phase is complete. But the interconnect will only stabilize if there is no burstiness on large timescales as in a Poisson process. In self-similar workloads with high Hurst parameters a "steady state" will not occur and therefore it is necessary to set N_{wu} manually in this case. When N_{wu} elements have been processed by the measurement element the simulation time T_{wu} is registered. The simulation proceeds until N_r more tokens have been processed by the measurement element when the simulation time T_r is registered and simulation is stopped. The workload runtime is then defined as: $T_r - T_{wu}$.

Since there is a risk that variability in any benchmark can lead to misleading conclusions unless it is taken into consideration [1], the benchmarks are run several times with one particular configuration but with different random-seeds. This is described in [3] as the replication method. Averages and standard deviations are calculated so that the variability of the results can be estimated. Given the mean runtime, its standard deviation, and the number of repeated experiments[1] confidence intervals can be calculated using the Students t-distribution which is found in standard mathematical tables [19].

3.5 Workload patterns

The workload elements can be organized into different workload patterns. While a workload patterns may model any abstract application model, this paper focuses on two patterns that are representative in many cases.

3.5.1 Fork-Join

A typical signal processing application can be parallelized by distributing the work in different "fingers". These fingers can be implemented in pipelines for further parallelization. The *Fork-Join* communication pattern is designed to mimic this type of applications. It begins with a workload element that emits tokens to four fingers. Each finger consists of two subsequent workload elements. Tokens are sent from the end of each finger to the last workload element which has a feedback connection to the first. This synthetic signal

[1] Also known as samples

processing application is accompanied with six workload elements that create interfering traffic. This communication pattern is illustrated in Figure 5.

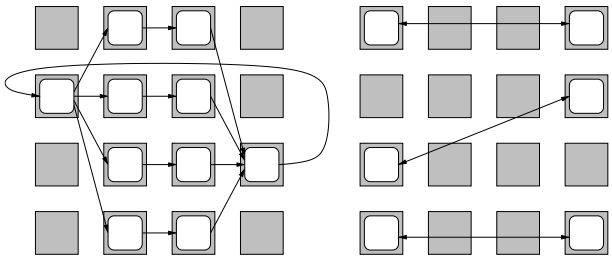


Figure 5. Fork-Join and its Background communication pattern.

3.5.2 Processor-Memory

This communication pattern models a processor that is connected to a memory. The workload element that represents the processor will emit tokens to the memory and cannot proceed until the token is returned. The processor and memory models are deliberately mapped to adjacent locations for efficiency reasons. Six workload elements provide easily configurable background traffic just like in the previous communication pattern.

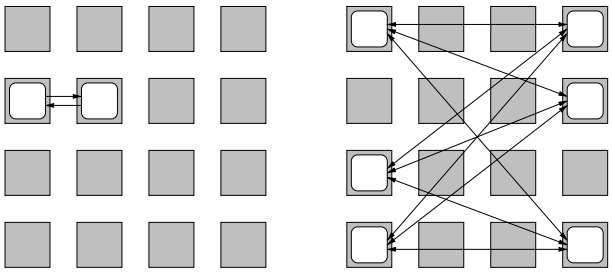


Figure 6. Processor - Memory and its Background communication pattern.

4 Case Study

To demonstrate the usefulness of the performance analysis method a small case study is conducted. A bus and a NoC, which are two fundamentally different communication infrastructures, are compared using the previously described communication patterns. This is a relevant study since busses and NoCs represent two different generations of interconnects and it is important to see how they respond to different workloads.

4.1 Communication infrastructures

The bus is 64-bits wide using round-robin arbitration and a 1 GHz clock frequency is used. One transaction is completed every clock cycle and succeeded by a transaction from another sender.

Nostrum [11] is a NoC platform that uses hot potato routing which has the advantages that requires no queues and avoids contention. If a switch has more than one packet that is preferably routed in a particular direction simultaneously only one will be sent there while the others will be routed elsewhere. In our tests, we have set the clock frequency to 1 GHz and each transaction is 64 bit [2]. *Nostrum* uses a mesh topology of variable size and here a size of $4x4$ is used.

Some general conclusions can be drawn just by looking at the design of these two communication infrastructures;

- The bus is suitable for situations where latency must be kept low and when the bandwidth demands are low.
- A NoC can be expected to handle a much larger bandwidth but its multi-hop topology will affect the lower bound on latency.

Still, benchmarks provide further insights on the difference between the two platforms. Since the main benefit of this performance analysis method is that burstiness can be explicitly modified, we shall study this effect and see how it affects both interconnects.

4.2 Experiments

The b parameter for the workload has been selected to 0.5, 0.7, and 0.9. Since the results do not differ very much between these values, only graphs where b is 0.7 for the workload are shown. The background b parameter is varied between 0.5 and 1. Simulations are repeated 12 times with different random seeds and the averaged value is used in the graphs.

Figure 7 shows how the performance varies when the workload burstiness is changed. It is remarkable how large influence burstiness has on performance for the bus solution. There is up to a 25% difference in runtime between the highest and the lowest degree of burstiness. In Nostrum this ratio is 9% at most. However, the bus is much faster than the NoC for this application since it is very dependent on low latency. The reason for this is that the processor in the model will stall when it waits for tokens to arrive from the memory. The big impact of bursty background traffic

[2] The actual link bit-width is 128 bytes but only the payload is counted as a transaction.

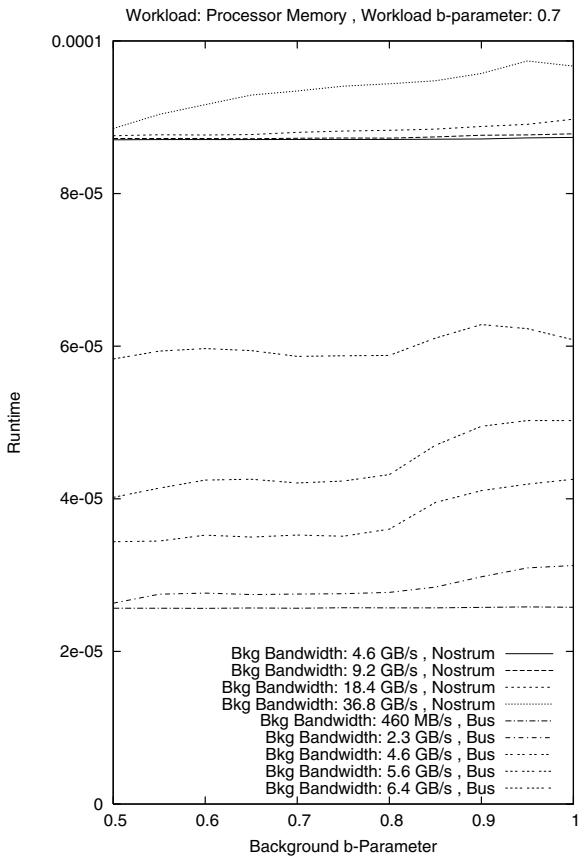


Figure 7. Impact of the burstiness of background traffic in the Processor - Memory pattern.

affirms our claims that this is a very important property that should be included in an accurate performance analysis.

Figure 8 depicts how the runtime varies when the bandwidth of the interfering traffic is changed. The range of the background traffic bandwidth is much smaller for the bus since it has a much lower saturation point than the NoC. Here, the confidence intervals are included in the plots and we can see that it is very small. Notice that the overall runtime is not very dependent on the background traffic for the NoC while it shows an almost exponential growth when the bus is used.

From these graphs we can conclude that the bus is a good candidate for the Processor-Memory application but it is very sensitive to its surrounding environment. It is clear that the NoC has drawbacks when it comes to latency but thanks to its adaptive nature it is not very sensitive to burstiness or the amount of background traffic.

The results form the Fork-Join pattern are very different from the previous. Figure 10 illustrates how the runtime varies with different background workloads. As in the former traffic pattern, the overall runtime is very sensitive to the amount of background traffic in the bus based design whereas the Nostrum based design is not affected very much.

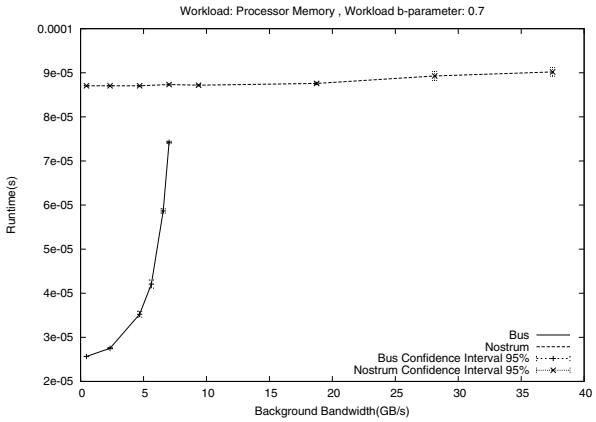


Figure 8. Runtime as a function of background bandwidth for Nostrum and the bus in the Processor - Memory pattern.

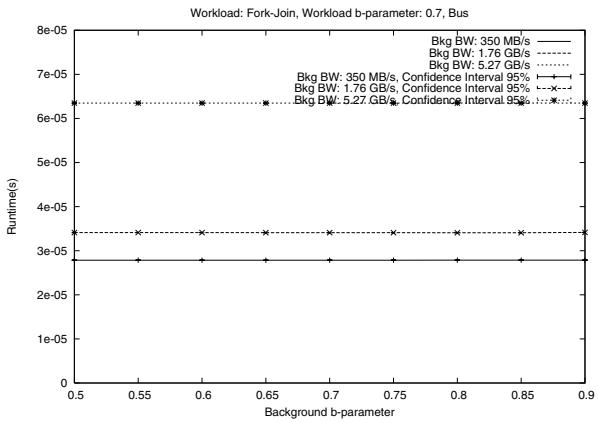
Figure 9 shows how the runtime varies with the b parameter. The burstiness does not affect the overall runtime in the bus based design since the graphs are completely flat. However, the situation in the Nostrum based design requires some explanation.

It may appear that the burstiness is affecting the execution time but since the confidence intervals overlap no conclusions can be drawn about the impact of burstiness or background bandwidth in this setting. The large confidence intervals are caused by a high variability in runtime when Nostrum is used. Since such large confidence intervals were found the hypothesis that variability must be dealt with was correct.

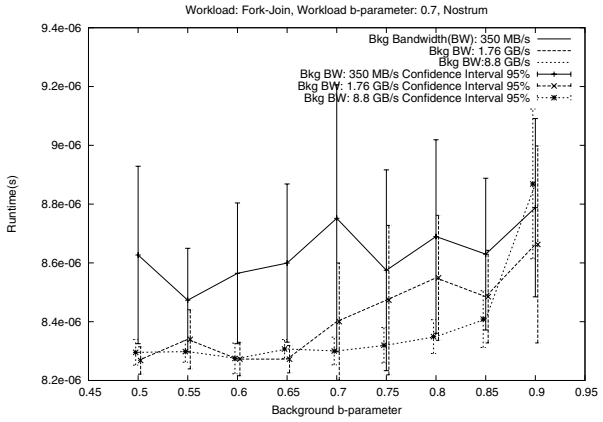
We can conclude that the bus is the fastest communication infrastructure for the Processor-Memory pattern while Nostrum handles the Fork-Join pattern most effectively. Also, it is clear that burstiness of the background traffic is a property that has an impact for performance in some instances and therefore it is very important in heterogeneous NoC environments.

5 Conclusions and future work

This paper introduces a novel method for performance analysis for networks on chip and traditional bus interconnects. Instead of focusing on performance of single specific applications or random traffic, our method uses syn-



(a) Bus



(b) Nostrum

Figure 9. The impact of burstiness in background traffic in (a) Bus and (b) Nostrum. Notice the large confidence intervals for Nostrum.

thetic workload models that are configurable so that they can model any system at an early stage in a design process. Our method accentuates the communication patterns and the data dependences between system components. The use of the b-model offers much more realistic and configurable bursty workloads than traditional stochastic processes do. Using our method, we have shown that the burstiness is a parameter that has a significant impact on results and that it must be considered. We have applied our method to two relevant application patterns and analyzed the performance on NoC and bus systems. These experiments show that already

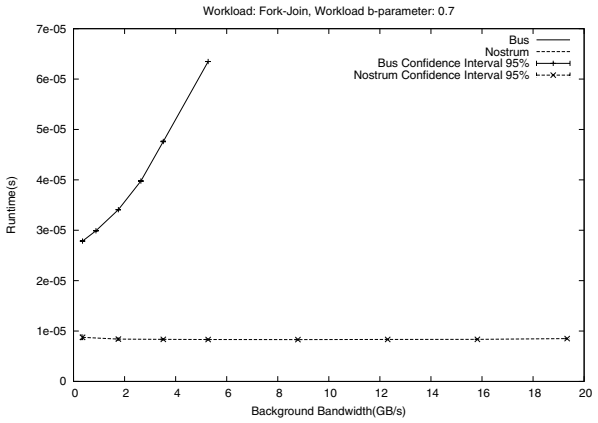


Figure 10. In the fork-join pattern, the background bandwidth has a similar impact as before but the NoC is always faster than the bus.

at an early design stage our method is capable to investigate important properties like the impact of burstiness for many applications.

We plan to apply modifications in the NoC architecture and investigate how overall performance and burstiness tolerance is affected. Our method can be very useful in such a study. It would also be interesting to try the method on other NoC architectures than Nostrum.

So far we did not have the opportunity to assess the accuracy of our workload models with real NoC applications. While it would be very interesting to conduct such a study, one should keep in mind that 100% accuracy can never be achieved without an exact workload model and that the strength of our method is that it enables performance analysis very early in the design process before any refinement has been done.

References

[1] A. Alameldeen and D. Wood. Addressing Workload Variability in Architectural Simulations . *IEEE Micro*, 23(6):94–98, November - December 2003.

[2] M. P. C. Lee and H. Mangione-Smith. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *Proceedings of the 30th Annual International Symposium on Microarchitecture*, December 1997.

[3] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[4] S. González Pestana, E. Rijpkema, A. Rădulescu, K. Goossens, and O. P. Gangwal. Cost-Performance Trade-Offs in Networks on Chip: A Simulation-Based Approach. In *Proceedings of the conference on Design, automation and test in Europe*, volume 2, pages 764–769, February 2004.

[5] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. MiBench: A free, commercially representative embedded benchmark suite. In *IEEE 4th Annual Workshop on Workload Characterization*, pages 3–14, December 2001.

[6] K. Lahiri, A. Raghunathan, and S. Dey. Evaluation of the Traffic-Performance Characteristics of System-on-Chip Communication Architectures. In *Fourteenth International Conference on VLSI Design*, pages 29–35, January 2001.

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

[8] S. Mahadevan, F. Angiolini, M. Storgaard, R. G. Olsen, J. Sparso, and J. Madsen. A Network Traffic Generator Model for Fast Network-on-Chip Simulation. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 780–785, 2005.

[9] J. Mahdavi and V. Paxson. RFC2678-2681. http://www.ietf.org/rfc.html, September 1999.

[10] M. A. Marsan, G. C. G. Balbo, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley Series in Parallel Computing. John Wiley and Sons, 1995.

[11] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum backbone - a communication protocol stack for networks on chip. In *Proceedings of the International Conference on VLSI Design*, pages 693–696, January 2004.

[12] K. Sadron, M. Martonosi, D. I. August, M. D. Hill, D. J. Lilja, and V. S. Pai. Challenges in Computer Architecture Evaluation. *IEEE Computer*, 36(8):30–36, August 2003.

[13] J. Voldman, B. Mandelbrot, L. Hoevel, J. Knight, and P. Rosenfeld. Fractal Nature of Software-Cache Interaction. *IBM Journal of Research and Development*, 27(2):164–170, March 1983.

[14] M. Wang, N. H. Chan, S. Papadimitriou, C. Faloutsos, and T. Madhyastha. Data Mining Meets Performance Evaluation: Fast Algorithms for Modeling Bursty Traffic. In *Proceedings of the 18th International Conference on Data Engineering*, pages 507–516, February 2002.

[15] R. P. Weicker. Dhrystone: A Synthetic Systems Programming Benchmark. *Communications of the ACM*, 27(10):1013–1030, October 1984.

[16] D. Wiklund, S. Sathe, and D. Liu. Network on chip simulations for benchmarking. In *Proceedings of the 4th IEEE International Workshop on System-on-Chip for Real-Time Applications*, pages 269–274. IEEE Computer Society, July 2004.

[17] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 24–36, June 1995.

[18] T. T. Ye, L. Benini, and G. D. Micheli. Packetization and routing analysis of on-chip multiprocessor networks. *Journal of Systems Architecture*, 50(2-3):81–104, 2004.

[19] D. Zwillinger, editor. *CRC Standard Mathematical Tables and Formulae*. Chemical Rubber Company Press, 30th edition, 1995.