

# Accurate Run-Time Performance Prediction for Multi-Application Multi- Processor Systems

Akash Kumar, Bart Mesman, Henk Corporaal and Yajun Ha



## ES Reports

ISSN 1574-9517

ESR-2008-07

16 June 2008

Eindhoven University of Technology  
Department of Electrical Engineering  
Electronic Systems



© 2008 Technische Universiteit Eindhoven, Electronic Systems.  
All rights reserved.

<http://www.es.ele.tue.nl/esreports>  
[esreports@es.ele.tue.nl](mailto:esreports@es.ele.tue.nl)

Eindhoven University of Technology  
Department of Electrical Engineering  
Electronic Systems  
PO Box 513  
NL-5600 MB Eindhoven  
The Netherlands

# Accurate Run-time Performance Prediction for Multi-Application Multi-Processor Systems

Akash Kumar<sup>1,2</sup>, Bart Mesman<sup>1</sup>, Henk Corporaal<sup>1</sup> and Yajun Ha<sup>2</sup>

<sup>1</sup>Department of Electrical Engineering

Eindhoven University of Technology, The Netherlands

<sup>2</sup>Department of Electrical and Computer Engineering

National University of Singapore, Singapore

Email: a.kumar@tue.nl

**Abstract**—Non-preemptive multi-processor platforms are increasingly being developed to support the performance requirements of modern systems with multiple applications. Due to a huge number of possible combinations of these multiple applications, it becomes a challenge to predict their performance in advance. This becomes even more important when applications may be dynamically started and stopped in the system. *Mis-prediction* may result in reduced quality of applications and lower the user-experience. Since modern embedded systems allow users to download and add applications at run-time, a complete design-time analysis is not possible.

In this paper, we present a technique to accurately predict the performance of applications at run-time before they execute in the system. The technique uses performance expressions computed off-line from the application specifications. A run-time iterative probabilistic analysis is used to estimate the time spent by tasks during contention phase, and thereby predict the performance of applications. The performance values predicted vary from the measured values by 2% on average and 3% at maximum. The analysis takes 3ms on a 50MHz processor for 10 applications. The approach is fast, yet extremely accurate.

Further, the prediction technique is used to design an admission controller that is completely implemented and tested on FPGA. Besides the approach and results, we provide a fully automated flow to generate such a controller on an FPGA multiprocessor platform. In addition, we present a complete and composable system design flow where applications may be added at run-time.

**Index Terms**—Heterogeneous multiprocessor, synchronous data flow graphs, multiple applications, admission controller, FPGA, system-design, performance prediction.<sup>1</sup>

## I. INTRODUCTION

Current developments in modern embedded devices like set-top box and mobile phone, for media systems integrate a number of applications or functions in a single device, some of which are not even known at design time. Therefore, an increasing number of processors are being integrated into a

single chip to build Multi-Processor Systems-on-Chip (MP-SoCs). To achieve high performance in such systems, the limited computational resources must be shared causing contention. Modeling and analyzing this interference is essential to building cost-effective systems which can deliver the desired performance of the applications.

However, with increasing number of applications running in parallel leading to a large number of possible *use-cases*, their performance analysis becomes a challenging task [2]. (A *use-case* is defined as a possible set of concurrently running applications.) The problem is compounded by the fact that applications may be started and stopped by the user at run-time. Future multimedia platforms may easily run 20 applications in parallel, corresponding to an order of  $2^{20}$  possible use-cases. It is clearly impossible to verify the correct operation of all these situations through testing and simulation. The product divisions in large companies already report 60% to 70% of their effort being spent in verifying potential use-cases. This has motivated researchers to emphasize the ability to analyze and predict the behavior of applications and platforms without extensive simulations of every use-case.

While this analysis is well understood (and relatively easier) for preemptive systems [3][4][5], non-preemptive scheduling has received considerably less attention. However, for high-performance embedded systems (like cell-processing engine (SPE) and graphics processor), non-preemptive systems are preferred over preemptive scheduling for a number of reasons [6]. In many practical systems, properties of device hardware and software either make the preemption impossible or prohibitively expensive. Further, non-preemptive scheduling algorithms are easier to implement than preemptive algorithms and have dramatically lower overhead at run-time [6]. Further, even in multi-processor systems with preemptive processors, some processors (or coprocessors/ accelerators) are usually non-preemptive; for such processors non-preemptive analysis is still needed. It is therefore important to investigate non-preemptive multi-processor systems.

### A. Need for Run-time

In modern multimedia systems, multiple applications are executing concurrently. While traditionally a mobile phone had to support only a handful of applications like communicating

<sup>1</sup>Some results, in particular Section IV of this research were published in *Proceedings of the ACM/IEEE Design Automation Conference (DAC) 2007*, pp. 726-731 [1]. This article presents several new contributions:

- 1) This article presents a new probabilistic technique which outperforms our earlier technique by a factor of five.
- 2) The approach is used to implement an admission controller, that is fully integrated in an FPGA MPSoC design flow.
- 3) A complete and composable system design flow is presented that allows addition of applications at run-time.

with the base station, sending and receiving short messages, and encoding and decoding voice; modern high-end mobile devices also act as a music and video player, camera and a complete personal digital assistant. To further complicate matters, the user also expects to be able to download applications at run-time that may be completely unknown to the system designer, for example, a security application running in the background to protect the mobile phone against theft. While some of these applications may not be so critical for the user-experience (e.g. browsing a web), others like playing video and audio are some functions where a reduced performance is easily noticed. Accurate performance prediction is therefore essential to be performed at run-time before starting a new application, and not always feasible at design-time.

To estimate the performance of multiple applications running concurrently, a design-time analysis has been proposed [7]. While a design-time analysis can sometimes provide good estimates for performance of all possible use-cases, it becomes harder with the increasing number of applications in the system. Further, it lacks the flexibility of adding new applications that have not been analyzed. To allow for such run-time addition of applications and deal with ever-increasing number of use-cases, a prediction mechanism is needed to ensure that when a new application is started, the existing applications and the starting application can still meet their performance requirements.

## B. Our Contribution

In this paper, we propose a technique to accurately predict performance of multiple applications executing on a multi-processor platform. The approach is very fast and can be used at run-time as has been demonstrated by our FPGA prototype. In our analysis, we model the applications as synchronous data flow (SDF) graphs, since this allows analysis of application properties like throughput, buffer-requirement, deadlock analysis, etc with ease. Each application contains a number of tasks that have a worst-case execution time. Our novel iterative probabilistic technique computes the expected waiting time when multiple tasks share a processing resource (The approach can be adapted for other types of resource like communication and memory as well). These waiting time estimates, together with the execution time are used to estimate the performance of applications at run-time. This performance prediction technique is used to implement an admission controller. When a new job is to be started, the admission controller checks the expected performance against the desired performance and makes a decision whether to admit the application or not.

This admission controller is integrated in MAMPS (Multi-Application Multi-Processor Synthesis) - an FPGA-based multiprocessor system generation flow [8]. The hardware needed for signaling and performance checking is also designed and instantiated in the flow automatically for a complete system generation. Further, this tool is available for use on-line, where anyone can upload their application models and a complete design for FPGAs (presently limited to Xilinx) is generated

which can be directly synthesized and executed on FPGA<sup>2</sup> [9].

Following are the key features of our admission controller.

- *Accurate*: The performance values predicted vary from the measured values by 2% on average and 3% at maximum.
- *Fast*: The algorithm has the complexity of  $O(n)$ , where  $n$  is the number of actors on each processor.
- *Scalable*: The algorithm is scalable in the number of actors per applications, the number of processing nodes and the number of applications in the system.
- *Suitable for Embedded Systems*: The algorithm requires very low memory and has low complexity making it ideal for implementation in embedded platforms.
- *Dynamic*: Our flow allows applications to be added at run-time without any prior knowledge at design-time.
- *Fully Integrated in FPGA synthesis flow*: The admission controller has been fully implemented on FPGA and integrated in automated MPSoC generation flow that proves its suitability for embedded platforms.

The remainder of the paper is organized as follows. Section II discusses related work about how performance analysis is done using SDF graphs traditionally - for single and multiple applications. Relevant research in resource management and the use of probability is also discussed in the same section. Section III explains how the system should be designed when multiple applications are to be supported, and applications are allowed to be added in the system at run-time. Section IV explains the probabilistic approach that is used to predict performance of multiple applications accurately. Section V explains the iterative probability technique that builds upon the probability technique to improve the accuracy of the technique even more. Section VI explains how the admission controller and the resource manager is integrated in the FPGA implementation flow. Section VII describes the experimental setup and results obtained, and finally, Section VIII presents major conclusions and gives directions for future work.

## II. RELATED WORK

In [10], the authors propose to analyze performance of a *single application* modeled as an SDF graph mapped on a multi-processor system by decomposing it into an homogeneous SDF graph (HSDFG) [11]. This can result in an exponential number of vertices [12], after which the throughput is calculated based on analysis of each cycle in the HSDFG [13]. Algorithms that have a polynomial complexity for HSDFGs, therefore have an exponential complexity for SDFGs. Algorithms have been proposed to reduce average case execution [14], but it still takes in practice  $O(n^2)$  time where  $n$  is the number of vertices in the graph. Extra edges can be added to model resource dependency such that a complete analysis taking resource dependency into account is possible. However, the number of ways this can be done even for a single application is exponential in the number of vertices [2]; for multiple applications the number of possibilities is endless. Further, only static order arbitration can be modeled using this

<sup>2</sup>A licensed Xilinx tool installation is still needed to synthesize the design

technique while the best performance of SDFG applications is obtained when actors are allowed to execute with least contention on their own [11]. Our approach allows for that behavior since no ordering is imposed.

For *multiple applications*, an approach that models resource contention by computing *worst-case-response-time* for TDMA scheduling (requires preemption) has been analyzed in [15]. This analysis also requires limited information from the other SDFGs, but gives a very conservative bound. As the number of applications increases, the bound increases much more than the average case performance. Further, this approach requires preemption for analysis. A similar worst-case analysis approach for round-robin is presented in [16], which also works on non-preemptive systems, but suffers from the same problem of lack of scalability. Real-time calculus has also been used to provide worst-case bounds for multiple applications [17][18][19]. Besides providing a very pessimistic bound, the analysis is also very intensive and requires a very high design-time effort. Our approach on the other hand is very simple. However, we should note that above approaches give a worst-case bound that is targeted at hard-real-time (RT) systems, while our estimation approach is aimed at designing soft-RT systems.

A common way to use probabilities for modeling dynamism in application is using stochastic task execution times [20][21][22]. In our case, however, we use probabilities to model the resource contention and provide estimates for the throughput of applications. This approach is orthogonal to the approach of using stochastic task execution times. In our approach we assume fixed execution time, though it is easy to extend this to varying task execution times as well. To the best of our knowledge, there is no efficient approach of analyzing multiple soft-RT applications on a non-preemptive heterogeneous multi-processor platform.

Recently, quite some work has been in the context of resource management for multi-processor systems [23][24][25]. The work in [23] only considers preemptive systems, while our work is targeted at non-preemptive systems. Non-preemptive systems are harder to analyze since the interference of other applications has to be taken into account. The work in [24] presents a run-time manager for MPSoC platforms, but they only consider one task mapped on one tile in the system; they do not allow sharing of processors. In [25] the authors deal with non-preemptive heterogeneous platforms where processors are shared, but only discuss the issue of budget enforcement and not of admission control.

The authors in [26] motivate the use of a scenario-oriented (or *use-case* in our paper) design flow for heterogeneous MPSoC platforms. They propose to analyze the scenarios at design-time. However, with the need to add applications at run-time, a design-flow is needed that can accommodate this dynamic addition of applications. We present such a flow in this paper.

### III. DESIGNING SYSTEMS WITH MULTIPLE APPLICATIONS

In this section, we explain our proposed flow for designing systems with multiple applications. The approach is designed

such that the system and the analysis remains composable. We *define composability as being able to reason about application behaviour using as little information from the other applications as possible*. This allows applications to be analyzed largely in isolation from other applications. The compute-intensive property derivation can be done off-line since no information from other applications is needed. These properties can then be used by the run-time manager to determine the system behavior when all the applications execute together.

As explained in Section I-A, it is often not feasible to know the complete set of applications that the system will execute. Even in cases, when the set of applications is known at design-time, the number of potential use-cases (or scenarios) may be large. We propose a combination of off-line and on-line (same as run-time) processing, such that the design-effort remains contained. Note that off-line is different from design-time; while system design-time is limited to the time until the system is rolled-out, off-line can also overlap with using the system. In a mobile phone for example, even after a consumer has already bought the mobile phone, he/she can download the applications whose properties may have been derived after the phone was already designed. In our methodology, all applications may not be known at design-time either. In those cases the properties of the applications are derived off-line, and the run-time manager checks whether the given application-mix is feasible.

As mentioned earlier in our analysis, we model the applications as a synchronous data flow (SDF) graph, since this allows analysis of various application properties like throughput, buffer-requirement, deadlock analysis, etc with ease. The following sub-section gives a quick overview of SDF graphs.

#### A. Synchronous Data Flow Graphs

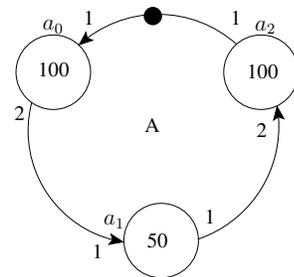


Fig. 1. Example of an SDF Graph

Synchronous Data Flow Graphs (SDFGs, see [27]) are often used for modeling modern DSP applications [11] and for designing concurrent multimedia applications implemented on multi-processor system-on-chip. Both pipelined streaming and cyclic dependencies between tasks can be easily modeled in SDFGs. Tasks are modeled by the vertices of an SDFG, which are called *actors*. SDFGs allow one to analyze a system in terms of throughput and other performance properties, e.g. latency, buffer requirements [28].

Figure 1 shows an example of an SDF Graph. There are three actors (also known as tasks) in this graph. As in a typical

data flow graph, a directed edge represents the dependency between actors. Actors also need some input data (or control information) before they can start and usually also produce some output data; such information is referred to as *tokens*. The number of tokens produced or consumed in one execution of actor is called *rate*. In the example,  $a_0$  has an input rate of 1 and output rate of 2. Actor execution is also called *firing*. An actor is called *ready* when it has sufficient input tokens on all its input edges and sufficient buffer space on all its output channels; an actor can only fire when it is ready.

The edges may also contain *initial tokens*, indicated by bullets on the edges, as seen on the edge from actor  $a_2$  to  $a_0$  in Figure 1. Buffer-sizes may be modeled as a back-edge with initial tokens. In such cases, the number of tokens on that edge indicates the buffer-size available. When an actor writes data on a channel, the available size reduces; when the receiving actor consumes this data, the available buffer increases, modeled by an increase in the number of tokens.

One of the most interesting properties of SDFGs relevant to this paper is throughput. Throughput is defined as the inverse of the long term period, i.e. the average time needed for one iteration of the application. (An iteration is defined as the minimum non-zero execution such that the original state of the graph is obtained.) This is the performance parameter that we use in this paper. More information and formal definitions can be found in Ref [7][11]. Following are the definitions most relevant for this paper.

*Definition 1: (ACTOR EXECUTION TIME)* Actor execution time,  $\tau(a)$  is defined as the time needed to complete execution of actor  $a$  on a specified node.  $\tau(a_0) = 100$ , for example, in Figure 1.

*Definition 2: (REPETITION VECTOR)* Repetition Vector  $q$  of an SDFG  $A$  is defined as the vector specifying the number of times an actor in  $A$  is executed for one iteration of  $A$ . For example, in Figure 1,  $q[a_0 \ a_1 \ a_2] = [1 \ 2 \ 1]$ .

*Definition 3: (APPLICATION PERIOD)* Application Period  $Per(A)$  is defined as the time SDFG  $A$  takes to complete one iteration on average.  $Per(A) = 300$  in Figure 1. (Note that actor  $a_1$  has to execute twice.) This is also equivalent to the inverse of throughput. An application with a throughput of 50 Hz takes 20 ms to complete one iteration.

In the following sub-sections we explain how and which properties are derived off-line from the applications, and how they can be used at run-time.

### B. Off-line Derivation of Properties

Figure 2 shows what properties from the application(s) are derived off-line. Individual applications are partitioned into tasks with respective program code tagged to each task and communication between them explicitly specified. A number of techniques are present in literature to do this partitioning. Compaan [29] is one such example that converts sequential description of an application into concurrent tasks by doing static code analysis and transformation. Sprint also allows code partitioning by letting the users tag the functions which are to be split into different actors [30]. Yet another technique has been presented that is based on execution profile [31]. The

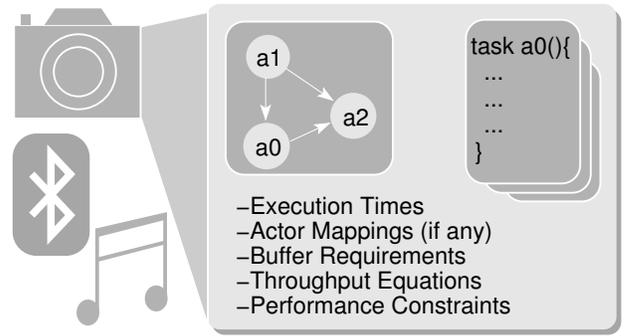


Fig. 2. Off-line application(s) partitioning and computation of application(s) properties. Three applications - photo taking, bluetooth and music playing are shown above. The partitioning and property derivation is done for all of them, as shown for photo taking application, for example.

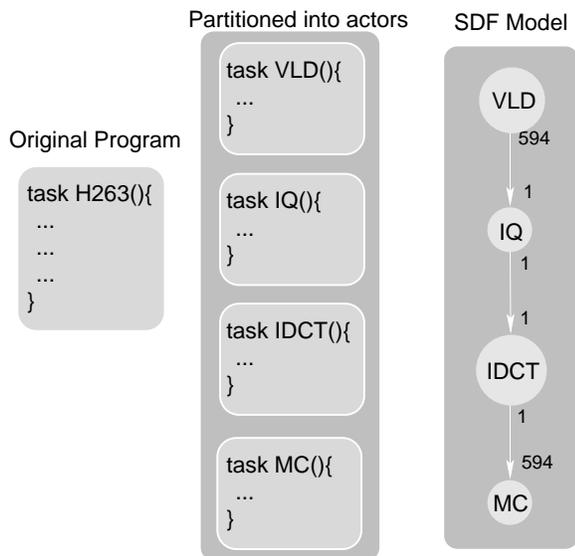
program code can be profiled (or statically analyzed) to obtain execution time estimates for the actors. For this paper, we shall assume that the application is modeled as a synchronous data flow graph, i.e. the application is already split into tasks with worst case execution time estimates.

Throughput computation of an SDF graph is very time consuming as explained in Section II. This is therefore often done off-line or at design-time for a particular graph. However, if the execution time of an actor changes, the entire analysis has to be repeated. Recently, a technique has been proposed to derive throughput equations for a range of execution times at design-time and these equations can be easily evaluated at run-time to compute the limiting cycle and hence the period [32].

As shown in Figure 2, following information is extracted from the application off-line.

- Partitioned program code into tasks
- SDF model of the application
- Mapping of these tasks on to the heterogeneous platform
- Buffer-sizes needed for the edges in the graph
- Throughput equations of the model
- Worst-case execution time estimates of each task
- Minimum performance (throughput) permissible for satisfactory user-experience

Note that there may be multiple pareto points with different mappings, buffer-sizes and throughput equations. Figure 3, for example, shows how the application partitioning and analysis is done for H263 decoder application. The sequential application code is split into task-level description, and an SDF model is derived for these communicating tasks. The corresponding production and consumption rates are also mentioned along the edges. The table alongside the figure shows the mapping and worst case execution times of each task. The buffer-size needed between each actor is also mentioned in the table. There are two throughput expressions that correspond to this buffer-size [28]. The minimum performance associated with this application is 25 frames per second. This is the constraint that should be respected when the application is executed. For these initial execution time estimates, the first expression forms the bottleneck and determines the period to be 646262 cycles. This implies that if each of these tasks is executed on a processor of 50 MHz, the maximum throughput of the



Overall Constraint: 25 fps

$$T_1 = 0 \times t_{vld} + 593 \times t_{iq} + 594 \times t_{idct} + 1 \times t_{mc}$$

$$T_2 = 1 \times t_{vld} + 594 \times t_{iq} + 593 \times t_{idct} + 0 \times t_{mc}$$

Task	Mapping	Execution cycles	Min outgoing buffer
VLD	ARM7	26018	594 tokens
IQ	ARM9	559	1 tokens
IDCT	TIC67	486	594 tokens
MC	TIC64	10958	-

Fig. 3. The properties of H263 decoder application computed off-line

application is 77 iterations per second<sup>3</sup>. Clearly, when this application is executing concurrently with other applications, it may not be possible.

An application can often be associated with multiple quality levels as has been explained in existing literature [33][34]. Each quality of the application will in that case be depicted with a different task graph with (potentially) different requirements of resources and different performance constraints. For example, a bluetooth application may be able to run at a higher or lower data rate depending on the availability of the resources. If a bluetooth device wants to connect to a mobile phone which is already running a lot of jobs in parallel, it may not be able to start at 3.0 Mbps (Bluetooth 2.0 specification [35]) due to degraded performance of existing applications, but only at 1.0 Mbps (Bluetooth 1.2 specification [35]). We consider these two as separate applications (except that these two are unlikely to execute together).

### C. On-line Resource Manager

A resource manager, as the name suggests, is needed for managing the diverse resources available in the platform. Typically it takes care of resource assignment, budget assignment and enforcement, and admission control. When an actor, for example, can be mapped on multiple processors, or when

<sup>3</sup>In practice, the frequency of different processors may be different. In that case, we should add time taken for each task in throughput expressions instead of cycles.

there are multiple of the same processor instances available, it chooses which one to assign the actor to. It also assigns and enforces budgets on say, for example, shared communication resources like a bus or on-chip network e.g.  $\text{\AA}$ ethereal [36]. However, for the scope of this paper, we focus on the task of admission control, i.e. to determine if a particular application should be admitted or not. Further, when an actor can be mapped on multiple resources (either because it can be mapped on different types of processors, or because there are multiple instances of the type of processors it can be mapped on, or both), we assume that the resource manager (or the compiler/designer) has already done the assignment. It is possible that while assignment to one processor makes an application non-admissible, another assignment would have potentially allowed the application to be admitted. Heuristics to explore mapping options are orthogonal to our approach and have been left out of the scope of this paper. Such heuristics can be used in combination with our approach. Here we assume that a mapping is already provided, and we are interested in finding out if the application can be admitted in the system with that mapping.

*Performance Predictor:* The performance predictor runs as part of admission controller and uses the off-line information of the applications to predict their performance at run-time. For example, imagine a scenario where you are in the middle of a phone call with your friend and you are streaming some mp3 music via the 3G connection to your friend, and at the same time synchronizing your calendar with the PC using bluetooth. If you also wish to now take a picture of your surrounding, traditional systems will simply start the application without considering whether there are enough resources to meet the requirements or not. As shown in Figure 4, with so many applications in the system executing concurrently, it is very likely that the performance of the camera and the bluetooth application may not be able to match their requirements.

With the on-line predictor, using the properties of applications computed off-line, we can check what is the expected performance before admitting the application. It can then be decided to either drop the incoming application, or perhaps try the incoming application (or one of the existing applications, if allowed) at a lower quality level. As shown in Figure 4, if the camera application is tested at 2.0 MPixel requirements, all the applications can meet their requirements. It is much better to know in advance and take some corrective measure, or simply warn the user that the system will not be able to cope up with these set of applications.

It can be seen how this flow allows addition of applications at run-time without sacrificing predictability. The user can download new applications as long as the application is analyzed off-line and the properties mentioned earlier are derived. Since the performance analysis is done at run-time, no extensive testing is needed at design-time to verify which applications will meet their performance requirements and which not.

## IV. PROBABILISTIC ANALYSIS

The on-line prediction mechanism needs a mechanism that can predict accurately the performance of multiple applications

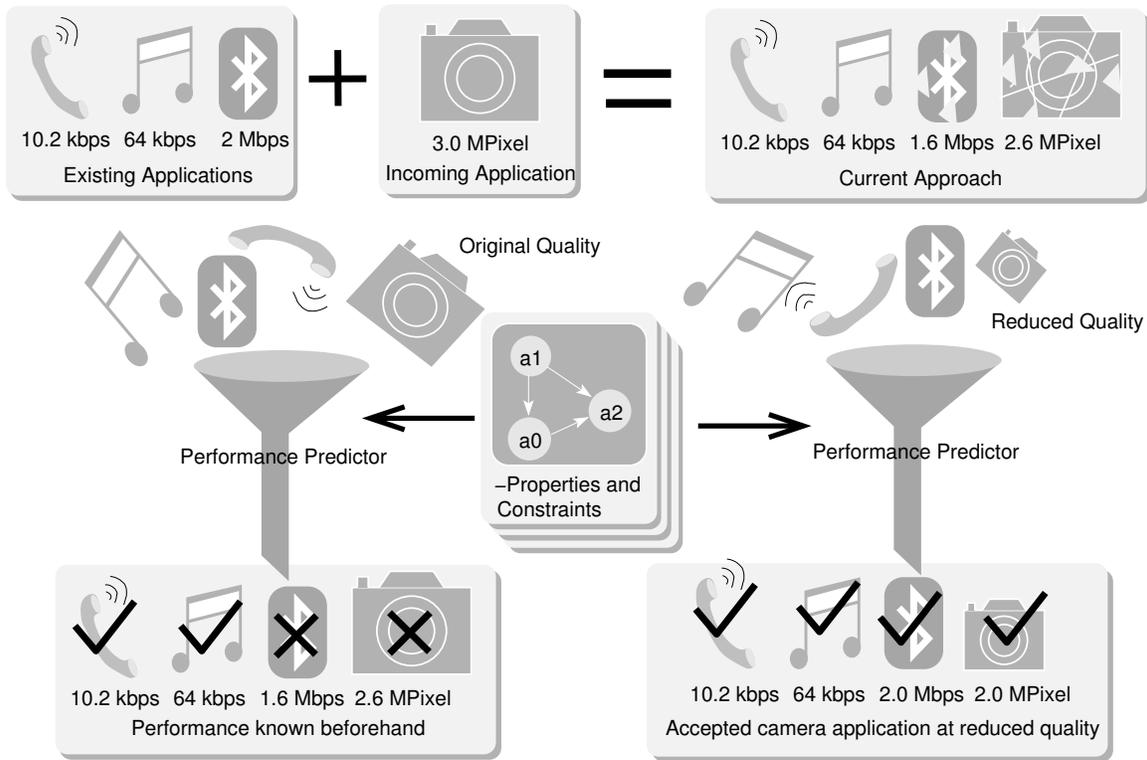


Fig. 4. On-line predictor for multiple application(s) performance

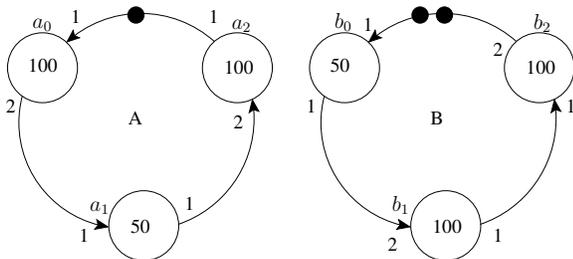


Fig. 5. Two application SDFGs A and B

executing concurrently on a heterogeneous multiprocessor platform. When multiple applications execute in parallel, it causes contention for the resources. Our probabilistic mechanism predicts this contention before the applications are actually executed. The time spent by an actor in contention is added to the execution time, and the total gives the response time. The equation below puts it more clearly.

$$t_{resp} = t_{exec} + t_{wait} \quad (1)$$

The  $t_{wait}$  is the time that is spent in contention when waiting for the resource to become free. The response time,  $t_{resp}$  indicates how long it takes to process an actor after it arrives on a node. When there is no contention, the response time is simply equal to the execution time. Using only the execution time gives us the maximum throughput that can be achieved with the given mapping. At design-time, since the run-time application-mix is not known, it is not possible to accurately predict the waiting-time, and hence the performance. In this section, we explain how this estimate is obtained using

probability.

We now refer to SDFGs  $A$  and  $B$  in Figure 5. Say  $a_0$  and  $b_0$  are mapped on a processor  $Proc_0$  and others have dedicated resources.  $a_0$  is active for time  $\tau(a_0)$  every  $Per(A)$  time units (since its repetition entry is 1). In Figure 5,  $\tau(a_0) = 100$  time units and  $Per(A) = 300$  time units.

The probability that  $Proc_0$  is used by  $a_0$  at any given time is  $\frac{100}{300} = \frac{1}{3}$ , since  $a_0$  is active for 100 cycles out of every 300 cycles. Since arrival of  $a_0$  and  $b_0$  are independent, this is also the probability of  $Proc_0$  being occupied when  $b_0$  arrives at it. Further, since  $b_0$  can arrive at any arbitrary point during execution of  $a_0$ , the time  $a_0$  takes to finish after  $b_0$  arrives on the node is uniformly distributed from  $[0, 100]$ . Therefore,  $b_0$  has to wait for 50 time units on average if  $Proc_0$  is found blocked. Since the probability that the resource is occupied is  $\frac{1}{3}$ , the average time actor  $b_0$  has to wait is given by  $\frac{50}{3} \approx 17$  time units. The expected response time of  $b_0$  will therefore be  $\approx 67$  time units.

#### A. Generalizing the Analysis

This sub-section generalizes the analysis presented above. As we can see in the above analysis, each actor has two attributes associated with it: 1) the probability that it blocks the resource and 2) the average time it takes before freeing up the resource it is blocking. In view of this we define the following terms:

*Definition 4: (BLOCKING PROBABILITY) Blocking Probability,  $P(a)$  is defined as the probability that actor  $a$  of application  $A$  blocks the resource it is mapped on.  $P(a) =$*

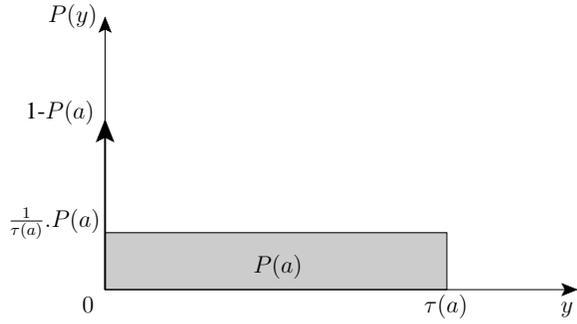


Fig. 6. Probability distribution of waiting time another actor has to wait when actor  $a$  is mapped on the resource.

$\tau(a) \cdot q(a) / Per(A)$ .  $P(a_0) = \frac{1}{3}$  in Figure 5.  $P(a)$  is also represented as  $P_a$  interchangeably.

**Definition 5: (AVERAGE BLOCKING TIME)** Average Blocking Time,  $\mu(a)$  is defined as the average time before the resource blocked by actor  $a$  is freed given the resource is found to be blocked. Again,  $\mu(a)$  is also represented as  $\mu_a$  interchangeably.  $\mu(a) = \tau(a)/2$  for constant execution time. In Figure 5,  $\mu(a_0) = 50$ .

If  $X$  denotes how long an actor  $b$  has to wait if the resource  $b$  is requesting is being blocked by actor  $a$ , the probability density function,  $w(x)$  of  $X$  can be defined as follows.

$$w(x) = \begin{cases} 0, & x \leq 0 \\ \frac{1}{\tau(a)}, & 0 < x \leq \tau(a) \\ 0, & x > \tau(a) \end{cases} \quad (2)$$

The average time  $b$  has to wait given resource is blocked, or  $\mu_a$  is therefore,

$$\begin{aligned} E(X) &= \int_{-\infty}^{\infty} x w(x) dx \\ &= \int_0^{\tau(a)} x \frac{1}{\tau(a)} dx \\ &= \frac{1}{\tau(a)} \left[ \frac{x^2}{2} \right]_0^{\tau(a)} \\ &= \frac{\tau(a)}{2} \end{aligned} \quad (3)$$

Figure 6 shows the overall probability distribution of  $b$  waiting for a resource that is shared with  $a$ . This includes a delta function of value  $1 - P(a)$  at the origin since that is the probability of the resource being available (not being occupied by  $a$ ) when  $b$  wants to execute. Clearly, the total area under the curve is 1, and the expected value of this variable gives the overall expected waiting time of  $b$  and can be computed as

$$t_{wait}(b) = E(Y) = \frac{\tau(a)}{2} \cdot P(a) \quad (4)$$

Let us revisit our example in Figure 5. Let us now assume actors  $a_i$  and  $b_i$  are mapped on  $Proc_i$  for  $i = 0, 1, 2$ . The blocking probabilities for actors  $a_i$  and  $b_i$  for  $i = 0, 1, 2$  are

$$\begin{aligned} P(a_i) &= \frac{\tau(a_i) \cdot q(a_i)}{Per(A)} = \frac{1}{3} \text{ for } i = 0, 1, 2, \\ P(b_i) &= \frac{\tau(b_i) \cdot q(b_i)}{Per(B)} = \frac{1}{3} \text{ for } i = 0, 1, 2. \end{aligned}$$

The average blocking time of actors in Figure 5 is

$$[\mu_{a_0} \mu_{a_1} \mu_{a_2}] = [50 \ 25 \ 50] \text{ and } [\mu_{b_0} \mu_{b_1} \mu_{b_2}] = [25 \ 50 \ 50]$$

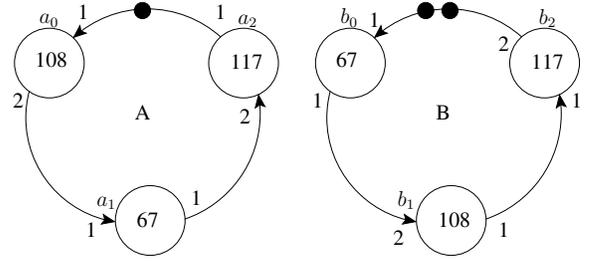


Fig. 7. SDFGs A and B with response times

In this case, since only one other actor is mapped on every node, the waiting time for each actor is easily derived.

$$\begin{aligned} t_{wait}(b_i) &= \mu(a_i) \cdot P(a_i) \text{ and } t_{wait}(a_i) = \mu(b_i) \cdot P(b_i) \\ t_{wait}[b_0 \ b_1 \ b_2] &= \left[ \frac{50}{3} \ \frac{25}{3} \ \frac{50}{3} \right] \text{ and } t_{wait}[a_0 \ a_1 \ a_2] = \left[ \frac{25}{3} \ \frac{50}{3} \ \frac{50}{3} \right] \end{aligned}$$

Figure 7 shows the response time of all actors taking waiting times into account. The new period of SDFG A and B is computed as 359 time units for both. In practice, the period that these application graphs would achieve is actually 300 time units. However, it must be noted that in our entire analysis we have ignored the intra-graph actor dependency. For example, if the cyclic dependency of SDFG B was changed to clockwise, all the values computed above would remain the same while the period of the graphs would change. The period then becomes 400 time units. The probabilistic estimate we have now obtained in this simple graph is roughly equal to the mean of period obtained in either of the cases.

Further, in this analysis we have assumed that arrival of actors on a node is independent. In practice, this assumption is not always valid. Resource contention will inevitably make the independent actors dependent on each other. Even so, the approach works very well, as we see in Section VII. A rough sketch of the algorithm used in our approach is outlined in Figure 8.

- 
- 1:  $a_{ij}$  is actor  $j$  of application  $A_i$
  - 2: **for all** actors  $a_{ij}$  **do**
  - 3:      $P(a_{ij}) = \text{BlockingProb}(\tau(a_{ij}), q(a_{ij}), Per(A_i))$
  - 4: **end for**
  - 5: //Now use this to compute waiting time
  - 6: **for all** Applications  $A_i$  **do**
  - 7:     **for all** Actors  $a_{ij}$  of  $A_i$  **do**
  - 8:          $t_{wait}(a_{ij}) = \text{WaitingTime}(\tau, P)$
  - 9:          $\tau(a_{ij}) = \tau(a_{ij}) + t_{wait}(a_{ij})$
  - 10:     **end for**
  - 11:      $Per(A_i) = \text{NewPeriod}(A_i)$
  - 12: **end for**
- 

Fig. 8. Algorithm for estimating Period using blocking probabilities

### B. Extending to $N$ Actors

Let us assume actors  $a$ ,  $b$  and  $c$  are mapped on the same node, and that we need to compute the waiting time for  $c$ .  $c$  may be blocked by either  $a$  or  $b$  or both. Analyzing the case of  $c$  being blocked by both  $a$  and  $b$  is slightly more complicated.

There are two sub-cases for it - one in which  $a$  is being served and  $b$  is queued, and another in which  $b$  is being served and  $a$  is queued. We therefore have four possible cases.

*Blocking only by a:*

$$t_{wait}(c_1) = \mu_a \cdot P_a \cdot (1 - P_b)$$

*Blocking only by b:*

$$t_{wait}(c_2) = \mu_b \cdot P_b \cdot (1 - P_a)$$

*a being served, b queued:* The time spent by  $b$  waiting behind  $a$  is given by  $\mu_a \cdot P_a$ . Therefore, the total probability of  $b$  behind  $a$  is,

$$P_{wait}(c_3) = \mu_a \cdot P_a \cdot \frac{q[b]}{Per(b)} = P_a \cdot P_b \cdot \frac{\mu_a}{2 \cdot \mu_b},$$

and the corresponding time is,

$$t_{wait}(c_3) = P_a \cdot P_b \cdot \frac{\mu_a}{2 \cdot \mu_b} \cdot (\mu_a + 2\mu_b)$$

*b being served, a queued:* This can be derived similar to above as follows:

$$t_{wait}(c_4) = P_b \cdot P_a \cdot \frac{\mu_b}{2 \cdot \mu_a} \cdot (\mu_b + 2\mu_a)$$

The time that  $c$  needs to wait when two actors are in queue varies depending on which actor is being served. For example, if  $a$  is ahead in the queue,  $c$  has to wait for  $\mu_a$  due to  $a$ , since  $a$  is being served. However, since the entire  $b$  remains to be served after  $a$  is finished,  $c$  needs to wait  $2 \cdot \mu_b$  for  $b$ . One can also observe that the waiting time due to actor  $a$  is  $\mu_a \cdot P_a$  when it is in front, and  $2 \cdot \mu_a \cdot P_a$  when behind. Adding all the above equations, we get

$$\begin{aligned} t_{wait}(c) &= \frac{1}{2} \cdot P_a \cdot P_b \cdot \left( \frac{\mu_a^2}{\mu_b} + \frac{\mu_b^2}{\mu_a} \right) + \mu_a \cdot P_a + \mu_b \cdot P_b \\ &= \mu_a \cdot P_a \cdot \left( 1 + \frac{\mu_a}{2\mu_b} P_b \right) + \mu_b \cdot P_b \cdot \left( 1 + \frac{\mu_b}{2\mu_a} P_a \right) \end{aligned}$$

In most cases, the execution time of actors are of similar granularity. Further, we observe that the probability terms (that are often  $< 1$ ) are multiplied. To make the analysis easier, we therefore assume that the probability of  $a$  behind  $b$ , and  $b$  behind  $a$  are nearly equal (which becomes even more true when tasks are of equal granularity, since then  $\mu_a \approx \mu_b$ . This assumption is not needed for the iterative analysis). Therefore, the above equation can be approximated as,

$$\begin{aligned} t_{wait}(c) &= \frac{1}{2} \cdot P_a \cdot P_b \cdot (\mu_a + \mu_b) + \mu_a \cdot P_a + \mu_b \cdot P_b \\ &= \mu_a \cdot P_a \cdot \left( 1 + \frac{1}{2} P_b \right) + \mu_b \cdot P_b \cdot \left( 1 + \frac{1}{2} P_a \right) \end{aligned}$$

The above can be also computed by observing that whenever an actor  $a$  is in the queue, the waiting time is simply  $\mu_a \cdot P_a$ , i.e. the probability of  $a$  being in the queue (regardless of other actors) and the waiting time due to it. However, when it is behind some other actor, there is an extra waiting time  $\mu_a$ , since the whole of  $a$  has to be executed. The probability of  $a$  being behind  $b$  is  $\frac{1}{2} \cdot P_a \cdot P_b$  and hence the total waiting time due to  $a$  is  $\mu_a \cdot P_a \cdot \left( 1 + \frac{1}{2} P_b \right)$ . The same follows for the contribution due to  $b$ .

TABLE I  
PROBABILITIES OF DIFFERENT QUEUES WITH  $a$

Queue	Probability (excl $P_a$ )	Extra waiting prob
a	$(1 - P_b)(1 - P_c)$	
ab	$P_b(1 - P_c)/2$	
ba	$P_b(1 - P_c)/2$	$P_b(1 - P_c)/2$
ac	$P_c(1 - P_b)/2$	
ca	$P_c(1 - P_b)/2$	$P_c(1 - P_b)/2$
abc-acb	$P_b \cdot P_c / 3$	
bca-cba	$\frac{2}{3} P_b \cdot P_c$	$\frac{2}{3} P_b \cdot P_c$
bac-cab		
<b>Total</b>		$\frac{1}{2}(P_b + P_c) - \frac{1}{3}P_b \cdot P_c$

For three actors waiting in the queue, it is best explained using a table. Table I shows all the possibilities of queue with  $a$  in it. The first column contains the ordering of actors in the queue, where the leftmost actor is the first one in the queue. All the possibilities are shown in it together with their probabilities. Please note that since  $a$  is in all the queues, the probability component  $P_a$  has been excluded. For the cases when  $a$  is not in front, the waiting time is increased by  $\mu_a \cdot P_a$ , and therefore, those probability terms are added again. The same can be easily derived for other actors too. We therefore obtain the following equation.

$$\begin{aligned} \mu_{abc} \cdot P_{abc} &= \mu_a \cdot P_a \cdot \left( 1 + \frac{1}{2}(P_b + P_c) - \frac{1}{3}P_b \cdot P_c \right) \\ &\quad + \mu_b \cdot P_b \cdot \left( 1 + \frac{1}{2}(P_a + P_c) - \frac{1}{3}P_a \cdot P_c \right) \\ &\quad + \mu_c \cdot P_c \cdot \left( 1 + \frac{1}{2}(P_a + P_b) - \frac{1}{3}P_a \cdot P_b \right) \end{aligned} \quad (5)$$

It can be further generalized for  $n$  actors  $a_1, a_2, \dots, a_n$  mapped on a resource to give

$$\begin{aligned} \mu_{a_1 \dots a_n} P_{a_1 \dots a_n} &= \sum_{i=1}^n \mu_{a_i} P_{a_i} \left( 1 + \sum_{j=1}^{n-1} \frac{(-1)^{j+1}}{j+1} \right. \\ &\quad \left. \prod_j (P_{a_1} \dots P_{a_{i-1}} P_{a_{i+1}} \dots P_{a_n}) \right) \end{aligned} \quad (6)$$

where  $\prod_j (x_1, \dots, x_n)$  is an elementary symmetric polynomial defined in [37]. We observe that as the number of actors mapped on a node increases, the complexity of analysis also becomes high. To be exact, the complexity of the above formula is  $O(n^{n+1})$ , where  $n$  is the number of actors mapped on a node. Since this is done for each actor, the overall complexity becomes  $O(n^{n+2})$ . In the next sub-section we show how this complexity can be reduced.

### C. Complexity Reduction

The complexity of the analysis plays an important role when putting an idea to practice. The total complexity for analysis in Equation 6 is  $O(n^{n+2})$ . Using some clever techniques for implementation the complexity can be reduced to  $O(n^2 + n^n)$  i.e.  $O(n^n)$ . This can be achieved by modifying the equation such that we first compute  $\prod_j (P_{a_1}, P_{a_2} \dots P_{a_n})$  including  $P_{a_i}$ . The extra component is then subtracted from the total for each  $a_i$  separately.

However, this is still infeasible and not scalable. An important observation that can be made is that higher order terms

start to appear in our analysis. The number of these terms in  $\Pi_j$  in Equation 6 increases exponentially. Since these terms are products of probabilities, higher order terms can likely be neglected. To limit the computational complexity, we provide a second order approximation of the formula.

$$\mu_{a_1 \dots a_n} P_{a_1 \dots a_n} \approx \sum_{i=1}^n \mu_{a_i} P_{a_i} \left( 1 + \frac{1}{2} \sum_{j=1, j \neq i}^n (P_{a_j}) \right)$$

The complexity of the above formula is  $O(n^3)$ , since we have to do it for  $n$  actors. For the above equation, we can modify the summation inside the loop such that the complexity is reduced. The new formula is re-written as

$$\mu_{a_1 \dots a_n} P_{a_1 \dots a_n} \approx \sum_{i=1}^n \mu_{a_i} P_{a_i} \left( 1 + \frac{1}{2} (Tot\_Summ - P_{a_i}) \right) \quad (7)$$

where

$$Tot\_Summ = \sum_{j=1}^n P_{a_j}$$

This makes the overall complexity  $O(n^2)$ . In general, the complexity can be reduced to  $O(n^m)$  for  $m \geq 2$  by using  $m$ -th order approximation. In Section VII we present results of second and fourth order approximations.

1) *Composability-based Approach*: In this approach, two actors are *composed* into one actor such that the properties of this new actor can be approximated by the sum of their individual properties. In particular, if we have two actors  $a$  and  $b$ , we would like to know their combined blocking probability  $P_{ab}$ , and combined waiting time due to them  $\mu_{ab} \cdot P_{ab}$ . We further define this composability operation for probability by  $\oplus$  and for waiting time by  $\otimes$ . We therefore get,

$$P_{ab} = P_a \oplus P_b = P_a + P_b - P_a \cdot P_b \quad (8)$$

$$\mu_{ab} \cdot P_{ab} = \mu_a \cdot P_a \otimes \mu_b \cdot P_b = \mu_a \cdot P_a \cdot \left( 1 + \frac{P_b}{2} \right) + \mu_b \cdot P_b \cdot \left( 1 + \frac{P_a}{2} \right) \quad (9)$$

(Strictly speaking  $\otimes$  operation also requires individual probabilities of the actors as inputs, but this has been omitted in the notation for simplicity.) Associativity of  $\oplus$  is easily proven by showing  $P_{abc} = P_{ab} \oplus P_c = P_a \oplus P_{bc}$ . Operation  $\otimes$  is associative only to second order approximation. This can be proven in a similar way by showing  $\mu_{abc} P_{abc} = \mu_{ab} P_{ab} \otimes \mu_c P_c = \mu_a P_a \otimes \mu_{bc} P_{bc}$ .

Associative property of these operations reduces the complexity even further. Complexity of Equation 8 and 9 is clearly  $O(1)$ . If waiting time of a particular actor is to be computed, all the other actors have to be combined giving a total complexity of  $O(n^2)$ , which is equivalent to the complexity of second-order approximation approach. However, in this approach the effect of actors is incrementally added. Therefore, when a new application has to be added to the analysis and new actors are added to the nodes, the complexity of the computation is  $O(n)$  as compared to  $O(n^2)$  in the case of second-order approximation, for which the entire analysis has to be repeated.

2) *Computing inverse of Formulae*: The complexity of this *Composability-based* approach can be further reduced when we can compute the inverse of the formulae in Equation 8 and 9. When the inverse function is known, all the actors can be *composed* into one actor by deriving their total blocking probability and total average blocking time. To compute the individual waiting time, only the inverse operation with their own parameters has to be performed. The total complexity of this approach is  $O(n) + n \cdot O(1) = O(n)$ . The inverse is also useful when applications enter and leave the analysis, since only an incremental *add* or *subtract* has to be done to update the waiting time instead of computing all the values.

The inverse for both operations are given below.

$$\begin{aligned} P_{a_1 \dots a_n b} &= P_{a_1 \dots a_n} \oplus P_b \\ \Rightarrow P_{a_1 \dots a_n} &= P_{a_1 \dots a_n b} \oplus^{-1} P_b = \frac{P_{a_1 \dots a_n b} - P_b}{1 - P_b} \quad (P_b \neq 1) \end{aligned} \quad (10)$$

$$\begin{aligned} \mu_{a_1 \dots a_n b} P_{a_1 \dots a_n b} &= \mu_{a_1 \dots a_n} P_{a_1 \dots a_n} \otimes \mu_b P_b \\ \Rightarrow \mu_{a_1 \dots a_n} P_{a_1 \dots a_n} &= \mu_{a_1 \dots a_n b} P_{a_1 \dots a_n b} \otimes^{-1} \mu_b P_b \\ \Rightarrow \mu_{a_1 \dots a_n} P_{a_1 \dots a_n} &= \frac{\mu_{a_1 \dots a_n b} P_{a_1 \dots a_n b} - \mu_b \cdot P_b \left( 1 + \frac{P_{a_1 \dots a_n}}{2} \right)}{1 + \frac{P_b}{2}} \end{aligned} \quad (11)$$

It should be mentioned that the inverse formula can only be applied when  $P_b \neq 1$ .

## V. ITERATIVE ANALYSIS

The iterative analysis takes advantage of two facts observed in the previous sections.

- An actor contributes to the waiting time for another actor in two ways - while it is being executed, and while it is waiting for the resource to become free.
- The application behavior itself changes when executing concurrently with other applications. In particular the period of the application changes (increases as compared to original period) when executing concurrently with interfering applications.

The increase in application period implies that the actors request the resource less frequently than analyzed in the earlier analysis. The application period as defined in Definition 3 is modified due to the difference in actor response times leading to a change in the actor blocking probability. Further, an actor can block another actor in two ways. Therefore, we define two different blocking probabilities.

*Definition 6: (EXECUTION BLOCKING PROBABILITY) Execution Blocking Probability,  $P_e(a)$  is defined as the probability that actor  $a$  of application  $A$  blocks the resource it is mapped on, and is being executed.  $P_e(a) = \tau(a) \cdot q(a) / Per_{New}(A)$ .  $P_e(a_0) = \frac{100}{359}$  in Figure 5, since  $Per_{New}(A) = 359$ .*

*Definition 7: (WAITING BLOCKING PROBABILITY) Waiting Blocking Probability,  $P_w(a)$  is defined as the probability that actor  $a$  of application  $A$  blocks the resource it is mapped on while waiting for it to become available.  $P_w(a) = t_{wait}(a) \cdot q(a) / Per_{New}(A)$ .  $P_w(a_0) = \frac{8}{359}$  in Figure 5.*

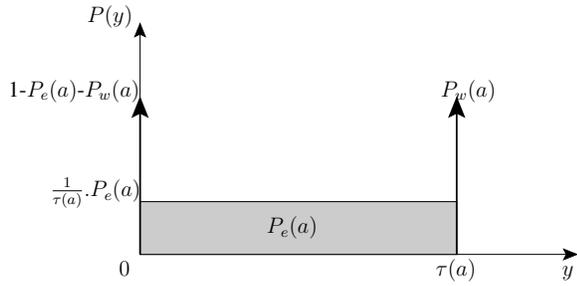


Fig. 9. Probability distribution of waiting time another actor has to wait when actor  $a$  is mapped on the resource with explicit waiting time probability.

When an actor arrives at a particular processor, it can either find a particular other actor being served, waiting in the queue or not in the queue at all. If an actor arrives when the other actor is waiting, then it has to wait for the entire execution time of that actor (since it is queued at the end). On the other hand when the actor is being served, the average waiting time due to that actor is half of the total execution time as shown in Equation 3.

There is a fundamental difference with the analysis presented in Section IV. In the earlier analysis an actor had two states - requesting a resource and not requesting a resource. In this analysis, there are three states - waiting in queue on the resource, executing on the resource and not requesting it at all. This explicit state of waiting for the resource, combined with the updated period, makes the blocking effect on another actor more accurate, and also understanding the analysis easier.

Figure 9 shows the updated probability distribution of the waiting time contributed by an actor with three explicit states. There is now an extra delta function at  $\tau(a)$  due to the waiting state of  $a$  as compared to the earlier distribution in Figure 6.

Taking the example above as shown in Figure 7, the new periods as computed from the probabilistic analysis in earlier sections are 359 time units for both  $A$  and  $B$ . So, we obtain

$$P_e[a_0 a_1 a_2] = \begin{bmatrix} 100 & 100 & 100 \\ 359 & 359 & 359 \end{bmatrix}, \quad P_e[b_0 b_1 b_2] = \begin{bmatrix} 100 & 100 & 100 \\ 359 & 359 & 359 \end{bmatrix}$$

$$P_w[a_0 a_1 a_2] = \begin{bmatrix} 8 & 34 & 17 \\ 359 & 359 & 359 \end{bmatrix}, \quad P_w[b_0 b_1 b_2] = \begin{bmatrix} 34 & 8 & 17 \\ 359 & 359 & 359 \end{bmatrix}$$

This gives the following waiting time estimates.

$$t_{wait}[a_0 a_1 a_2] = [11.7 \ 16.2 \ 18.6] \text{ and}$$

$$t_{wait}[b_0 b_1 b_2] = [16.2 \ 11.7 \ 18.6]$$

The period for both  $A$  and  $B$  evaluates to 362.7 time units. Repeating this analysis for another iteration gives the period as 364.3 time units. Repeating the analysis iteratively gives 364.14, 364.21, 364.19, 364.20, and 364.20 thereby converging at 364.20. In this example, we started our iterative analysis from the basic probabilistic estimate. If we simply start the analysis from the original graph, i.e. assuming no waiting time for the first iteration, we obtain the periods as 358.33, 362.79, 364.26, 364.14, 364.21, 364.20 and 364.20 again converging at 364.20 in about the same number of iterations. Figure 10 shows the updated application graphs after the iterative technique is applied.

For three actor system, when waiting time of an actor  $c$  has to be computed like above, the following formula can be derived from Figure 9. Note that  $\tau(a) = 2 \cdot \mu_a$ .

$$t_{wait}(c) = \mu_a \cdot P_e(a) + 2 \cdot \mu_a \cdot P_w(a) + \mu_b \cdot P_e(b) + 2 \cdot \mu_b \cdot P_w(b)$$

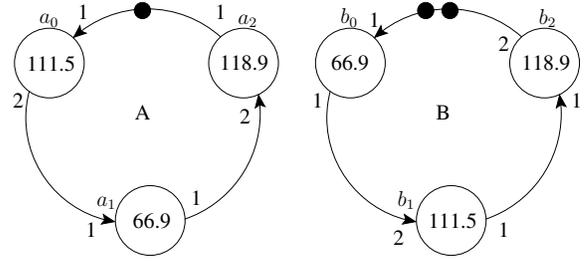


Fig. 10. SDF application graphs A and B updated after applying iterative analysis technique

For  $N$  actors the waiting time becomes as follows.

$$t_{wait} = \sum_{i=1}^n (\mu_{a_i} P_e(a_i) + 2\mu_{a_i} P_w(a_i)) \quad (12)$$

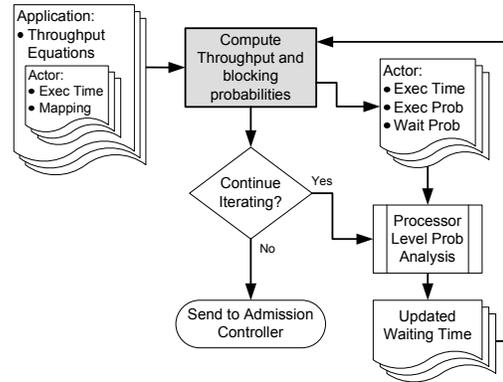


Fig. 11. Iterative Probability Method. Waiting times and throughput are updated until needed.

The change in period as mentioned earlier leads to a change in the execution and waiting probabilities of actors. This in turn, changes the response times of actors, which in turn may change the period. This very nature of this technique defines its name *iterative probability*. The cycle is therefore repeated until the period of all applications stabilises. Figure 11 shows the flow for iterative probability approach. The input to this flow is the output of the off-line flow - namely the application throughput expressions, and the execution time and mapping of each actor in all the applications. These, like in the approach mentioned earlier, are first used to compute the base period (i.e. the minimum period without any contention) and the blocking probability of the actor. Using the mapping information, a list of actors is compiled from all the applications and grouped according to their resource mapping. For each processor, the probability analysis is done according to Equation 12. The waiting time thus computed are used again to compute the throughput of the application and the blocking probabilities. The analysis can be run for a fixed number of iterations or terminate using some heuristic e.g. the maximum or average change in application period.

#### A. Conservative Iterative Analysis

For some applications, the user might be interested in having a conservative bound on the period. In such cases, we provide here a conservative analysis using our iterative technique. The



Fig. 12. Probability distribution of waiting time another actor has to wait when actor  $a$  is mapped on the resource with explicit waiting time probability for the conservative iterative analysis.

motivation behind this analysis is that for some applications, it is better to have a less accurate pessimistic estimate than an accurate optimistic estimate; a much better quality than predicted is more acceptable as compared to even a little worse quality than predicted.

In earlier analysis, when an actor  $b$  arrives at a particular resource and finds it occupied by say actor  $a$ , we assume that  $a$  can be anywhere in the middle of its execution, and therefore,  $b$  has to wait on average half of execution time of  $a$ . In the conservative approach, we assume that  $b$  has to always wait for full execution of  $a$ . In the probability distribution as presented in Figure 9, the rectangular uniform distribution of  $P_e(a)$  is replaced by another delta function at  $\tau(a)$  of value  $P_e(a)$ . This is shown in Figure 12. The waiting time equation is therefore updated to following.

$$t_{wait} = \sum_{i=1}^n 2\mu_{a_i} (P_e(a_i) + P_w(a_i)) \quad (13)$$

Applying this analysis to our earlier example starting from original graph, we obtain the periods as 416.7, 408, 410.3, 409.7, 409.8 and settles at that value. Starting from probabilistic analysis values it also settles at 409.8 in 5 iterations. Note that in our example, the actual period will be 300 in the best case and 400 in the worst case. The conservative iterative analysis correctly finds the bound of about 410, which is only 2.5% more than the actual worst case. If we apply real worst-case analysis in this approach, we would then get a period of 600 time units, which is 50% over-estimated.

In the following section, we explain the hardware implementation of the resource manager including the admission controller.

## VI. IMPLEMENTATION

We implemented the proposed resource manager with the admission controller on an FPGA-based multiprocessor design flow [8]. The flow is named MAMPS for Multi-Application Multi-Processor Synthesis. An overview of the existing flow is presented in Figure 13.

The flow generates multiprocessor systems from a specification of multiple applications. Applications are described in the form of SDF graphs in xml format. A snippet of application specification of *App10* is shown in Figure 14, corresponding to the application in Figure 13. The specification file contains details about how many actors are present in the application, and how they are connected to the other actors. The execution time of the actors and their memory usage on the processing

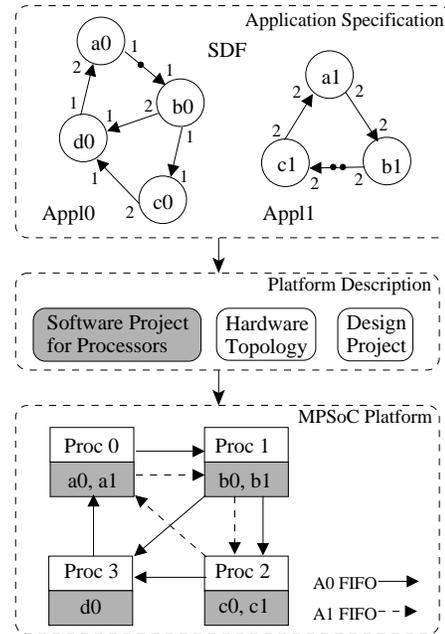


Fig. 13. Design flow

core is also specified. For each channel present in the graph, the file describes if there are any initial tokens present on it. The buffer capacity of a particular channel is specified as well.

```
<application id="App10">
  <actor name="a0">
    <port name="d0" type="in" rate="2"/>
    <port name="b0" type="out" rate="1"/>
    <executionTime time="1200"/>
    <memoryUsage byte="200"/>
  </actor>
  <actor name="b0">
    <port name="a0" type="in" rate="1"/>
    <port name="c0" type="out" rate="1"/>
    <port name="d0" type="out" rate="2"/>
    <executionTime time="9600"/>
    <memoryUsage byte="600"/>
  </actor>
</application>
```

Fig. 14. Snippet of *App10* application specification.

From these application-descriptions, a multiprocessor system is generated. For processors that have multiple actors mapped onto them, an arbitration scheme is also generated. All the edges in an application are mapped on a unique FIFO channel. This creates an architecture that mimics the applications directly. Unlike processor sharing for multiple applications, the FIFO links are dedicated as can be seen in Figure 13. As opposed to a network or a bus-based infrastructure, the dedicated links remove the possible sources of contention that can limit the performance. Since we have multiple applications running concurrently, there is often more than one link between some processors. Even in such cases, multiple FIFO channels are created. This avoids head-of-line blocking that can occur if one FIFO is shared for multiple channels [38].

In addition to the hardware topology, the software for each processor is also generated. The software simulates the SDF model of the actor execution and the arbitration. If the source code of an actor is available it may also be inserted in the description. Other miscellaneous files that are necessary for

synthesis are also generated. An example of this in case of FPGA is the pin-constraints file.

### A. Resource Manager

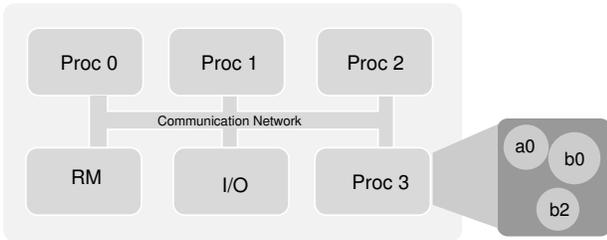


Fig. 15. Architecture with Resource Manager

The design is extended to allocate one processor for the resource manager (RM). Figure 15 shows the modified architecture when resource manager is used in the system. The FIFO links in Section 2.2 are abstracted away with a communication fabric. The application description and properties like the actor execution times, mapping and throughput expressions are stored in a CF card.

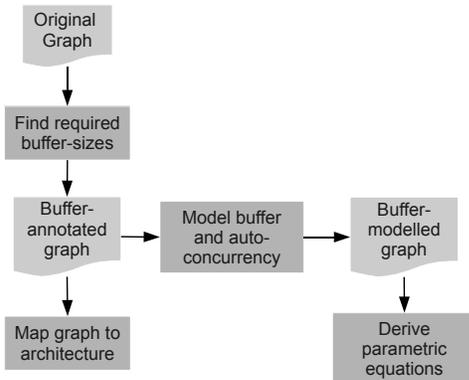


Fig. 16. An overview of the design flow to analyze the application graph and map it on the hardware.

Figure 16 shows the flow that is used to do the experiments. For each application, as explained in Section III-B, the buffer-sizes needed for the required performance are computed. These sizes are annotated in the graph description and used for the hardware flow described above. These buffer-sizes are modeled in the graph using a back-edge with the number of initial tokens on that edge equal to the buffer-size needed on the forward edge as explained above in Section III-A. Further, we limit the auto-concurrency of actors to 1 since at any point in time, only one execution of an actor can be active. These constraints are modeled in the graph before the parametric throughput expressions are derived. Note that the graph used for computing the parametric expressions is not the same as the one that is mapped to architecture, but it leads to the same application behavior since the constraints modeled in the graph come from the architecture itself.

*Admission controller:* Figure 17 shows a simple admission controller that we implemented. The controller takes

the expected and required performance of all applications as input. For each application, it is checked whether the expected performance of the application meets the desired performance. If the performance of any of the applications is not expected to meet the requirement by adding a new application, the new application is rejected (or retried at a lower quality level), and otherwise accepted. This analysis can also be adapted to include some margin for error. For example, if  $x\%$  margin is desired, the comparison function can be adapted to check  $(100 - x)/100 \cdot \text{exp}(i) \geq \text{reqd}(i)$  for a pessimistic analysis.

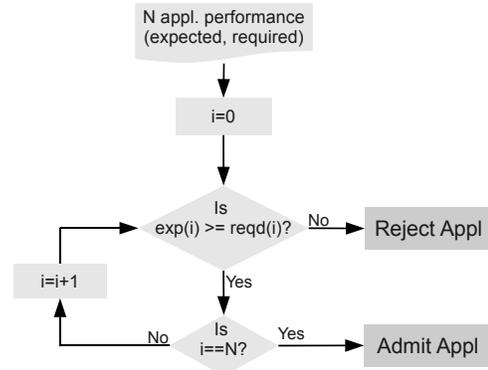


Fig. 17. A simple admission controller

If an application is to be started and the controller concludes that it is safe to admit the new application, the program code has to be transferred to the relevant processors and connections setup for communication. This is abstracted in our system and the actor behavior is already defined on the processor in the system, and the applications are simply disabled at system-startup. As and when an application is admitted in the system, the resource manager signals the relevant processors to enable the application.

## VII. EXPERIMENTS

In this section, we describe our experimental setup and some results obtained both for basic probability, as explained in Section IV. The iterative technique as explained in Section V improves upon this. First, we only show results of basic probability analysis since iterative analysis results are almost exactly same as the measured results. Superimposing iterative analysis results on the same scale makes the graph difficult to understand. In basic analysis results, the graph is scaled to the original period, while in iterative analysis it is scaled to the measured period. The results for the hardware implementation of the admission controller are also provided. For some experiments, we were limited by FPGA synthesis time. Therefore, we developed another tool using POOSL [40] to provide quick simulation results.

### A. Setup

In this section we present the results of above analysis obtained as compared to simulation results for a number of use-cases. For this purpose, ten random SDFGs were generated with eight to ten actors each using the *SDF*<sup>3</sup> tool [39], mimicking DSP and multimedia applications. Each graph is

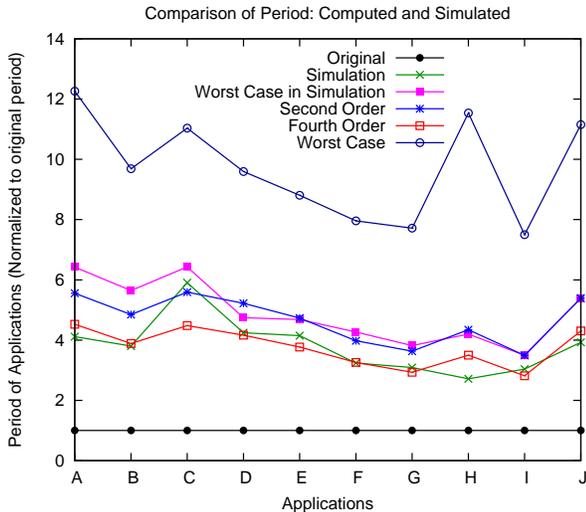


Fig. 18. Comparison of period computed using different analysis techniques as compared to POOSL simulation result (all 10 applications running concurrently). The periods obtained through analysis and simulation are normalized to the original period.

a strongly connected component i.e. every actor in the graph can be reached from every actor. The execution time and the rates of actors were also set randomly. The  $SDF^3$  tool was also used to analytically compute the periods of the graphs. Using these ten SDFGs, over a thousand use-cases ( $2^{10}$ ) were generated. Simulations were performed using POOSL [40] to give actual performance achieved for each use-case. Two different probabilistic approaches were used - the second order and the fourth order approximations of Equation 6. Results of worst-case-response-time analysis [16] for non-preemptive systems are also presented for comparison.

The simulation of all possible use-cases, each for 500,000 cycles took a total of 23 hours on a Pentium 4 3.4 GHz with 3 GB of RAM. In contrast, analysis for all the approaches was completed in only about 10 minutes.

### B. Results and Discussion - Basic Analysis

Figure 18 shows a comparison between periods computed analytically using different approaches as described in the paper (without the iterative analysis), and the simulation result. The use-case for this figure is the one in which all applications are executing concurrently. This is the case with maximum contention. The period shown in the figure is normalized to the original period of each application that is achieved in isolation. The worst case observed in simulation is also shown.

A number of observations can be made from the figure. We see how the period is much higher when multiple applications are run. For application *C*, the period is six times the original period, while for application *H*, it is only three-fold (simulation results). The analytical estimates computed using different approaches are also shown in the same graph. The estimates using the worst-case-response-time [15] is much higher than that achieved in practice and therefore, overly pessimistic. The estimates of the two probabilistic approaches are very close to the observed performance.

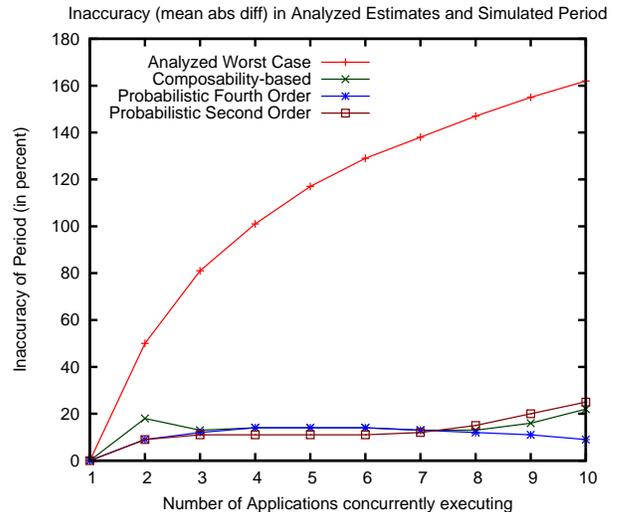


Fig. 19. Inaccuracy in application periods obtained through simulation and different analysis techniques

We further notice that the second order estimate is always more conservative than the fourth order estimate, which is expected, since it overestimates the resource contention. The fourth order estimates of probability is the closest to the simulation results except in applications *C* and *H*.

Figure 19 shows the variation in period that is estimated and observed as the number of applications simultaneously executing in the system increases. The metric displayed in the figure is the mean of absolute differences between estimated and observed period. When there is only one application active in the system, the inaccuracy is zero for all the approaches, since there is no contention. As the number of applications increases, the worst-case-response-time estimate deviates a lot from the simulation result. This indicates why this approach is not scalable with number of applications in the system. For the other three approaches, we observe that the variation is usually within 20% of simulation result. We also notice that the second order estimate is almost exactly equal to the composability-based approach - both of which are more conservative than the fourth-order approximation. The maximum deviation in the fourth order approximation is about 14% as compared to about 160% in the worst-case approach - a ten-fold improvement.

### C. Results and Discussion - Iterative Analysis

Figure 20 shows the strength of the iterative analysis. The results are now shown with respect to the results achieved in simulation as opposed to the original period. The fourth-order probability result are also shown on the same graph to put things in perspective since that is the closest to the simulation result. As can be seen, while the maximum deviation in fourth-order is about 30%, the average error is very low. The results of applying iterative analysis starting from fourth order, after 1, 5 and 10 iterations are also shown. The estimates get closer to the actual performance after every iteration. After 5 iterations, the maximum error that can be seen is in Application H of about 3%, and the average error is to the tune of 2%.

Results of conservative version of the iterative technique are also shown on the same graph. This is the result obtained

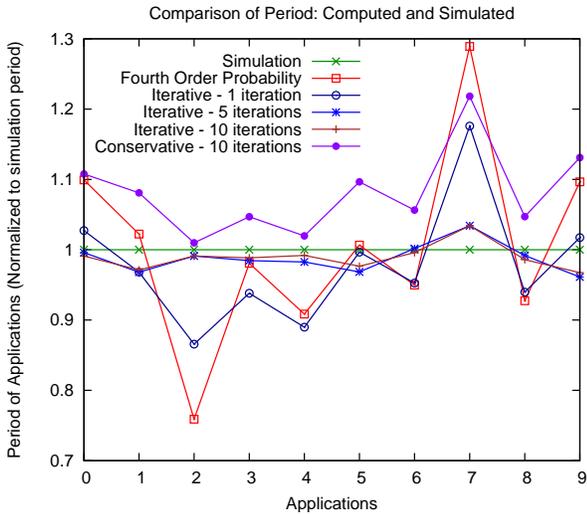


Fig. 20. Comparison of period computed using iterative analysis techniques as compared to simulation result (all 10 applications running concurrently)

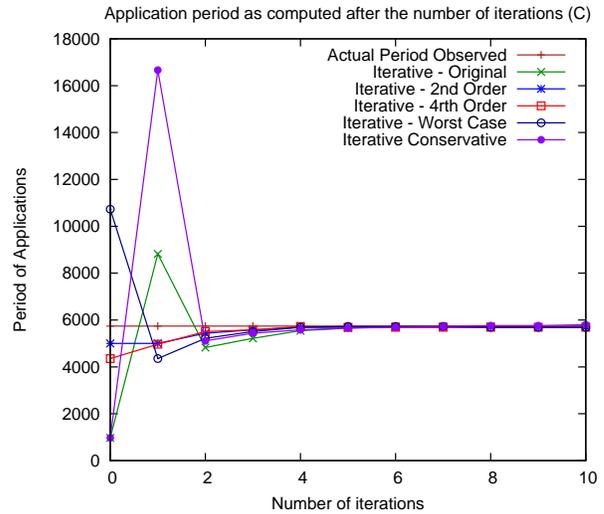


Fig. 22. Comparison of period computed using iterative analysis technique as compared to simulation result for application C.

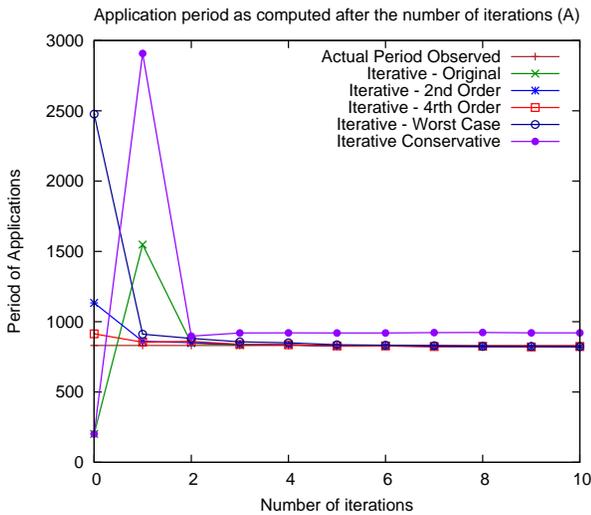


Fig. 21. Comparison of period computed using iterative analysis technique as compared to simulation result for application A.

after ten iterations of the conservative technique. The estimate provided by this technique is always above the simulation result. On average, in this figure the conservative approach over-estimates the period by about 8% - a small price to pay when compared with the worst-case bound that is 162% over-estimated.

Figure 21 shows the results of iterative analysis with increasing number of iterations for application A. Five different techniques are compared with the simulation result - iterative technique starting from the original graph, second order probabilistic estimate, fourth order probabilistic estimate and worst case initial estimate, including the conservative analysis of iterative technique starting from the original graph. While most of the curves converge almost exactly on the simulation result, the conservative estimate converges on a value slightly higher, as expected. Similar graph is shown for another application C. In this application, it takes somewhat longer before the estimate converges. For this application the

TABLE II

MEASURED INACCURACY FOR PERIOD IN % AS COMPARED WITH SIMULATION RESULTS FOR ITERATIVE ANALYSIS. BOTH THE AVERAGE AND MAXIMUM ARE SHOWN.

Iterations	2nd Order	4th Order	Worst Case	Original	Conser.
0	22.3/44.5	9.9/28.9	72.6/83.1	163/325	72.6/83.1
1	6.2/19	6.7/17.6	88.4/144	12.6/36	252/352
2	3.7/13.3	3.5/11.9	6.3/17.6	6.7/23.2	7.9/23.2
3	3/7.7	2.9/6.2	4.5/11.9	4.3/13.3	8.8/24.7
4	2.2/6.2	2/4.8	2.5/7.7	3.1/9.1	8.4/23.2
5	2.2/4.8	1.9/3.9	2.2/4.8	2.5/6.2	8.3/23.2
6	1.7/3.6	1.6/3.6	1.7/3.4	2/4.8	8.1/21.8
7	1.8/4	1.9/4	1.8/3.4	1.7/3.9	8/21.8
8	1.7/3.6	1.7/3.6	1.7/3.4	1.8/3.6	8/21.8
9	1.8/3.4	1.9/3.4	1.7/3.6	1.7/3.4	8/21.8
10	1.6/3.3	1.7/3.4	1.3/3.1	1.9/3.4	8.1/21.8
20	1.7/3	1.4/2.9	1.4/2.9	1.5/3	8.1/21.8
30	1.4/3	1.6/3	1.6/3	1.4/3	8.1/21.8

conservative estimate is almost exactly equal to the simulation result.

A couple of observations can be made from this graph. First, the iterative analysis approach is converging. Regardless of how far and which side the initial estimate of the application behavior is, it converges within a few iterations close to the actual value. Second, the final value estimate is independent of the starting estimate. The graph shows that iterative technique can be applied from any initial estimate (even the original graph directly) and still achieve accurate results. This is a very important observation since this implies, that if we have constraints on program memory, we can manage with only the iterative analysis technique. If there is no such constraint, one can always start with the fourth-order estimate in order to get faster convergence. (This is probably only suitable for cases when applications have a large number of throughput equations, and when throughput computation takes more cycles than fourth order estimate.)

The error in the iterative analysis (defined as mean absolute difference), is averaged and presented in Table II. In general, as the number of iterations increase the error decreases. Different starting points for iterative analysis are taken. As can be seen,

TABLE III

THE NUMBER OF CLOCK CYCLES CONSUMED ON A MICROBLAZE PROCESSOR DURING VARIOUS STAGES, AND THE PERCENTAGE OF ERROR (BOTH AVERAGE AND MAXIMUM) AND THE COMPLEXITY.

Algorithm/Stage	Clock cycles	Error (%) avg/max	Complexity
Load from CF Card	1903500	-	$O(N.n.k)$
Throughput Computation	12688	-	$O(N.n.k)$
Worst Case	2090	72.6/83.1	$O(m.M)$
Second Order	45697	22.3/44.5	$O(m^2.M)$
Fourth Order	1740232	9.9/28.9	$O(m^4.M)$
Iterative - 1 Iteration	15258	12.6/36	$O(m.M)$
Iterative - 1 Iteration*	27946	12.6/36	$O(m.M + N.n.k)$
Iterative - 5 Iterations*	139730	2.2/3.4	$O(m.M + N.n.k)$
Iterative - 10 Iterations*	279460	1.9/3.0	$O(m.M + N.n.k)$

\*Including throughput computation time

$N$ -number of applications

$n$ -number of actors in an application

$k$ -number of throughput equations for an application

$m$ -number of actors mapped on a processor

$M$ -number of processors

the fourth order initial estimate converges the fastest among all approaches. If we define 2% error margin as acceptable, we find that fourth order estimate requires only 4 iterations to converge while others require 6 iterations.

#### D. Hardware Implementation Results

The proposed admission controller is fully implemented on a processor as described in Section VI. All the algorithms were ported to Microblaze; this required some extra tuning to optimize the implementation for timing and reduced memory use. Table III shows the time taken during various stages and algorithms. The algorithmic complexity of each stage and also the error as compared to the simulation result is also shown. The default time taken for second and fourth order, for example, was 72M and 11M cycles respectively.

The error in various techniques as compared to the performance achieved is also shown in Table III. As can be seen the basic probability analysis with fourth order gives an average error of about 10% and maximum error of 29%. The iterative technique after just five iterations predicts performance that is within 2% of measured performance on average and has only 3% maximum deviation in the entire set of applications.

This system consisted of the same 10 applications as used in the previous sub-section. The loading of application properties from the CF card took the most amount of time. However, this is only done at the start of system, and hence does not cause any bottleneck during admission control. On our system operating at 50 MHz, it takes about 40ms to load the applications-specification. Parametric throughput computation is quite fast, and takes about 12K cycles for all 10 applications. The analysis itself for all the applications is quite fast, except the fourth-order analysis.

For the iterative analysis, each iteration takes only 15K cycles i.e. 300 micro-seconds. If 5 iterations are carried out, it takes a total of 140K cycles for all 10 applications including the time spent in computing throughput. This translates to a latency of about 3 ms in starting of applications when 10 applications' performance is to be computed and checked.

## VIII. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a probabilistic technique to estimate the performance of applications when sharing resources. An iterative analysis is presented that can predict the performance of applications very accurately. Besides, a conservative flavour of the iterative analysis is presented that can also provide conservative prediction for applications for which the mis-prediction penalty may be high. Further, we proposed an admission controller to estimate the performance of multiple applications before they execute on the system.

The basic probability analysis gives results with average error of 10%, and the maximum error of 29%. The average error in prediction using iterative probability is, however, only 2% and maximum of about 3%. Further, it takes about four to six iterations for the prediction to converge. The execution-time complexity of the algorithm is low, and the area overhead of the admission controller is at most 7%. The latency in starting applications is very low - only 3ms with 10 applications on a 50MHz processor.

We also present a flow for designing systems with multiple applications, such that the entire analysis remains composable. This allows easy and quick analysis of an application-mix while properties of individual applications are derived in isolation. Our flow also allows addition of unknown applications at run-time, even when little is known about their properties at design-time.

One of the limitations of this approach is that it does not provide any guarantees. In future, we intend to extend our technique to provide guarantees for soft real time tasks using probability. Further, in this approach we have considered first-come-first-serve arbitration mechanism implicitly in the analysis. A more predictable arbitration mechanism, for example, credit-based priority, may be more suitable for tasks with varying priority, and we would like to extend our analysis to such arbitration mechanisms.

## REFERENCES

- [1] A. Kumar, B. Mesman, B. Theelen, H. Corporaal, and Y. Ha, "A Probabilistic Approach to Model Resource Contention for Performance Estimation of Multi-featured Media Devices," in *Proceedings of Design Automation Conference*. New York, NY, USA: ACM Press, 2007, pp. 726–731.
- [2] A. Kumar, B. Mesman, H. Corporaal, J. van Meerbergen, and H. Yajun, "Global Analysis of Resource Arbitration for MPSoC," in *Proceedings of 9th EUROMICRO Conference on Digital System Design (DSD)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 71–78.
- [3] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in a hard-real-time environment," *J. ACM*, vol. 20, no. 1, pp. 46–61, 1973.
- [4] S. Davari and S. K. Dhall, "An on line algorithm for real-time tasks allocation," *IEEE Real-time Systems Symposium*, pp. 194–200, 1986.
- [5] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, "Proportionate progress: A notion of fairness in resource allocation," *Algorithmica*, vol. 15, pp. 600–625, 1996.
- [6] K. Jeffay, D. F. Stanat, and C. U. Martel, "On non-preemptive scheduling of periodic and sporadic tasks," in *Proceedings of 12th IEEE Real-Time Systems Symposium*, 1991, pp. 129–139.
- [7] S. Stuijk, "Predictable mapping of streaming applications on multiprocessors," Ph.D. dissertation, Eindhoven University of Technology, 2007.

- [8] A. Kumar, S. Fernando, Y. Ha, B. Mesman, and H. Corporaal, "Multi-processor System-level Synthesis for Multiple Applications on Platform FPGA," in *Proceedings of 17th International Conference on Field Programmable Logic and Applications*. New York, NY, USA: IEEE Circuits and Systems Society, 2007, pp. 92–97.
- [9] MAMPS, "Multiple Applications Mutli-Processor Synthesis [Online]." Username: tcad, Password: guest. Available at: <http://www.es.ele.tue.nl/mamps/>, 2008.
- [10] N. Bambha, V. Kianzad, M. Khandelia, and S. S. Bhattacharyya, "Intermediate representations for design automation of multiprocessor DSP systems," *Design Automation for Embedded Systems*, vol. 7, no. 4, pp. 307–323, 2002.
- [11] S. Sriram and S. Bhattacharyya, *Embedded Multiprocessors; Scheduling and Synchronization*. New York, NY, USA: Marcel Dekker, 2000.
- [12] J. Pino and E. Lee, "Hierarchical static scheduling of dataflow graphs onto multipleprocessors," in *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, vol. 4. Detroit, MI, USA: IEEE, 1995, pp. 2643–2646.
- [13] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 9, no. 4, pp. 385–418, 2004.
- [14] A. Ghamarian *et al.*, "Throughput Analysis of Synchronous Data Flow Graphs," in *6th ACSD 2006*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 25–36.
- [15] M. Bekooij, R. Hoes, O. Moreira, P. Poplavko, M. Pastrnak, B. Mesman, J. D. Mol, S. Stuijk, V. Gheorghita, and J. van Meerbergen, "Dataflow analysis for real-time embedded multiprocessor system design," in *Dynamic and Robust Streaming in and between Connected Consumer-Electronic Devices*. Springer, 2005, pp. 81–108.
- [16] R. Hoes, "Predictable Dynamic Behavior in NoC-based MP-SoC." Available from: [www.es.ele.tue.nl/epicurus/](http://www.es.ele.tue.nl/epicurus/), 2004.
- [17] K. Richter, M. Jersak, and R. Ernst, "A formal approach to MPSoC performance verification," *Computer*, vol. 36, no. 4, pp. 60–67, 2003.
- [18] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proceedings of ISCAS 2000 Geneva.*, vol. 4. Geneva, Switzerland: IEEE, 2000, pp. 101–104.
- [19] S. Kunzli, F. Poletti, L. Benini, and L. Thiele, "Combining Simulation and Formal Methods for System-level Performance Analysis," in *Proceedings of Design Automation and Test in Europe*, vol. 1. IEEE, 2006, pp. 1–6.
- [20] L. Abeni and G. Buttazzo, "QoS guarantee using probabilistic deadlines," in *Proceedings of the 11th Euromicro Conference on Real-Time Systems, 1999*. York, UK: IEEE, 1999, pp. 242–249.
- [21] S. Manolache, P. Eles, and Z. Peng, "Schedulability analysis of applications with stochastic task execution times," *Trans. on Embedded Computing Sys.*, vol. 3, no. 4, pp. 706–735, 2004.
- [22] S. Hua, G. Qu, and S. S. Bhattacharyya, "Probabilistic design of multimedia embedded systems," *Trans. on Embedded Computing Sys.*, vol. 6, no. 3, p. 15, 2007.
- [23] O. Moreira, J. J.-D. Mol, and M. Bekooij, "Online resource management in a multiprocessor with a network-on-chip," in *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing*. New York, NY, USA: ACM, 2007, pp. 1557–1564.
- [24] V. Nollet, P. Avasare, H. Eeckhaut, D. Verkest, and H. Corporaal, "Run-time management of a mpsoC containing fpga fabric tiles," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 16, no. 1, pp. 24–33, Jan. 2008.
- [25] A. Kumar, B. Mesman, B. Theelen, H. Corporaal, and Y. Ha, "Resource manager for non-preemptive heterogeneous multiprocessor system-on-chip," in *Proceedings of 4th Workshop on Embedded Systems for Real-Time Multimedia (Estimedia)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 33–38.
- [26] J. M. Paul, D. E. Thomas, and A. Bobrek, "Scenario-oriented design for single-chip heterogeneous multiprocessors," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 14, no. 8, pp. 868–880, Aug. 2006.
- [27] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous dataflow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 36, no. 1, pp. 24–35, Feb 1987.
- [28] S. Stuijk, M. Geilen, and T. Basten, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," in *Proceedings of Design Automation Conference*. New York, NY, USA: ACM Press, 2006, pp. 899–904.
- [29] T. Stefanov, C. Zissulescu, A. Turjan, B. Kienhuis, and E. Deprette, "System design using Kahn process networks: the Compaan/Laura approach," in *Proceedings of Design Automation and Test in Europe*. Los Alamitos, CA, USA: IEEE Computer Society, 2004, pp. 340–345.
- [30] J. Cockx, K. Denolf, B. Vanhoof, and R. Stahl, "Sprint: a tool to generate concurrent transaction-level models from sequential code," *EURASIP Journal on Applied Signal Processing*, vol. 2007, no. 1, pp. 213–213, 2007.
- [31] S. Rul, H. Vandierendonck, and K. D. Bosschere, "Function level parallelism driven by data dependencies," *ACM SIGARCH Computer Architecture News*, vol. 35, no. 1, pp. 55–62, 2007.
- [32] A. Ghamarian, M. Geilen, T. Basten, and S. Stuijk, "Parametric throughput analysis of synchronous data flow graphs," in *Proceedings of Design Automation and Test in Europe*. Los Alamitos, CA, USA: IEEE Computer Society, 10-14 March 2008, pp. 116–121.
- [33] V. Nollet, "Run-time management for Future MPSoC Platforms," Ph.D. dissertation, Eindhoven University of Technology, 2008.
- [34] R. Lauwereins, C. Wong, P. Marchal, J. Vounckx, P. David, S. Himpe, F. Catthoor, and P. Yang, "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *Proceedings of the 15th international symposium on System Synthesis*. Los Alamitos, CA, USA: IEEE Computer Society, 2002, pp. 112–119.
- [35] Bluetooth, "Bluetooth specification [Online]." Available from: <http://www.bluetooth.com/bluetooth>, 2008.
- [36] K. Goossens, J. Dielissen, and A. Radulescu, "Aetheral network on chip: concepts, architectures, and implementations," *IEEE Design and Test of Computers*, vol. 22, no. 5, pp. 414–421, 2005.
- [37] D. Terr and E. W. Weisstein, "Symmetric polynomial," Available from: [mathworld.wolfram.com/SymmetricPolynomial.html](http://mathworld.wolfram.com/SymmetricPolynomial.html), 2008.
- [38] HOL, "Head-of-line blocking [Online]." Available from: [http://en.wikipedia.org/wiki/Head-of-line\\_blocking](http://en.wikipedia.org/wiki/Head-of-line_blocking), 2008.
- [39] S. Stuijk, M. Geilen, and T. Basten, "SDF3: SDF For Free," in *Sixth International Conference on Application of Concurrency to System Design (ACSD)*. Los Alamitos, CA, USA: IEEE Computer Society, 2006, pp. 276–278.
- [40] B. Theelen, O. Florescu, M. Geilen, J. Huang, P. van der Putten, and J. Voeten, "Software/Hardware Engineering with the Parallel Object-Oriented Specification Language," in *Proceedings of the Fifth ACM-IEEE International Conference on Formal Methods and Models for Codesign*. Los Alamitos, CA, USA: IEEE Computer Society, 2007, pp. 139–148.