Comparing Analytical Modeling with Simulation for Network Processors: A Case Study

Matthias Gries[1], Chidamber Kulkarni[1], Christian Sauer[2], Kurt Keutzer[1]
[1] University of California, Berkeley
[2] Infineon Technologies, Corporate Research, Munich
{gries, kulkarni, sauer, keutzer}@eecs.berkeley.edu

Abstract

Programming network processors remains an art due to the variety of different network processor architectures and due to little support to reason and explore implementations on such architectures. We present a case study of mapping an IPv4 forwarding switch application on the Intel IXP1200 network processor and we compare this implementation with an analytical model of both the application and architecture used to evaluate different design alternatives. Our results not only show that we are able to model the IXP1200 and our application within 15% of the accuracy compared to that of IXP1200 simulation, but also find closely matching trends for different workloads. This shows the clear potential of such analytical techniques for design space exploration.

1. Introduction

Contemporary network processors (NPs) exhibit a wide range of architectures for performing similar tasks: from simple RISC cores with dedicated peripherals, in pipelined and/or parallel organization, to heterogeneous multiprocessors, based on complex multi-threaded cores with customized instruction sets. Evaluating such disparate architectures via extensive benchmarking is time consuming and tedious, due to absence of a proper programming model.

Programming such concurrent systems remains an art. The programmer not only is required to partition and balance the load of the application manually, it is also necessary to implement each task, often in assembly, in order to get reliable performance estimation. Hence, a robust application mapping strategy for such architectures requires a balance between thread partitioning, scheduling, memory accesses and I/O. With the current state-of-the-art tools this task becomes time consuming and error prone, due to trial-and-error method employed by system implementers based on simulation runs. Therefore, methods to address the above issues need to be investigated.

For the next generation of network processor based system implementations, we strongly believe that a considerable emphasis will be put on performance per cost (for example, power consumption) aspects and on support of appropriate programming models. Therefore, it is essential to investigate methods that help in identifying limitations and bottlenecks in system implementation without going all the way down to complete implementations, as is the current practice. Consequently, high-level design space exploration tools are required that support a wide range of heterogeneous architectures and enables precise reasoning about different implementation styles and their performances. Data generated by such tools while evaluating single design points should ideally be indicative of final achievable quality of results.

Related works focus on three different approaches namely simulation, trace analysis and analytical models. Simulation and trace analysis are somewhat similar, since for generating a trace one needs an (cycle) accurate simulator. Simulation based approaches, such as [1], require application specification in a high-level language or assembly, compiled to the particular architecture. In addition, different workloads need to be specified or generated for both the simulation and trace based analysis. Trace based analysis (like [2]) is limited by the fact that they capture details of a single execution for the particular workload. Thus for event driven systems with varying workloads, a large amount of traces need to be generated for any useful analysis and hence the gap between simulation and trace is no longer that large.

In contrast to simulation and trace analysis, analytical models promise a fast evaluation that allows for a larger design space to be explored. In the packet processing domain, [3] presents an approach to explore different cache configurations based on general purpose computing elements. Lakshamanamurthy et al. [4] present an ad-hoc approach limited to the IXP2400 network processor.

Thiele et al. [5] present a generic approach for modeling applications and architectures in this domain. This model

although specialized to some extent for our domain, provides a natural way to specify different architectural parameters, workloads and application task graphs. To date, no comparison of such techniques to real-life network processor designs exists. Although the comparison by Chakraborty et al. [11] focuses on the packet transport from network interfaces to a general-purpose CPU core, ours is the only work that is able to capture the characteristics of the network processing domain by emphasizing a fine-grained description of a multiprocessor system and the packet processing itself. Our chosen, representative network processor scenario thus considers a frequent interleaving of computation and memory accesses using multithreading on a heterogeneous architecture. We have extended the architecture modeling of [5] to enable modeling a network processor such as IXP1200. Indeed, a primary goal of this work is to compare, the implementation of IPv4 forwarding switch application on Intel IXP1200 network processor, with an analytical model of the application mapped to the IXP1200 model. In this process, we hope to understand the main limitations of current state-of-the-art tools and evaluate the suitability of analytical approaches for high-level design decisions.

The paper is organized as follows: First, we introduce the analytical framework. Second, we describe how the IPv4 application and the IXP1200 architecture are modeled within this framework. We then provide and compare results of our case study. We conclude with observations and scope for future work.

2. Analytical Framework

In the following we will briefly introduce task and architecture models, service and arrival curves as well as the network calculus in order to model and determine the workload, the application, the NP architecture, and the performance of a given mapping of the application onto NP architecture. We restrict the description of the calculus to the basics and refer the reader to [5][6] for a more refined description, providing tighter bounds by using upper and lower arrival and service curves.

2.1. Workload description

A workload on the network processor is defined by arrival curves of the incoming network traffic and the task model associated with network traffic flows. Arrival curves describe a kind of worst-case envelope by which traffic patterns are restricted in terms of, for instance, burstiness and average rate. Arrival curves are used in the context of Integrated Services [7] in the Internet and thus provide a natural abstraction of network traffic in the application domain of network processing.

Arrival function x: The arrival function *x(t)* of a network flow is equal to the number of packets seen on the flow within the time interval *[0,t]* at a defined place in the network processor micro-architecture.

Arrival curve a: Given a non-decreasing function $a(t)$ defined for $t \geq 0$, a flow with arrival function x is constrained by the arrival curve a if and only if for all $s \leq t: \; x(t) - x(s) \leq a(t-s)$.

That means, during any time window of width t the amount of traffic for the flow is limited by $a(t)$. A common traffic specification is defined by the IETF (TSPEC [8]) that restricts peak p and average rates r as well as the burstiness b of a traffic flow as shown in Fig. 1.

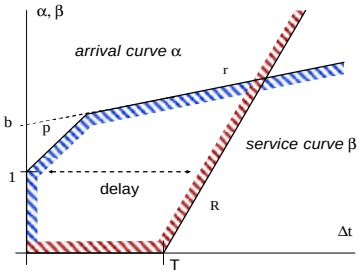

Fig. 1 Arrival and service curves.

Task model: Let F be a set of flows and T be a set of tasks. To each flow $f \in F$ there is one directed acyclic graph *G(f) = (V(f), E(f))* with task nodes $V(f) \subseteq T$ and communication edges *E(f)*. The tasks $v \in V(f)$ must be executed for each packet of flow *f* while respecting the precedence relations in *E(f)*. Associated with each task *v* there is a weight *w(v)* describing the computation demand of *v* on a given computing resource, e.g. specified in clock cycles. With each communication edge $e \in E(f)$ there is a weight *w(e)* defining the communication demand between the two connected tasks, e.g. given in units of bytes.

2.2. Micro-architecture model

Architecture components: The micro-architecture of a network processor consists of computation (CPU cores, special units, etc.) and communication components (buses) as well as separate RAMs. Associated with each architecture component there is one service curve describing the service capabilities of the component.

Service curve b: Given a component *S* and a flow through *S* with arrival function *x(t)* at the input of *S* and arrival function *y(t)* after processing at the output of *S*, *S* offers a service curve *b* to the traffic flow if and only if for all $t \geq 0$ there is some $t_0 \leq t$ such that $y(t) - x(t_0) \geq b(t - t_0)$. That means in particular, a flow backlogged during any time interval t receives at least a

flow-through of *b(t)*, e.g. specified in clock cycles/sec or bytes/sec. A curve representing Round-Robin scheduling with average rate *R* is sketched in Fig. 1.

2.3. Determining system properties

Worst-case bounds for a flow's backlog at component *S* and the delay experienced by a packet arriving at the component can be determined. We only show a formula for the delay (see [5][6] for details).

Bounded delay d: A flow constrained by *a* passes a component offering service *b* to the flow. The delay *d(t)* experienced by a packet in the component satisfies for all *t*: $d(t) \le \sup_{s \ge 0}(\inf\{t : t \ge 0 \text{ and } a(s) \le b(s+t)\})$.

Since arrival curves are specified in units of packets/sec, either the arrival or the service curve must be normalized to the unit of the other curve by using the computation or communication demand *w* respectively. For all curves used in this paper determining delay bounds reduces to the calculation of the maximum horizontal distance between corresponding *a(t)* and *b(t)* curves (see Fig. 1).

In order to determine delay properties as well as the utilization of the whole network processor, the accumulated service curve offered by the NP to a flow as well as the arrival curves of processed flows can be calculated by iterating through all service curves offered to the flows on different components [5][6]. As an example we give the formula for the calculation of the remaining service *b'(t).*

Remaining service b': The remaining service *b'(t)* of a component offering a service *b* to a flow *f* (which is constrained by *a(t)*) after processing one task *v* with demand *w(v)* for all packets of flow *f* is given by $b'(t) = \sup_{0 \le u \le t}(b(u) - w(v) \cdot a(u))$.

2.4. Capabilities and limitations

Since the analytical approach is derived from the traffic and service model used in the Quality of Service framework for the Internet, this method is in particular suited to describe workloads and performance properties in the packet processing domain. Some inherent features include:

- *Multithreading can be captured:* If several tasks are mapped to the same computation component they share a single service curve. If more tasks were mapped to a component than it can support using separate thread contexts, any additional tasks could be punished by an increase in its computation demand to account for the required thread context swap.
- *Heterogeneous micro-architectures can be considered:* Besides different types of computation components such as general purpose computation cores or dedicated hardware units, several concurrent communication buses and single-port memory interfaces can be modeled as well.
- *Pipelining of processing elements is possible:* Tasks of the execution path of a flow may be mapped to different computation components that are interconnected by point-to-point connections.
- *Incorporation of shared resource arbitration:* The order by which service curves are processed as well as the shape of the curves determine the thread scheduling and access arbitration scheme used by computation and communication components respectively. In [5], examples for fixed priority scheduling and generalized processor sharing (GPS) are shown.

The following extensions have been added to the framework in [5]: Round-Robin and GPS scheduling, a general, non-template based approach to model communication in order to enable automated design space exploration of the computation and the communication structure and tighter memory bounds by performing a lifetime-like analysis on task execution chains. Further improvements could include the modeling of multi-port RAMs.

Due to their accumulative nature service curves cannot express any locality of accesses. This is not a drawback for most applications in the packet-processing domain. The incorporation of caches would require the interplay with other analytical models.

The service curve approach is in particular suited to determine worst-case corner cases of a design since its foundations are in the real-time domain. Simulation is better suited to capture sporadic and arbitrary dynamic effects during the run-time of the system. The analytical approach only needs per-packet processing and communication demands which could be determined by analysis of pseudo code or some other estimation technique. A simulation framework however always requires an executable model of the design.

3. Modeling IPv4 Forwarding application

A 16 fast Ethernet port IPv4 forwarding switch application is used in this work. Our functional specification of the application is based on RFC 1812 [9]. The main components of this functionality are: Packets with invalid IP version numbers, broadcast packets, packets with time-to-live (TTL) field less than one and packets with invalid addresses are dropped; packet header checksum is calculated and the packet is dropped if the checksum is invalid; the TTL field of packet headers with valid output ports based on routing table entries are decremented and are routed to appropriate output port. The output port is determined by performing a longest prefix match on the IP destination address field. Figure 2 illustrates the main components in

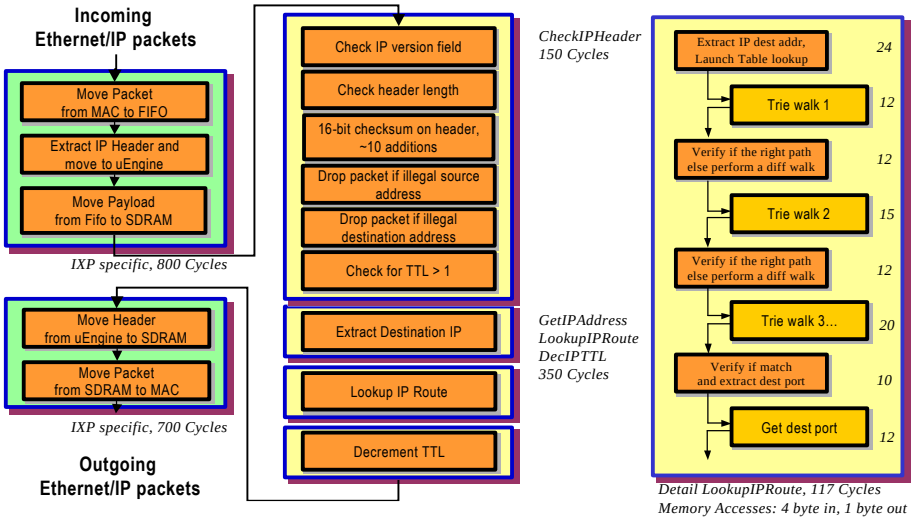

Fig. 2 Instance of IPv4 task graph derived from the application for analysis. A detailed graph of IP lookup is shown on the right.

the functionality of our benchmark annotated with cycle counts for 64byte packet size.

The components above represent the core functionality of the benchmark. In addition, a number of steps are required to receive the packets from the external MAC unit into the IXP1200 and extract the packet header, on which the above stated operations are performed. Lastly, the modified packet header and the packet payload need to be written back into the MAC unit via the IX bus unit. These additional operations, in fact, result in most of the programming effort for our application. For example, 14 detailed tasks are required to perform the core functionality of our benchmark whereas we need 42 detailed tasks to perform the ingress and egress operations on each packet. Thus modeling such complex application tasks is a challenge for analytical methods and needs to be investigated.

4. Modeling IXP1200 architecture

The Intel IXP1200 processor [10] is targeted for applications performing packet forwarding and classification at layers three and below of the OSI model. In this paper, we introduce only the main components of the IXP1200 utilized by our application as needed for modeling. The IXP1200 comprises six micro-engines, with four threads on every micro-engine, for computation. There are four unidirectional on-chip buses connecting both the off-chip memories (SRAM and SDRAM) to the micro-engines. External media access control units (MAC) are connected to the IXP1200 via the IX Bus. The IX bus interface unit has the required logic and memories to receive and transmit packets to the external MAC unit. The IX bus unit has a scratchpad memory (SRAM) and two FIFO memories, with each having 16 entries of size 64 bytes. In addition, the SDRAM unit is connected to the IX bus unit via a separate on-chip bus, used to transfer packet payloads directly based on micro-engine commands. An on-chip command bus carries events and signals between micro-engines and the IX bus unit.

In this work, we focus only on the data plane of the IXP1200 network processor. Hence, aspects related to the StrongARM processor are not modeled. Also, we have not modeled the PCI bus interface and the hash engine since we do not utilize these peripherals.

A typical packet flow through the IXP1200 follows the following steps: the external media access control (MAC) signals the receipt of a packet to the IX bus unit, which in turn receives the packet directed by the micro-engine. The packet header is extracted and read into the micro-engine directly from the IX bus unit; the payload is written directly from the receive FIFO into the SDRAM. The micro-engine performs all the packet header checks, as described earlier and launches a route table lookup, stored in SRAM. The micro-engine thread writes the packet header to the SDRAM and the packet descriptor with output port information to the SRAM.

On the transmit side, a micro-engine thread keeps polling the SRAM for packet descriptors. As soon as the scheduler notices a packet ready for transmission, it signals one of the available threads to perform the transmit operation. A micro-engine thread then initiates the transfer of the complete packet from the external SDRAM to the IX bus unit transmit FIFO. A transmit state machine manages the transfer of data from TFIFO to the external MAC.

5. Case study

A primary goal of this work is to implement a IPv4 forwarding switch using state-of-the-art tools on a network processor based system and compare this implementation with an analytical framework based model to enhance the quality of implementation (or results). In this process, we have first implemented the application on IXP1200 and then derived a task model as an input to the analytical framework, as described in earlier sections.

5.1. Experimental setup

The IXP1200 implementation environment comprises a micro-engine C language compiler and a simulation envi-

ronment that displays detailed timing as well as other execution statistics like micro-engine and memory utilizations. In our case study, we first developed the application in micro-engine C following the above specification based on the Intel reference code. The application was partitioned so that sixteen threads on four micro-engines were assigned one port each on receive (and forwarding) part. The transmit part of the application was assigned eight threads on two micro-engines. This partitioning holds since the end-to-end delay for a packet on the receive part is more than twice that on the transmit part.

Performance on the IXP1200 was measured using version 2.01 of the Developer Workbench assuming a clock frequency of 200 MHz; the IX bus is 64-bit wide and has a clock frequency of 80 MHz. Also, an off-chip and an off-chip SRAM are used. Two IXF440 external media access control unit (with eight fast Ethernet ports each) are connected to the IX bus and Ethernet IP packets are streamed from this unit to the IXP1200 and back. The packets for the application contain destination addresses evenly distributed across the IPv4 32-bit address space. We employ different packet sizes namely from 40 bytes to 256 bytes. There is a single packet source for each input port that generates an evenly distributed load across all the output ports.

The following parameters are extracted for different packet sizes to model the benchmark in the analytical approach: a detailed profiling of operations on one individual packet, namely the individual time spent in different tasks, the size of memory read or write and time required for different events and signals with respect to the micro-engine.

5.2. Results and Analysis

In order to have a reference set of design points we determined the maximum possible throughput for IPv4 forwarding without packet loss by simulation. We varied the packet length to account for payload storage versus header processing trade-offs. The results are shown in Fig. 3. As one can see, we approach line speed only for larger packet sizes where the micro-engines can keep up with the processing demand of the reduced number of packet arrivals (compared with small packet lengths). We can also recognize the influence of the 64byte receive and transmit FIFOs in the IX bus unit. As soon as an additional 64byte segment is needed, there is a drop in the throughput, due to the basic unit of data transfer between SDRAM and FIFOs being 64bytes. Thus for a given delay of two 64byte transfers we are transferring only 65bytes (instead of 128bytes).

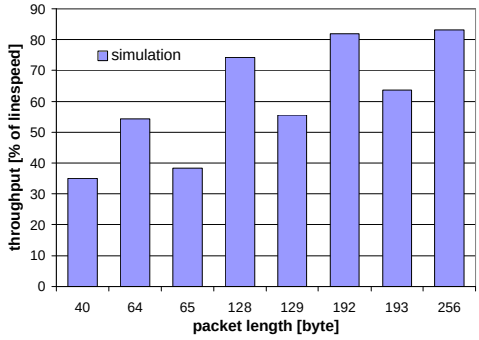

Fig. 3 Throughput for IPv4 forwarding on IXP.

Given these throughput numbers we compare the results from simulation with the performance values calculated with the analytical model of IP forwarding on the IXP micro-architecture by matching the corresponding traffic throughput values. Differences in the accuracy of the approaches thus become apparent by looking at delay values experienced by packets and the load generated by packet processing on the micro-architecture components.

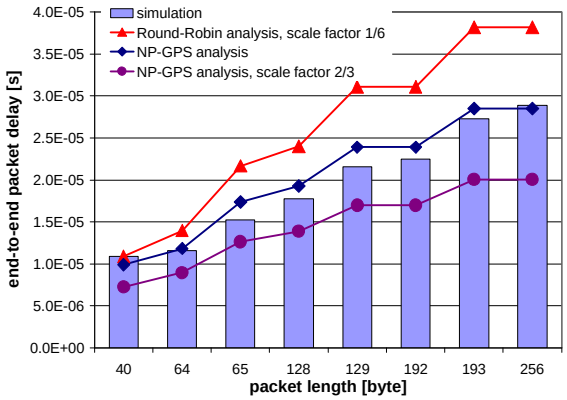

Fig. 4 Total delay experienced by packets.

Fig. 4 shows the end-to-end delay experienced by packets for different packet lengths. The first analysis series is based on Round-Robin (RR). The worst-case assumption for the arbitration penalty to equal the service time of a full RR frame is too pessimistic. For this series, we reduced the penalty (parameter *T* in Fig. 1) to 1/6 of the original value for each resource. The second series uses non-preemptive generalized processor sharing (NP-GPS). Thus, the arbitration penalty reduces to the service length of only a single task/packet of maximum size. This assumption quite well matches the results from simulation. Close examination by simulation indeed confirms that the average waiting time of a task is only in the order of one task length. The third series gives us an optimistic bound on the packet delay by reducing the arbitration penalty further by a factor of 2/3.

An example of the component load is given in Fig. 5 which compares the load of the receive micro-engines. The analysis results are based on original NP-GPS scheduling. The simulation results are subdivided into polling effects and the load generated by the actual computation. Polling artifacts are not considered in the analytical model.

Both figures show a deviation of less than 15% between the cycle-accurate simulation and the analytical model. Moreover, the values obtained by simulation and by analysis show the same increasing or decreasing trends in the same order of magnitude. For instance, the end-to-end delay sharply increases as soon as the processing of an additional 64byte segment is required.

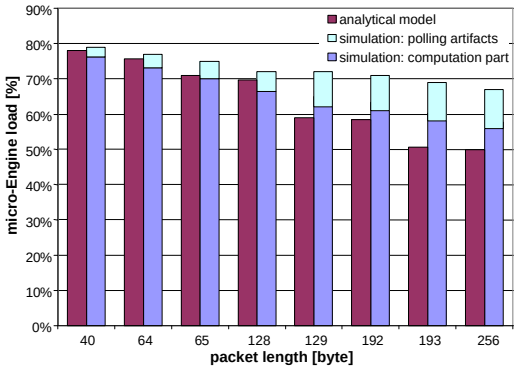

Fig. 5 Load of the receive-mEngines.

5.3. Discussion

In our study we have shown that the evaluation of a challenging real-world example showed only minor differences between simulation-based and analytical model-based approaches. The evaluation also revealed the sensitivity of the design on the chosen scheduling strategy which underpins the usefulness of the approach to quickly identify corner cases of possible implementations.

Since we had to perform a reverse engineering of the IXP to do the comparison in such detail, some uncertainties remain such as the rotating priority-based arbitration scheme of the command bus which cannot be accurately modeled in the current framework. More important than matching exact numbers however is the matching of trends when modifying parameters. This is why we believe that the service curve approach is indeed a reasonable candidate for design space exploration of network processors where the main goal is to find a first ranking of suitable designs.

6. Summary and future work

We have implemented IPv4 packet forwarding on the Intel IXP1200 network processor using an ad-hoc partitioning between threads and micro-engines. We have compared the performance obtained by simulation with a network calculus-based analytical approach. Comparing both results we conclude:

- For small packet sizes, the chosen partition cannot keep up with the line speed of the incoming ports.
- Artifacts of the 64byte segment organization are clearly visible and are in particular apparent for small packets.
- The network calculus-based analytical approach is able to capture trends in resource utilization and packet delay with a suitable level of detail for being used in high-level design space exploration (DSE).

As a next step, we will use the analytical models to find an optimal partitioning of processing steps to micro-engine threads. We will continue to broaden our DSE scope by investigating varying workloads analytically for which a trace-based simulation approach is too inflexible.

References

[1] P. Crowley, M. Fiuczynski, J. Baer, B. Bershad, "Characterizing Processor Architectures for Programmable Network Interfaces," Proceedings of the *2000 International Conference on Supercomputing*, Santa Fe, N.M., May 2000.

[2] 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): 768-783, June 2001.

[3] M. Franklin, T. Wolf, "A Network Processor Performance and Design Model with Benchmark Parameterization," *First Workshop on Network Processors* at the 8th International Symposium on High Performance Computer Architecture (HPCA8), Cambridge MA, USA, February 2002.

[4] S. Lakshmanamurthy, K.-Y. Liu, Y. Pun, L. Huston, U. Naik, "Network Processor Performance Analysis Methodology," *Intel Technology Journal*, 6(3): 19-28, August 2002.

[5] L. Thiele, S. Chakraborty, M. Gries, S. Künzli, "Design Space Exploration of Network Processor Architectures," *First Workshop on Network Processors* at the 8th International Symposium on High Performance Computer Architecture (HPCA8), Cambridge MA, USA, February, 2002.

[6] J.-Y. Le Boudec, P. Thiran, "Network Calculus: A Theory of Deterministic Queuing Systems for the Internet," *LNCS 2050*, Springer Verlag, 2001

[7] B. Braden, D. Clark, S. Shenker, "Integrated Services in the Internet architecture: an overview," *RFC1633*, Internet Engineering Task Force (IETF), June 1994.

[8] S. Shenker, J. Wroclawski, "General characterization parameters for integrated service network elements," *RFC2215*, Internet Engineering Task Force (IETF), Sept. 1997.

[9] F. Baker, "Requirements for IP Version 4 Routers," *RFC1812*, Internet Engineering Task Force (IETF), June 1995.

[10] Intel Corporation, "Intel IXP1200 Network Processor Family: Hardware Reference Manual," Revision 8, pp. 225-228, August 2001.

[11] S. Chakraborty, S. Künzli, L. Thiele, A. Herkersdorf, P. Sagmeister, "Performance Evaluation of Network Processor Architectures: Combining Simulation with Analytical Estimation," to appear, *Computer Networks, Elsevier Science*, 2003