PERFORMANCE EVALUATION OF NETWORK PROCESSOR'S BUS ARBITER POLICIES USING WORST CASE DETERMINISTIC ANALYTICAL METHODOLOGIES.

Frederico de Faria, Marius Strum, Wang Jiang Chau

Microelectronics Laboratory, Polytechnic School of University of São Paulo

ffaria@lme.usp.br, strum@lme.usp.br, jcwang@lme.usp.br

ABSTRACT

Abstract: In this paper an analytical modeling technique based on deterministic worst case scenarios using Network Calculus is applied, using concepts like convolution and deconvolution to the study of time-based performance analysis of network processors. We present conceptual models that mimic network processors running IPv4 forwarding applications, with emphasis on detailed study of bus and bus arbiter policy latencies. The growing number of electronic systems with communication capabilities has raised the necessity of exploration of huge number of different possible micro-architectures that include many communication and other different components (like cryptography, audio and video decoders) in a manner that makes it possible the elimination of undesired architectures, determination of high level system parameters (buffer utilization, end-to-end latencies, percentage of use), localization of bottlenecks and answer "what-if" questions, all of them in a timely fashion. The problems shown above can be addressed by using analytical modeling techniques.

SECTION I - INTRODUCTION

The electronic systems that incorporate communication capabilities are becoming more and more common in the marketplace, as it is the widespread use of wireless and networked devices. Network processors are an intrinsic component in this context since they are present in devices ranging from access gateways to core devices, being fundamental pieces that connect all networked devices. The constant advances in VLSI processing technology have allowed greater levels of chip integration, which has led complex systems as network processors to be built by inserting other modules or intellectual property cores on a single die, being known as system-on-chip (SOC) implementation.

SoC designs have been growing in terms of size (number of blocks, interfaces, standards, transistor count), design time and complexity. As such, tight project cycles and time to market pressures have challenged engineers to deliver designs on shorter schedules. Phases like specification, architecture definition and implementation have become more complex and designers have been pressured to shrink them in aggressive time-frames. Therefore, design time and complexity have been major concerns for designers. In order to make the design cycle more efficient, the integration of reusable blocks (both in-house and licensed cores) has turned to be a must, since the development of current electronic systems is practically infeasible from the ground. One of the most common approaches adopted by the design industry is the platform based design where the set of reusable components and their communication interfaces are standardized, and the system integration designer explore the trade-offs of different architectural configurations. Most current complex SoCs include an important architectural component, generic programmable microprocessors running a software application.

Performance analysis of systems composed of reusable blocks, made on very early development stages is highly desirable. A performance analysis performed at high abstraction level (like the analytical domain) can contribute to a significant reduction of design space exploration alternatives for feasible architectures. Those that don't comply with specifications are eliminated and, more important, design time is reduced since several undesired iterations between high level (behavioral, for instance) and low level (RTL) implementation cycles are discarded. In this phase, it is possible to offer the design team a sum of answers to "what-if" questions and a broad analysis of tradeoff architectures. In [8], it has been shown that even at later stages of project development, the refinements of analytical model in terms of accuracy is indeed desired for the next generation of product. Platform-based designs are then one viable design methodology to reach such aggressive goals.

With modeling and execution times shorter than other methods like instruction level and cycle-accurate simulation, hundreds of different architectures can be explored in a matter of minutes, allowing the former methods to investigate architectures that really conform to specifications [5], [6]. Many design exploration tools like EXPO [4] have been used to prove that analytical modeling techniques allow the exploration of hundreds of different architectures, submitted to constraints, in a matter of minutes. The models and techniques applied in this papers derive maximum and minimum values to system parameters like buffer utilization, end-to-end latencies, percentage of use , both component and systemwide. The results obtained in Section V are derived from the use of an in-house tool that implements the theoretical concepts in this paper.

Related works: In [3], the technique results are compared against results obtained from a tool provided by a network processor supplier, for an application of IPv4 forwarding, like fundamental routing tasks. It is shown that high levels of fidelity for the model can be achieved, even though some simplifications in the modeling of bus arbiter are applied.

In [2] the present model technique is shown, and a case study, applied to network processors, are evaluated. Again, some enhancement to the model in terms of bus and bus arbiter detailing can move the technique one step further.

This technique has been applied to the analytical study of NP and its components like generic programmable processors, dedicated microengines, ciphers, checksum modules among others., and hence is flexible enough to support more detailed approaches, oriented, in particular, to explore the influence of bus arbiter policies and associated latencies.

Many analytical techniques like Stochastic Automata networks, Calculus of Communication Systems, Communicating Sequential Processes and Algebra of Communicating Processes have being used to model communication systems, but the Network Calculus can bridge together characteristics presented in communication systems and also present in computing systems.

In [9], is presented a analytical performance technique that compares the results with a SystemC based model. With a security processor in detail, this technique is based on probabilistic-statistical analysis. Although this last approach have being used by scientific community and industrial tools to perform analytical studies for many years now, the growing complexity of systems and exponential growth in number of "nodes" is rapidly making it an not-so-feasible alternative due to increasing computational demands and at the same time that other techniques bring similar results in a matter of seconds requiring much less computational resources.

The remainder of this paper is organized as follows. In Section II we approach theoretical concepts that form the basis of the present technique. We present in Section III details of an application of this theory to an electronic system like NP. Next, in Section IV different approaches of modeling an NP are shown. The results of this comparison are presented and discussed in Section V and finally in Section VI the conclusions of this paper are presented.

SECTION II - THEORY

ln [1] arrival and service curves are defined in the context of integrated services networks.

(ARRIVAL CURVE). Given a wide-sense increasing function α defined for $t \geq 0$, we say that a flow of network packets R is constrained by α if and only if for all $s \leq t$:

$R(t) - R(s) \leq \alpha(t - s)$. In this case, R has α as an arrival curve, or also that R is α-smooth.

(SERVICE CURVE). Consider a system S and a flow through S with input and output function R and $R*$. We say that S offers to the flow a service curve β if and only if β is wide sense increasing, $\beta(0) = 0$ and $R* \geq R \otimes \beta$.

Many algebraic operations are applied over these two curves, like convolution and deconvolution [1]. When applied over a certain resource, other curves are derived, like final arrival and final service curves..

DEFINITION - MIN-PLUS CONVOLUTION - Let f and g be two functions or sequences of F. The min-plus convolution of f and g is the function $(f \otimes g)(t) = \inf_{0 \leq s \leq t} \{f(t - s) + g(s)\}$. (If $t < 0$, $(f \otimes g)(t) = 0$).

DEFINITION - MIN-PLUS DECONVOLUTION - Let f and g be two functions or sequences of F. The min-plus deconvolution of f by g is the function $(f \oslash g)(t) = \sup_{u \geq 0} \{f(t + u) - g(u)\}$.

In the context of network processors , arrival curves can be interpreted as the throughput which data (network packets, bytes) arrive in a port of a micro-architecture resource in intervals of time. They have many slopes, ranging from the nominal rate of a certain channel to lower values, depending on network characteristics. Since network traffic characteristics like packet size, type of flows and throughput can vary a lot, arrival curves are represented with different slopes. representing the sustained transfer rate on intervals of time. Service curves represent the capacity of a resource to process data in a period of time. Different categories of resource have their own types of service curves, either peak rate (slope) , burst-delay (step), rate-latency (slope shifted right), affine functions (slope shifted left), among others. Arrival and service curves can be represented as lower and upper arrival curves, representing the minimum and maximum flow rates and computing capacity. In the context of the present work, physically, the curve resulting from the convolution of two curves is interpreted as the workload left from a certain resource.

For a refined derivation of following equations, please refer to [1].

Graphically, upper and lower curves are represented as three parameters forming two segments [Figure 1 bellow].

- $\alpha_l(t) = L[q_l^{\alpha}, r_l^{\alpha}, s_l^{\alpha}] = \max_{0 \leq t \leq \Delta} \{r_l \times t, q_l^{\alpha} + s_l \times t, 0\}$;

$\alpha_u(t) = U[q_u^{\alpha}, r_u^{\alpha}, s_u^{\alpha}] = \min_{0 \leq t \leq \Delta} \{r_u \times t, q_u^{\alpha} + s_u \times t\}$

(1)

- $\beta_l(t) = L[q_l^{\beta}, r_l^{\beta}, s_l^{\beta}] = \max_{0 \leq t \leq \Delta} \{r_l \times t, q_l^{\beta} + s_l \times t, 0\}$;

$\beta_u(t) = U[q_u^{\beta}, r_u^{\beta}, s_u^{\beta}] = \min_{0 \leq t \leq \Delta} \{r_u \times t, q_u^{\beta} + s_u \times t\}$

(2)

Where *q* represents the largest packet size for the flow, r and s represent the nominal transfer rate and the sustained transfer rate, respectively. Network traces obtained from either synthetical or real world traffic flows can be utilized to derive packet sizes and slopes other than the nominal.

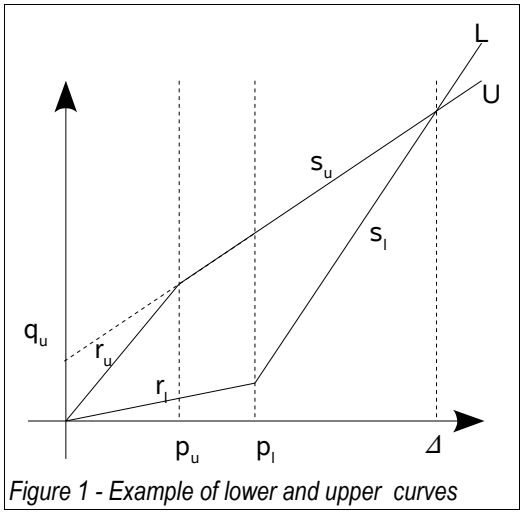


Figure 1 - Example of lower and upper curves

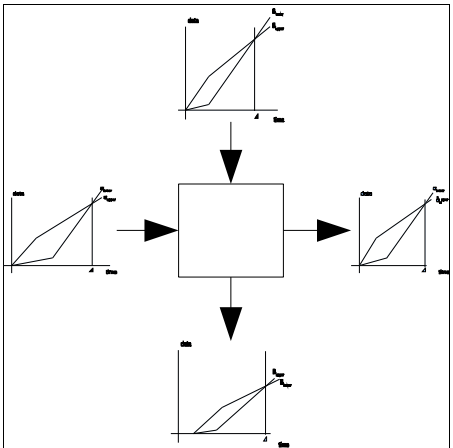


Figure 2 - Representation of a vertice in Scheduling Network Graph and associated arrival and service curves

For an analysis to take place, some stability criteria must be satisfied.

Let $\aleph$ be a curve, upper curves must be concave and lower curves must be convex, and

(1) $r_{lower} \leq r_{upper}$;

(2) $q_{upper} \geq 0$, $r_{upper} > s_{upper} \geq 0$; $r_{upper} = s_{upper} \leftrightarrow q_{upper} = 0$;

(3) $q_{lower} \leq 0$, $s_{lower} > r_{lower} \geq 0$; $r_{lower} = s_{lower} \leftrightarrow q_{lower} = 0$;

In case condition (1) is violated, there is not an intersection Δ, meaning that there is no point p , $p \neq 0$, in upper curve that satisfies $\aleph_U(p) > \aleph_L(p)$.

For a given resource, there are four other curves that are obtained from the original arrival and service curves (upper and lower) and represent the workload processed in that resource (final arrival curves) and the remaining resource capacity after processing the original workload (final service curves).

The relationship between arrival and service curves are defined as the following equations (the notation ' indicates a final curve) [1]

- $\alpha'_l(\Delta) = \inf_{0 \leq t \leq \Delta} \{\alpha_l(t) + \beta_l(\Delta - t)\} = (\beta_l \otimes \alpha_l)(t)$ (3)
- $\alpha'_u(\Delta) = \inf_{0 \leq t \leq \Delta} \{ \sup_{v \geq 0} \{\alpha_u(t+v) - \beta_l(v)\} + \beta_u(\Delta - t), \beta_u(\Delta)\} = \inf_{0 \leq t \leq \Delta} \{(\alpha_u \oslash \beta_l) \otimes \beta_u, \beta_u(\Delta)\}$ (4)
- $\beta'_l(\Delta) = \sup_{0 \leq t \leq \Delta} \{\beta_l(t) - \alpha_u(t)\}$ (5)
- $\beta'_u(\Delta) = \sup_{0 \leq t \leq \Delta} \{\beta_u(t) - \alpha_l(t)\}$ (6)

Notice that for convolution, critical circumstances happen before Δ, (Figure 1). The processing capacity is smaller than the incoming data flow, hence forcing the existence of buffer B and latency δ. After Δ, the processing capacity is greater than the incoming workloads, and therefore B and δ are not critical.

Also, the following values are derived from the curves.

- delay $\delta \leq \sup_{t \geq 0} \{\inf\{\tau \geq 0 : \alpha_u(t) \leq \beta_l(t+\tau)\}\}$ (7)

Which is the largest horizontal distance between the two curves, representing the maximum end-to-end flow latency.

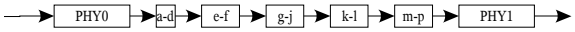
- backlog $B \leq \sup_{t \geq 0} \{\alpha_u(t) - \beta_l(t)\}$ (8)

Which is the largest vertical distance between the two curves, representing the maximum buffer required in order to process an incoming flow.

- utilization $\mu \leq \lim_{\Delta \to \infty} [\beta_u(\Delta) - \beta'_l(\Delta)] / \beta_u(\Delta)$ (9)

Is the maximum utilization in function of time for a given resource.

A first step for constructing the model is the generation of a task graph that comprises all task that are executed in the system to be modeled. Task graphs have vertices containing tasks performed by resources present in the system.



•Data Link Layer 2

- .a receive set of frames from PHY and checksums
- .b removes preamble and sequences frames
- .c send ack to sender
- d. stores assembled packet in output FIFO
- e . Transmit buffer descriptor and payload are written into memory

•Network Layer 3 (Internet Layer)

- f. processor reads buffer descriptor and ip header from memory
- g.Check IP header
- •.h Checks target IP
- •.i IP destination
- j decrements TTL
- k IP header and receive buffer descriptor are written into memory
- •.l IP packet (header e payload) and receive buffer descriptor are moved from memory to MAC

•Data Link Layer 2

- .m receives ack of frame from RCV
- .n generate frames frompacket
- .o inserts preamble
- .p calculates checksums from frames
- . send frames to PHY

Figure 3 - Task graph and associated tasks

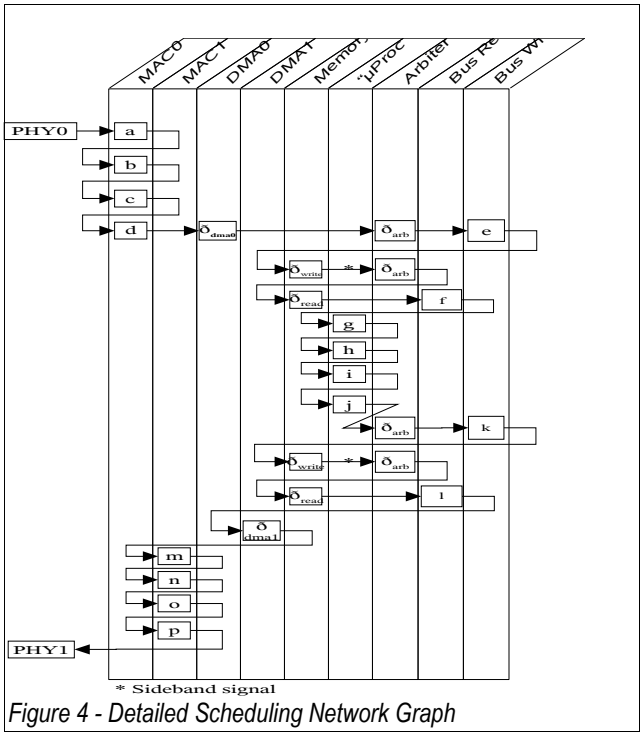


Figure 4 - Detailed Scheduling Network Graph

SECTION III – GENERAL SCHEDULING NETWORK GRAPH.

In a SNG, the flow of the data through a system is modeled as a sequence of tasks, represented by vertices, as it is shown in Figure 4. On a Scheduling Network graph (SNG), vertices are disposed on columns and lines, having specific purposes. Columns represent the different resources allocated at the micro-architecture. These resources can be divided into four categories. Processing modules, communication structures, communication controllers and storage. Examples of processing modules are generic programmable microprocessors and dedicated microengines. Communication structures are multiplexers and wires. Communication controllers are modules that handle interactions between communication structures and other types of resources. They are bus arbiters, decoders, DMA controllers, memory controller, bridges. Storage is dedicated memory arrays. In the context of SNG this distinction is important since each of them will have different conceptual service curves. Namely, modules like processing modules or communication structures when pipelined, keep state information about different "jobs" in an interval of time (rate-latency functions). Other modules like communication controllers and storage are "stateless", hence their service curves are burst-delay functions (step functions). Basically, the latter resources add delay to the data-flow without processing them. The determination of those latencies have different approaches and difficulties. While memory and bus transport latencies are closely related to the physical implementation themselves, with known maximum intervals, arbiter latencies vary significantly, even for the same master, depending on the profile of data, resources in the micro-architecture and contention (bus arbiter latencies are stochastic). For this reason, studying the behavior of these latencies is of great importance and in the present work, different latencies are modeled to analyze their impact in terms of delays, component utilization and buffer requirements.

The order how resources are displayed is important and in Figure 4 they are disposed in a way that easy the user comprehension of both micro-architecture and its dynamics.

Lines in SNG are not bound to time but to the order which tasks are executed. Tasks (inherited from task graph) executed by a specific resource must be disposed in different lines. Although different tasks executed by different resources might be on different lines, they should preferably belong to the same. For a matter of reference, consecutive vertices are disposed on the same lineup to the right-most resource, with the next line vertices on the left-most resource for that given task.

Each vertice in Figure 4 represents a specific task *t* being executed by the resource *s*. A task *t* has a weight *w(s,t)* associated to the resource and can be measured in terms of cycles, for example. In case the resource is a bus, *w(s,t)* represents the number of cycles the transfer will take. In the case of a processor, it measures the effort in cycles realized by the processor to execute that task. Each vertice has associated parameters like arrival curves (4), service curves (4), resource, task and weight, as graph related neighbors (one successor, one predecessor, maximum one inferior, maximum one superior). The graph does not support conditional deviation, since it is a worst case deterministic modeling and the worst case condition must be considered in each vertice. Vertices sharing resources with rate-latency service curves inherit the remaining service curves from the immediately superior vertice. Other vertices sharing resources with burst-delay service curves doesn't inherit the remaining service curves from the immediately superior vertice since the remaining service curve is equal to the original service curve. Successor vertices have the predecessors final arrival curves as incoming arrival curves.

For each resource in a given SNG, the most superior vertice must be bound to a initial lower service curve and initial upper service curve. The very first vertice of this graph must be bound to a initial lower arrival curve and initial upper arrival curve, as explained earlier. All other arrival and service curves, buffers, latencies and utilization rates are derived using the algebraic methods mentioned in the previous section. After computation of the last vertice of resource, delays and resource utilizations in the system can be extracted, using equations (7)-(9). If a more detailed study is needed, it is possible to evaluate latency, buffer and utilization for each vertice in the SNG, exposing the requirements of each task. Albeit more computing time is required, a deeper understanding of micro-architecture dynamics is achieved.

The instant information about each packet flow is not concerned in arrival curves, but the lower and upper rates, maximum packet size and maximum burst size. Hence, the whole flow is calculated on each vertice for that given task, thus making this approach extremely fast when compared to other simulation methods, at the same time, providing high levels of accuracy [4].

For a given flow, only one SNG need to be followed to obtain the values formulated in (7)-(9), making this a non-iterative algorithm. Different flows lead to different arrival curves, and since a network processor usually supports many different flows, each one with its own task graph and SNG, hundreds of architecture explorations can be performed in a matter of minutes.

The SNG represent information from micro-architecture shown in Figure 6 and from task graph shown in Figure 3 in a much more detailed way, since the relations between different resources are exploited.

SECTION IV – APPLIED SNG

In this section, more detailed concepts related to bus and bus arbiter modeling are presented. It is known that network processors have a very specific set of tasks, related to the network protocols they are implemented to support. Network processors can be represented in a SNG as different flows of packets, each of them with different task order, in a preemptive fashion. In [1] it is shown that it is algebraically advantageous gather different vertices pertaining to similar resources categories. As a result it is shown in Figure 5 the equivalent SNG. Another work [3] has adopted the latency of a priority based arbiter policy as a constant, which is a simplification that can reduce the fidelity of results, since arbitration policy may account for a significant part of end-to-end latency, as shown in Section V.

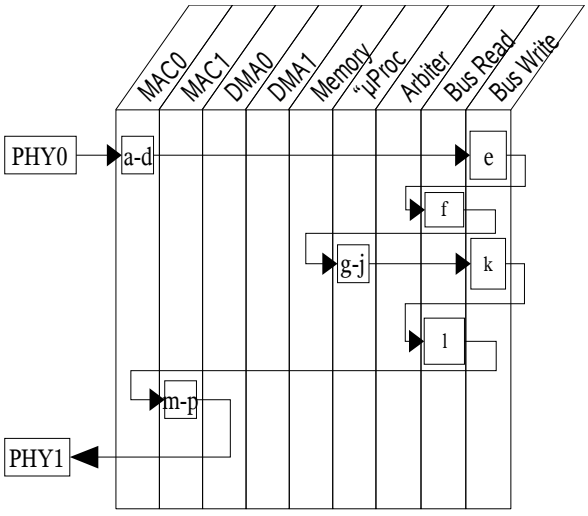


Figure 5 - Condensed SNG optimized to reduce computing demand.

In the present work, three bus arbiter policies are modeled. In the first, named *Two-Level* [7], because it is TDMA and round robin, a variable time slot with sequenced masters (without priority) arbiter policy is modeled. Each master has a specific time slot interval $T_{master\text{-}i}$ granted to perform bus operations and will have access granted again only after all other masters intervals have been spent, in a sequenced fashion. This is a non-priority based arbiter policy since all master have identical priorities. The arbiter latency δ_{arb} is constant and equal to the sum of each slot interval..

The second model is a *hybrid* bus arbiter policy since it grants the default master (the one with highest priority) a maximum latency , after when is has a bus request granted. In the worst case its latency will be the largest bus operation. This is because there are no split bus operations. A two level bus arbiter policy is enforced for the other masters.

The third model is the ideal case , and a *zero* bus arbiter latency is enforced for all bus operations.

Memory Write

The modeling of memory write is detailed in Figure 4 by vertices [d-δ_{write}]. A certain processing module (MAC, for instance) has data to write to memory and, through the DMA, awaits the bus arbiter to grant access to the memory, which is the slave. After an interval of time δ_{arb1}, the access is granted to the DMA and words flow through bus write wires all the way into the memory array, where they are written. In a similar fashion, this process takes place when the processor needs to write to the memory.

Here it is relevant to explain the meaning of each specific vertice in SNG. This process is dependent upon a sideband signal between the DMA/MAC and the processor. MAC has an IP packet and a transmit buffer descriptor (TxBD) in its output buffer ready to be sent to memory via DMA. In the present case, DMA is implemented in RTL inside the MAC block, but the representation in SNG would the same if they were two different blocks built apart. DMA has a latency δ_{dma} since different blocks and sub-blocks (like a MAC RX and MAC TX) are requesting read (sideband signals) and write to the memory. These solicitations will be granted due to a policy implemented in DMA. Notice, however, that δ_{dma} comprises not only the time to apply the policy, but also the time it takes to grant access after a given request. That depends on many factors outside the DMA itself, like resource hazards and different arbiter policies. Here, there is only one MAC for each DMA, bringing a constant value for δ_{dma}. The value of δ_{arb} is dependent on bus arbiter policy.

After the DMA request for bus access has been granted, a maximum latency δ_{arb}, the bus is granted and the data flow

between the ports. According to the arbiter policy, latency is constant (worst case) and is independent of the many factors outside the arbiter block itself, for example, the traffic between different blocks, IP packet sizes and resource hazards, to name a few. The time spent by the arbiter/decoder block to process the policy and grant access is also considered in δ_{arb}. Since the slave must be free of performing operations for other masters, the time it takes from the write grant until it becomes idle again is $T_{master\text{-}i}$. The total amount of information delivered is represented by vertice "e", which contains a rate-latency function of latency that equals to the time required by bus to effectively transfer data, and a slope that is a function of bus width and frequency.

The fact that those latencies are not dependent only on the block itself, but more intense on outside factors, makes its analysis extremely important. The present SNG is an viable alternative to study the influence of this latencies, considering them constant between each other but with possibly different time slots intervals for each master.

The third latency represented in this operation, δ_{write}, is the number of cycles it takes for memory to move data from the source port to arrays, which is a small cycle count, most likely two, since bus is pipelined, this latency is accounted only once per operation. After data is written in memory a sideband signal travels to the processor adverting it that some data is ready to be read.

Memory Read

The modeling of memory read is detailed in Figure 3, from sideband signal to vertice g.

After the processor has updated the IPv4 header and the transmit buffer descriptor (TxBD), or right after the payload and receive buffer descriptor (RxBD) have been written into the memory, a sideband signal is sent, either from the processor to MAC (former case) or from MAC to processor (latter case) , which starts a read operation against the memory.

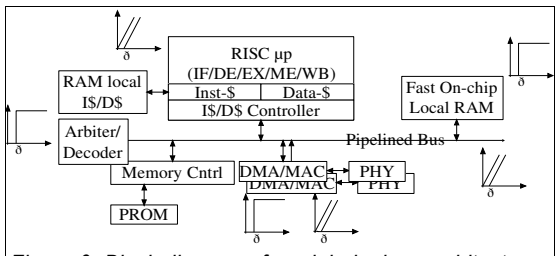


Figure 6- Block diagram of modeled micro-architecture and associated service curves.

First, the processor awaits the arbiter grant for a memory read operation. After maximum δ_{arb} cycles, access is granted and it takes δ_{read} cycles to make the required data available on the memory port and another "f" cycles to transfer the amount of data from memory port to processor cache controller port through the bus wires. Processor then applies the modification to the IPv4 packet header and receive buffer descriptor (RxBD) and a new memory write operation is performed.

Modeling of components :

Network processors have limited scope in terms of applications running on its generic programmable processor or dedicated microengines. In the present case, an IPv4 forwarding application described in Figure 6 is running, providing routing functionality.

Fast on-chip local RAM is modeled as an infinite address space. After the SNG is analyzed, the total buffer actually used is known. It has different read and write latencies and accepts read/write calls from DMA and processor, stores IPv4 packet payloads and buffer descriptors.

Bus Arbiter/decoder is modeled as latency only. There are four arbiter calls in SNG: DMA requests memory write, processor requests memory read, processor requests memory write and DMA requests memory read. All operations have latency δ_{arb} (dependent on the bus arbiter policy enforced). When the time slot expires, the next master must have access granted to the memory. The optimization of each $T_{master\text{-}i}$ can be achieved with the help of IPv4 packet sizes collected from real world flows. While DMAs read and writes the total IPv4 packet size and buffer descriptors, processors read and write only the IPv4 header and buffer descriptor, which represent only a fraction of DMA transfers.

Bus is pipelined and has a initial setup latency and a transfer rate of bandwidth multiplied by bus frequency.

MAC and DMA are modeled as independent modules, being processing module and communication controller module, respectively and hence have a worst case processing cycles and throughput rate.

SECTION V – MODEL DETAILS, RESULTS AND ANALYSIS

The modeled micro-architecture is equivalent to that shown in Figure 6 above. Each MAC module is able to support 16 PHY circuits. There are two MAC modules compatible with IEEE 802.3 and 802.3u (CSMA/CD), each one with one DMA controller integrated. All flows arrive on MAC0 and departs from MAC1. During experiments, MAC frequencies vary appropriately conforming PHY frequency. Bus width assumes 32bits and 64bits, with a constant 80MHz clock. Processor frequency is 200MHz. Incoming flow rates assume 50Mbps, 100Mbps and 200Mbps, with packet sizes ranging from 256bytes to 1400 bytes. Buffer descriptors (both receive and transmit) have constant size equal to 8 bytes, and IPv4 packet headers have 20 bytes of size. For each task, an appropriate number of cycles for the resource is adopted, and latency is proportional to resource frequency. With these parameters, lower and upper arrival and service curves are obtained.

In the following two figures, end-to-end latencies and memory requirements for the system described above are shown. Three bus arbiter policies, three different packet sizes and two different bus widths are considered. In these two analysis, the flow and MAC capacity are raised simultaneously. Although these results are not compared against any simulation, the results are equivalent to those presented in [3] and [2]

Given an increase in link transfer rate, latency tends to decrease for the same packet size, although for the same link, it raises with packet size, as seen on Figure 7 and 8 bellow.

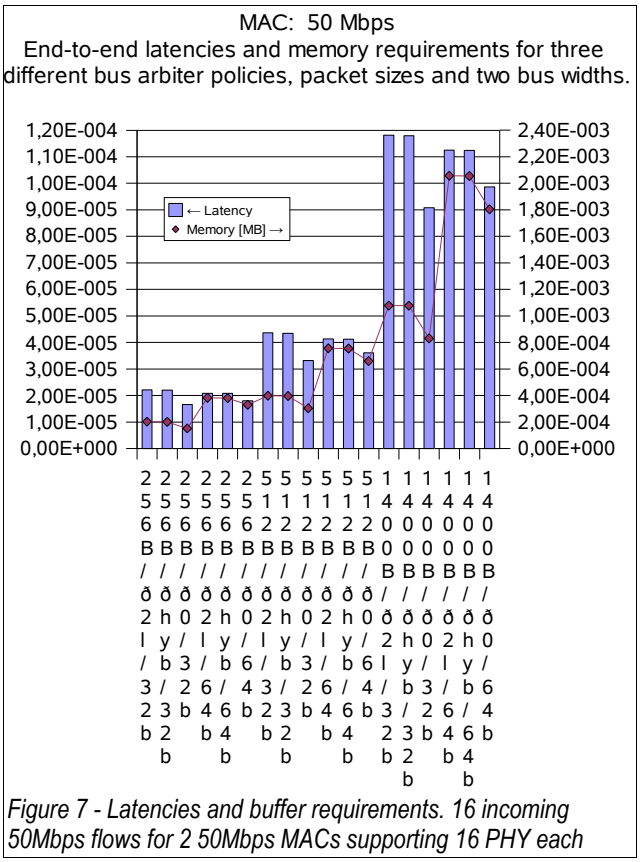


Figure 7 - Latencies and buffer requirements. 16 incoming 50Mbps flows for 2 50Mbps MACs supporting 16 PHY each

It can be observed that for the 50Mbps case, for the same packet size, latencies are very close to each other, but the memory requirements almost double when the bus width doubles. This is because all other system components had their parameters kept constant, and each component will have to store more and more data in each of their internal buffers (internal accumulation).

In all cases, when changing bus width from 32b to 64b, for bus with two level and hybrid arbiter policy, a reduction in end-to-end latency is perceived, but for ideal bus arbiter policy, there is a sensible raise. This shows a relationship between bus arbiter policy and bus width. This happens because, in the ideal case there is no waiting for bus grants, it is more sensitive to the contention on each component due to the fact that they don have their capacity augmented. Memory requirements also change with bus arbiter policy, since for the same packet size, a smaller latency causes less contention on each component individually.

The larger the workload on the system, the larger is the end-to-end latency resulting from the raise in the overall contention. This can be noticed by the greater difference among ideal arbiter latency and the other two, for the same packet size and bus width, when comparing the results in Figure 7 and 8. This shows the importance of arbiter policy definition for systems under heavy workload. The proportion between ideal arbiter latency and the other two almost doubled with the workload raise. Consequently, the bus arbiter policy can represent a bottleneck on the system.

The overall effects are similar for different packet sizes, what shows that the system is reasonably balanced in respect to the internal transfers, since in this kind of application, outside the MAC module, only buffer descriptors and IPv4 headers are actually processed.

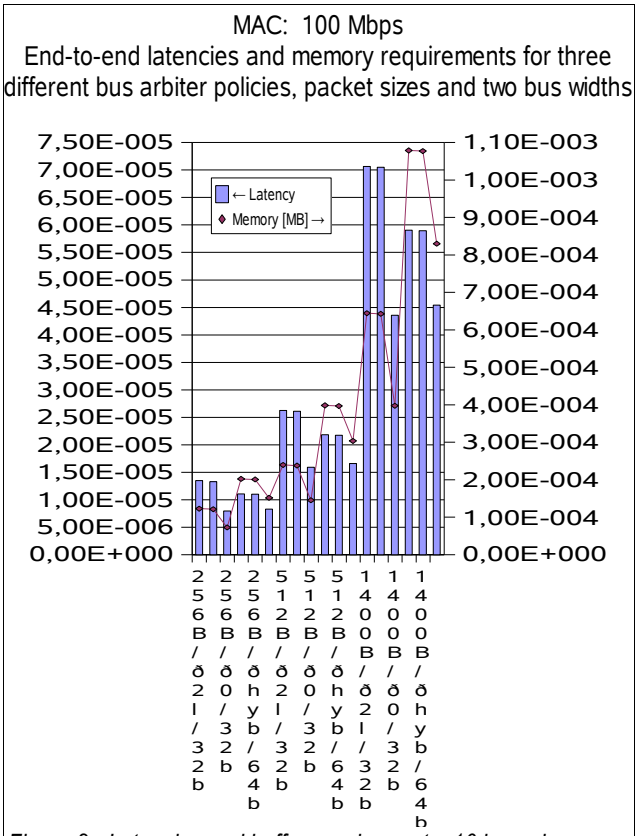


Figure 8 - Latencies and buffer requirements. 16 incoming 100Mbps flows for 2 100Mbps MACs, supporting 16 PHY each.

In Figure 9 end-to-end latency and memory requirements are estimated for a system under three different workloads. In this case, the MAC module supports 100Mbps, but that is under flows

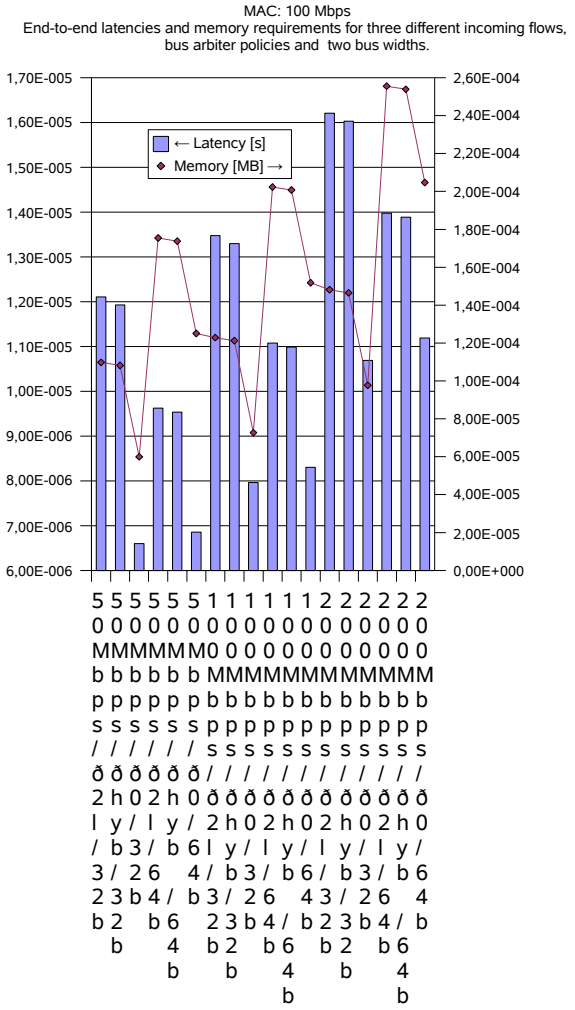


Figure 9 - Latencies and buffer requirements for 100Mbps MAC. Packet size is 256bytes, and flows are 50Mbps, 100Mbps, 200Mbps.

of 50Mbps, 100Mbps and 200Mbps with a constant packet size of 256 bytes. A distinctive behavior is perceived in these results since latency tends increase due to greater component contention.

Similar effects are repeated in this model, like the raise in arbiter latency for ideal policy when bus width changed from 32b to 64b.

Under heavier workload, the benefit of 64b bus width is minimized due to the negative impact of bus arbiter policy (two levels and hybrid). It is noticeable that for ideal arbiter policy, the latency proportion between 32b and 64b for the three workloads is practically constant, however, this proportion is reduced for the other two policies. This effect happens to the memory in a similar fashion: albeit the ideal policy requires more memory, the proportion is constant; on the other side, the excess of workload requires more memory for the other two latency policies.

SECTION VI – CONCLUSION

We have presented an enhancement in an analytical modeling technique that is capable of derive consistent results in respect of a network processor dynamics, allowing the analysis of many different micro-architectures, being the execution time of computation in the order of a few seconds on a Pentium IV machine.

Beyond that, we have shown that the detailed modeling of bus components and their relations is of great importance and possible through the use of this analytical technique, as shown in the results.

As future works, some enhancements to the model can be achieved, like the raise in the number of curve segments as well as the simulation of micro-architectures in more detailed levels, in order to confront the results form analytical models against simulation results.

REFERENCES

[1] J.-Y. Le Boudec and P. Thiran. *Network Calculus - A Theory of Deterministic Queuing Systems for the Internet. Lecture Notes in Computer Science 2050*, Springer Verlag, 2001.

[2] S. Chakraborty, S. KÂ¨unzli, L. Thiele, A. Herkersdorf, and P. Sagmeister. “Performance evaluation of network processor architectures: Combining simulation with analytical estimation.” Computer Networks, Elsevier, 41(5):641– 665, April 2003.

[3] M. Gries, C. Kulkarni, C. Sauer, and K. Keutzer. “Comparing analytical modeling with simulation for network processors: A case study.” In Proc. of the Designer’s Forum at the 6th Design, Automation and Test in Europe (DATE), Munich, Germany, March 2003.

[4] L. Thiele, S. Chakraborty, M. Gries, and S. KÂ¨unzli. “A framework for evaluating design tradeoffs in packet processing architectures.” In Proc. 39th Design Automation Conference (DAC), pages 880–885, New Orleans, LA, June 2002. ACM Press.

[5] L. Thiele, S. Chakraborty, M. Gries, and S. KÂ¨unzli. “Design space exploration of network processor architectures.” In Crowley et al. [48], chapter 4, pages 55–90. A preliminary version of this paper appeared in the Proc. 1st Workshop on Network Processors, held in conjunction with the 8th International Symposium on High-Performance Computer Architecture, Cambridge, Massachusetts, 2002.

[6] T. Wolf, M.A. Franklin, and E.W. Spitznagel. “Design tradeoffs for embedded network processors.” Technical Report WUCS-00- 4, Department of Computer Science, Washington University in St. Louis, 2000.

[7] Conti, M., Caldari, M., Vece, G. B., Orcioni, S., and Turchetti, C. 2004. “Performance analysis of different arbitration

algorithms of the AMBA AHB bus.” In Proceedings of the 41st Annual Conference on Design Automation (San Diego, CA, USA, June 07 - 11, 2004). DAC '04. ACM Press, New York, NY, 618-621.

[8] M. Sakamoto, A. Katsuno, A. Inoue, T. Asakawa, H. Ueno, K. Morita, Y. Kimura ;*"Microarchitecture and performance analysis of a SPARC-V9 microprocessor for enterprise server systems"* The Ninth International Symposium on High-Performance Computer Architecture, HPCA-9; 8-12 Fev. 2003 página(s):141 - 152 .

[9] Yung Chia Lin; Chung Wen Huang; Jenq Kuen Lee; “System-level design space exploration for security processor prototyping in analytical approaches” Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific Volume 1, 18-21 Jan. 2005 Page(s):376 - 380 Vol. 1