ELSEVIER

Available at
www.ComputerScienceWeb.com
POWERED BY SCIENCE @ DIRECT®

Computer Networks 41 (2003) 641–665

COMPUTER
NETWORKS

www.elsevier.com/locate/comnet

Performance evaluation of network processor architectures: combining simulation with analytical estimation

Samarjit Chakraborty [a,*], Simon Künzli [a], Lothar Thiele [a],
Andreas Herkersdorf [b], Patricia Sagmeister [b]

[a] *Computer Engineering and Networks Laboratory, Swiss Federal Institute of Technology (ETH) Zürich, Gloriastrasse 35, CH-8092 Zürich, Switzerland*
[b] *IBM Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlilion, Switzerland*

Abstract

The designs of most systems-on-a-chip (SoC) architectures rely on simulation as a means for performance estimation. Such designs usually start with a parameterizable template architecture, and the design space exploration is restricted to identifying the suitable parameters for all the architectural components. However, in the case of heterogeneous SoC architectures such as network processors the design space exploration also involves a combinatorial aspect—which architectural components are to be chosen, how should they be interconnected, task mapping decisions—thereby increasing the design space. Moreover, in the case of network processor architectures there is also an associated uncertainty in terms of the application scenario and the traffic it will be required to process. As a result, simulation is no longer a feasible option for evaluating such architectures in any automated or semi-automated design space exploration process due to the high simulation times involved. To address this problem, in this paper we hypothesize that the design space exploration for network processors should be separated into multiple stages, each having a different level of abstraction. Further, it would be appropriate to use analytical evaluation frameworks during the initial stages and resort to simulation techniques only when a relatively small set of potential architectures is identified. None of the known performance evaluation methods for network processors have been positioned from this perspective.

We show that there are already suitable analytical models for network processor performance evaluation which may be used to support our hypothesis. To this end, we choose a reference system-level model of a network processor architecture and compare its performance evaluation results derived using a known analytical model [Thiele et al., Design space exploration of network processor architectures, in: Proc. 1st Workshop on Network Processors, Cambridge, MA, February 2002; Thiele et al., A framework for evaluating design tradeoffs in packet processing architectures, in: Proc. 39th Design Automation Conference (DAC), New Orleans, USA, ACM Press, 2002] with the results derived by detailed simulation. Based on this comparison, we propose a scheme for the design space exploration of network processor architectures where both analytical performance evaluation techniques and simulation techniques have unique roles to play.

Keywords: Performance analysis; Network processors; High-performance routers; Application-specific instruction set processors

---

[*] Corresponding author.

E-mail addresses: samarjit@tik.ee.ethz.ch (S. Chakraborty), kuenzli@tik.ee.ethz.ch (S. Künzli), thiele@tik.ee.ethz.ch (L. Thiele), anh@zurich.ibm.com (A. Herkersdorf), psa@zurich.ibm.com (P. Sagmeister).

1389-1286/03/$ - see front matter
doi:10.1016/S1389-1286(02)00454-1

1. Introduction

Today, network processors play an important role in the design of modern routers. Therefore, of late there has been a lot of interest in the study of network processor architectures. Designing such architectures is partly complicated by the fact that they involve complex tradeoffs between flexibility and efficiency. Typically, such processors are required to process packets at high line speeds, support complex packet processing functions, and to a large extent be programmable to incorporate new functionality. To support such stringent and to some extent conflicting demands, most network processors available today consist of a collection of heterogeneous processing elements, memory subsystems and on-chip communication infrastructure. There is no agreement on a *generic network processor architecture* and a wide variation exists among the processors available in the market today. However, typically they are built around one or more processor cores and micro-engines, some dedicated hardware for packet processing tasks such as header parsing, table lookup, etc., some on-chip cache/memory, and specialized buses and bridges for on-chip communication and interfacing with external memories and ports.

Because of these new and different requirements, designing and evaluating a network processor architecture calls for specialized modeling techniques and frameworks which do not fall under the preview of traditional embedded processor design. As a result, recently there has been a number of proposals for performance models and design frameworks specific to network processor architectures (see [6,7,9,26,27,31,32]). The goal of these frameworks and models is to aid a designer in understanding the performance tradeoffs involved in a design and come up with an optimal architecture that suits the application scenario at hand.

Realizing these goals in the context of traditional embedded processor design typically involves a method for performance evaluation of the architecture, and a means for automatic design space exploration. In most of these cases it is possible to formulate a parameterized *architecture template*, where the design space exploration is restricted to finding appropriate values of the parameters such as bus width, cache associativity and cache size. The resulting design space is therefore reasonably small and it is usually feasible to exhaustively evaluate all the possible designs by simulation. In cases where the design space is relatively larger, techniques such as partitioning the architecture into disjoint subsystems and using independent design space explorations for the different subsystems have been used [11,15,22]. Even in these cases, the choice of the different architectural components is usually fixed (for example, see [1,24], where the system always consists of a VLIW processor, a systolic array and a cache subsystem and the design space exploration consists of identifying appropriate parameters for each of these components), and there are no uncertainties associated with the application, as is the case with network processor architectures.

1.1. Network processor design

In the case of network processor design, the issue of design space exploration, however, might have a different complexity because of several reasons. Network processor architectures are very heterogeneous in nature and usually it is not possible to define a parameterizable template architecture. As a result the design space is larger compared to those in the case of typical embedded processor architectures and involves a combinatorial aspect in addition to traversing the parameter spaces of the different components. These processors might also be used in multiple application scenarios (such as core or access networks), might be required to support different traffic classes (where some classes might have quality-of-service requirements and others have minimum throughput requirements), and at the same time should be flexible to be able to incorporate new functionality. In order to account for all of these issues in the design phase, we believe that new design methodologies are required. In particular, resorting to exhaustive simulations of all possible designs will no longer be a feasible option for automated design space exploration. Hence, using other means of performance evaluation such as analytical models is necessary.

It is known that typically the design flow of complex systems-on-a-chip (SoC) architectures starts with an abstract description of the application and some performance requirements, which are then used to drive a system-level design space exploration for identifying a suitable architecture. This involves evaluating many prospective architectures on a system-level and an iteration loop between the exploration and the evaluation steps. Once the main architectural decisions have been made, the resulting architectures are then more accurately evaluated, possibly on the basis of many other criteria which were not considered previously. The design space exploration at this stage, in contrast to the previous, might only involve tuning the parameters of different cores in a core-based SoC design.

1.1.1. Our contributions and relation to previous work

In this paper we argue that in the case of heterogeneous SoC architectures, this separation of the design space exploration into multiple stages is all the more important in order to tackle the large and the different nature of the design space. In particular, we hypothesize that in the context of network processors, the underlying framework for performance evaluation should vary depending on the stage of the design space exploration—it will be more appropriate to use analytical methods during the initial stages and resort to simulation when a relatively small set of promising alternatives has been identified. None of the known performance evaluation frameworks for network processors have been evaluated or positioned from this perspective. From a designer's point of view it would be useful to know if any of the known modeling techniques are more suitable for a particular stage of the architecture design.

In any of the design phases, for a potential architecture at hand, the performance evaluation needs to answer questions such as: Does this architecture meet the required line speeds and maximum allowable delays experienced by packets? What are the limits to the improvement in processor or bus utilization as the number of processor cores is increased? How does the cache/memory organization impact these limits? Will a given hardware assist improve the system performance compared to a software implementation? We believe that the exact nature of these questions, how accurately they need to be answered, and what is the allowable computation/simulation time required to answer them strongly depend on the design phase. For network processors many of these can be adequately answered with a system-level model, and we show that there exist suitable analytical frameworks for doing this. These are orders of magnitude faster when compared to simulations and are hence appropriate for a system-level design space exploration where the design space can be very large.

In support of our hypothesis, we compare the results obtained by a system-level analytical performance model proposed in [26,27] with detailed cycle accurate simulations, on a realistic network processor architecture. We consider three performance metrics: (i) the line speed or the end-to-end throughput that can be supported by the architecture, which is measured using the utilization of its different components (processors, buses) and thereby also identifying which component acts as the bottleneck, (ii) the end-to-end packet latencies, and (iii) the on-chip cache/memory requirement of the architecture. Many important questions that arise in the context of network processor architectures pertain to these metrics. The usefulness of the results obtained from the analytical model should be evaluated with respect to their relative accuracy when compared to the simulation results, and the time it takes to compute these results compared to simulation times (under the assumption that there is a high confidence in the simulation results).

One of the major criticisms of the analytical framework [26,27] we consider here, has been that although it is sufficiently general, it still remains to be seen if it can be applied to analyse any realistic network processor architecture (see, for example, [29]). Our work in this paper addresses this issue and additionally places this framework in an appropriate stage of the design flow.

In the next section we review the existing modeling techniques known for network processor performance evaluation. For the sake of completeness, we present the analytical model considered in this paper in Sections 3 and 4. In Section 5 we describe the setup used for the simulation. A comparison of the results obtained by the two methods on a reference architecture is made in Section 6. Finally, in Section 7 the results of the comparative study are analysed, and on the basis of these results we propose a methodology for the design space exploration of network processor architectures which relies on both the compared techniques.

2. Existing approaches

As mentioned in the last section, lately there has been a number of proposals for performance analysis techniques specific to network processor architectures—both analytical models and simulation based frameworks. There is also a large body of work devoted to system-level performance evaluation of SoC architectures (see [10] and the references therein) and also of more specific aspects of an architecture such as the on-chip communication infrastructure (see [17] and the references therein). However, in this section we focus on the work done specifically in the context of network processors.

In [6] a modeling framework is presented which is composed of independent application, system and traffic models. The application is modeled using the Click modular router from MIT [16]. Click consists of a collection of software modules for describing various router functionalities. Such modules in Click are called *elements*, and by putting together different elements in the form of a graph (which is called a *configuration*) it is possible to construct IP routers, firewalls, QoS routers, etc.

The framework in [6] is based on compiling Click modules for the Alpha ISA [3]. The architecture to be evaluated is simulated using SimpleScalar [5] and it implements the Alpha instruction set. The compiled Click modules are then executed on this architecture. By simulating this execution using different traffic traces, the profiled code yields various information such as instruction count, details regarding cache behavior, etc. These are then used to compute various performance metrics for the architecture being evaluated, related to packet latency, bandwidth and resource utilization. For elements which do not have any software implementation (such as dedicated hardware units for header parsing) and cannot be simulated, the profile and external dependencies need to be provided manually by the user.

In contrast to this approach, the work done in [32] models an architecture in SystemC [12]. This work mostly focuses on the communication subsystem and the memory organization of an architecture. The models are then simulated on packet traces and performance metrics such as bus utilization, memory fill levels, and packet delays are evaluated. This work forms the basis of the simulations results that we present in this paper and further details on it are given in Section 5.

These two approaches as they exist now are complementary to each other. Crowley and Baer [6] use an accurate processor model but a very elementary model of the communication subsystem of an architecture (such as buses, bridges, etc.). On the other hand, the framework in [32] implements cycle accurate models for buses and bridges, but has a very simple processor and application model.

Clearly, there exists an opportunity to combine the above frameworks which will then consist of a detailed model of processors as well as other components of the communication subsystem, such as buses, for which SystemC models already exist. Additionally, in the same way as there exists a SimpleScalar model of the Alpha processor, there already exists a SystemC model of PowerPC which can be simulated on executable code. This easily opens the opportunity of integrating this model into the network processor architecture model and have detailed applications models, as for example is done in [6] using Click.

An analytical performance model of a network processor is considered in [9]. Here the different components that make up the architecture, and the interconnection among these components (the so called *architecture template*) is fixed. The design decisions that are to be made in deriving a concrete architecture from such a template, consist of choosing the values of the various parameters such as the bus width, cache sizes, etc. The architecture considered consist of a number of multithreaded processors organized in clusters. Each cluster consists of a number of processors, each having its own cache, and the cluster communicates with an off-chip memory using its own memory interface. The parameters that can be changed are the number of threads running in each processor, the cache sizes, the number of processors in each cluster, the number of clusters in the network processor, the width of the memory channels, etc. For evaluating a set of parameters, an analytical model for multithreaded processors proposed by Agarwal [2] is used. Most of this work can be viewed as a model for the cache/memory subsystem of a network processor architecture. The analytical model is then evaluated on a benchmark workload [30] consisting of a mix of header-processing and payload-processing applications. For each application, properties such as load and store instruction frequencies and instruction and data cache miss rates are measured using processor and cache simulators. These values are then used to evaluate an architecture in terms of its processing power per unit chip area. This model can therefore be viewed as a combination of analytical methods and simulation, and might be restrictive for use in a fast design space exploration method because of the simulation times involved.

The analytical model [26,27] used for the comparative study in this paper uses a model for both the architecture and the traffic traces on which the architecture is evaluated. In contrast to the work in [9] the architecture layout or the topology in this case is not fixed. Therefore, different combinations of processors, buses and their interconnection can be modeled and evaluated. The details of this model are presented in the next two sections.

3. A model for characterizing network traffic and packet processing resources

In this section we describe a model for characterizing packet flows, and based on similar concepts, a model for describing the computation/communication power of resources used to process/transmit the packets entering a resource. These models are based on the concept of *arrival* and *service* curves due to Cruz [8], and Parekh and Gallager [19].

Let f be a flow entering a given resource. The resource is either a computation resource such as a processor on which some packet processing task is implemented, or it is a communication resource such as a bus that is used to transmit packets between two computation resources or a computation resource and a memory module. Let $R_f(t)$ denote the number of packets from f arriving at the resource during the time interval $[0,t]$. The maximum number of packets arriving at the resource is assumed to be bounded by a right-continuous subadditive function called the *upper arrival curve* denoted by α_f^{u}. Similarly, a lower bound on the number of packets arriving at the resource is given by a *lower arrival curve* denoted by α_f^{l}. α_f^{l} and α_f^{u} together can be referred to as the *traffic constraint functions* and they satisfy the following inequality:

$$\alpha_f^{\mathrm{l}}(t-s) \leqslant R_f(t) - R_f(s) \leqslant \alpha_f^{\mathrm{u}}(t-s), \quad \forall 0 \leqslant s \leqslant t.$$

For any $\Delta \geqslant 0$, $\alpha_f^{\mathrm{l}}(\Delta) \geqslant 0$ and $\alpha_f^{\mathrm{l}}(0) = \alpha_f^{\mathrm{u}}(0) = 0$. Therefore, $\alpha_f^{\mathrm{l}}(\Delta)$ gives a lower bound on the number of packets that can arrive at the resource from the flow f within any time interval of length Δ. $\alpha_f^{\mathrm{u}}(\Delta)$ gives the corresponding upper bound.

Similar to the arrival curves describing packet flows, the computation or communication capability of a resource is described using *service curves*. Given a resource r, let $C_r(t)$ denote the number of packets (or the number of bytes, depending on the resource) that can be processed by r during the time interval $[0,t]$. Then the upper and lower service curves β_r^{u} and β_r^{l} describing the processing capabilities of the resource satisfy the following inequality:

$$\beta_r^{\mathrm{l}}(t-s) \leqslant C_r(t) - C_r(s) \leqslant \beta_r^{\mathrm{u}}(t-s), \quad \forall 0 \leqslant s \leqslant t.$$

Further, for any $\Delta \geqslant 0$, $\beta_r^{\mathrm{l}}(\Delta) \geqslant 0$ and $\beta_r^{\mathrm{l}}(0) = \beta_r^{\mathrm{u}}(0) = 0$. Therefore, $\beta_r^{\mathrm{l}}(\Delta)$ is a lower bound on the number of packets that can be processed by r (or the number of bytes that can be transmitted in case r is a communication resource) within any time interval of length Δ. Likewise, $\beta_r^{\mathrm{u}}(\Delta)$ is the corresponding upper bound. Hence, the processing capability of r over any time interval of length Δ is always greater than or equal to $\beta_r^{\mathrm{l}}(\Delta)$ and less than or equal to $\beta_r^{\mathrm{u}}(\Delta)$.

As mentioned above, the arrival curves of a flow f entering a resource r is described in terms of either the number of bytes entering r within any time interval of length Δ or the number of packets within any time interval of length Δ. This depends on the packet processing task implemented on r. For a task such as packet header processing, where the load on r depends on the number of packets entering and not on the sizes of the packets, the arrival curves are defined in terms of number of packets. On the other hand, if r is a communication resource such as a bus, or a payload processing task such as encryption is implemented on r, then the arrival curves are defined in terms of number of bytes.

Now suppose that for each packet (or each byte, as the case maybe) entering the resource r, w units of processing resource have to be spent by r to process the packet. This might be described as the number of processor cycles, bus cycles, or processor instructions. Then if a flow f has arrival curves $\bar{\alpha}^{\mathrm{l}}_f$ and $\bar{\alpha}^{\mathrm{u}}_f$ described in terms of number of packets, these maybe transformed as follows to represent the processing request demanded by f from the resource r:

$$\alpha^{\mathrm{l}}_f = w\bar{\alpha}^{\mathrm{l}}_f, \quad \alpha^{\mathrm{u}}_f = w\bar{\alpha}^{\mathrm{u}}_f.$$

Hence, α^{l}_f and α^{u}_f now describe the arrival curves of flow f in terms of the processing request (for example, number of processor cycles) demanded from r. If β^{l}_r and β^{u}_r describe the processing capability of r in terms of the same units (i.e., processor cycles) then the maximum delay and maximum backlog suffered by packets of flow f at the resource r can be given by the following inequalities:

$$\text{delay} \leqslant \sup_{t \geqslant 0} \left\{ \inf \left\{ \tau \geqslant 0 : \alpha^{\mathrm{u}}_f(t) \leqslant \beta^{\mathrm{l}}_r(t+\tau) \right\} \right\}, \tag{1}$$

$$\text{backlog} \leqslant \sup_{t \geqslant 0} \left\{ \alpha^{\mathrm{u}}_f(t) - \beta^{\mathrm{l}}_r(t) \right\}. \tag{2}$$

A physical interpretation of these inequalities can be given as follows: the delay experienced by packets waiting to be served by r can be bounded by the maximum horizontal distance between the curves α^{u}_f and β^{l}_r, and the backlog is bounded by the maximum vertical distance between them.

The packets of flow f, after being processed by the resource r, emerge out of this resource. Let $\alpha^{\mathrm{u}'}_f$ and $\alpha^{\mathrm{l}'}_f$ be the upper and lower arrival curves of this processed flow. Similarly, let $\beta^{\mathrm{u}'}_r$ and $\beta^{\mathrm{l}'}_r$ be the upper and lower remaining service curves of the resource r, denoting the remaining processing capability of r after processing packets of flow f. Then these can be given as follows (see [28] for further details and proofs):

$$\alpha^{\mathrm{l}'}_f(\Delta) = \inf_{0 \leqslant t \leqslant \Delta} \left\{ \alpha^{\mathrm{l}}_f(t) + \beta^{\mathrm{l}}_r(\Delta - t) \right\}, \tag{3}$$

$$\alpha^{\mathrm{u}'}_f(\Delta) = \inf_{0 \leqslant t \leqslant \Delta} \left\{ \sup_{v \geqslant 0} \left\{ \alpha^{\mathrm{u}}_f(t+v) - \beta^{\mathrm{l}}_r(v) \right\} + \beta^{\mathrm{u}}_r(\Delta - t), \beta^{\mathrm{u}}_r(\Delta) \right\}, \tag{4}$$

$$\beta^{\mathrm{l}'}_r(\Delta) = \sup_{0 \leqslant t \leqslant \Delta} \left\{ \beta^{\mathrm{l}}_r(t) - \alpha^{\mathrm{u}}_f(t) \right\}, \tag{5}$$

$$\beta^{\mathrm{u}'}_r(\Delta) = \sup_{0 \leqslant t \leqslant \Delta} \left\{ \beta^{\mathrm{u}}_r(t) - \alpha^{\mathrm{l}}_f(t) \right\}. \tag{6}$$

The maximum utilization of the resource r due to the processing of flow f is given by

$$\text{utilization} \leqslant \lim_{\Delta \to \infty} \frac{\beta^{\mathrm{u}}_r(\Delta) - \beta^{\mathrm{l}'}_r(\Delta)}{\beta^{\mathrm{u}}_r(\Delta)}. \tag{7}$$

4. An analytical model for network processor performance evaluation

The modeling concepts used in the last section are known from work done in the area of communication networks. The results derived on the basis of these models (see also [28]) have their background in network calculus [4] which use the concept of arrival and service curves in a network theoretic context. In this section we show how these results can be used to formulate an analytical performance evaluation model for network processors. The results here were originally derived in [26,27] and we refer the reader to these papers for additional details.

Here we view a network processor as a collection of different processing elements (such as CPU cores, micro-engines, and dedicated units like hardware classifier, cipher, etc.) and memory modules connected to each other by communication buses. On each of these processing elements one or more packet processing tasks are implemented. Depending on the sequence in which these tasks process a packet, and the mapping of these tasks onto the different processing elements of the network processor, any packet entering the network processor follows a specific sequence through the different processing elements. The flow to which this packet belongs is associated with its arrival curves. Similarly all the resources have their associated service curves. As the packets of the flow move from one processing element to the next, and also cross communication resources such as buses, both, the arrival curves of the flow and the service curves of the resources get modified following the Eqs. (3)–(6) given in Section 3. Given this, the maximum end-to-end delay experienced by any packet, the on-chip memory requirement of the network processor, and the utilization of the different resources (both computation and communication) can now be computed using Eqs. (1), (2) and (7).

To formally state the above procedure, consider that for the set of flows F entering the network processor, there is a task graph $G = (V, E)$. Any vertex $v \in V$ denotes a packet processing (or communication) task. For any flow $f \in F$, let $V(f) \subseteq V$ denote the set of packet processing tasks that have to be implemented on any packet from F. Additionally, a subset of directed edges from E defines the order in which the tasks in $V(f)$ should be implemented on any packet from f. Therefore, if $u, v \in V(f)$ represent two tasks such that for any packet belonging to f, the task v should be implemented immediately after task u, then the directed edge (u, v) belongs to the set E. Hence, for each flow f there is a unique path through the graph G starting from one of its source vertices and ending at one of its sink vertices. The vertices on this path represent the packet processing tasks that are to be implemented on packets from f.

Fig. 1 shows a hypothetical network processor architecture built out of blocks from an existing core library [13,14]. Here PPC refers to the PowerPC 440 core, and PLB and OPB refer to two buses called the processor local bus and the on-chip peripheral bus provided by the CoreConnect [14] architecture for interconnecting cores and custom logic. The numbers on the arrows in this figure indicate actions that are to be performed by the different blocks as a packet flows through the architecture, and they are ordered according to the numbering.

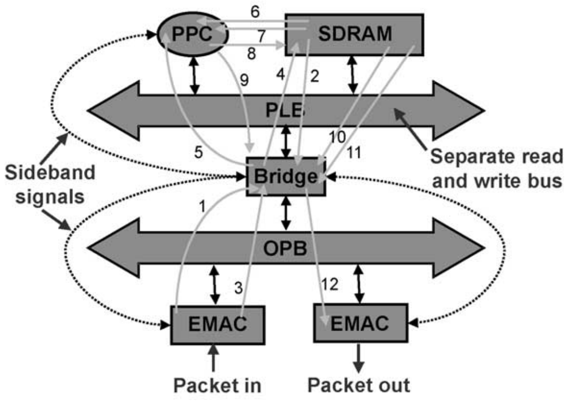


Fig. 1. A system-level model of a network processor. The figure shows the path that a packet follows through the architecture. The numbers on the arrows indicate the different actions involved (which are explained in Table 1) while the packet travels through the architecture, and specify the order in which these actions are executed. Further details of this model can be found in Section 6.

From Fig. 1 it is possible to construct a task graph considering the appropriate packet transfers from one resource to another. This task graph can then be used to compute the load on the different buses (such as the OPB and the PLB), the on-chip memory requirement of this architecture to store the buffered packets in front of each resource, and the end-to-end packet delays.

4.1. Analysis using a scheduling network

In Section 3 we described how to compute the delay and backlog experienced by a flow passing through a single resource node processing the flow. For this we characterized the flow using its arrival curves and the resource node using its service curves and also derived formulas for the maximum utilization of this resource and the outgoing arrival and resource curves. Now we extend these results to consider the case where the flow passes through multiple resource nodes as shown in Fig. 1.

The outgoing arrival curve captures the characteristics of the processed packet flow (for example its burstiness and long-term rate), which might be different from the characteristics the flow has before entering the resource. Similarly the outgoing service curve indicates the remaining processing capability of the resource after processing the flow. The idea now is to use this outgoing arrival curve as an input to another resource node (more precisely, the resource node where the next packet processing task, as given by task graph described above, is implemented). In the same way, the outgoing service curve of the first resource is used to process packets from a possibly second flow. This procedure can be illustrated using a *scheduling network*. For example, Fig. 2 shows the scheduling network corresponding to the packet traversal through the architecture shown in Fig. 1.

In general, multiple flows enter a network processor and are processed by the different resources in the order specified by the task graph described above. As packets from several flows arrive at a resource, they are served in an order determined by the scheduling policy implemented at the resource. For example, many buses use a fixed priority bus arbitration scheme. Other scheduling policies might be first-come-first-serve (FCFS) and round robin. Here, we illustrate the analytical model assuming that all the resources use fixed priority. However, the model can be extended to incorporate other scheduling policies as well.

Let us assume that there are n flows $f_1, \ldots, f_n$ arriving at a resource r, which serves these flows in the order of decreasing priorities, i.e., f_1 has the highest priority and f_n the lowest. For each packet of the flow f_i, some packet processing task t_i implemented on resource r processes the packet and this requires $w(t_i, r)$ processing units from r. For example, $w(t_i, r)$ might be the number of processor instructions, or bus cycles in case r is a communication resource. We henceforth denote $w(t_i, r)$ by w_i when it is clear which resource is being referred to. Each flow f_i arriving at r is associated with its upper and lower arrival curves $\bar{\alpha}_i^{\mathrm{u}}$ and $\bar{\alpha}_i^{\mathrm{l}}$ respectively and receives a *service* from r which can be bounded by the upper and lower service curves β_i^{u} and β_i^{l} respectively. The service available from r in the unloaded state (i.e., before any of the flows $f_1, \ldots, f_n$ are served) is bounded by the upper and lower service curves β^{u} and β^{l} respectively.

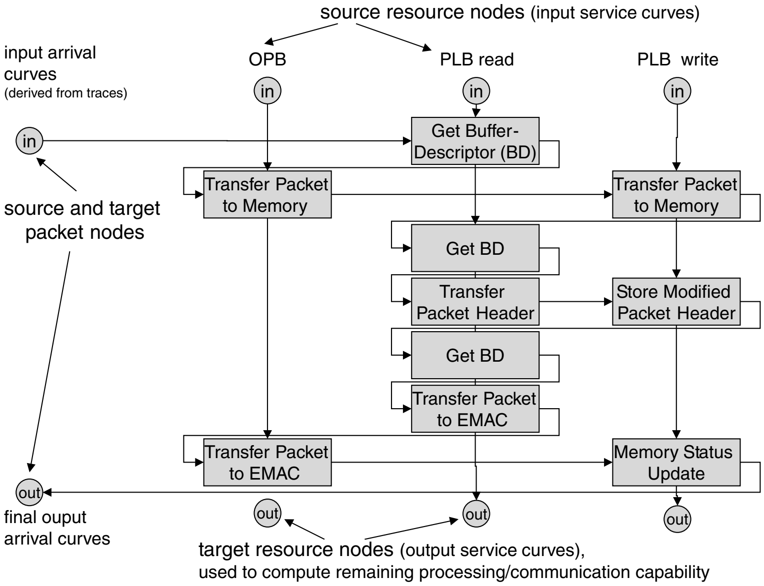


Fig. 2. The scheduling network for the architecture given in Fig. 1.

In the fixed priority scheme, r serves the flows in the order of decreasing priorities and the remaining service curve after processing a flow is used to serve the lower priority flows. The resulting arrival curves and the remaining service curves can be computed using Eqs. (3)–(6) given in Section 3. Since packets from different flows might have different processing requirements given by $w_1, \ldots, w_n$, the arrival curves first need to be scaled as described in Section 3. Similarly, the outgoing arrival curves need to be scaled back as follows. If $\alpha_i^{u'}$ and $\alpha_i^{l'}$ are the outgoing arrival curves from a resource node calculated using Eqs. (3) and (4), then $\bar{\alpha}_i^{u'} = \lceil \alpha_i^{u'}/w_i \rceil$ and $\bar{\alpha}_i^{l'} = \lfloor \alpha_i^{l'}/w_i \rfloor$. The floor/ceiling functions are used since a subsequent resource node can start processing a packet only after the task implemented on r finishes processing it.

Finally, given the service curves for the unloaded resource β^u and β^l, and the arrival curves $\bar{\alpha}_i^u$, $\bar{\alpha}_i^l$, $i = 1, \ldots, n$, we show how the service curves β_i^u and β_i^l for $i = 1, \ldots, n$ can be determined. As described before,

$$\alpha_i^u = w_i \bar{\alpha}_i^u, \quad \alpha_i^l = w_i \bar{\alpha}_i^l, \quad i = 1, \ldots, n,$$

$$\bar{\alpha}_i^{u'} = \lceil \alpha_i^{u'}/w_i \rceil, \quad \bar{\alpha}_i^{l'} = \lfloor \alpha_i^{l'}/w_i \rfloor, \quad i = 1, \ldots, n,$$

$$\beta_1^u = \beta^u, \quad \beta_1^l = \beta^l,$$

$$\beta_i^u = \beta_{i-1}^{u'}, \quad \beta_i^l = \beta_{i-1}^{l'}, \quad i = 2, \ldots, n,$$

where $\beta_{i-1}^{u'}$ and $\beta_{i-1}^{l'}$ for $i = 2, \ldots, n$ are determined from β_{i-1}^u, β_{i-1}^l, α_{i-1}^u and α_{i-1}^l by applying Eqs. (5) and (6) from Section 3. Lastly, the remaining service curve after processing all the flows is given as follows:

$$\beta^{u'} = \beta_n^{u'}, \quad \beta^{l'} = \beta_n^{l'}.$$

These can be used to compute the maximum utilization of the resource using the inequality (7). The processed flows with their resulting arrival curves $\bar{\alpha}_i^{u'}$ and $\bar{\alpha}_i^{l'}$ now enter other resource nodes for further processing.

4.2. Scheduling network construction

Using the results in the last section we now describe the procedure for constructing a scheduling network. This can then be used to determine properties of the architecture such as the on-chip memory requirement, the end-to-end delay experienced by packets and the utilization of the different on-chip resources such as processors and buses.

The inputs necessary for constructing such a network are the task graph which denotes for each flow the sequence of packet processing tasks that are to be executed on any packet of the flow and the target architecture on which these tasks are mapped.

The scheduling network contains one *source resource node* and one *target resource node* for each resource used in the architecture. Similarly, for each packet flow there is a *source packet node* and a *target packet node*. For each packet processing task of a flow there is a node in the network marked with the task and the resource on which this task is implemented. For two consecutive tasks u and v of a flow, if u is implemented on a resource r_u and v on a resource r_v then there is an edge (drawn horizontally in the Fig. 2) in the scheduling network from the node (u, r_u) to (v, r_v). For a given flow, if u and v are two tasks implemented on the same resource r and u precedes v in the task graph, then there is an edge (drawn vertically in the Fig. 2) from the node (u, r) to the node (v, r).

The arrival curves of the flows and the service curves of the resources pass from one node to the next in the scheduling network and get modified in the process, following Eqs. (3)–(6).

For a given flow f, let α_f^u be its upper arrival curve before entering the network processor. Suppose this flow passes through nodes of the scheduling network which have their input lower service curves equal to $\beta_1^l, \ldots, \beta_m^l$. Then the *accumulated lower service curve* β^l used to serve this flow can be computed as follows:

$$\bar{\beta}_1^l = \beta_1^l,$$

$$\bar{\beta}_{i+1}^l = \inf_{0 \leqslant t \leqslant \Delta} \left\{ \bar{\beta}_i^l(t) + \beta_{i+1}^l(\Delta - t) \right\},$$

$$i = 2, \ldots, m-1,$$

$$\beta^l = \bar{\beta}_m^l.$$

Now the maximum end-to-end delay and the total backlog experienced by packets from the flow f can be given by

$$\text{delay} \leqslant \sup_{t \geqslant 0} \left\{ \inf \left\{ \tau \geqslant 0 : \alpha^{u}_{f}(t) \leqslant \beta^{l}(t+\tau) \right\} \right\}, \quad (8)$$

$$\text{backlog} \leqslant \sup_{t \geqslant 0} \left\{ \alpha^{u}_{f}(t) - \beta^{l}(t) \right\}. \quad (9)$$

Compared to independently deriving the delay and backlog at single resources using inequalities (1) and (2) from Section 3 and adding them, the inequalities (8) and (9) give tighter bounds.

4.3. Approximating the arrival and service curves

The Eqs. (3)–(6) are clearly expensive to compute for general arrival and service curves. Moreover, these equations need to be computed for all the nodes of a scheduling network. Additionally, if these curves are to be meaningfully derived out of packet traces (as shown later in this paper), then the resulting curves can be described by a few parameters such as the maximum packet size, the short-term burst rate, and the long-term packet arrival rate. In view of this, we propose a piecewise linear approximation of all arrival and service curves. Using these approximations, the Eqs. (3)–(6) can be efficiently computed, thereby avoiding the computational bottleneck involved in dealing with general curves.

Each curve in this case is approximated using a combination of three straight line segments. This allows us to exactly model an arrival curve in the form of a T-SPEC [23], which is widely used in the area of communication networks. Fig. 3 shows the resulting form of the upper and lower curves (both arrival and service). Here q^{u}_{1} represents the maximum possible load on a resource for processing one packet. The slope r^{u} of the middle segment of the upper curve can be interpreted as the burst rate, and the slope s^{u} as the (load on a resource due to the) long-term packet arrival rate. In the case of communication resources, q^{u}_{1} represents the maximum packet size. The values of p^{u} and p^{l}_{1}, p^{l}_{2} can be computed from these parameters.

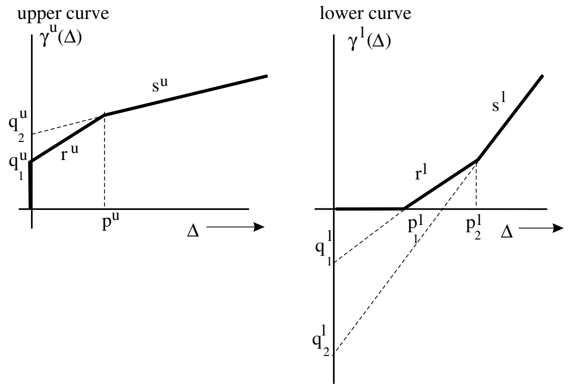


Fig. 3. Piecewise linear approximations of the upper and lower (arrival and service) curves.

Any upper (say γ^{u}) and lower (say γ^{l}) curves can now be written as the following:

$$\gamma^{u}(\Delta) = \min\left\{q^{u}_{1} + r^{u}\Delta, q^{u}_{2} + s^{u}\Delta\right\},$$
$$\gamma^{l}(\Delta) = \max\left\{q^{l}_{2} + s^{l}\Delta, q^{l}_{1} + r^{l}\Delta, 0\right\},$$

where

$$q^{u}_{2} \geqslant q^{u}_{1} \geqslant 0, \quad r^{u} \geqslant s^{u} \geqslant 0, \quad r^{u} = s^{u} \Longleftrightarrow q^{u}_{1} = q^{u}_{2},$$
$$q^{l}_{2} \leqslant q^{l}_{1} \leqslant 0, \quad 0 \leqslant r^{l} \leqslant s^{l}, \quad r^{l} = s^{l} \Longleftrightarrow q^{l}_{1} = q^{l}_{2}.$$

Using these curves, the Eqs. (3)–(6) as well as maximum delay and backlog can be evaluated efficiently using symbolic techniques.

5. The simulation setup

Even in cases where analytical models exist, performance evaluation of processor architectures using simulation still continues to be the most widely used procedure. This is primarily motivated by the accuracy of the results that can be obtained using simulation, and the second reason being flexibility. In many cases analytical models cannot capture all the aspects of an architecture and are often restricted to one particular level of abstraction.

In this section we outline the methodology for model composition and performance evaluation of network processor architectures based on simulation, which forms the basis for comparing the results obtained from the analytical framework presented in the last two sections. This section is based on the work presented in [32] and more detailed explanations concerning the models can be found there. Here the primary goal is to illustrate the abstraction level at which the different components of the architecture are modeled. Section 6 presents the results obtained by evaluating a reference network processor architecture using this simulation method along with the results obtained by the analytical model on the same architecture.

5.1. Modeling environment and software organization

The overall approach is based on using a library of reusable component models written in SystemC [12,25]. These include models for buses, different interfaces, processor cores, etc. Each of these models can be an abstract match of an already existing hardware component which can be found from a core library [13], or can also be a model of a new component which does not exist yet. In an architecture composition step the selected component models are combined into a system model which is then evaluated by simulation. The workload used to drive the simulation can either be synthetically generated, or can be obtained from real traffic traces. During simulation, the model execution performance data can be collected which can then be evaluated later.

It is not necessary that every component model is implemented on the same level of abstraction. But all models are implemented in the form of a *black-box* having well defined abstract interfaces and allow for component-specific local refinements. This supports easy exchangeability among different models of the same component. Every component model is separated into two layers—a so-called abstract functional layer and a data collection layer. The functional layer describes the functional behavior of a component and defines the component interfaces, while the data collection layer exclusively deals with gathering statistics which are used to evaluate different performance metrics. This separation enables independent and selective refinements on the functional layer and also flexible customization of the data collection mechanisms.

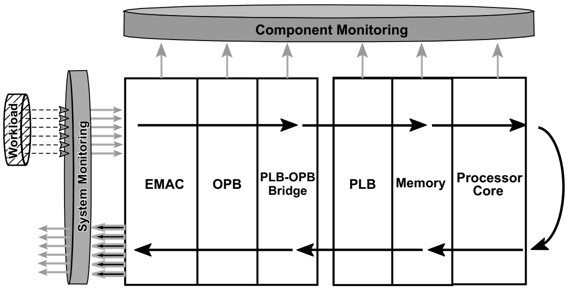


Fig. 4. System evaluation and component evaluation of a network processor architecture.

The performance evaluation done using the data collected during simulation can be distinguished into either a component level evaluation or a system evaluation. These are illustrated in Fig. 4 (note that this shows the organization of the framework; the models of the different architectural components might vary in their level of abstraction and details). The component evaluation is based on the data collected through the data collection layer of each component model. Examples of such evaluation metrics can be memory fill levels, bus rearbitration counts, and the load on the different components. System specific aspects of an architecture like the overall system throughput, or end-to-end packet delays are evaluated using system evaluation mechanisms. In contrast to the component evaluation approach, the data in this case is not gathered within a specific component but is collected using the workload traveling through the entire system.

An overview of the entire simulation framework is given in Fig. 5. Based on specification documents of already existing components, a set of abstract models are created and combined with component- and system-specific data collection metrics. Such models constitute, what is referred to in the figure as the "Model Library". The architectural models from this library can be composed together and then simulated on a workload.

5.2. Component modeling

In this section we briefly describe how each component of the reference network processor architecture that is evaluated in Section 6 is modeled in the simulation. This would enable a meaningful comparison between the results obtained by simulation and those obtained from the analytical model. The models usually available from core libraries are in the form of hard or soft cores, whereas SystemC models used in the

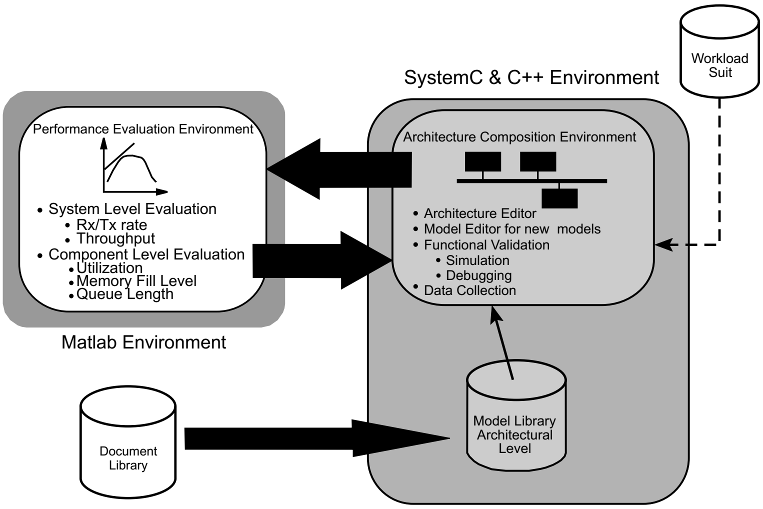


Fig. 5. Overview of the simulation framework that is used for comparing the results obtained by the analytical method with simulation results.

simulation were created for the work done in [32]. The discussion below refers to these SystemC models.

5.2.1. Bus models

The bus models used in the reference architecture are based on the CoreConnect bus architecture [14], designed to facilitate core based designs. CoreConnect involves three buses: The processor local bus (PLB) is for interconnecting high performance cores with high bandwidth and low latency demands, such as CPU cores, memory controllers and PCI bridges. The on-chip peripheral bus (OPB) hosts lower data rate peripherals such as serial and parallel ports and UARTs. The PLB is fully synchronous, has decoupled address, read and write data lines and transfers on each data line are pipelined. Different masters maybe active on the PLB address, read and write data lines and access to the PLB is granted through a central arbitration mechanism that allow masters to compete for bus ownership.

The models used for the PLB and the OPB are both cycle accurate. Both these models do not transfer any data nor consider any address, but model the signal interchanging according to the bus protocol.

There is a third bus in CoreConnect, which we do not use in our study. This is called the device control register (DCR) bus, and is used for reading and writing low performance status and configuration registers.

The features of the PLB that are modeled for our study include the arbitration mechanism, the decoupled address and read and write lines, a fixed length burst protocol, and a slave enforced arbitration mechanism. The OPB is much less sophisticated compared to the PLB and most of its features are modeled. For both the buses, the arbitration algorithm uses a round robin strategy among the requesting masters on a given priority level, and typically there are four priority levels.

5.2.2. Ethernet core (EMAC) model

The Ethernet core or EMAC is a generic implementation of the Ethernet media access control (MAC) protocol, supporting both half-duplex (CSMA/CD) and full duplex operations for Ethernet, Fast Ethernet and Gigabit-Ethernet. The EMAC has two large FIFOs to buffer packets, and two OPB interfaces—one for providing access to its configurations and status registers, and the other is a bidirectional interface to transfer data to and from the PLB–OPB bridge.

The model of the EMAC only contains receiving and transmitting channels, where a receive channel can be considered as an input port and a transmit channel as an output port.

The set of receiving channels constitutes the traffic input to the network processor. Each one reads packet information from a real traffic trace at a parameterizable rate. Within a receive channel there are two threads of activity. The first one reads the input packet traces, and writes each resulting packet into a FIFO. The second thread implements the communication protocol with the PLB–OPB bridge and transfers packets to memory as long as the FIFO is not empty. The transmit path consists only of a *transmit packet* thread, which is active as long as packets are waiting to be transferred to the appropriate port.

5.2.3. PLB–OPB bridge model

The PLB–OPB bridge is a specialized module which combines pure bridge functionality with DMA capability, and it can effectively handle packet transfer overheads. An EMAC communicates with a PLB through the PLB–OPB bridge. Since as an OPB slave, the EMAC cannot inform the bridge of its willingness to transfer a packet, the EMAC to PLB–OPB bridge interface has *sideband* signals to serve this purpose. These do not form a part of the OPB bus, and nearly all signals are driven by the EMAC and sampled by the bridge.

The PLB–OPB bridge is modeled as two independent paths, receive and transmit, and both offer the bridge functionality and arbitrate among active channels. Each path implements the PLB–OPB bridge to EMAC communication protocol by driving the required sideband signals and accessing buses. Bus accesses are concurrent and therefore both paths can contend for their access, especially on the OPB.

5.2.4. Memory model

The memory is accessed via the PLB and can either be on-chip or off-chip. It is modeled in a high-level way, where only parameters like average or worst case transfer latency are considered.

5.2.5. Software application and timing

A simple high-level model of software application is used. It primarily consists of the following. For each packet, the software can cause a pure delay without generating any PLB load, representing processing time in the CPU. Second, there can be a PLB load generated by the software (for example, this might consist of reading and writing packets by the CPU to and from the memory). Lastly, the timing model is based on using the PLB clock as the system time.

6. A comparative study

This section presents the main result of this paper—a comparison of the performance evaluation data obtained by the analytical framework presented in Sections 3 and 4 on a reference network processor architecture, with the results obtained by detailed simulations based on the models discussed in Section 5. This comparative study is based on the assumption that there is a high confidence in the simulation results. We do not compare the results obtained by either the analytical method or the simulations with any real hardware implementation because of two reasons: (i) During a design phase actual hardware does not exist, and the best one can do is to validate the results obtained from an analytical model against those obtained using simulation, and vice versa. (ii) Simulations are widely used in practice, and assuming that the processor model being simulated is accurate and detailed enough, the results are expected to match the real hardware with high confidence.

Our choice of the reference architecture which is the basis of the comparison is a hypothetical system-level model of a network processor which can be matched by many existing network processors (such as the family of processors described in [21]). The architectural components modeled in this study are from an existing core library [13]. Since the particular system-level model we study here is sufficiently general, we believe that the conclusions based on the results obtained from this study would hold for many other models. This enables us to make the general claim that it is possible (and more appropriate in terms of the benefits in running time) to use analytical modeling frameworks during the early stages of architecture design space exploration in the context of network processors, in order to tackle the design complexity.

We evaluate the reference architecture using three different performance metrics: (i) The line speed or the end-to-end throughput that can be supported by the architecture. This is measured using the utilization of the different components of the architecture and hence also identifies the component which acts as the bottleneck. During a design space exploration, identifying the utilization of the different components goes beyond measuring the overall throughput of the system because a designer is generally interested in identifying whether all the components in the architecture have a moderately high utilization at the maximum supported line speed, or whether it is one single component that acts as a bottleneck. (ii) The maximum end-to-end delay that is experienced by packets from the different flows being processed by the architecture. (iii) The total on-chip buffer/memory requirements, or in other words, the on-chip memory fill level. The results of the analytical framework should be judged on the basis of how closely the data related to these performance metrics for the reference architecture match those obtained using simulation. Rather than absolute values, it is more interesting to analyse the behavior of the architecture (for example with increasing line speeds, or increasing packet sizes for the same line speed), and see if the same conclusions can be drawn from both evaluation techniques. The second axis for our comparisons is the time it takes to compute the evaluation data by the analytical framework, compared to the simulation time required to generate this data.

6.1. Reference architecture and parameters

The system-level model of a network processor that is used for this study is shown in Fig. 1. The different actions that are executed while each packet travels through this architecture, and the order in which they are executed is given in Table 1. The model effectively deals with the communication subsystem of the architecture, and the software application running on the CPU core (indicated by PPC-PowerPC) is modeled as simply performing two reads from the SDRAM and one write to the SDRAM for every processed packet. The amount of data read or written (and hence the traffic generated on the PLB), however, depends on the packet size and this is appropriately modeled.

Table 1
Sequence of actions for every processed packet in the architecture model shown in Fig. 1

Step	Action
1	Sideband signal from EMAC to bridge (indicating that a new packet has arrived)
2	Bridge gets a "buffer descriptor" (BD) from the SDRAM
3	Packet is sent from EMAC to bridge over the OPB
4	Packet is sent from bridge to SDRAM over the PLB write bus
5	Sideband signal from Bridge to PPC (indicating that the new packet has been stored)
6	CPU get buffer descriptor over the PLB read bus
7	CPU gets packet header over the PLB read bus
8	CPU processes header, and stores it back to SDRAM over the PLB write bus
9	Sideband signal from bridge to CPU (indicating that the packet can be sent out)
10	Bridge gets buffer descriptor over the PLB read bus
11	Bridge gets packet over the PLB read bus
12	Packet sent out to specified a EMAC over the OPB

As seen in Fig. 1, the architecture is composed of two Ethernet media access controllers (EMACs), a slow on-chip peripheral bus (the OPB), a fast processor local bus (the PLB) consisting of separate read and write lines, a PLB–OPB bridge, a SDRAM and a processor core (PPC). Each EMAC consists of one receive and one transmit channel, and is capable of reading packets at parameterizable input rates.

In the simulation, the entire path of a packet through the modeled architecture can be described as follows. First a receive channel of an EMAC reads a packet from a file containing the packet traces (only packet lengths are used, and all packets arrive back-to-back with a fixed inter-frame gap; this is described in further details later), and allocates a packet record which contains the packet length and the source EMAC identification. This packet record models the packet inside the processor architecture and generates a load equivalent to the size of the packet. The channel then requests service to the PLB–OPB bridge via a sideband signal, which is served following a bridge internal arbitration procedure. The PLB–OPB bridge fetches a *buffer descriptor* for the packet (which is a data structure containing a memory address in the SDRAM, where the received packet is to be stored). This fetching operation involves the SDRAM and generates traffic on the PLB read bus, equal to the size of the buffer descriptor. Following this, the received packet is stored in the SDRAM at the location specified by the buffer descriptor. This involves the packet traversing through the OPB to the PLB–OPB bridge, and then through the PLB write bus to the SDRAM. This generates a load equal to the size of the packet, on both the buses. Since data on the PLB is sent in bursts, the PLB–OPB bridge schedules a PLB transfer only when sufficient data is gathered. As the EMAC channel is served over and over again, the packet is written part by part into the SDRAM. After the packet transfer is complete, the bridge informs the EMAC receive channel via a sideband signal, and also notifies the application software running on the processor core (again by a sideband signal) that the packet is now available in the memory. It is then processed by the software as soon as the processor becomes available. This processing involves a buffer descriptor transfer from the SDRAM to the processor core via the PLB read bus, followed by a packet header transfer, again from the SDRAM to the processor core via the PLB read bus. The packet header is then processed in the processor core (for example, implementing some lookup operation) and written back into the SDRAM over the PLB write bus.

After the completion of this processing, the software notifies the bridge (via a sideband signal) that the packet is now processed and is ready to be sent out through the chosen transmit channel of the EMAC. As in the receive path of the packet, the bridge gets the buffer descriptor of the packet from the SDRAM via the PLB read bus, and then the packet traverses over the PLB read bus and the OPB to an EMAC transmit channel. After the completion of the packet transfer, the EMAC notifies the bridge via a sideband signal, which then reads certain status information and releases the buffer descriptors.

This entire process happens concurrently for two packet flows entering through the two EMACs of the architecture. All the components involved also work concurrently and the two buses (the PLB and the OPB) use FCFS as a bus arbitration policy. The main complexity in the analysis of this system is due to concurrent operation of the different components. Hence it is non-trivial to evaluate how the system behaves with increasing line speeds, variations in packet sizes, and what is the maximum line speed that it can support without packet dropping.

6.1.1. Parameters

As already mentioned, the EMAC can read packets at different input line speeds. The line speeds used for the evaluation range from 100 to 400 Mbps, the former representing a nominal load situation and the later a loaded situation.

The OPB modeled has a width of 32 bits and a frequency of 66.5 MHz. The read and the write data paths of the PLB are of 128 bits and operate at 133 MHz. The size of a PLB burst is limited to a maximum of 64 bytes. Therefore, the PLB–OPB bridge gathers up to 64 bytes (which is only one OPB burst transfer) before scheduling the PLB transfer. There are two different kinds of buffer descriptors, small and large ones. The small buffer descriptors refer to memory locations/buffers with 64 bytes of size, while the large ones refer to buffers with a size of 1472 bytes. As a consequence, 64 byte packets require only one small buffer descriptor and packets larger than 64 bytes require an additional large buffer descriptor. Both *small* and *large* buffer descriptors are of size 64 bytes each. Therefore, the traffic generated by a packet on any of the buses depends not only on its own size, but also on the buffer descriptors associated with it. All packets and the buffer descriptors reside in the SDRAM described above.

6.2. Evaluation method and comparisons

The reference architecture described above is evaluated using simulation and the analytical framework, using two different workload types—synthetic traces with same-sized packets, and real traces from NLANR [18]. For the synthetic traces, packet sizes of 64, 128, 512, 1024, 1280 and 1500 bytes are used. The real traces are used only to exploit the impact of real world packet size distributions on the system performance. They are time compressed and adjusted and only the packet sizes are retained. Therefore, in both cases packets arrive back-to-back (to exert the maximum stress on the architecture) with an interframe gap equal to 20 bytes.

The overall scheme for comparing the results obtained using the analytical framework with those obtained from simulation is shown in Fig. 6. The different components of the architecture are modeled in either SystemC in the simulation based evaluation, or analytically using the model presented in Section 3. To compute the required parameters of an analytical component model (such as the transfer time of a single packet over an unloaded bus), either simulation results or data sheets of the component are used. The component models are then composed together (using the methods described in Section 4 in the case of the analytical framework, and using standard SystemC composition techniques in the case of simulation) to obtain a system model of the architecture.

Recall that the analytical model considered here does not use real packet traces, but instead uses arrival curves modeling the traces in terms of their maximum packet size, burstiness, and long-term arrival rate. These parameters were extracted from the traces as shown in Fig. 7 and fed into the model for evaluation. For the upper arrival curve, the maximum number of bytes that can arrive (at the network processor) at any time instant is given by the largest sized packet, the short-term burst rate is given by the maximum number of largest sized packets that can be seen occurring back-to-back in the trace, and the long-term arrival rate is given by the total length (in bytes) of the trace divided by the time interval over which all the packets in this trace arrive.

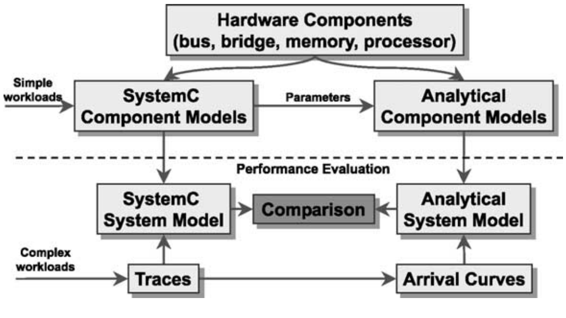


Fig. 6. The overall scheme for comparing the results from the analytical framework with those obtained by simulation.

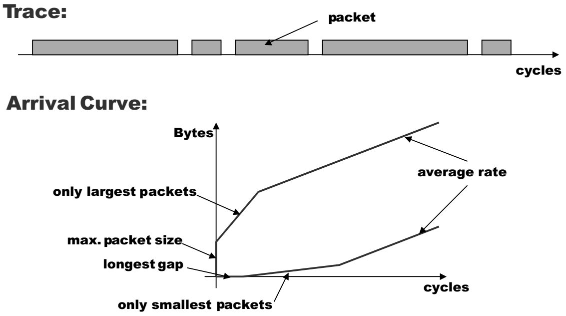


Fig. 7. Obtaining arrival curves from packet traces in the analytical framework.

Similarly, for the lower arrival curve, the maximum time interval over which no traffic can arrive is equal to the interframe gap in the trace (equal to 20 bytes), the bound on the minimum number of bytes that can arrive over a time interval is given by the maximum number of minimum sized packets occurring back-to-back in the trace, and the long-term arrival rate is equal to that in the upper arrival curve.

Given any packet trace, arrival curves such as those shown in Fig. 7 can be derived from the trace and they capture the traffic arrival pattern given by the trace. Note that here we restrict each arrival curve to be made up of a combination of three line segments in order to simplify the computations involving these curves. However, in general they can be arbitrarily complex to capture the exact details of a trace (albeit, at the cost of increasing the computational complexity). As mentioned in Section 4 the analytical model composes the different component models, resulting in a *scheduling network*. For the architecture we study here (Fig. 1), such a scheduling network is given in Fig. 2.

6.3. Evaluation results

Table 2 gives the utilization values of the three buses (the OPB, and the PLB read and write bus) when the model is fed with synthetic traces con-

Table 2

The utilization values of the OPB, and the PLB read and write buses, when the model is fed with six different synthetic traces consisting of fixed size packets (ranging from 64 to 1500 bytes)

	OPB		PLB read		PLB write	
	AnM	Sim	AnM	Sim	AnM	Sim
Packet size 64 bytes						
100 Mbps	18	18	14	13	6	5
150 Mbps	27	28	20	19	9	8
200 Mbps	36	37	27	25	12	10
250 Mbps	45	46	34	31	15	13
300 Mbps	54	55	41	37	17	15
350 Mbps	63	65	48	40	20	17
400 Mbps	72	76	55	47	23	20
Packet size 128 bytes						
100 Mbps	19	19	15	14	5	5
150 Mbps	29	28	22	21	7	7
200 Mbps	38	38	30	28	10	10
250 Mbps	48	47	37	35	12	12
300 Mbps	57	56	45	42	15	15
350 Mbps	69	66	52	48	17	16
400 Mbps	79	75	59	53	20	18
Packet size 512 bytes						
100 Mbps	20	20	7	7	3	3
150 Mbps	30	29	11	11	5	4
200 Mbps	40	39	15	15	7	6
250 Mbps	50	49	19	19	8	7
300 Mbps	60	59	22	22	10	8
350 Mbps	71	69	26	26	12	10
400 Mbps	82	79	30	30	13	11
Packet size 1024 bytes						
100 Mbps	20	20	6	6	3	2
150 Mbps	30	30	9	9	4	4
200 Mbps	40	40	12	12	6	5
250 Mbps	50	50	15	15	7	6
300 Mbps	60	59	18	18	9	7
350 Mbps	71	69	21	21	10	8
400 Mbps	83	79	25	24	12	9
Packet size 1280 bytes						
100 Mbps	20	20	6	6	3	2
150 Mbps	30	30	9	9	4	4
200 Mbps	40	40	12	12	6	5
250 Mbps	50	50	15	15	7	6
300 Mbps	60	60	18	18	9	7
350 Mbps	71	69	20	21	10	8
400 Mbps	83	79	23	24	12	9
Packet size 1500 bytes						
100 Mbps	20	20	6	6	3	2
150 Mbps	30	30	9	9	4	4
200 Mbps	40	40	12	12	6	5
250 Mbps	50	50	14	15	7	6
300 Mbps	61	60	17	18	9	7
350 Mbps	72	70	20	21	10	8
400 Mbps	83	80	23	24	12	9

Table 2 (*continued*)

For each trace and for each bus, the first column (marked as AnM-analytical method) gives the results obtained using the analytical model, and the second column (marked as Sim) gives the results obtained by simulation.

sisting of fixed sized packets. Here, six different packet sizes have been used, from 64 to 1500 bytes. For each packet trace and bus combination, the table compares the results obtained from the analytical method with those resulting out of simulation for different line speeds. To give an impression of how the utilization of the different buses increase with the line speed for the same packet size, in Fig. 8 we plot the utilization values for the trace containing 512 byte sized packets. As can be seen from Table 2, the results for the other traces are very similar, and hence we do not plot them.

There are two things to be noted from these values. First, with increasing line speeds the utilization of the different buses also increase, and as expected, this increase is proportional to the increase in the line speed. Second, the results from

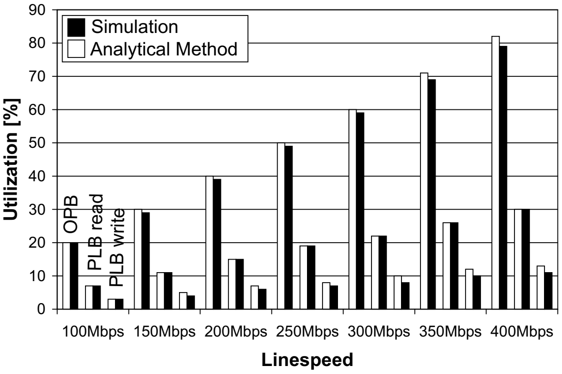


Fig. 8. Utilization values for different line speeds for the trace containing 512 byte packets. In this bar graph, for each line speed, the first bar indicates the utilization of the OPB, the second bar shows the utilization of the PLB read bus and the third bar corresponds to the PLB write bus. For each bus, the white bar gives the result computed by the analytical method, and the black bar gives the result obtained from simulation.

the analytical method and the simulation match very well for the utilization values. In Fig. 8, for each line speed there are three bars, each corresponding to the OPB, the PLB read and the PLB write bus. It may be noted from the figure that the maximum line speed that this architecture can sustain is in the range of 400 Mbps and the OPB acts as a bottleneck (as most of the traffic is generated on it).

In Fig. 9 we show the utilization values of the PLB read bus for fixed line speeds, as the packet size is increased. For a fixed line speed, as the packet size is increased, the component of the utilization that comes from the packet traversal increases, since there is less total interframe gap in the whole trace (assuming that the trace size in bytes remains the same). This is because the number of packets in the trace decrease. However, because of this, the number of buffer descriptor and packet header traversals also decrease and therefore the component of the bus utilization that comes from these also decrease. These effects can be seen in Fig. 9. As the packet size is doubled from 64 to 128 bytes, the first component mentioned above plays a dominating role and hence the utilization slightly increases. Thereafter, the effect of the second component dominates and the utilization falls, and then remains almost constant since there is no significant change in the number of packets as the packet size is increased from 1024 to 1280 bytes and then from 1280 to 1500 bytes. The same results for the OPB is shown in Fig. 10. However, in this case the utilization increases with increasing packet size because these are no packet header or buffer descriptor transfers over this bus.

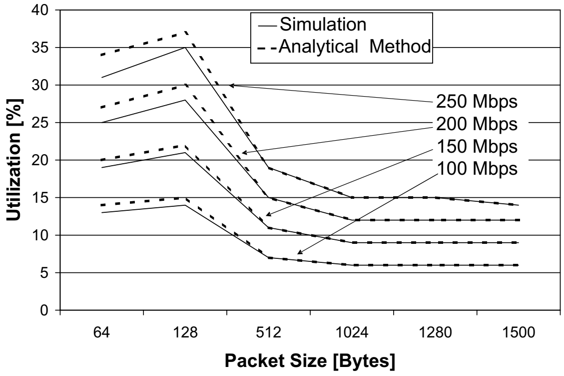


Fig. 9. The variation of the PLB read bus utilization with increasing packet size for four different line speeds.

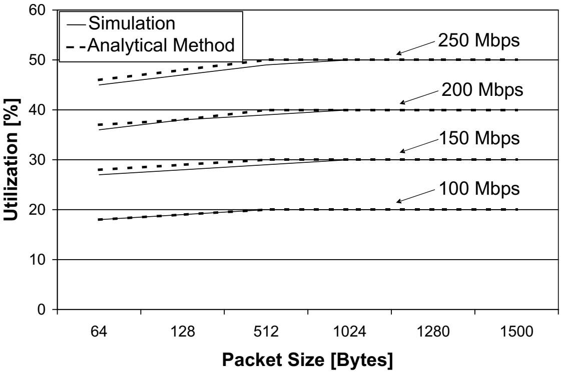


Fig. 10. The variation of the OPB utilization with increasing packet size for four different line speeds.

It is to be noted that the match between the analytical results and the simulation is always close enough to deduce the above conclusions from the analytical results itself (with significant savings in evaluation time).

For the fixed size packet traces, we do not consider the end-to-end packet latencies and the memory fill levels, since for all low load situations they remain constant and do not depend on the input line speed.

We next consider the results generated by real traces obtained from NLANR [18]. We use three different traces—FL (traces from a number of Florida universities), SDC (traces collected from the San Diego Supercomputer Center) and TAU (traces from the Tel Aviv University). Each trace is made up of traffic patterns for two different lines and these are fed into the two EMACs in our architecture. The main motivation behind using these traces is to see the effect of real-life packet size distributions on the architecture. The line speeds used for all the traces vary from 100 to 400 Mbps as before.

Figs. 11–13 show the variation in the utilization of the three different buses for the different line speeds. It maybe noted that, as before, there is a close match between the results from the simulation and the analytical framework. Secondly, the architecture behaves almost identically for the different traces.

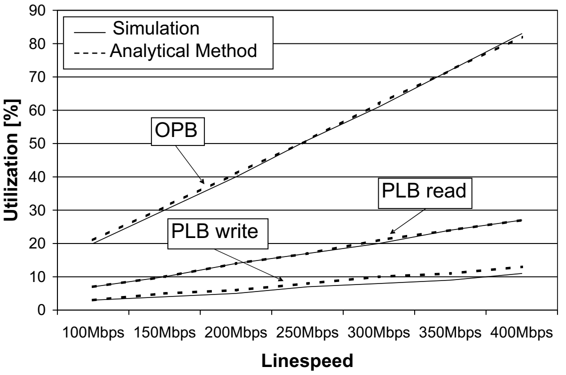


Fig. 11. The utilization of the OPB and the PLB read and write buses under different line speeds for the FL packet trace.

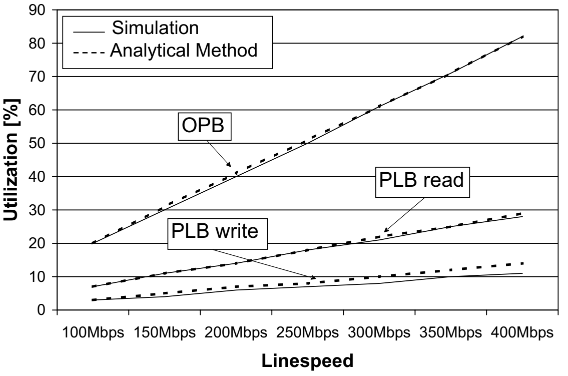


Fig. 12. The utilization of the OPB and the PLB read and write buses under different line speeds for the SDC packet trace.

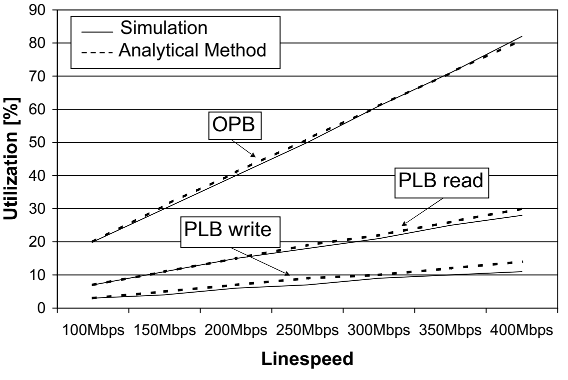


Fig. 13. The utilization of the OPB and the PLB read and write buses under different line speeds for the TAU packet trace.

Recall that the bus arbitration mechanism used in our reference architecture is always FCFS. Unfortunately, for FCFS there do not yet exist tight bounds for delay and backlog in the analytical framework that we consider here (there do exist tight bounds for static priority, round-robin, time division multiplexing, etc.). To get around this problem, we use fixed priority based arbitration mechanisms in the analytical model and compare them with FCFS used in the simulation. Towards this, one of the packet flows (recall that each trace is made up of two flows) in a trace is assigned a higher priority over the other in all the buses. For computing the end-to-end packet latency, the maximum delay experienced by the lower priority flow now gives an upper bound on the maximum delay experienced by any packet when FCFS is used. Similarly, we use the maximum delay experienced by the higher priority flow as a lower bound on the maximum delay experienced by any packet in the case of FCFS. These results are shown in Figs. 14–16 for the three different traces. Note that in all the three cases, the delay values obtained through simulation lie in between the delay values (obtained from the analytical method) for the high and the low priority flows. These results indicate that the architecture is sufficiently provisioned for one flow, even for high line speeds, since the maximum delays experienced

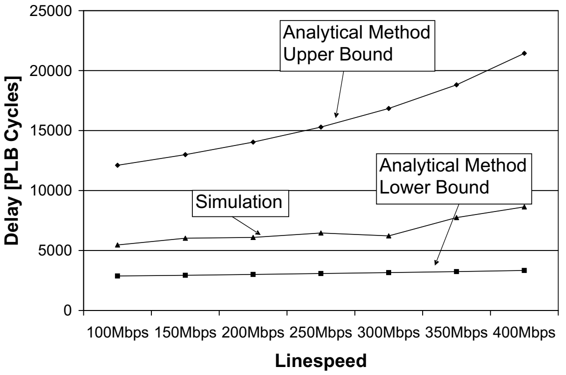


Fig. 14. The maximum end-to-end delays experienced by packets of the FL trace under different line speeds. For the two flows that make up this trace, the analytical results show the delay experienced by the higher and the lower priority flows when using priority based arbitration at the buses. The simulation results are based on FCFS implemented at all the buses.

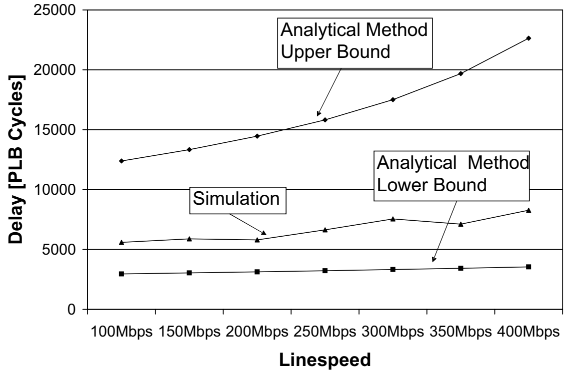


Fig. 15. The maximum end-to-end packet delays experienced by packets of the SDC trace under different line speeds.

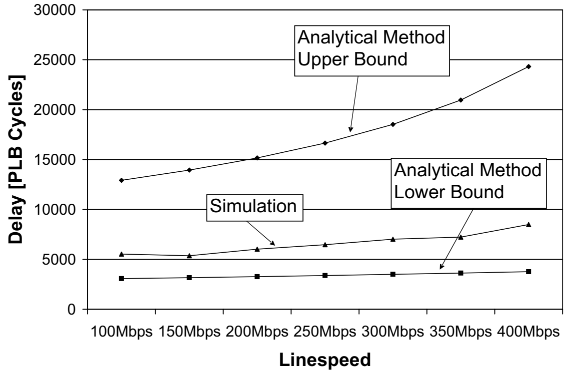


Fig. 16. The maximum end-to-end packet delays experienced by packets of the TAU trace under different line speeds.

by packets from the high priority remain constant with increasing line speeds for all the three buses. For the low priority flow, as expected, the delay values increase with increasing line speeds. When FCFS arbitration policy is used, the delays suffered are more than those suffered by packets from the high priority flow, but less than those suffered by the low priority flow.

In Fig. 17, for each trace we first assign a high and a low priority to the two flows and measure the maximum delay experienced under this priority assignment using the analytical method. Then we reverse this priority assignment and again measure the maximum delay, and finally average the two maximum delays for each flow. Fig. 17 shows this averaged maximum delay (we choose the flow for

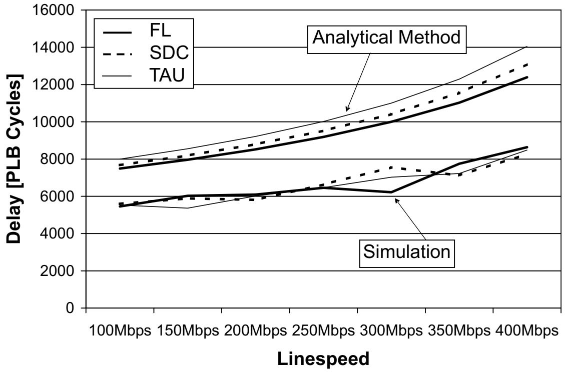


Fig. 17. The maximum end-to-end packet delays experienced by packets of all the traces under different line speeds. The plots corresponding to the analytical method shows the average of the maximum delays experienced by packets from the low and the high priority flows, when using priority based arbitration at the different buses. The simulation results are based on using FCFS at all three buses.

which this averaged maximum delay has a higher value) for the analytical method. The simulation results, as before, are based on FCFS at all the buses.

Lastly, Fig. 18 shows the on-chip memory requirements (measured in terms of the backlog) obtained by the analytical method and by simulation. In the analytical case, as before, we use priority based arbitration at the different buses and the simulation results are based on FCFS.

It maybe noted that although the analytical and the simulation results for the end-to-end packet delays and the memory fill levels do not match as

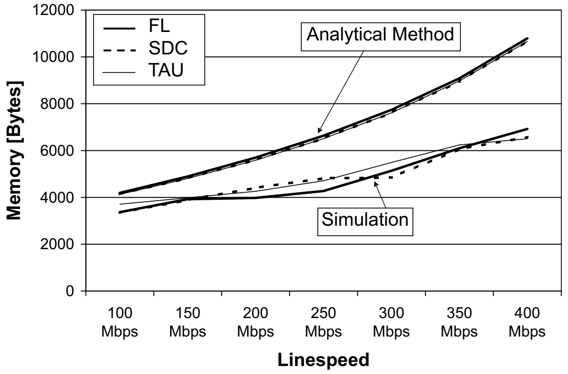


Fig. 18. The buffer memory requirements/memory fill levels for the different traces under different line speeds.

closely as the results for the utilization values, the "trends" indicated by both the methods do match. We believe that such trends, to a large extent, would suffice to make the architectural decisions that are involved in any system-level design space exploration. One of the reasons why there is a mismatch between the values obtained by the analytical method and those obtained by simulation is that the former computes *worst case* delays and backlogs. In any particular simulation run, such worst case results may not occur. Additionally, the results obtained using the analytical framework to a very large extent depend on how tight are the different bounds for calculating the delay, backlog and resource utilization for the different scheduling policies. There is still a significant amount of work to be done in this direction (see, e.g., [4]), and we hope that there will be more results in the future to improve this framework.

For all the results reported here, the simulations run in time in the order of a few minutes to several hours. In contrast to these, the analytical procedure completes execution in time less than a second for all the traces and is the only feasible option for performance evaluation in any automated design space exploration process.

7. The role of multiple evaluation frameworks in a design flow

7.1. Accuracy and evaluation times in the context of design space exploration

Traditionally, design flows for embedded processors involve architecture exploration by iterative improvement. In this approach, an initial target architecture is generated and specified in a machine description language following an analysis of the applications. Application code is then compiled for this target architecture and executed on an instruction level simulator. The performance data and statistics gathered from such a simulator are then fed into a hardware model of the target architecture to derive values of performance metrics such as power consumption, die size, etc. These values help in evaluating the architecture, identifying bottlenecks and making improvements. New architectures generated through these improvements are again evaluated following the same steps and this cycle is repeated until no further improvements are possible. This whole iteration process is largely manual and ad-hoc.

However, modern embedded systems are increasingly becoming more complex and heterogeneous, and are additionally programmable. Network processors are certainly a good showcase for such trends. This makes predicting performance behavior more difficult since designing accurate simulators and performance evaluation tools for such heterogeneous systems are difficult and expensive. Further, high simulation times make automatic design space exploration impractical even during very early stages of the design.

To address these problems, new approaches to explore the design space of such systems are being considered (see for example [20]). Here, the goal is to determine an appropriate system-level architecture template during the initial phase of the design space exploration. This involves answering questions such as: Which architectural components are to be chosen? How should they be interconnected? Which tasks should be mapped to which components? Once such an architecture template is determined, further details of this template, to convert it into a detailed architecture, can then be considered. This process involves different performance evaluation techniques for the different levels of abstraction. Performance evaluation at multiple abstraction levels enable the desired speed for a design space exploration during the early stages of a design, and the accuracy required at the later stages. Modeling platforms and system-level languages such as SystemC are being developed to address these issues (see [25]).

A second possibility is to make use of fast analytical performance evaluation methods during the early stages of a design, where designers only need rough performance estimates of a template architecture and therefore a lot of architectural details can be abstracted away, making good analytical models relatively easy to construct. Once a reasonably small region of the architecture design space (captured by the template) is identified, architectures from this space can be subjected to a more extensive evaluation using cycle accurate simulation techniques. Our study in this paper, comparing the results obtained from a system-level analytical performance evaluation method with those obtained by simulation, suggests this possibility in the context of network processors. The analytical framework considered here, to a large extent can be adapted to various abstraction levels. This depends on what each vertex in the underlying task graph described in Section 4 is used to model. In the experiments reported in Section 6, the vertices in this task graph modeled relatively low level details of an architecture such as bus communication. Nevertheless, most of the performance values obtained at this level of abstraction match fairly well with the results obtained using detailed simulation, and within a time which is orders of magnitude faster. For example, the utilization values of the different buses obtained using the analytical method match simulation results very closely. For the other performance metrics, the analytical method provides results based upon which similar conclusions about the architecture can be drawn, as those from detailed simulations.

7.2. A design flow for network processors

Based on the above results, one can meaningfully conclude that currently there exist analytical performance evaluation models for network processor architectures which can enable an automatic system-level design space exploration during the early to intermediate phases of a design. Since the design space at these stages can be fairly large due to the combinatorial nature of the decisions concerning the architecture that are to be made, such models can help in evaluating a large number of designs within a short span of time. Once an interesting region of the design space is identified, in the form of one or more parameterizable template architectures, one can then resort to simulation based techniques to obtain accurate performance values of these architectures where all the lower level details are also taken into account.

This proposed design trajectory can be visualized as Fig. 19. It is primarily based on the above argument that during the later stages of a design, some form of architectural template already exists—for example, it is known how many processor cores and other dedicated hardware units are to be used and how they are interconnected using the on-chip communication infrastructure, and which tasks are mapped to which processors. Design issues at this stage mostly concern the tuning of different component parameters such as bus width, cache sizes, etc., or optimally determining where

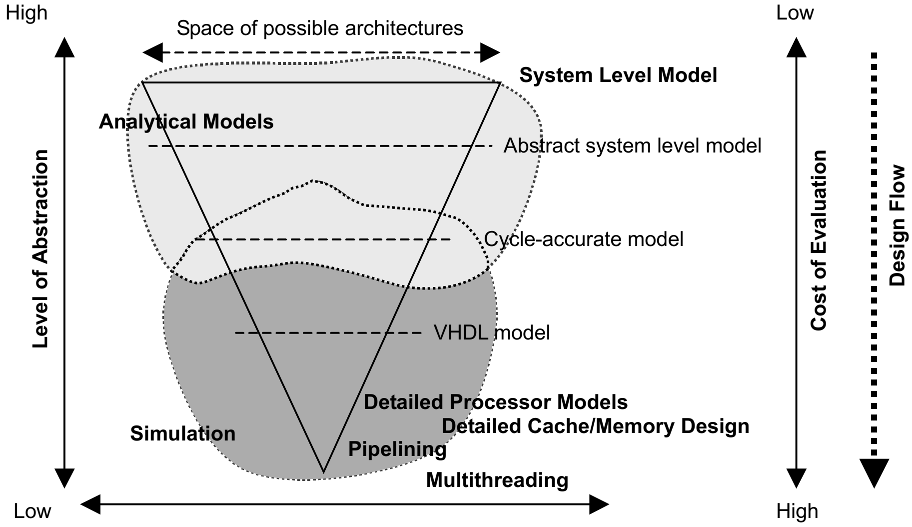


Fig. 19. Different stages of design space exploration and the associated evaluation technique.

various packet related data structures need to be stored. These possibilities can be exhaustively simulated to determine the optimal configuration.

8. Concluding remarks

We have presented a detailed comparison study of an analytical performance evaluation framework for network processor architectures with a simulation based technique. The underlying assumption here has been that there is a high confidence in the simulation results. But, obtaining these results is time intensive because of the high simulation times involved, thereby rendering simulation based models to be inappropriate for the early stages of automated design space exploration. The main contribution of this paper is a validation of the analytical model against simulation results. We believe that such cross-checking is required to establish the usefulness of such models, and also helps in identifying the appropriate design phase where such models can be used and where it is necessary to use detailed simulation to obtain meaningful results. Based on the results obtained, we proposed a design flow for network processors which relies on different classes of performance evaluation frameworks, and the two models studied here can be considered to be representatives of these classes.

We also believe that the two models considered here lie at two different extremes of a spectrum of possibilities. For evaluating different aspects of an architecture, different models might be more suitable. We envisage a suitable combination of analytical and simulation based frameworks not only across different abstraction levels of a design flow, but also within the same abstraction level, for evaluating network processor architectures. For example, it might be more suitable to use simulation to evaluate the cache/memory subsystem of an architecture, while it might be sufficient to use an analytical model to evaluate the on-chip communication architecture. The essential ingredients for such a *hybrid* framework in the context of network processors are already available. But, more work needs to be done in this area to devise ways for meaningfully combining them.

Acknowledgements

The work presented in this paper was partly supported by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation under grant number 5005-67322. The authors are also thankful to the two anonymous reviewers, whose suggestions greatly improved the contents of this paper.

References

[1] S.G. Abraham, B.R. Rau, R. Shreiber, Fast design space exploration through validity and quality filtering of subsystem designs, Technical Report HPL-2001-220, Compiler and Architecture Research Program, Hewlett Packard Laboratories, August 2000.

[2] A. Agarwal, Performance tradeoffs in multithreaded processors, IEEE Transactions on Parallel and Distributed Systems 3 (5) (1992) 525–539.

[3] Alpha Architecture Reference Manual, Digital Press, 1992.

[4] J.Y. Le Boudec, P. Thiran, Network Calculus—A Theory of Deterministic Queuing Systems for the Internet, Lecture Notes in Computer Science, vol. 2050, Springer, Berlin, 2001.

[5] D.C. Burger, T.M. Austin, The SimpleScalar tool set, Technical Report CS-TR-1997-1342, University of Wisconsin, Madison, 1997.

[6] P. Crowley, J.-L. Baer, A modeling framework for network processor systems, in: Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture, Cambridge, MA, February 2002. An enhanced version of this paper appears in the book "Network Processor Design: Issues and Practices, Volume 1", Morgan Kaufmann, San Mateo, CA, October 2002.

[7] P. Crowley, M. Fiuczynski, J.-L. Baer, On the performance of multithreaded architectures for network processors, Technical Report 2000-10-01, Department of Computer Science, University of Washington, 2000.

[8] R.L. Cruz, A calculus for network delay, IEEE Transaction on Information Theory 37 (1) (1991) 114–141.

[9] M.A. Franklin, T. Wolf, A network processor performance and design model with benchmark parameterization, in: Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture, Cambridge, MA, February 2002. An enhanced version of this paper appears in the book "Network Processor Design: Issues and Practices, Volume 1", Morgan Kaufmann, San Mateo, CA, October 2002.

[10] D.D. Gajski, F. Vahid, S. Narayan, J. Gong, Specification and Design of Embedded Systems, Prentice Hall, Englewood Cliffs, NJ, 1994.

[11] T. Givargis, F. Vahid, J. Henkel, System-level exploration for pareto-optimal configurations in parameterized system-on-a-chip, in: Proc. International Conference on Computer-Aided Design (ICCAD), San Jose, 2001, IEEE Transactions on Very Large Scale Integration Systems, in press.

[12] T. Grötker, S. Liao, G. Martin, S. Swan, System Design with SystemC, Kluwer Academic Publishers, Boston, 2002.

[13] Blue Logic technology, IBM. Available from <http://www.chips.ibm.com/bluelogic/>.

[14] Coreconnect bus architecture, IBM. Available from <http://www.chips.ibm.com/products/coreconnect/>.

[15] V. Kathail, S. Aditya, R. Schreiber, B.R. Rau, D.C. Cronquist, M. Sivaraman, PICO: Automatically designing custom computers, IEEE Computer 35 (9) (2002) 39–47.

[16] E. Kohler, R. Morris, B. Chen, J. Jannotti, M.F. Kaashoek, The Click modular router, ACM Transactions on Computer Systems 18 (3) (2000) 263–297.

[17] K. Lahiri, A. Raghunathan, S. Dey, System level performance analysis for designing on-chip communication architectures, IEEE Transactions on Computer Aided-Design of Integrated Circuits and Systems 20 (6) (2001) 768–783.

[18] National Laboratory for Applied Network Research (NLANR), Traces collected in June 2000. Available from <http://pma.nlanr.net/pma/>.

[19] A.K. Parekh, R.G. Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single-node case, IEEE/ACM Transactions on Networking 1 (3) (1993) 344–357.

[20] A.D. Pimentel, P. Lieverse, P. van der Wolf, L.O. Hertzberger, E.F. Deprettere, Exploring embedded-systems architectures with artemis, IEEE Computer 34 (11) (2001) 57–63.

[21] IBM PowerNP NPe405 Embedded Processors. Available from <http://www-3.ibm.com/chips/techlib/techlib.nsf/products/PowerNP_NPe405_Embedded_Processor>.

[22] B.R. Rau, M.S. Schlansker, Embedded computer architecture and automation, IEEE Computer 34 (4) (2001) 75–83.

[23] S. Shenker, J. Wroclawski, General characterization parameters for integrated service network elements, RFC 2215, IETF, September 1997.

[24] G. Snider, Spacewalker: Automated design space exploration for embedded computer systems, Technical Report HPL-2001-220, Compiler and Architecture Research Program, Hewlett Packard Laboratories, September 2001.

[25] SystemC homepage http://www.systemc.org.

[26] L. Thiele, S. Chakraborty, M. Gries, S. Künzli, Design space exploration of network processor architectures, in: Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture, Cambridge, MA, February 2002. An enhanced version of this paper appears in the book "Network Processor Design: Issues and Practices, Volume 1", Morgan Kaufmann, San Mateo, CA, October 2002.

[27] L. Thiele, S. Chakraborty, M. Gries, S. Künzli. A framework for evaluating design tradeoffs in packet processing architectures, in: Proc. 39th Design Automation Conference (DAC), New Orleans, USA, ACM Press, 2002.

[28] L. Thiele, S. Chakraborty, M. Gries, A. Maxiaguine, J. Greutert. Embedded software in network processors—models and algorithms, in: First Workshop on Embedded Software, Lecture Notes in Computer Science 2211, Lake Tahoe, CA, USA, Springer, Berlin, 2001, pp. 416–434.

[29] T. Wolf, Design and performance of a scalable high-performance programmable router, Ph.D. thesis, Department of Computer Science, Washington University, St. Louis, 2002.

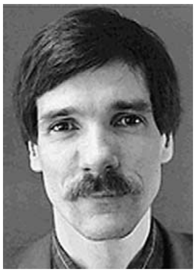
[30] T. Wolf, M. Franklin. CommBench—A telecommunications benchmark for network processors, in: Proc. IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS), Austin, TX, 2000, pp. 154–162.

[31] T. Wolf, M.A. Franklin, E.W. Spitznagel, Design tradeoffs for embedded network processors, Technical Report WUCS-00-24, Department of Computer Science, Washington University, St. Louis, 2000.

[32] F. Worm, A performance evaluation of memory organizations in the context of core based network processor designs, Master's thesis, Institut Eurécom, Sophia-Antipolis, France, This work was done at IBM Research Laboratory Zürich, 2001.

Samarjit Chakraborty received his Bachelor's degree in Computer Science and Engineering from Jadavpur University, Calcutta, in 1996, and his Master's degree in the same subject from the Indian Institute of Technology Kanpur, in 1998. Since then, he is a Ph.D. candidate, working with the Computer Engineering and Networks Laboratory at ETH Zurich. His research interests are in system-level design and analysis of real-time embedded systems, and in computer networks.

Simon Künzli received a Diplom-Ingenieur degree in Electrical Engineering from the Swiss Federal Institute of Technology (ETH) Zurich, Switzerland, in 2001. He is now a Ph.D. student at the Computer Engineering and Networks Laboratory of ETH Zurich. His main areas of interest are system-level modeling languages and performance models for embedded systems, especially in the context of network processors.



Lothar Thiele received his Diplom-Ingenieur and Dr.-Ing. degrees in Electrical Engineering from the Rhine Westphalia Institute of Technology in Aachen and the Technical University of Munich in 1981 and 1985 respectively. After completing his Habilitation thesis from the Institute of Network Theory and Circuit Design of the Technical University Munich, he joined the Information Systems Laboratory of Stanford University in 1987. In 1988, he took up the chair of microelectronics at the Faculty of Engineering, University of Saarland, Saarbrücken, Germany. He joined ETH Zurich, Switzerland, as a full Professor of Computer Engineering, in 1994. His research interests include models, methods and software tools for the design of embedded systems, and more recently, also bioinspired optimization techniques. In 1986 he received the "Dissertation Award" of the Technical University of Munich, in 1987, the "Outstanding Young Author Award" of the IEEE Circuits and Systems Society, in 1988, the Browder J. Thompson Memorial Award of the IEEE, and in 2000, the "IBM Faculty Partnership Award".

Andreas Herkersdorf received a Diplom-Ingenieur degree in Electrical Engineering from the Technical University of Munich, Germany, in 1987, and a Ph.D. in Electrical Engineering from the Swiss Federal Institute of Technology (ETH) in Zurich, Switzerland, in 1991. Since 1988, he has been with the IBM Zurich Research Laboratory, Switzerland, where he currently manages the Network Processor Hardware group. His areas of interest are high-speed communication networks and systems, and VLSI design methodologies.

Patricia Sagmeister received a Diplom-Information, degree in Computer Science from the University of Passau, Germany, in 1993 and her Ph.D. in Computer Science from the University of Stuttgart, Germany, in 2000. She joined IBM in 1999 where she is currently a member of the Network Processor Hardware group. Her areas of interest are system-level design, architectural performance evaluation and hardware/software codesign.