A Queuing-Theoretic Performance Model for Context-Flow System-On-Chip Platforms

Rami Beidas, Jianwen Zhu
Electrical and Computer Engineering
University of Toronto, ON M5S 3G4, Canada
{rbeidas, jzhu}@eecg.toronto.edu

Abstract—**Few analytical performance models that relate performance figure of merit to architectural design decisions are reported in recent studies of network-on-chip, which prevents the development of effective system-level synthesis techniques. In this paper, we propose an analytical performance model based on queuing theory for a network-on-chip platform recently reported, which features an extremely simple programming model, while providing superior performance measures when compared with alternative architectures. We developed a multi-processor simulation framework, which can simulate an application at the instruction set level given an architecture configuration, to validate the analytical performance model. The accuracy and applicability of the proposed model is illustrated by two real-life applications, namely an SSL security acceleration processor and MP3 decoder.**

I. Introduction

With the vast complexity growth of System-On-Chip (SOC) platforms, the number of critical design decisions and alternative implementations and configurations considered in order to map an application to the corresponding platform increases exponentially. Therefore, the ability to evaluate the effect of these possible alternatives accurately in a reasonably short time becomes indispensable.

So far, simulation-based approaches have been the dominant choices by the industry for performance analysis of SOCs. These approaches are highly accurate, but also prohibitively time consuming for large systems, which prevents the evaluation of a large number of possible system configurations. Intuition and experience are usually relied on to select a few configurations to simulate out of the feasible many. However, such *ad hoc* decisions become less effective as the systems become larger. What is badly needed is a performance model that can give insight on how performance metric is related to architectural mapping decisions, commonly referred to as *analytical performance model*.

In recognition to the limitations of simulation-based SOC exploration procedure, we developed an analytical performance model to statically model systems implemented on the recently proposed Context-Flow Architecture (CFA). Our model is based on Queueing Networks, a field that received extensive research over many decades, and whose models were used extensively in computer systems and networks modeling. Queueing network models were proved to be general, simple, accurate, and detailed, reporting various aspects of the target system and application performance measures.

In contrast to the previous work reported in the area, the following contributions are made in this paper. First, the proposed performance model is extremely *simple*. In fact, the solution of important metrics involves only simple equations. Second, the proposed performance model is *synthesis-friendly*. An optimization procedure can be readily developed, in contrast to the manual "architectural exploration" approach commonly practiced. Third, the proposed model is highly *accurate*. In fact, for the key metrics of interest, our model is as accurate as the input statistics. Fourth, our performance model is *validated* against real-life applications with a detailed multiprocessor simulator. The credibility and applicability of the proposed model is therefore guaranteed.

The remainder of the paper is organized as follows. Section II provides some background material of CFAs and queueing networks. In Section III, we derive the analytical model for performance measures of CFAs. Section IV describes our CFA simulation framework used to validate our model before presenting experimental results. Related works are described in Section V followed by a conclusion in Section VI.

II. Background

A. Context-Flow Architectures

In this section, we briefly describe the rationale and design of the context-flow SOC architecture. Our goal is to provide enough background so that the analytical performance model, the main subject of the paper, can be related to a realistic architecture. Interested readers are referred to [1] for a detailed discussion of the architecture.

The context-flow architecture [1] was proposed to address two omissions in recent researches in communication-centric SOC platforms, or network-on-chip. First, while traditional computer architecture is well abstracted with a programming model, new SOC architectures have not made much progress on that front. An SOC platform is either modeled in system-level languages, such as SystemC [2] or SpecC [3], where a distinction between application, architecture and hardware does not exist, or using traditional parallel programming models, such as MPI [4], which are usually very complex to implement. Second, while traditional networks in supercomputers are designed with the bandwidth limitation imposed by chip pin count, new SOC platforms, which are based on similar topologies, do not take full advantage of the much relaxed physical constraints and almost unlimited on-chip bandwidth.

We introduce a new programming model revolving around a new concept, called *context*, which is essentially an abstraction of autonomous dynamic data structures closed under the point-to relation. A context-flow program (CFP) can be viewed as a set of procedures operating on a set of contexts in a multi-threaded form, collaborating through remote procedure call abstraction (RPC) to achieve the overall system behavior. Unlike an application in traditional programming models, a CFP is *highly parallelizable*, since different procedures, each accessing their own private data structures maintained in different context, can be run in a CFA on different processing elements (PEs) in parallel, without the concern of dependency hazard or cache coherence that frequently occur in the traditional shared or distributed memory architectures. The accesses of contexts do switch from one procedure to another when a procedure call occurs.

The key problem in the design of a CFA is the design of its on-chip network. We start by first defining an instruction set, which abstracts how the on-chip network interacts with the PEs that it connects (Figure 1). The instruction set is simple enough to contain only 7 instructions. It is encoded by the values of the wires on each port that connects a PE to the network. From the perspective of the network, it encodes a command or request from a PE. From the perspective of a PE, the instruction set is a complement of its own for which it can assume the availability of a co-processor for actual execution – effectively by driving the right wires in the corresponding ports.

In Figure 1, `cfiAllocBank` allocates a bank for a single context until deallocated by `cfiFreeBank`. `cfiMalloc` is used for subsequent allocations of arbitrary objects on the target context. `cfiLoad` and `cfiStore` are simple memory accesses. `cfiRPC` and `cfiRet` are used to implement the remote procedure call abstraction, where the context currently accessed by the caller is passed to the callee for further processing.

```
int     cfiAllocBank( void );
void    cfiFreeBank( int bankid );
void*   cfiMalloc( int size );
word    cfiLoad( int addr );
void    cfiStore( int addr, word data );
void    cfiRPC( int procid );
void    cfiRet( int procid );
```

Fig. 1. Context-flow Instruction Set

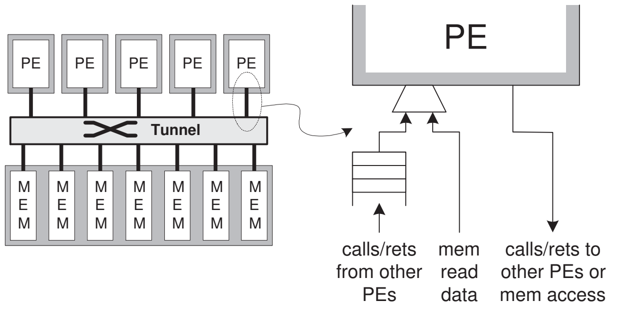

Fig. 2. Tunnel-Based Context-Flow Architecture

Previous efforts do not take full advantage of the fact that the network we are designing is on-chip, and the PEs are physically close to each other. In [1], we proposed a new on-chip network, called a CFA *tunnel*, that can implement this instruction set efficiently. As shown in Figure 2, the tunnel maintains a pool of separate memory banks, as well as an intelligent crossbar switch. Each context is dynamically mapped to a single memory until it is deallocated, and the crossbar ensures the access to the memory is dynamically switched to the callee whenever an RPC occurs. Note that our crossbar should not be confused with crossbars in previous efforts, such as switch fabrics of network routers, which are utilized still for the purpose of data transfer. Instead, the goal of our crossbar is to provide the direct, wired access to memories. RPC, or the flow of contexts from one PE to another, can then be achieved at virtually no cost! Experimental results in [1] showed superior performance measures of CFA-based implementation when compared with alternative architectures.

It is important to note that there is a physical limit for the scalability of the CFA tunnel. As the network gets larger, the delay of the crossbar grows quickly, thereby increasing the cost of each memory access. This is contained by employing a two-layer strategy, where PEs are partitioned into clusters based on the communication traffic among them, and intracluster network is based on the tunnel, whereas the intercluster network is based on packet-switch, such as those presented in [5], [6], [7]. In this paper, we focus only on the study of the flat network, which we believe is appropriate for state-of-the-art multimedia and network applications, such as those presented in the Section IV.

B. Queueing Networks

In this section, we provide some background on Queuing Network, an efficient and accurate approach to computer system modeling. It has been used in the design of systems ranging from single network servers to wide area communication networks [8].

A queueing network consists of a set of communicating nodes of service providers. A job arrives at a node, waits in the corresponding queue when all servers are busy, gets processed, and departs for another node or out of the system[1]. Figure 3 shows an example of a simple queueing system with some feedback flows.

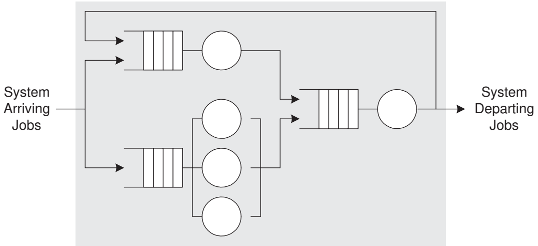

Fig. 3. A Queueing Network

A key feature and reason to the success of queueing network models is that they abstract away many of the low level details associated with the various modeled system. All it needs is a set timed parameters that affect the system performance.

The basic characterization entities of queueing network models are *service providers*, which represent the modeled system processing resources, and *customers*, which represent the system jobs (contexts in our case). A typical set of inputs of a queueing model are [8]:

- λ, *arrival rate*, specifies the arrival intensity in customers per unit time.
- Dem_i, *service demand* at server i, which specifies the service time for each customer.

The outputs obtained by solving the system are:

- R, *average system response time*, which specifies the travel time between the system input and output.
- U_i, *utilization* of server i, i.e. the percentage of overall time the server is busy.
- Wq_i, *queueing time* of server i, which specifies the average waiting time at server i before a job gets serviced.
- R_i, *residence time* of server i, which is simply the sum of average waiting time and average service time at server i.
- Lq_i, *queue length* of server i.

If the jobs arriving to the system have some classification, usually referred to as *Multi-Class Systems*, the model inputs need to specify the job mix and required services, and the outputs will be returned per class as well as overall system measures. It is worth noting that the input and output measure mentioned above are just the essential requirements for the least detailed models. Further parameters and results are associated with other models used in various analysis tools, as presented below.

Finally, it is a common practice in queuing theory to describe a queue using Kendall's Notation ($A/S/m/B/K/SD$); where:
A describes the distribution of interarrival times of customers. S is the distribution of service times. m is the number of servers. B is the maximum number of customers which can be accommodated by the annotated queue. K is the population size, and SD is the service discipline. For example M/D/2/10/500/FCFS is for exponentially distributed interarrival time, deterministic service time, two servers, buffer size of 10, population 500, and first-come-first-served discipline. Default values, such as infinite queue size and FCFS service discipline, can be omitted from this notation.

III. Analytical Performance Model

A. The Modeling Process

The close correspondence between the attributes of queueing networks and those of our CFA, as shown in Section II, suggests that

[1]Unless explicitly stated otherwise, when we talk about queueing networks we always refer to Open Queueing Networks, as opposed to Closed Queueing Networks which do not interact with the outside world.

queueing networks could be ideal modeling tools to describe our system.

The modeling process could be viewed as a conversion from system specifications in the Context-Flow domain to those recognized by queueing systems. The output of this stage would be a fully specified queueing network that can be easily solved using simple equations. Whether the resulting system is single-class or multi-class depends on the application being mapped on a CFA.

The inputs of our modeling process are:

• *Workload Specification*, which defines the arrival jobs mix and their corresponding arrival rates. This can be obtained by a process called *workload characterization*, which is a complex process of profiling to arrive at a typical workload. A second possibility is that a typical workload would be defined initially as part of the system specifications [8].

• *Procedure Frequency*, which defines the number of calls made to each context-flow procedure per unit time. Again, this measure can be obtained by profiling of a typical workload, or by static prediction of the probability of edges of the application call graph for a typical workload.

• *Mapping*, which describes the assignment of procedures to target system processing elements.

The output of our modeling is a fully characterized queueing network. Solving the model returns the performance estimates of various aspects of the system.

B. Stochastic Model

Traditional applications of queueing networks to model computer systems assumed the arrival of a Poisson process at the system inputs, and exponentially distributed service times at the service centers [8]. These assumptions imply that the resulting interconnection of processing elements forms a *Jackson Network* [9]. In this class of networks each queue can be analyzed separately as an M/M/m queue. This model is parameterized only by the average arrival rate and average service rate, returning average waiting time, average queue length, and server utilization. This approach was proved quite successful in modeling such systems. For example, requests sent by users to a mainframe did have a random arrival pattern that was captured using a Poisson process. And the size of jobs to be serviced was also a randomized process. However, the immediate application of the same simplifying assumptions to model our architecture was unsuccessful. In a SOC, the arrival process and/or service times could easily be deterministic! For example, arrival rate for an MPEG decoder is usually deterministic, and service rate for ATM packet processing stages is also deterministic.

In [10], W. Whitt described the Queueing Network Analyzer (QNA), a software package developed at Bell Laboratories to analyze complex queueing networks. The package uses a GI/G/m approximation models to describe and analyze the given system. The arrival process is assumed to be a generalized interarrival (GI) process, and the service may have any general (G) distribution. The approximation made by this approach is that only the *mean* and *squared coefficient of variance* ($SQV = var/(mean)^2$) of the arrival and service processes are required for the our calculations (a two-moment model). In addition to the basic input parameters described in Section II-B, we need to provide the SQV of interarrival time of the external arrival process to each node i, c_{0i}^2, and the SQV of the service time, c_{si}^2. The analysis process calculates the parameters of internal nodes, which enables the calculations of all required system measures. The model is capable of handling even more complicated system features, including superposition and splitting, which is outside the scope of this paper.

For our purpose, the proposed model seemed to be a suitable fit. The additional required parameters could easily be driven by workload characterization. The question left is the model accuracy, which will be reported in Section IV. In the sequel, we provide our approach to transform our CFA and application description into a fully described queueing network model.

C. Derivation of Analytical Performance Metrics

Let's assume that our system consists of N procedures with execution frequency $[f_0 f_1 \dots f_{N-1}]^T$, implemented on an M-port tunnel-based CFA. We define an *MxN* mapping matrix, *MAP*, where $MAP_{i,j}$ represents the mapping of procedure j to PE i.

$$MAP = \begin{pmatrix} m_{0,0} & \cdots & m_{0,N-1} \\ \vdots & \ddots & \vdots \\ m_{M-1,0} & \cdots & m_{M-1,N-1} \end{pmatrix} \tag{1}$$

For example, if we have an application realized in five procedures $[p_0 p_1 p_2 p_3 p_4]$ an implemented on a 3-port CFA such that p_0 and p_2 run on PE0, p_1 and p_4 run on PE1, and p_3 runs on PE2, then the mapping matrix is:

$$MAP_1 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2}$$

Note that in this model $\sum_{i=0}^{M-1} m_{i,j}$ must add to 1. Values less that 1 imply logic/functionality replication and workload distribution. For example, if we want to replicate procedure p_3 and divide the arrival requests such that one third of the requests go to PE1 and the rest to PE2, then the new mapping matrix will be:

$$MAP_2 = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0.33 & 1 \\ 0 & 0 & 0 & 0.67 & 0 \end{pmatrix} \tag{3}$$

To force single instantiation of procedure logic, we allow mapping figures to take only binary values, 0, 1.

Using the summing rule, when two procedures are assigned to a single PE, the arrival rate will be the sum of their frequencies. This conversion from the abstract domain to the queueing system domain can be captured using the mapping matrix, as shown in 4.

$$\begin{pmatrix} m_{0,0} & \cdots & m_{0,N-1} \\ \vdots & \ddots & \vdots \\ m_{M-1,0} & \cdots & m_{M-1,N-1} \end{pmatrix} \begin{pmatrix} f_0 \\ \vdots \\ f_{N-1} \end{pmatrix} = \begin{pmatrix} \lambda_0 \\ \vdots \\ \lambda_{M-1} \end{pmatrix} \tag{4}$$

After deriving the arrival rate for each PE/queue, we can calculate all performance measures the fully characterize the system behavior. The average execution time at each PE can easily be obtained using the equation:

$$D_i = \frac{\sum_{j=0}^{N-1} (m_{i,j} \cdot f_j \cdot Dp_j)}{\sum_{k=0}^{N-1} (m_{i,k} \cdot f_k)} \tag{5}$$

Where Dp_j is the average processing time of job by procedure j. Although Dp_j is assumed to be constant, the model can be easily extended to make procedure delays a function of the mapping. On heterogeneous systems, a single procedure could be mapped to different embedded processors with different architectural features, or even to custom logic. To take that into account we can define Dp_j in terms of $d_{i,j}$; the average processing time of job by procedure j when running on PE i.

In Queueing Theory it is a common practice to use service rate instead of service or processing time:

$$\mu_i = \frac{\sum_{k=0}^{N-1} (m_{i,k} \cdot f_k)}{\sum_{j=0}^{N-1} (m_{i,j} \cdot f_j \cdot Dp_j)} \tag{6}$$

Using these numbers we can derive major performance measures of processing elements using very simple formulas. The equation describing processing element utilization would be:

$$Utilization_i = \rho_i = \frac{\lambda_i}{\mu_i} = \sum_{j=0}^{N-1} (m_{i,j} \cdot f_j \cdot D_j) \tag{7}$$

Using equations 6 and 7, and c_{ai}^2 and c_{si}^2 for each node i, we can calculate further estimates of PE statistics. For example, the average waiting time at PEi is:

$$AveWaitingTime_i = Wq_i \quad = \quad \left(\frac{c_{ai}^2 + c_{si}^2}{2}\right) Wq_i^{M/M/1} \tag{8}$$

Where:

$$Wq_i^{M/M/1} \quad = \quad \frac{\rho_i}{\mu_i(1-\rho_i)} \tag{9}$$

$$AveQueueLength_i = Lq_i = \lambda_i W_{qi} \tag{10}$$

We can also derive performance estimates of the overall system. An average processing elements utilization is:

$$Utilization = \rho = \frac{\sum_{i=0}^{M-1} \rho_i}{M} \tag{11}$$

And the average service time for a request is:

$$AveServiceTime = D = \frac{\sum_{i=0}^{M-1} \lambda_i \cdot (D_i + Wq_i)}{\lambda} \tag{12}$$

Using this model we can easily get performance measures for each procedure, each processing element, each job class, and the overall system. Further processing is needed if the more detailed probability distribution of the above quantities is required, which is outside the scope of this work.

IV. Model Validation

In this section we support our proposal by demonstrating the model accuracy through experimental results. We start by introducing our performance evaluation framework. Then we introduce the application we use as a test case followed by results and discussion.

A. Performance Evaluation Framework

At the system level design we target complex applications usually described in C using high-level language features such as pointer references and complex data structures. The speculated performance model accuracy can only be validated on such applications. A performance evaluation environment, which can simulate CFA with reasonable architectural details for any CFP application, is therefore needed.

A good example of an architectural evaluation environment is the SimpleScalar tool set developed at Wisconsin [11]. It is designed to study new innovations in micro-architecture such as pipelining, branch prediction, out-of-order issue etc. The environment provides a complete compiler tool chain that can compile a C application into a binary in the PISA instruction set. An instruction set simulator can then be used to simulate the binary, while collecting performance metric of interest.

We consider a homogeneous CFA where each PE is implemented by a processor equipped with the PISA instruction complemented by the context-flow instruction set defined in Figure 1. While each PE has their own private address space, an unused memory space segment of each PE, from address 0x00000000 to 0x03FFFFFF, is mapped to context memory pool. With this approach, high-level language features, such as array references, pointer indirection and structure member references, can still be used directly in the source code to access objects within the context. We also coded a cycle-accurate implementation of the tunnel-based on-chip network defined in Section II-A. The SimpleScalar annotation interface was used to introduce the context-flow instruction set to each PE. Further details can be found in [1].

Our simulator collects all performance statistics that we need to fully describe the system performance during simulation. These statistics are compared with those derived in our queueing network-based estimation model for validation purposes.

B. Test Cases

We pursue the validation of our model through real-life applications, namely Cryptography Acceleration Processor, and MP3 Decoder.

B.1 Cryptography Acceleration Processor

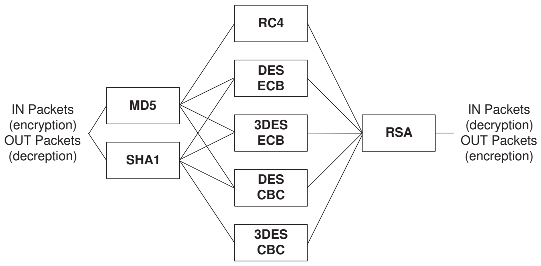

Fig. 4. Crypto Accelerator Flow

Cryptography acceleration processors are becoming of central interest with the increase of SSL-based traffic over the internet. In our benchmark, we implemented a number of symmetric and asymmetric algorithms commonly used in SSL and IPSec. The implemented functions and the possible flows of packets are shown in Figure 4. Delay of processing methods were mainly obtained from actual RTL implementations [12]. The longest path of an input packet is to go through all three categories of processing, namely hashing (MD5 or SHA1), symmetric or private-key encryption (DESECB, DESCBC, 3DESECB, 3DESCBC, or RC4), asymmetric or public-key encryption (RSA). Packets could skip hashing, public-key encryption, or both.

B.2 MPEG1-LayerIII Decoder

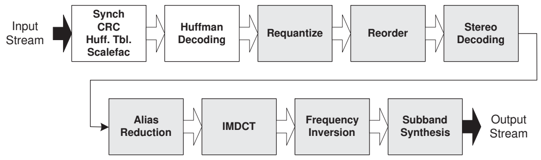

Fig. 5. MP3 Decoder

MPEG1-LayerIII, commonly referred to as MP3, is the de-facto standard of high-quality high-compression of audio data. MP3 decoders became of interest after their popular use in portable multimedia devices. An overview of the decoder stages is presented in Figure 5 The highlighted stages were implemented in our testbench. Each stage is implemented in a single procedure processing one data granule at a time.

C. Results and Discussion

To carry out the experiments on the SSL accelerator, we implemented a packet generator that generates a workload, or packet mix, which uses various processing paths according to given distribution parameters. For the MP3 decoder, on the other hand, we used some of the input files distributed along with the standard MP3 software.

In case of the SSL accelerator, for a given workload we used the different mappings described in Table I. For example, in mapping 1 we map the RSA procedure to PE0, MD5 to PE1, SHA1 to PE2, and so on. The corresponding simulation and estimation results are reported in Table III, and the average estimation errors for each mapping over all PEs are presented in Figure 6. In Table III, for each mapping we report the average simulated residence time and that estimated by our model for each PE (other measures, such as response time and utilization, can be easily derived from model inputs and reported results). For example, for mapping 1 of the SSL accelerator, the average residence time at PE0 was 10255.5 cycles, while the estimated value was 7027.6, residence time at PE1 was 462.1 cycles, while the estimated value was 413.6, and so on. Figure 6 reports the average estimation error for each mapping over all PEs. For example, estimation error for mapping 1 over all PEs was 11.58%. Similarly, for the MP3 decoder we tried the mappings described in Table II, and the corresponding estimation results are reported also in Table III and Figure 6.

Mapping	Target PE							
	RSA	MD5	SHA1	RC4	ECB	3ECB	CBC	3CBC
1	PE0	PE1	PE2	PE3	PE4	PE5	PE6	PE7
2	PE0	PE1	PE1	PE2	PE3	PE3	PE4	PE5
3	PE0	PE1	PE1	PE2	PE2	PE2	PE1	PE3

TABLE I

SSL ACCELERATOR MAPPINGS FOR PERFORMANCE MODEL EVALUATION

Mapping	Target PE						
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
1	PE0	PE0	PE0	PE1	PE2	PE1	PE3
2	PE0	PE1	PE2	PE3	PE3	PE4	PE5
3	PE0	PE1	PE1	PE1	PE2	PE3	PE4

TABLE II

MP3 DECODER MAPPINGS FOR PERFORMANCE MODEL EVALUATION

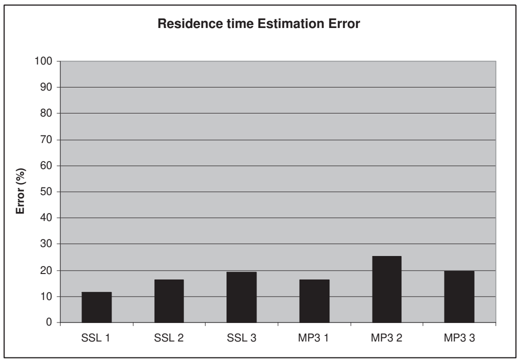

Fig. 6. Queueing Model Accuracy for SSL Accelerator and MP3 Decoder

Application	Mapping	Proc. Element	Sim. Residence Time	Est. Residence Time
SSL Accelerator	1	PE0	10255.5	7027.6
		PE1	462.1	413.6
		PE2	547.8	510.5
		PE3	1864.9	1844.9
		PE4	1144.5	1270.2
		PE5	1244.4	1360.1
		PE6	3396.5	3968.6
		PE7	7645.9	7209.9
	2	PE0	8360.8	7020.0
		PE1	650.9	453.9
		PE2	1843.5	1840.1
		PE3	2450.6	1923.6
		PE4	3561.0	3736.5
		PE5	8189.6	6164.9
	3	PE0	5646.9	6360.2
		PE1	2399.0	2119.0
		PE2	4020.1	2890.8
		PE3	6459.6	4850.8
MP3 Decoder	1	PE0	3621.8	2544.7
		PE1	1301.9	1781.6
		PE2	9246.7	9891.0
		PE3	2756.0	4925.3
	2	PE0	1271.4	1288.6
		PE1	832.5	972.6
		PE2	1417.4	1576.9
		PE3	11519.9	5918.8
		PE4	764.8	786.0
		PE5	2868.6	3340.9
	3	PE0	1271.4	1288.6
		PE1	2242.9	2340.8
		PE2	11533.5	3000.9
		PE3	764.8	785.9
		PE4	2873.2	3337.8

TABLE III

SIMULATED AND ESTIMATED RESIDENCE TIME

From the reported results, we can see that the estimation results were accurate in some cases, and varied (either high or low) in others, but correctly reported the relative time values at different PEs with acceptable average error (Figure 6), taking only few seconds as opposed to many simulation hours. It turned out that the way the solver handles multi-class networks through simple aggregation could potentially be improved. To illustrate this issue, mapping 3 of the SSL test case was intentionally configured such that procedures with largely different processing times were mapped to the same PEs. Also, the use of a single variability parameter to characterize the variability of an arrival process to a queue was not optimal. More advanced solutions were reported in [13], and further enhancements to queueing network solvers are being proposed in this active area of research, which is outside the scope of this work. However, as we observed in our experiments, the used solver still serves as a first order approximation of the queueing time at each PE. For example, the solver does not report a waiting time in thousands of cycles while the actual value is only in hundreds, or vice versa.

Although higher accuracy levels would have been appreciated, our proposed model is still valid, and it gets as accurate, flexible, and powerful as queueing theory itself. Even at the reported accuracy measures, the model will provide important optimization directions as part of a system-level optimization framework.

V. RELATED WORK

While a rich literature on performance modeling in general has been reported in the field of hardware-software codesign [14], [15], [16], [17], [18], very little work has been carried out focusing on the performance modeling of SOC architectures [19]. In the following we give a brief review of those efforts focusing on the performance modeling of network-on-chip. We start by first developing a taxonomy to help categorize these works.

- A performance model is *dynamic*, if it relies on the use of simulation. It is *static* otherwise. In general, a dynamic performance model is more accurate with respect to specific input trace. A static performance model is faster to evaluate.
- A performance model is *analytical*, or *architecture-aware*, if the result depends not only on the characteristics of the application, but also the architecture and how application is mapped to the architecture.
- A performance model is *automatic*, if it can be automatically constructed from the application and architectural mapping. It is *manual* otherwise.
- A performance model is *validated*, if its accuracy has been confirmed by detailed simulation.

Stochastic Automata Networks (SANs) were used in [20] to analyze application and derive probability distribution for various performance aspects of the target application. This model is static, however, not architecture-aware. Furthermore, the construction of a SAN network from an application is not yet an automated process.

A static performance model for network packet processing architectures was derived in [21] using Network Calculus results. The proposed approach uses deterministic bounds to describe the arrival and service processes of the target system. The model is also analytical, yet incomplete in the sense that conflicts over communication resources are ignored, which could easily result in large errors of the estimated measures. As a result, estimation results of the test cases were not validated.

The work in [22] proposes a hybrid static/dynamic performance analysis methodology for bus-based SOC communication architectures. Although the flow was validated and accurate estimates were reported, a speedup of only 2 over hardware/software co-simulation was obtained.

In this work we propose a performance model of a concrete SOC platform equipped with both an efficient on-chip network and a simple application programming model. The proposed model is static, architecture-aware, automatically evaluated, and can be easily incorporated in a system-level synthesis framework.

VI. Conclusion and Further Work

In this work we proposed the use of queueing networks to derive analytical performance models for a novel SOC platform. We illustrated the model usability and accuracy with real-life applications using a cycle-accurate simulation environment. The model is as flexible and powerful as queueing theory. It can easily be used in exploring the design space of CFAs for system-level synthesis, which represent a promising future work in this field.

After having a better understanding of the behavior of intercluster traffic on candidate second-level networks, such as torus or mesh [5], future work will consider the incorporation of the queueing theoretic model in a complete static performance analysis of larger systems. At that stage, the enhanced model will become an essential part of a complete system-level design exploration framework.

References

[1] R. Beidas and J. Zhu, "Performance Efficiency of Context-Flow System-On-Chip Platform," in *Proceedings of the International Conference on Computer-Aided Design*, November 2003.

[2] `http://www.systemc.org`.

[3] D. Gajski, J. Zhu, D. Doemer, A. Gerstlauer, and S. Zhao, *SpecC: Specification Language and Methodology*, Kluwer Academic Publishers, March 2000.

[4] *Message Passing Interface (MPI) Web Site*, `http://www-unix.mcs.anl.gov/mpi`.

[5] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proceeding of the 38th Design Automation Conference*, June 2001.

[6] P. P. Pande, C. Grecu, A. Ivanov, and R. Saleh, "Switch-Based Interconnect Architecture for Future Systems on Chip," in *Proceedings of SPIE, VLSI Circuits and Systems*, 2003.

[7] 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 VLSI Design Conference*, January 2004.

[8] E. D. Lazowska, J. Zahorjan, G. S. Graham, and K. C. Sevcik, *Quantitative System Performance, Computer System Analysis Using Queueing Network Models*, Prentice-Hall, Inc., February 1984.

[9] S. K. Bose, *An Introduction to Queueing Systems*, Kluwer Academic Publishers, December 2001.

[10] W. Whitt, "The Queueing Network Analyser," *The Bell System Technical Journal*, vol. 62, no. 9, pp. 2779–2815, November 1983.

[11] D. Burger and T. M. Austin, "The simplescalar tool set, version 2.0," Tech. Rep., Computer Science Department, University of Wisconsin, 1997.

[12] R. Usselmann, "DES/Triple DES IP Cores," September 2001.

[13] W. Whitt, "Towards Better Multi-Class Parametric-Decomposition Approximations For Open Queueing Networks," *Annals of Operations Research*, vol. 48, pp. 221–248, 1994.

[14] S. Malik, M. Martonosi, and Y.-T. Li, "Static timing analysis for embedded software," in *Proceeding of the 34th Design Automation Conference*, June 1997.

[15] T.-Y. Yen and W. Wolf, "Performance estimation for real-time distributed embedded systems," in *International Conference on Computer Design*, June 1995.

[16] A. Kalavade and P. Moghe, "Hardware-software codesign of embedded systems," in *Proceeding of the 35th Design Automation Conference*, June 1998.

[17] A. Mathur, A. Dasdan, and R. Gupta, "Rate analysis of embedded systems," *ACM Transaction on Design Automation of Eletronic Systems*, vol. 44, no. 3, July 1998.

[18] A. Baghdadi, N.-E. Zergainoh, W. O. Cesario, and A. A. Jerraya, "Combining a Performance Estimation Methodology with a Hardware/Software Codesign Flow Supporting Multiprocessor Systems," *IEEE Transactions on Software Engineering*, vol. 28, no. 9, September 2002.

[19] J. Russell, "Literature Survey: Software Performance Estimation," Tech. Rep., University of Texas at Austin, June 2001.

[20] R. Marculescu and A. Nandi, "Probabilistic Application Modeling for System-Level Performance Analysis," in *Proceedings of the Design Automation and Test Conference in Europe*, March 2001.

[21] L. Thiele, S. Chakraborty, M. Gries, and S. Kunzli, "A Framework for Evaluating Design Tradeoffs in Packet Processing Architectures," in *Proceeding of the 39th Design Automation Conference*, June 2002.

[22] K. Lahiri, A. Raghunathan, and S. Dey, "Fast Performance Analysis of Bus-Based System-On-Chip Communication Architectures," in *Proceedings of the International Conference on Computer-Aided Design*, November 1999.