Performance Models for Network Processor Design

Tilman Wolf, *Member*, *IEEE*, and Mark A. Franklin, *Fellow*, *IEEE*

Abstract—To provide a variety of new and advanced communications services, computer networks are required to perform increasingly complex packet processing. This processing typically takes place on network routers and their associated components. An increasingly central component in router design is a chip-multiprocessor (CMP) referred to as "network processor" or NP. In addition to multiple processors, NPs have multiple forms of on-chip memory, various network and off-chip memory interfaces, and other specialized logic components such as CAMs (Content Addressable Memories). The design space for NPs (e.g., number of processors, caches, cache sizes, etc.) is large due to the diverse workload, application requirements, and system characteristics. System design constraints relate to the maximum chip area and the power consumption that are permissible while achieving defined line rates and executing required packet functions. In this paper, an analytic performance model that captures the processing performance, chip area, and power consumption for a prototypical NP is developed and used to provide quantitative insights into system design trade offs. The model, parameterized with a networking application benchmark, provides the basis for the design of a scalable, high-performance network processor and presents insights into how best to configure the numerous design elements associated with NPs.

Index Terms—Network processor design, performance model, design optimization, power optimization, network processor benchmark.

1 Introduction

Over the last several years, network processors (NPs) have become important components in router designs. By providing for programmability of the data path, they permit adaptation to new functional requirements and standards. Additionally, network processors provide a powerful chip-multiprocessor architecture, typically containing logic components and instructions specialized to the networking environment to satisfy a range of performance requirements. At this point, there are a number of companies producing a variety of network processors, such as the Intel IXP2800 [1], the Hifn 5NP4G (formerly IBM PowerNP [2]), the AMCC np7510 [3], and the EZchip NP-1 [4].

1.1 Motivation

At the hardware level, there are four key concerns in the design of NPs. These have a direct impact on the system architecture and the configuration of a particular network processor.

- **Line Speed Maintenance:** The NP must perform the required computational tasks fast enough to keep up with input communication line speeds.
- **Functional Power:** The NP must be able to perform the required functional tasks associated with its targeted environment (e.g., process packets or cells by implementing IPv4, IPv6, MPLS, etc.).
- **Cost:** The cost of the chip should be reasonable. In this paper, we deal with only manufacturing costs and consider chip area to be a proxy for these costs.
- **Electrical Power Dissipation:** The NP must not consume an excessive amount of power. Limitations here are associated with the individual router line card on which the NP resides and extends to power constraints on racks and cabinets containing such line cards.

The principal contribution of this paper is in providing an analytic model that quantifies the relationships between the four elements listed above and relates them both to specific design elements (e.g., number of processors on the NP, caches sizes, multithreading level, memory I/O interfaces, etc.) and to the anticipated application workload. With this model, the NP architecture design space can be explored and an understanding of design trade offs derived. Additionally, the impact of new applications on NP performance characteristics can be evaluated. In contrast with complex simulation-based models, providing an analytic model permits such design studies to be done quickly and permits fast exploration of "optimal" designs.

1.2 System Architecture

We consider the prototypical NP architecture shown in Fig. 1, which is a generalization of various commercial designs. It contains a number of identical multithreaded general-purpose processor cores, each having its own instruction and data caches. To satisfy off-chip memory bandwidth requirements, groups of processors are clustered together and share a memory interface. A scheduler assigns packets from independent flows to the different processors. Thus, after assignment of a flow to a processor, all packets of the same flow are routed to the same processor. Speedup and computational power is achieved

- *T. Wolf is with the Department of Electrical and Computer Engineering, Knowles Engineering Building 211C, 151 Holdsworth Way, University of Massachusetts, Amherst, MA 01003. E-mail: wolf@ecs.umass.edu.*
- *M.A. Franklin is with the Department of Computer Science and Engineering, Campus Box 1045, Washington University, St. Louis, MO 63130. E-mail: jbf@cse.wustl.edu.*

Manuscript received 31 Aug. 2005; revised 31 Dec. 2005; accepted 10 Jan. 2006; published online 26 Apr. 2006.
Recommended for acceptance by A. Gonzalez.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-0392-0805.

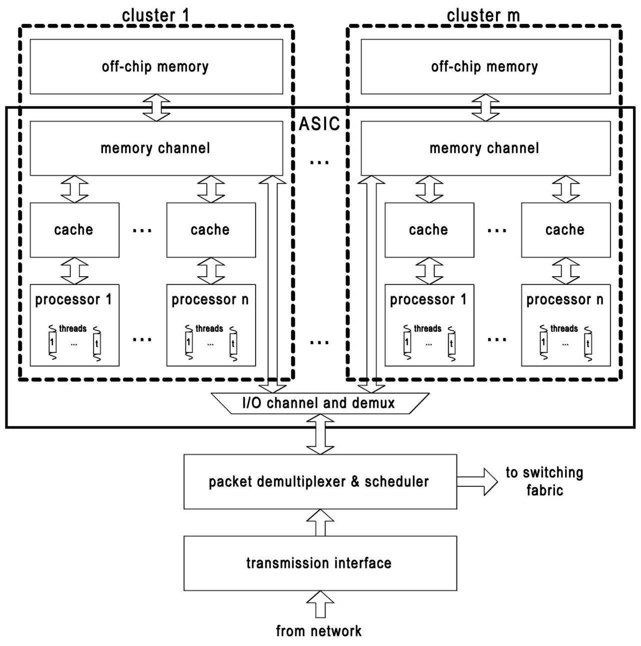


Fig. 1. Overall network processor architecture.

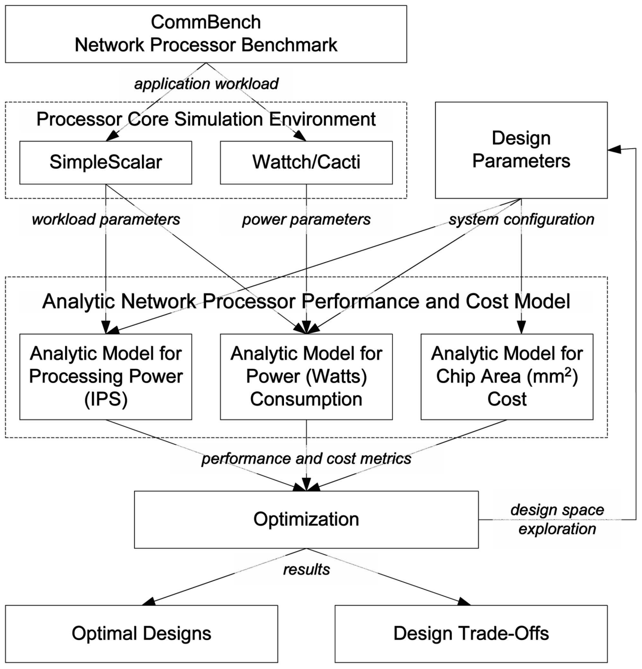

CommBench Network Processor Benchmark
application workload
Processor Core Simulation Environment
SimpleScalar
Wattch/Cacti
Design Parameters
workload parameters
power parameters
system configuration
Analytic Network Processor Performance and Cost Model
Analytic Model for Processing Power (IPS)
Analytic Model for Power (Watts) Consumption
Analytic Model for Chip Area (mm²) Cost
performance and cost metrics
Optimization
design space exploration
results
Optimal Designs
Design Trade-Offs

Fig. 2. Model development and optimization process.

by exploiting parallelism at the flow level. Note that additional speedup can be obtained by also exploiting packet level parallelism, however, this is not considered here. All of the processors are assumed to be identical and capable of executing the programs necessary for implementing NP functions.

Commercial network processors typically contain a few additional units to implement queue memory management, hardware support for checksum and hash computations, and other specialized coprocessors. These components help speed-up certain tasks, but complicate a general comparison of design trade offs. Practice has also shown that simpler, more uniform architectures are often easier to program. Our system architecture is therefore intentionally kept general without considering any product-specific components.

1.3 Methodology

The methodology used in exploration of the network processor design space is shown in Fig. 2. The main components are the analytic processing performance and cost models shown in the center of the figure. Three elements are present: a model for processing power that yields the performance of a particular configuration in terms of IPS (instructions per second), a cost model that estimates power consumption in terms of Watts, and a cost model that evaluates chip area in terms of square millimeters required for implementation. In order to derive results from these models (either in the form of one optimal design for a given set of parameters or more general design trade offs), a number of input parameters must be specified. Workload parameters (e.g., cache miss rates) and power parameters (e.g., cache power requirements) depend on the application workload. These parameters are derived from simulations that utilize well-established tools (e.g., SimpleScalar [5], Wattch [6], and CACTI [7]). The workload itself, an input to these simulators, is generated from an NP benchmark, *CommBench*, that has been developed by the authors and is reported in [8]. The system configuration parameters correspond to the various elements shown in Fig. 1 (e.g., number of processor cores, size of memory caches, etc.). These parameters are varied to explore a broad design space and results are derived as part of this iterative process. The processor core simulation environment is also used to verify the accuracy of the analytic power model.

The remainder of this paper presents the processing performance, power, and area cost models, the benchmark, and selected results and associated insights into the NP design process. Section 2 begins with a review of selected related work. The analytic models are presented in Section 3. Section 4 introduces *CommBench*, the NP application benchmark used in obtaining workload parameters. Section 5 presents a verification of the models and the design results that we have obtained. Section 6 summarizes and concludes the paper.

2 Related Work

Processing of traffic on a network node includes a range of functions that go beyond simple packet forwarding. For example, it is common for routers to perform firewalling [9], network address translation (NAT) [10], Web switching [11], IP traceback [12], and other functions. With increasingly heterogeneous end-systems (e.g., mobile devices and "thin" clients), computationally more demanding services have been moved into the network. Examples of these are content transcoding, advertisement insertion, and cryptographic processing. As customers demand more services, it is likely that this trend toward including more functionality on the router will continue. The IETF OPES (Open Pluggable Edge Services working group) is working to define such advanced networking service platforms.

In response to these demands for increasing flexibility and processing performance, the general trend has been to use network processors [1], [2], [3], [4]. These CMPs typically follow advanced RISC design principles for the core processors, but also often provide for some specialization. Specialization features include memory organization (e.g., cache size, associativity, design), word size (e.g., 16, 24, 32 bit register/instruction size), functional components and coprocessors [13] (e.g., floating point, timers, special logic blocks [14]), and new instructions (e.g., ability to define new processor instructions [15]). This set of alternatives represents the first generation of choices associated with the development of NP designs and is currently being considered.

A key design issue is just how to select from the numerous alternatives, given the available chip area and processing performance implications of each decision. Crowley et al. have evaluated different processor architectures for their performance under networking workloads [16]. This work primarily focuses on the trade offs between RISC, superscalar, and multithreaded architectures. In more recent work, a modeling framework is proposed that considers the data flow through the system [17]. Thiele et al. have proposed a general processing performance model for NPs [18] that takes into account the system workload in terms of data traffic streams, the performance of a processor under different scenarios, and effects of the queueing system. In our work, we provide a very broad, analytically based performance model that can be quickly solved and that includes the key attributes of computational performance, chip area, and electrical power consumption. It provides a systematic approach to developing a *quantitative* understanding of the design choices. This work has recently been extended to consider applications that are partitioned over multiple processor cores [19] and their implications on design trade offs between parallel and pipelined NP topologies [20].

In conjunction with performance models, it is necessary to obtain realistic system and workload parameters. Traditional benchmarks (e.g., SPEC [21]) do not reflect the simplicity and repetitiveness of networking processing environments. Another significant shortcoming is the missing focus on clearly defined I/O. We therefore developed a new benchmark, CommBench, that matches the workload characteristics of a network processor. CommBench includes streaming data flow-based applications and packet-based processing tasks. Memik et al. have proposed NetBench [22] and Lee and John have proposed NpBench [23], both of which have been published more recently. A commercial benchmark for network processors is the networking benchmark by the Embedded Microprocessor Benchmarking Consortium (EEMBC) [24]. This benchmark focuses on both data-plane and control-plane operations.

3 Analytic Performance and Cost Models

In this section, analytic models for both processing performance (i.e., instructions per second) and cost (i.e., power consumption in Watts and area in square millimeters) are developed. The models are based on the system configuration of Fig. 1 and use the parameters and notations shown in Table 1.

TABLE 1
System Parameters

Component	Symbol	Description
processor p	clk_p	processor clock frequency
	t	number of hardware threads on processor
	ρ_p	processor utilization
program a	f_{load_a}	frequency of load instructions
	f_{store_a}	frequency of store instructions
	$mi_{c,a}$	i-cache miss probability for cache size c_i
	$md_{c,a}$	d-cache miss probability for cache size c_d
	$dirty_{c,a}$	prob. of dirty bit set in d-cache of size c_d
	$compl_a$	complexity (instr. per byte of packet)
caches	c_i	instruction cache size
	c_d	data cache size
	$linesize$	cache line size of i- and d-cache
off-chip memory	τ_{DRAM}	access time of off-chip memory
memory channel	$width_{mchl}$	width of memory channel
	clk_{mchl}	memory channel clock frequency
	ρ_{mchl}	load on memory channel
I/O channel	$width_{io}$	width of I/O channel
	clk_{io}	clock frequency of I/O channel
	ρ_{io}	load on I/O channel
cluster	n	number of processors per cluster
ASIC	m	number of clusters and memory channels
	A_x	actual size of component x, with $x \in \{ASIC, p, c_i, c_d, io, mchl\}$

The network processor considered has m clusters with n RISC processors in each cluster. Each cluster has a single memory interface that is shared among the processors of each cluster. The entire chip has one I/O interface through which packet data is both received and transmitted. Each processor has its own instruction and data caches of size c_i and c_d bytes. The caches are shared among the t threads that can be supported in hardware by each processor. We assume that context-switching is done in hardware with zero cycle overhead. Thus, if one thread stalls on a memory miss, another thread can immediately start executing with zero cycle delay. The processor is taken to be a typical RISC processor that ideally executes one instruction per cycle when no hazards are present. We also assume that the on-chip memory cache can be accessed in a single cycle.

3.1 Processing Performance

For a single processor, processing power can be expressed as the product of the processor's utilization, ρ_p, and its clock frequency, clk_p. The processing power of the entire NP, IPS_{NP}, can be expressed as the sum of processing power of all the processors on the chip. Thus, with m clusters of processors and n processors per cluster:

$$IPS_{NP} = \sum_{j=1}^{m} \sum_{k=1}^{n} \rho_{p_{j,k}} \cdot clk_{p_{j,k}}. \qquad (1)$$

If all processors are identical and execute the same workload, then, on average, the processing power is:

$$IPS_{NP} = m \cdot n \cdot \rho_p \cdot clk_p. \qquad (2)$$

A key question is how to determine processor utilization. In the extreme case where there are a large number of threads per processor, for large caches that reduce memory misses and low memory miss penalties, the utilization approaches 1. However, a large number of thread contexts and larger caches require more chip area. Other stalls due to hazards, such as branch misprediction, are not considered here. Our model targets processors with shallow pipelines where branch mispredictions generally have a relatively small effect compared to the effects of cache misses.

Using the model proposed and verified by Agarwal [25], the utilization $\rho_p(t)$ of a multithreaded processor is given as a function of the cache miss rate p_{miss}, the off-chip memory access time τ_{mem}, and the number of threads t as:

$$\rho_p(t) = 1 - \frac{1}{\sum_{i=0}^{t} \left(\frac{1}{p_{miss} \cdot \tau_{mem}}\right)^i \frac{t!}{(t-i)!}}. \quad (3)$$

To illustrate the overall trend in this equation, we can simplify (3) by ignoring the second and higher order terms of the summation. Thus:

$$\rho_p(t) = \frac{t}{(t + \tau_{mem} \cdot p_{miss})}. \quad (4)$$

Note from this expression that, as expected, the utilization decreases with increasing miss rates and with increasing miss penalties for off-chip memory accesses. However, the larger the number of threads, t, the less the impact of τ_{mem} and p_{miss}, since more threads are available for processing and processor stalls are less likely. In the limit as $t \to \infty$, $\rho_p(t) = 1$. While it is desirable to run processors at high utilization, there is an area and power cost associated with large numbers of thread contexts. This impacts overall processing performance since the additional processor area leads to less area available for caches and, thus, higher miss rates. On the other hand, more threads can also help mask cache misses and, thus, can be beneficial. This design trade off can be understood more fully only after expressions for the memory access time, τ_{mem}, and the cache miss rate, p_{miss}, are obtained.

3.1.1 Off-Chip Memory Access

We assume the memory channel implements a FIFO service order on the memory requests and that the requests are interleaved in a split transaction fashion. It is assumed that the memory controller can generate an ideal job schedule and the system does not use multiple memory banks. The total off-chip memory request time, τ_{mem}, thus has three components: the bus access time, τ_Q, the physical memory access time, τ_{DRAM}, and the cache line transmission time, $\tau_{transmit}$ (all expressed in terms of processor clock cycles):

$$\tau_{mem} = \tau_Q + \tau_{DRAM} + \tau_{transmit}. \quad (5)$$

The DRAM access time and the cache line transmission time are straightforward to determine. The queuing time, however, depends on the memory channel load which, in turn, depends on the number of processors that share the memory channel, the number of threads per processor, and the cache miss rates.

The queuing time can be approximately modeled as a single server M/D/1 queuing system with n processors each generating requests at a mean rate of p_{miss}, for a total mean request rate of $n \cdot p_{miss}$. While the request distribution

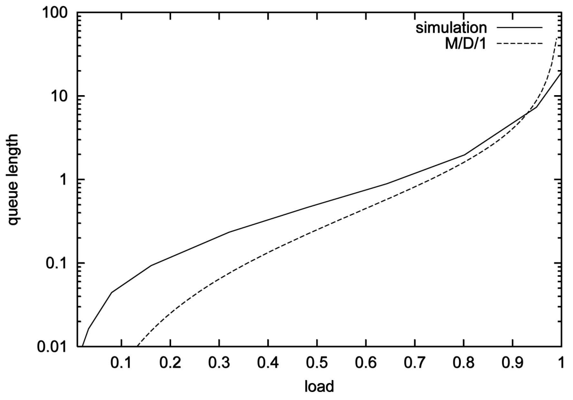


Fig. 3. Comparison of average memory queue length for different queuing models.

can be modeled as geometrically distributed random variables (suggested in [25]) with the distribution parameter p_{miss}, for simplicity, we use a continuous exponential distribution with this same mean request rate.

Note that the system is more accurately modeled as a finite source system, the machine repairman model, since there are a finite number of processor sources and threads $(n \cdot t)$. However, comparing the M/D/1 model with a more realistic simulation-based finite source model indicates that the M/D/1 model is reasonably accurate. This is shown in Fig. 3, where the average queue length for both models ($t = 8$, $n = 4$, and $\tau_{transmit} = 40$) are presented. For typical loads of $\rho_{mchl} = 0.5 \ldots 0.9$, the difference between the two models is relatively small and, below 50 percent load, the queue length is small enough so that the M/D/1 model is an acceptable model component in determining overall memory channel performance.

The bus access time, τ_Q, is now expressed as the queuing time of an M/D/1 system as:

$$\tau_Q = \frac{\rho_{mchl}^2}{2(1 - \rho_{mchl})} \cdot \frac{linesize}{width_{mchl}} \cdot \frac{clk_p}{clk_{mchl}}. \quad (6)$$

With a fixed DRAM access time, τ_{DRAM}, and a transmission time of

$$\tau_{transmit} = \frac{linesize}{width_{mchl}} \cdot \frac{clk_p}{clk_{mchl}}, \quad (7)$$

we can substitute in (5) to obtain the memory access time:

$$\tau_{mem} = \tau_{DRAM} + \left(1 + \frac{\rho_{mchl}^2}{2(1 - \rho_{mchl})}\right) \cdot \frac{linesize}{width_{mchl}} \cdot \frac{clk_p}{clk_{mchl}}. \quad (8)$$

3.1.2 On-Chip Cache

The remaining component needed to evaluate the utilization expression (3) is the cache miss rate p_{miss}. For a simple RISC-style load-store processor running application a, the miss probability is given as [26]:

$$p_{miss,a} = mi_{c,a} + (f_{load_a} + f_{store_a}) \cdot md_{c,a}, \quad (9)$$

where $mi_{c,a}$ and $md_{c,a}$ are the instruction and data cache miss rates and f_{load_a} and f_{store_a} are the frequency of occurrence of load and store instructions associated with application a. The instruction and data cache miss rates depend on the application, the cache sizes that have been implemented, and the effects of cache pollution due to multithreading.

Cache pollution from multithreading reduces the effective cache size that is available to each thread. On every memory stall, a thread gets to request one new cache line (replacing the least recently used line). While the thread is stalled, $t-1$ other threads can replace one line and, in steady-state, each thread can use on average $\frac{1}{t}$ of the available cache. If the working set size of a thread is very small, its effective cache usage could be less than $\frac{1}{t}$ (and the other threads use slightly more). In a network processor, due to chip area constraints, cache sizes are expected to be smaller than the working set size. This leads to equal sharing of the available cache between threads. Thus, the effective cache size that is available to a thread is:

$$c_{i,eff} = \frac{c_i}{t}, \quad c_{d,eff} = \frac{c_d}{t}. \tag{10}$$

This is a conservative estimate for the effective instruction cache as it is also possible that positive cache pollution occurs when multiple threads execute the same code. The application characteristics that are necessary for evaluating (9) are derived from our network processing benchmark that is discussed in Section 4.

3.1.3 Memory and I/O Channels

The expression for miss rate, p_{miss}, (9) and for total memory access time, τ_{mem}, (5) can now be substituted into (3) to obtain processor utilization. In order to do this, we need to fix the memory channel load, ρ_{mchl}, since τ_Q depends on ρ_{mchl}. Thus, with the memory channel load given, we can determine the utilization of a single processor. This yields the memory bandwidth, $bw_{mchl,1}$, required by a single processor:

$$bw_{mchl,1} = \rho_p \cdot clk_p \cdot linesize \cdot (mi_c + (f_{load} + f_{store}) \cdot md_c \cdot (1 + dirty_c)). \tag{11}$$

Note that, through use of the parameter $dirty_c$, the probability of the dirty bit being set, this equation models the situation where a dirty cache line needs to be written back to memory. In (9), considering dirty cache lines was not necessary, since a write-back does not stall the processor. If write-back can be ignored, (5) can be approximated by:

$$bw^*_{mchl,1} = \rho_p \cdot clk_p \cdot linesize \cdot p_{miss}. \tag{12}$$

The number of processors, n, in a cluster is then the number of processors that can share the memory channel without exceeding the specified load:

$$n = \left\lfloor \frac{width_{mchl} \cdot clk_{mchl} \cdot \rho_{mchl}}{bw_{mchl,1}} \right\rfloor. \tag{13}$$

This provides a cluster configuration model for all ranges of cache sizes and thread contexts. Finally, however, for a complete model, an expression of the I/O channel bandwidth is required. The I/O channel transmits packets both to and from the processing engines and, thus, each packet traverses the channel twice. What is needed next is a relation between the number of instructions executed in processing a packet and the size of the packet. We define a parameter, $compl_a$, as the "complexity" of an application. It is expressed as the number of processing instructions that are necessary per byte of packet data (the formal definition is found in (30) in Section 4). The I/O channel is operated at a load of ρ_{IO}; thus, the I/O channel bandwidth for the entire network processor is:

$$bw_{IO} = 2 \cdot \frac{IPS}{compl_a \cdot \rho_{IO}}. \tag{14}$$

With the processing performance of the system determined by IPS, we can now turn toward the power consumption and chip area cost.

3.2 Power Consumption

Overall power consumption can be derived by determining the consumption of each major component of the system and adding them together. The principal components considered in the power model are:

1. Processor ALUs,
2. processor clock,
3. register files,
4. processor instruction and data caches (level 1, on-chip), and
5. off-chip memory and I/O bus.

Since we are interested in relative performance of alternative configurations for the architecture of Fig. 1, power associated with off-chip components and with driving the chip pins are not considered. Additionally, the contributions of branch predictors, etc., are ignored since, for simple NP RISC cores, it is not necessary to consider such complex components as the overall system power consumption is dominated by memory accesses and I/O operations. Thus, the overall network processor power consumption, P_{NP}, is modeled as a sum of the five components listed above (scaled to the appropriate number of processors and cache sizes). This constitutes 94 percent to 97 percent of overall power consumption with the remaining 3 percent to 6 percent being consumed miscellaneous other components.

For CMOS technology, dynamic power consumption P_d is defined as:

$$P_d = C \cdot V_{dd}^2 \cdot a \cdot f, \tag{15}$$

where C is the aggregate load capacitance associated with each component, V_{dd} is the supply voltage, a is the switching activity for each clock tick ($0 \leq a \leq 1$ and can be considered to be the utilization of the component) and f is the clock frequency. The energy expended per cycle is:

$$E_d = C \cdot V_{dd}^2 \cdot a. \tag{16}$$

By obtaining parameter values for (15) and (16), the power consumption models for each of the components is determined. Power becomes more important as feature sizes shrink below $.18\mu$m. For these cases, a more accurate power modeling tool than the one used here is necessary in order to account for leakage currents. However, the general results and design trends are not expected to not change significantly.

3.2.1 ALU Power Model

ALU power depends on the voltage, V_{dd}, processor clock frequency, f, the ALU utilization, a_{ALU}, and its capacitance:

$$P_{ALU} = C_{ALU} \cdot V_{dd}^2 \cdot a_{ALU} \cdot f. \tag{17}$$

The capacitance for the ALU can be derived with the Wattch toolkit as discussed in Section 4. The ALU utilization is set to $a_{ALU} = 1$ since the processors are continuously active. Even when stalled, the processor needs to check if a thread is reactivated. If the processor could sleep during stalls, the ALU utilization could be obtained from (3) ($a_{ALU} = \rho_p$), however, this design is not considered here.

3.2.2 Clock Power Model

In a similar fashion, clock power consumption can be obtained:

$$P_{clk} = C_{clk} \cdot V_{dd}^2 \cdot a_{clk} \cdot f. \tag{18}$$

Since the clock is changing state in every cycle, $a_{clk} = 1$. With differing cache configurations, the clock power consumption in Wattch can vary by up to ± 8 percent, however the model does not consider this effect. As will be shown in Section 5, overall power consumption that is predicted corresponds well to that obtained with Wattch.

3.2.3 Register File Power Model

The register file of a processor must maintain state for each of the t threads. We assume that the power consumption of the register file scales proportionally with the number of threads. The per-thread power consumption, P_{reg_t}, is

$$P_{reg_t} = C_{reg} \cdot V_{dd}^2 \cdot a_{reg} \cdot f. \tag{19}$$

The capacitance of the register file for one thread can be derived with the Wattch toolkit as discussed in Section 4. The register file utilization is equal to the ALU utilization ($a_{reg} = a_{ALU}$).

3.2.4 Cache Power Model

The power consumption of caches can be modeled by considering the different cache subcomponents (tag arrays, data arrays, etc.). This has been done in the CACTI 3.0 toolkit [7]. With this toolkit, the energy consumption of a cache access, E_c, on a cache of particular size and associativity can be derived. The power consumption depends on the frequency of the such accesses and is

$$P_c = E_c \cdot a_c \cdot f. \tag{20}$$

The dynamic power consumption of caches is due to memory accesses. In addition to instruction accesses that result in a hit, pipeline stalls are present due to i-cache misses or branch misprediction (we do not consider misprediction effects on cache power in this analysis). Adding in the effects of cache usage occurring after a miss, one obtains:

$$a_{c_i} = \rho_p \cdot (1 + mi_{ci,a}), \tag{21}$$

where $mi_{ci,a}$ is the instruction cache miss probability associated with application a and instruction cache size ci.

The data cache is accessed for each read/write (load/store) instruction and for each d-cache miss, thus:

$$a_{c_d} = \rho_p \cdot ((f_{load_a} + f_{store_a}) \cdot (1 + md_{cd,a})). \tag{22}$$

3.2.5 Memory and I/O Bus Power Model

Based on (15), the power consumption of the memory and I/O busses can now be calculated. The memory channel is characterized by its width, $width_{mchl}$, its physical length on the chip, $length_{mchl}$, its clock frequency, f_{mchl}, and its utilization $a_{mchl} = \rho_{mchl}$:

$$P_{mchl} = C_{mchl} \cdot V_{dd}^2 \cdot a_{mchl} \cdot f_{mchl}. \tag{23}$$

The capacitance, C_{mchl}, is based on the width and the length parameters and is given by:

$$C_{mchl} = 2 \cdot C_{tech} \cdot width_{mchl} \cdot length_{mchl}. \tag{24}$$

The factor of 2 is due to the coupling capacitance between wires. C_{tech} is the wire capacitance for a particular CMOS technology. Similarly, the power consumption of the I/O bus, P_{IO}, can be determined.

3.2.6 Total Power Consumption

The overall power consumption of the network processor can then be computed to be

$$P_{NP} = P_{IO} + m \cdot (P_{mchl} + n \cdot (P_{ALU} + t \cdot P_{reg_t} + P_{c_i} + P_{c_d})). \tag{25}$$

3.3 Chip Area

With each network processor component, there is an associated chip area cost. Each processor uses an area of A_p for its core, and A_t for each thread context:

$$A_{p,t} = A_p + t \cdot A_t. \tag{26}$$

The instruction and data caches use A_{c_i} and A_{c_d}, which can be derived from CACTI 3.0 [7]. The size of a memory or I/O bus consists of a basis area for the bus logic plus the on-chip area of the pin drivers and pads. The total size depends on the width of the bus:

$$A_{mchl,width} = A_{mchl} + width_{mchl} \cdot A_{pin}. \tag{27}$$

The on-chip area equation for a NP configuration in our general architecture is the summation over all the system component areas:

$$A_{NP} = A_{IO} + m \cdot (A_{mchl,width} + n \cdot (A_{p,t} + A_{c_i} + A_{c_d})). \tag{28}$$

3.4 Performance Metrics

Several performance metrics are used to evaluate alternative design choices. With an expression for processing performance (IPS), power consumption (P), and chip area (A), performance metrics of the following form can be derived:

$$Performance = IPS^{\alpha} \cdot P^{\beta} \cdot A^{\gamma}. \tag{29}$$

In particular, we are interested in the metrics that consider processing performance as a benefit ($\alpha > 0$) and area and power consumption as a cost ($\beta, \gamma \leq 0$). Several common processor performance metrics are possible:

- Processing/area or $IPS \cdot A^{-1}$: This metric considers only area and no power consumption.
- Processing/power or $IPS \cdot P^{-1}$: This metric assumes an equal weight to processing performance and power consumption.
- Processing/(power)^2 or $IPS \cdot P^{-2}$: This metric is similar to the above, however, power consumption is given a higher weight.
- Processing/(area $\cdot$ power) or $IPS \cdot A^{-1} \cdot P^{-1}$: This metric combines both area and power costs.
- Processing/(area $\cdot$ power)$^{\frac{1}{2}}$ or $IPS \cdot A^{\frac{-1}{2}} \cdot P^{-\frac{1}{2}}$: This metric also combines both area and power costs, but balances the overall cost with the processing performance.

For the design results in Section 5, the first two metrics are explored; however, with the model equations provided, all can be evaluated. To complete the evaluation process, a set of realistic workload-based parameters are needed. The next section discusses how these parameters are obtained.

4 Model Parameterization and Validation

4.1 Application Parameters

The performance and cost models require several parameters which depend on the system workload (e.g., cache miss rates, load, and store frequency). In order to derive realistic values that are typical of network processing workloads, we have developed a benchmark called CommBench.

4.1.1 CommBench Network Processor Benchmark

A desirable property of any application in a benchmark is its representativeness of a wider application class in the domain of interest. CommBench applications have been chosen with this in mind. A detailed description of all eight applications can be found in the Appendix. CommBench contains common lookup algorithms to represent typical routing lookups as well as more complex data transcoding algorithms (*Header-Processing Applications* (HPA) and *Payload-Processing Applications* (PPA)).

4.1.2 Data and Tools

To collect application parameters, all the benchmark programs were run on the SimpleScalar [27] simulator using the ARM target. SimpleScalar was configured to simulate a typical embedded RISC core. The C compiler used was *gcc* 2.8.1 (optimization level O2). The O2 level was selected because the compiler only performs optimizations that are independent of the target processor and does not exploit particular architectural features (e.g., loop unrolling for superscalar machines).

4.1.3 Application Parameters

For each application, the properties required for the performance model have been measured experimentally: computational complexity ($compl_a$), load and store instruction frequencies (f_{load_a}, f_{store_a}), instruction cache and data cache miss rate ($mi_{ci,a}, md_{cd,a}$), and dirty bit probability ($dirty_{cd,a}$). The application complexity is defined as

$$compl_a = \frac{instructions\ executed\ in\ application\ a}{packet\ size}. \quad (30)$$

TABLE 2
Average Computational Complexity and Load and Store Frequencies of Workloads

Workload	$compl$	f_{load}	f_{store}
W1 (HPA)	3.77	0.2118	0.0838
W2 (PPA)	203	0.1822	0.0569

These parameter values were obtained with SimpleScalar for cache sizes ranging from 1kB to 1MB. A 2-way associative write-back cache with a linesize of 32 bytes was simulated. The cache miss rates were measured such that cold cache misses were amortized over a long program run. Thus, they can be assumed to represent the steady-state miss rates of these applications.

4.1.4 Workloads

To simplify the analysis of the results from our model, we aggregate the CommBench applications into two workloads, W1 and W2. Workload W1 is a combination of the four header-processing applications and workload W2 consists of the four payload processing applications. The applications within the workloads are weighted such that each application processes an equal number of instructions over time. W1 applications process only packet headers and are generally less computationally demanding than W2 applications that process all of the data in a packet.

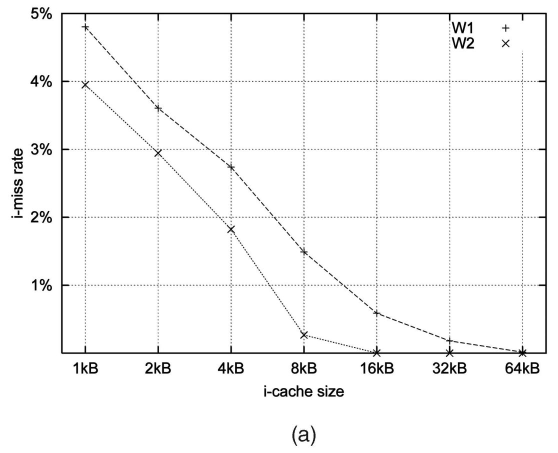
The average values for the model parameters were obtained for each of the benchmarks (W1 and W2) by averaging over the benchmark application values assuming equal probabilities for each application. Table 2 shows the aggregate complexity and load and store frequencies of the workloads. Note that the complexity of payload processing is significantly higher than for header processing. This is due to the fact that payload processing actually touches every byte of the packet payload (e.g., transcoding, encryption).

The aggregate cache miss rates for instruction and data cache are shown in Fig. 4. Both workloads achieve instruction miss rates below 1 percent for cache sizes of above 8kB. The data cache miss rates for the workloads drops below 2 percent for 32kB.

4.2 Power Parameters

Most of the power model parameters are based on usage of the Wattch toolkit [6] and CACTI [28], [7]. These values correspond to the use of an Alpha 21264 [29] processor and a .35μm technology. Since we are primarily interested in comparative NP configurations and what they illustrate about NP design, smaller feature size technologies are not initially considered. However, the analytic models presented apply, with parameter value adjustments, to other technologies (e.g., .18μm and $V_{dd} = 2.0$V).

Using Wattch, the capacitance of the ALU simulated by Wattch can be obtained as $C_{ALU} = 310$pF. V_{dd} for this case is 2.5 volts. Similarly, the capacitance of the register file is $C_{reg} = 142$pF (per thread). The cache access energy, E_{c_i} and E_{c_d}, is shown in Table 3. These numbers are given by the CACTI tool [7] for .35μm technology. The cache line size is 32 bytes and associativity level is 2. For instruction caches, one read/write port and one read port are assumed. For data caches, two read/write ports are assumed. Cache sizes up to



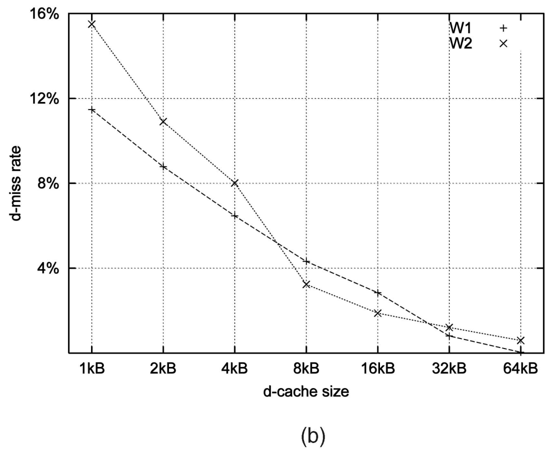


Fig. 4. Aggregate cache performance of workloads. Cache sizes up to 1MB are considered, but only 1kB to 64kB are shown. (a) Instruction cache. (b) Data cache.

1MB are considered, but only 1kB to 64kB are shown. The capacitance parameter associated with using .35μm technology is obtained from scaling the capacitance associated with Wattch's "result bus," yielding $C_{.35\mu m} = 0.275\text{fF}/\mu\text{m}$. The processor clock frequency is $clk_p = 600\text{MHz}$.

4.3 Area Parameters

The area required for a processor core is taken to be $A_p = 3\text{mm}^2$ with each register bank for a thread adding $A_t = 0.5\text{mm}^2$. The base area for a memory or I/O channel is $A_{mchl} = 20\text{mm}^2$ plus an area of $A_{pin} = 0.25\text{mm}^2$ for each pin pad. Area parameters for caches are obtained from CACTI.

The length of the memory channel depends on the number of processors that share the memory channel and the size of each processor and its caches. The length of the memory channel is taken to be $length_{mchl} = 5\text{mm}$, which is the expected distance to a processor from the edge of a chip. We also explored a larger channel length of 20mm. This, however, only affects the overall results shown below by about 1 percent. The width is set to 32 bits.

4.4 Validation

To compare the validity of the above power model, the energy results obtained with Wattch are compared with the model results. In the validation experiment, all applications in the benchmark were executed for cache configurations ranging from 1kB to 64kB. Fig. 5 shows the Wattch results versus the model results. Ideally, each cross point would lie on the dashed line which corresponds to the model and Wattch having the same results. It should be noted that Wattch simulates a complex superscalar processor. To make a reasonable comparison to the RISC core that we are modeling, only the ALU, clock, and cache access power from Wattch was considered. Since there is no shared memory bus modeled in Wattch, we cannot compare the results for this component.

The maximum error is 15.8 percent for the smallest cache size. This is due to differences in the results from the CACTI toolkit versus the Wattch toolkit. For larger caches, the differences are much smaller. With an average error of only 8 percent, the analytic approximation of power consumption is a useful tool for NP design space exploration.

5 Results

The design space has been exhaustively explored over a wide range of parameter configurations (number of threads, cache sizes, memory channel load, etc.). The simplicity of analytic performance modeling makes this approach feasible and the results are presented in the following sections.

5.1 Trends of Metrics

Fig. 6 illustrates the basic trends for the components of the performance metrics from (29). To illustrate basic trends,

TABLE 3
Cache Access Energy and Area for .35 μm Technology

Cache size in kB	i-cache access energy in nJ	d-cache access energy in nJ	i-cache area in mm^2	d-cache area in mm^2
1	2.314	2.372	3.49	4.45
2	2.479	2.538	4.13	5.20
4	2.755	2.815	5.72	7.02
8	3.417	3.575	8.97	10.86
16	4.271	4.629	17.62	18.99
32	6.011	6.441	31.22	36.61
64	8.465	8.741	63.06	72.87

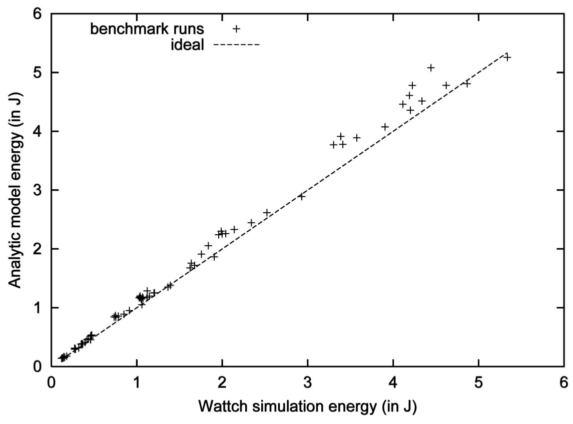


Fig. 5. Comparison of benchmark application execution energy for wattch toolkit and analytic model.

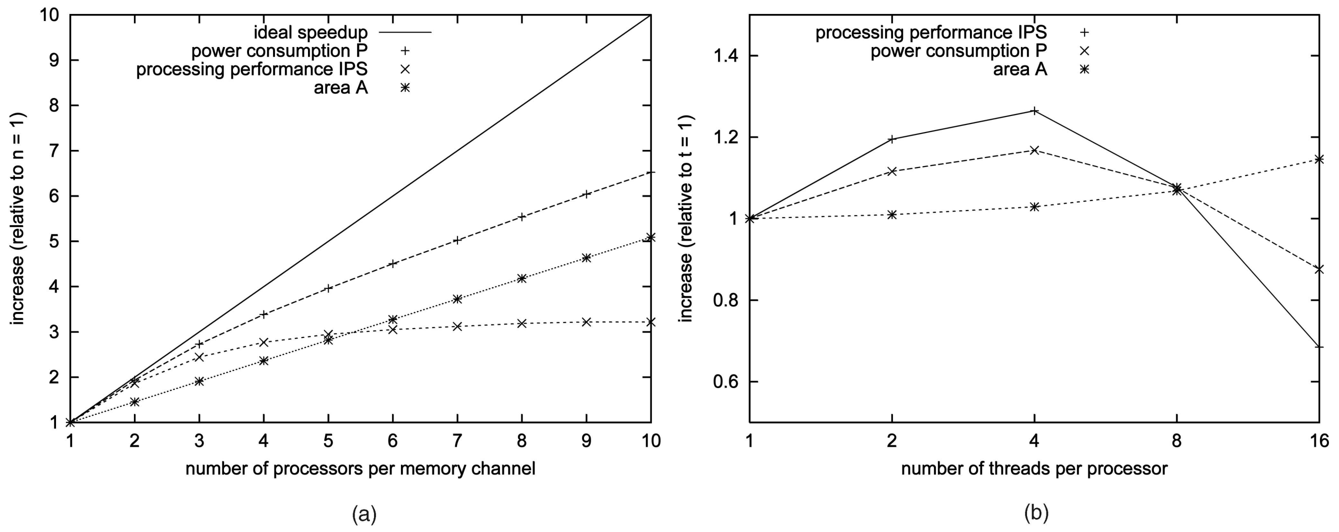


Fig. 6. Trends of processing performance, area, and power. The workload is W1 and cache sizes are c_i=8kB and c_d=8kB. The trend for a different number of processors per memory channel is shown in (a), and the trend for different numbers of threads is shown in (b).

the cache sizes in this figure are set to 8kB for both the instruction and data caches. The number of processors, n, (Fig. 6a) that share a memory channel (i.e., processors in a cluster) and the number of threads per processors, t, (Fig. 6b) are shown on the x-axis. The y-axis shows the increase in processing performance, power consumption, and area relative to a configuration with a single processor ($n = 1$) or single thread ($t = 1$).

As expected, the area cost A increases linearly with the number of processors (the offset and slope depend on the proportion of processor and cache sizes to the memory channel). The instructions per second curve, IPS, initially increases more rapidly than A, however levels out after six processors. This is due to increasing contention on the shared memory channel and an increase in the memory channel load, ρ_{mchl}. At saturation, the memory system responds to requests at its maximum rate and, hence, the IPS remains steady. The trends in Fig. 6a show that power consumption grows faster than other metrics. This is related to memory channel contention. If more processors share a memory channel, the processor stall time on a cache miss increases. During stall times, the processor does no useful computation, but the memory interface consumes energy. As a result, the total processing performance does not increase very much, but power consumption does. These trends are very similar for all cache configurations. The plateaus for processing performance are higher for larger caches since miss rates are lower and, thereby, contention on the memory channel is less. In all cases, however, power consumption grows faster than processing performance.

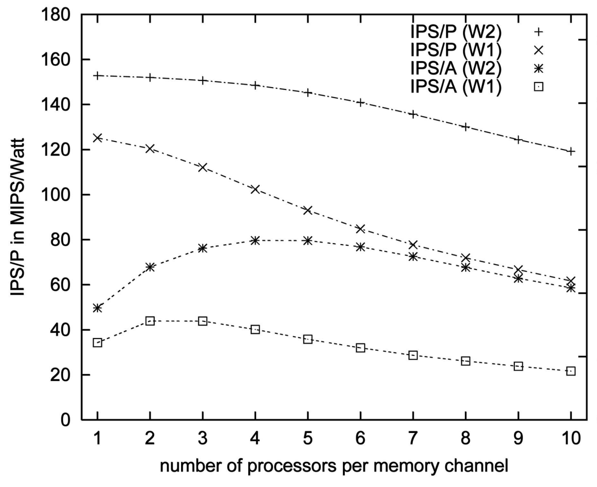
For an increasing number of threads, processing performance and power consumption at first increase and then drop. The increase in processing performance and power consumption is due to a higher utilization of the processor, which peaks in this example for $t = 4$ as the processor is fully utilized. The processing performance decrease for a larger number of threads is due to cache pollution. Since threads compete for cache memory, the effective memory per thread decreases (see (10)). As a result, there are more cache misses and this leads to more memory accesses and lower processor utilization. Note that, due to processor idling, processing power also decreases. The area increase is continuous and due to the additional area required for thread states. This indicates that relatively low multithreading levels ($t = 2$ or $t = 4$) can significantly improve processor utilization while requiring only a small amount of additional area. For configurations with large numbers of threads, it is important to have enough memory available to avoid the processing performance decrease from cache pollution.

The trends of combined metrics, processing per power and processing per area as defined in (29), are shown in Fig. 7. From Fig. 6a, we see that with more processors, power increases faster than processing performance. Thus, the combined processing and power metric drops with higher number of processors. This means that, from the point of view of power consumption, fewer processors per memory channel are preferable. Looking at the impact of area in Fig. 7a, however, fewer processors are not necessarily best. There is a clear optimum for three (workload W1) or five (workload W2) processors. The differences between the workloads are due to different cache miss rates. For different numbers of threads (Fig. 7b), the processing performance per power shows little difference across the number of threads as processing power and power consumption show similar trends in Fig. 6b. For the metric that considers area, there is a peak at about two to four threads, where good processor utilization is achieved with little area cost for thread contexts.

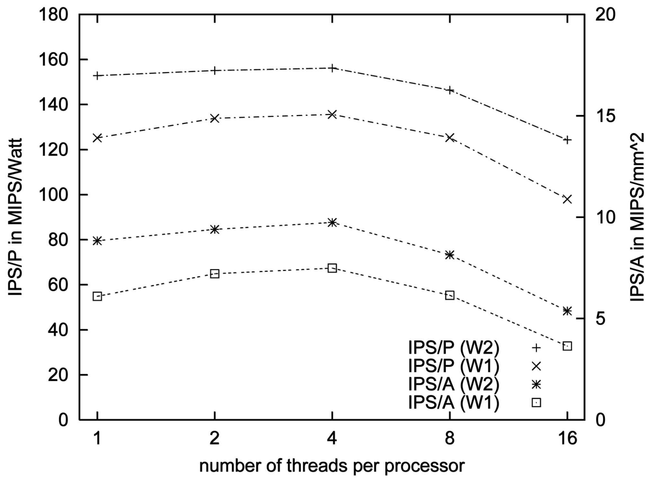
5.2 Impact of Design Parameters

The following results show "slices" through the design space to explore the impact of a particular design parameter. One key question for system-on-a-chip design is how to find a good balance between processing logic and on-chip memory. NP designs are constrained by the maximum chip size. More processing engines mean more processing cycles, but also smaller caches, higher cache miss rates, more memory contention, and higher energy consumption. Using our model, we can find the optimal cache configuration for a given metric. Figs. 8a, 8b, 8c, and 8d show the performance of various cache configurations for the different performance metrics.

The most important observation in Figs. 8a, 8b, 8c, and 8d is that the performance of an NP system is extremely sensitive to the cache size. A change in cache size by a factor

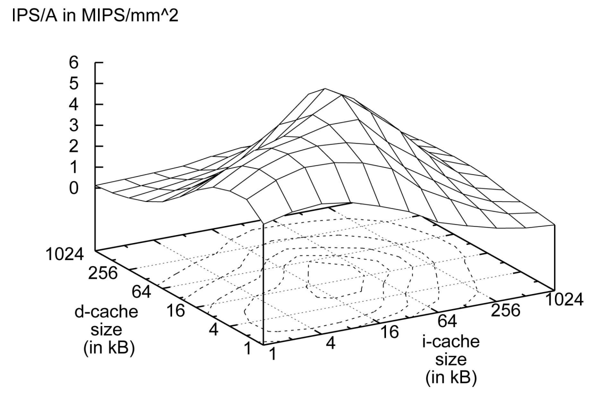


(a)

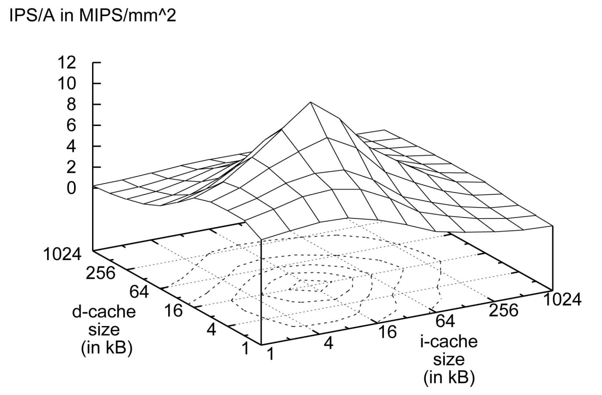


(b)

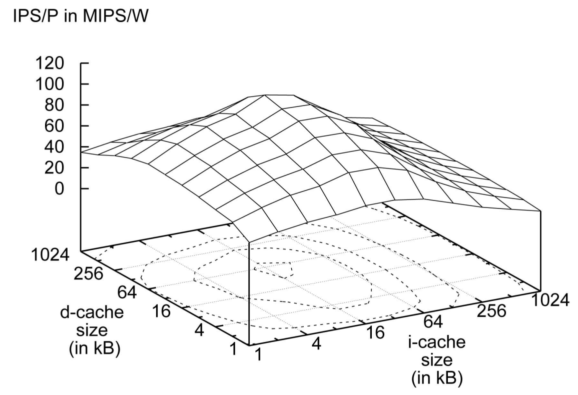
Fig. 7. Trends of combined metrics. The workload is W1 and cache sizes are c_i=8kB and c_d=8kB. The trend for a different number of processors per memory channel is shown in (a), and the trend for different numbers of threads is shown in (b).



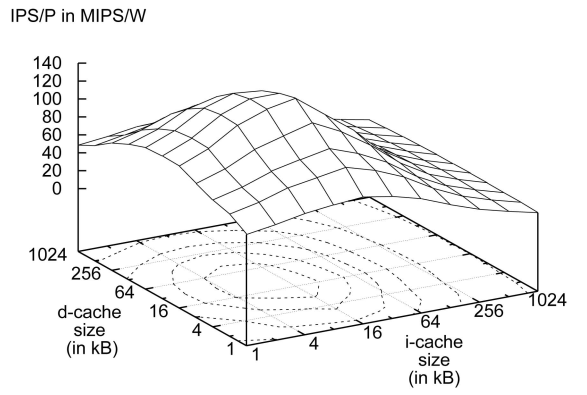
(a)



(b)



(c)



(d)

Fig. 8. Performance of cache configurations for various performance metrics and workloads. The number of processors per memory channel is set to four, and the number of threads per processor is two. (a) IPS/A (W1). (b) IPS/A (W2). (c) IPS/P (W1). (d) IPS/P (W2).

two to four from the optimum configuration can cut the performance in half.

When considering processing per area, the optimal value of the instruction cache is 32kB for workload W1 and 16kB for workload W2. With two threads, this yields an effective instruction cache size of 16kB and 8kB per thread. The cache miss rates for this size instruction cache are very small (less than 1 percent, see Fig. 4). The optimal data cache size is 16kB for both workloads. Even though this cache size leads to more data cache misses (around 4 percent), the absolute number of data cache misses is less than the number of instruction cache misses because only one of four instruction is a load/store operation. This shows that the optimal configuration balances the ratio of instruction and data misses.

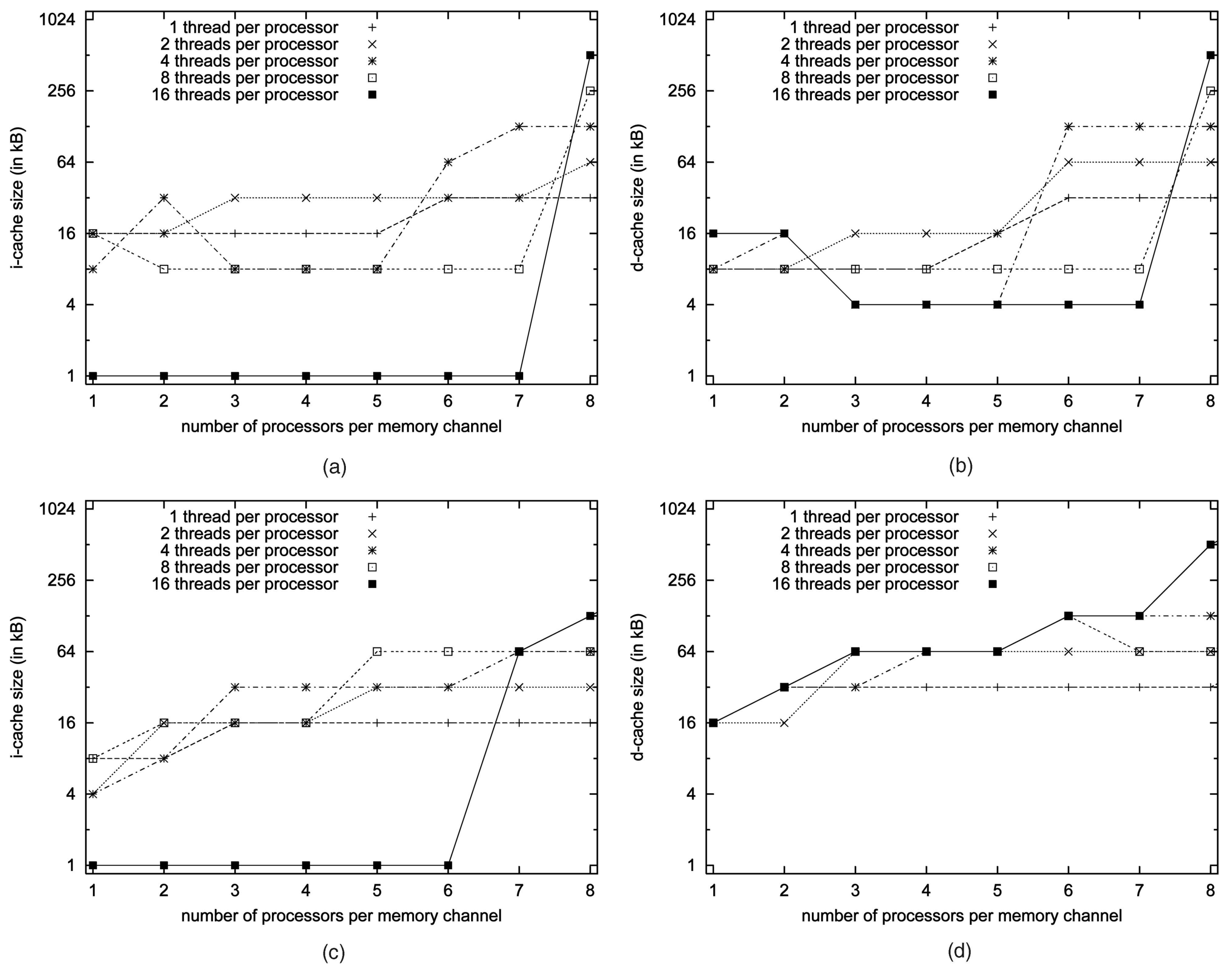


Fig. 9. Optimal instruction and data cache configuration for different numbers of threads per processor and different numbers of processors per memory channel. The workload is W1. (a) Optimal instruction cache size (IPS/A). (b) Optimal data cache size (IPS/A). (c) Optimal instruction cache size (IPS/P). (d) Optimal data cache size (IPS/P).

When considering processing per power (i.e., MIPS per Watt), the optimization results in a configuration with 16kB instruction caches for both workloads and a 64kB data cache for workload W1 and a 16kB data cache for workload W2. The increase in the data cache for workload W1 is due to the significant drop of data miss rates at this size. Even though the power cost of a cache does depend somewhat on the size of the cache, the benefit of fewer stalls yield this solution.

While the trends shown in these optimization results apply generally to all configurations, the optimal cache sizes are different depending on other system parameters. Figs. 9a, 9b, 9c, and 9d show the optimal configurations for different numbers of processors per memory channel and threads per processor. One very interesting observation is the distinct jump in cache size for configurations with many threads when increasing the number of processors (e.g., for $t = 16$ and IPS/A, $ci_{opt} = 1$kB for $n = 1 \dots 7$ and $ci_{opt} = 512$kB for $n = 8$). Small cache configurations incur a large number of cache misses, especially when many threads share the same memory. At the same time, the latency hiding due to multithreading manages to avoid processor stalls. However, when the number of processors on one memory interface exceeds a certain limit, the queuing delay increases the memory access times by so much that this approach does not work anymore. Instead, larger caches are used to reduce the number of cache misses. For configurations with fewer threads, this transition is smoother. Even when the queuing delay is minimal (few processors per memory channel), the limited number of threads cannot hide the latency completely and, thus, larger caches are better for such configurations.

5.3 Optimal Configuration

With an understanding of the different system design trade offs, we now focus on the overall optimal configuration results.

5.3.1 *Single Cluster Optimization*

Table 4 shows optimal cluster configurations for both workloads and the IPS/A and IPS/P metric. There is no optimal result for the IPS metric since it would be the configuration with maximum size caches, threads, etc. in the design space.

For the IPS/A metric, the optimal configuration uses a small number of processors ($n = 3, 4$) with a single thread. The single thread ensures that there is no cache pollution occurring on the caches of size $ci = 16$kB, 8kB, and

TABLE 4
Optimal Cluster Configurations for Different Optimization Metrics

Workload	metric	cluster configuration				IPS	A	P	IPS/A	IPS/P
		n	ci	cd	t	in MIPS	in mm^2	in W	in MIPS/mm^2	in MIPS/W
W1	IPS/A	3	16kB	8kB	1	1005	124	8.59	8.11	117
	IPS/P	1	1kB	16kB	16	350	61.5	2.48	5.69	141
W2	IPS/A	4	8kB	8kB	1	1719	121	11.6	14.2	149
	IPS/P	1	4kB	16kB	2	492	56.7	2.97	8.68	166

TABLE 5
Optimal System Configurations for Different Optimization Metrics

Workload	metric	system configuration					IPS	A	P	throughput
		m	n	ci	cd	t	in MIPS	in mm^2	in W	in Gbps
W1	IPS/A	3	3	16kB	8kB	1	3015	371	25.7	6.40
	IPS/P	6	1	1kB	16kB	16	2099	369	14.9	4.45
W2	IPS/A	3	4	8kB	8kB	1	5155	364	34.7	0.203
	IPS/P	7	1	4kB	16kB	2	3445	397	20.7	0.135

$cd = 8$kB. Due to the low miss rates at this cache size, multiple processors can share the memory channel without causing too much contention.

For the IPS/P metric, the cluster configuration is scaled in a different dimension. Instead of having multiple processors share one memory channel, a single processor is used with a larger number of threads $t = 16$ and $t = 2$. More processors per memory interface increases the relative power consumption for the network processors. This comes from power dissipation that occurs during stall cycles while waiting for memory access. Thus, the best configuration is a single processor per memory interface with a larger number of threads to ensure full utilization. This reduces the overall power consumption to about one quarter of that of the IPS/A optimized cluster. The area is however only reduced to one half.

These results indicate that the different optimization goals (area versus power) lead to different system designs. In practice, both area and power need to be considered and, thus, the best configuration lies between the two extremes presented here. Using our performance model, such intermediate results can be derived by using, for example, the $IPS \cdot A^{-\frac{1}{2}} \cdot P^{-\frac{1}{2}}$ metrics, which combine both optimization goals.

5.3.2 NP System Optimization

The results presented above focus on optimal cluster configurations. Table 5 shows optimal configurations that focus on the entire chip where there is a chip area constraint $A \leq 400$mm^2. The table shows the optimal configurations in terms of number of memory interfaces, m, and processors per memory channel, n. For all metrics and workloads, the overall throughput of such a system is also shown, which is determined by the complexity of the workload and the overall processing power ($IPS/compl$). Note that the complexity for workload W2 is about 50 times higher than that of W1, which results in the large differences in throughput. Even within a workload, there are significant differences between applications. For workload W1 and IPS/A, the throughput varies from 0.299 Gbps for the TCP application to 19.3 Gbps for the DRR application. While header processing applications can achieve throughput rates of several gigabits per second, payload processing applications have rates well under that for all performance metrics. This is consistent with the notion that these types of applications (e.g., encryption) often require special purpose coprocessors to achieve high throughput rates.

For power-related metrics, the trends in Fig. 6 result in optimal configurations with only one processor per interface. This, however, yields a lower throughput than when optimizing for area only. On the other hand, power consumption for the area-optimized configuration is about twice as high as that for power-optimized configurations. The overall power consumption of the optimal configurations, 15 to 35 watts, is higher than current commercial systems, which consume on the order of 10W. This is due to commercial NPs using more advanced CMOS technologies with smaller feature and overall chip sizes (e.g., Intel IXP2400: $.18\mu$m versus $.35\mu$m and 1.3V versus 2.5V [1]). The model and optimization results can be adapted to other CMOS technologies by using different power models (e.g., one that consider leakage current for $.18\mu$m CMOS and below).

5.4 General System Design Results

From the above results, there are several general "rules of thumb" for network processor system design that can be derived. These are:

- The shared memory interface poses a significant bottleneck when increasing the number of processors per interface (Fig. 6a). To achieve scalability, larger instruction and data caches are necessary to reduce the number of off-chip memory accesses per processor.
- System performance is very sensitive to caches configurations (Fig. 8). Small caches can lead to memory system bottlenecks; large caches use area that could be used more effectively for additional processors.
- Multithreaded configurations with two or four threads per processor are satisfactory (Fig. 6b). They achieve better processor utilization at only slightly higher area and power consumption. Configurations with more threads are limited in processing performance due to cache pollution. Some current NPs have higher multithreading levels, however, this

may be appropriate with their particular technology parameters.

- The system configurations suggest that an optimal NP design involves several memory channels, uses small number of processors per memory channels, and implements a small level of multithreading (Table 5). The cache configurations are small and do not exceed 16kB for any cache.

These general, qualitative observations can be explored in a quantitative way using our analytic performance model as shown above. Further, it is possible to parameterize the model for other CMOS technology generations. This makes our approach a useful tool for projecting design trade offs for future network processor systems.

6 Summary and Conclusions

In this work, we have presented a methodology to explore the design space of network processors using analytic performance modeling that is parameterized by realistic workload and technology parameters. The performance model considers the key aspects of a highly parallel NP system: multithreaded processors, on-chip memory, and shared memory and I/O channel for off-chip communication. The quantitative results obtained through the model provide an in-depth understanding of the design trade offs and the impact of different system components on the overall performance. To derive realistic results, the model was parameterized with workload parameters that were obtained from our network processing benchmark. The technology parameters were obtained from state-of-the-art power and area estimation tools. The detailed consideration of workload, power, and area parameters in our performance model ensures that it can be used for future generations of NP workloads and CMOS technologies.

Appendix

Benchmark Applications

The header-processing applications represent operations that are done on a per-packet basis and are mainly independent of the size and type of the packet payload. These applications involve a good deal of "random" logic, header field interrogation and processing, table lookup, and control. We have selected the public domain programs listed below, which are likely to be operationally similar to proprietary programs. The "kernel" of each application is the part of the code that contributes the majority of dynamic instructions.

- *RTR* is a Radix-Tree Routing table lookup program. Routing table lookups are important operations performed on every packet in a datagram-based network and on every connection in a connection-based network. *RTR* is the radix-tree routing algorithm from the public domain NetBSD distribution. *Kernel*: lookup operations on tree data structure.
- *FRAG* is a IP packet fragmentation application. IP packets are split into multiple fragments for which some header fields have to be adjusted and a header checksum computed. The checksum computation that dominates this application is performed as part of all IP packet application programs other than just forwarding. *Kernel*: packet header modifications and checksum computation.
- *DRR* is a Deficit Round Robin fair scheduling algorithm [30] that is commonly used for bandwidth scheduling on network links. The algorithm is implemented in one form or another in various switches currently available (e.g., Cisco 12000 series). *Kernel*: queue maintenance and packet scheduling for fair resource utilization.
- *TCP* is a TCP traffic monitoring application that is representative of the class of monitoring and management applications. We use *tcpdump*, a widely used tool, that is standard in BSD distributions and is based on the BSD packet filter [31]. *Kernel*: pattern-matching on header data fields.

Payload-processing applications access and possibly modify the contents of a packet during network node processing. The applications are typically executed on a stream of packets. Note that each of these applications has an encoding and a decoding section. While each of these sections is executed separately, they are considered together as a single program unless they have significantly different processing performance characteristics.

- *CAST* is a program based on the CAST-128 block cipher algorithm that uses a 128 bit key to encrypt data for secure transmission [32]. CAST-128 operates similar to other block cipher algorithms used in current networks, such as IDEA and RC5, however, CAST is in the public domain. *Kernel*: encryption arithmetic.
- *ZIP* is a data compression program based on the commonly used Lempel-Ziv (LZ77) algorithm [33]. The implementation can achieve different levels of data compression by varying the algorithm's computational complexity and exemplifies applications that permit trade offs between computational power and bandwidth. *Kernel*: data compression.
- *REED* is an implementation of the Reed-Solomon Forward Error Correction scheme that adds redundancy to data to allow recovery from transmission errors [34]. This is commonly used on unreliable data links which can be found in wireless networks. *Kernel*: redundancy coding.
- *JPEG* is a lossy compression algorithm [35] for image data. It represents the class of media transcoding applications. *Kernel*: discrete cosine transform (DCT) and Huffmann coding.

Acknowledgments

This research was supported in part by the US National Science Foundation grants CCR-0217334 and CCF-0427794.

References

[1] Intel Corp., *Intel IXP2800 Network Processor,* 2002, http://developer.intel.com/design/network/products/npfamily/ixp2800.htm.

[2] J. Allen, B. Bass, C. Basso, R. Boivie, J. Calvignac, G. Davis, L. Frelechoux, M. Heddes, A. Herkersdorf, A. Kind, J. Logan, M. Peyravian, M. Rinaldi, R. Sabhikhi, M. Siegel, and M. Waldvogel, "IBM PowerNP Network Processor: Hardware, Software, and Applications," *IBM J. Research and Development,* vol. 47, nos. 2/3, pp. 177-194, 2003.

[3] AMCC, *np7510 10 Gbps Network Processor,* 2003, http://www.amcc.com.

[4] EZchip Technologies Ltd., Yokneam, Israel, *NP-1 10-Gigabit 7-Layer Network Processor,* 2002, http://www.ezchip.com/html/pr_np-1.html.

[5] D. Burger and T.M. Austin, "The SimpleScalar Tool Set, Version 2.0," Technical Report 1342, Dept. of Computer Science, Univ. of Wisconsin in Madison, June 1997.

[6] D. Brooks, V. Tiwari, and M. Martonosi, "Wattch: A Framework for Architectural-Level Power Analysis and Optimizations," *Proc. ACM Int'l Symp. Computer Architecture,* pp. 83-94, June 2000.

[7] P. Shivakumar and N.P. Jouppi, "CACTI 3.0: An Integrated Cache Timing, Power and Area Model," Technical Report WRL Research Report 2001/2, Palo Alto, Calif.: Western Research Laboratory, Aug. 2001.

[8] T. Wolf and M.A. Franklin, "CommBench—A Telecommunications Benchmark for Network Processors," *Proc. IEEE Int'l Symp. Performance Analysis of Systems and Software (ISPASS),* pp. 154-162, Apr. 2000.

[9] J.C. Mogul, "Simple and Flexible Datagram Access Controls for UNIX-Based Gateways," *USENIX Conf. Proc.,* pp. 203-221, June 1989.

[10] K.B. Egevang and P. Francis, "The IP Network Address Translator (NAT)," RFC 1631, Network Working Group, May 1994.

[11] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha, "Design, Implementation and Performance of a Content-Based Switch," *Proc. IEEE INFOCOM 2000,* pp. 1117-1126, Mar. 2000.

[12] A.S. Snoeren, C. Partridge, L.A. Sanchez, C.E. Jones, F. Tchakountio, S.T. Kent, and W.T. Strayer, "Hash-Based IP Traceback," *Proc. ACM SIGCOMM 2001,* pp. 3-14, Aug. 2001.

[13] G. Kane and J. Heinrich, *MIPS RISC Architecture.* Prentice Hall, Sept. 1991.

[14] Triscend Corp., *Triscend E5 Configurable System-on-Chip Family,* 1999, http://www.triscend.com/products/dse5csoc.pdf.

[15] Tensilica, Inc., *Application Specific Microprocessor Solutions—Data Sheet for Xtensa V1,* http://www.tensilica.com/datasheet.pdf, 1998.

[16] P. Crowley, M.E. Fiuczynski, J.-L. Baer, and B.N. Bershad, "Characterizing Processor Architectures for Programmable Network Interfaces," *Proc. 2000 Int'l Conf. Supercomputing,* pp. 54-65, May 2000.

[17] P. Crowley and J.-L. Baer, "A Modelling Framework for Network Processor Systems," *Proc. First Network Processor Workshop (NP-1) in Conjunction with Eighth Int'l Symp. High Performance Computer Architecture (HPCA-8),* pp. 86-96, Feb. 2002.

[18] L. Thiele, S. Chakraborty, M. Gries, and S. Künzli, "Design Space Exploration of Network Processor Architectures," *Proc. First Network Processor Workshop (NP-1) in Conjunction with Eighth Int'l Symp. High Performance Computer Architecture (HPCA-8),* pp. 30-41, Feb. 2002.

[19] N. Weng and T. Wolf, "Profiling and Mapping of Parallel Workloads on Network Processors," *Proc. 20th Ann. ACM Symp. Applied Computing (SAC),* pp. 890-896, Mar. 2005.

[20] N. Weng and T. Wolf, "Pipelining versus Multiprocessors —Choosing the Right Network Processor System Topology," *Proc. Advanced Networking and Comm. Hardware Workshop (ANCHOR 2004) in Conjunction with the 31st Ann. Int'l Symp. Computer Architecture (ISCA 2004),* June 2004.

[21] Standard Performance Evaluation Corp., *SPEC CPU2000—Version 1.2,* Dec. 2001.

[22] G. Memik, W.H. Mangione-Smith, and W. Hu, "NetBench: A Benchmarking Suite for Network Processors," *Proc. Int'l Conf. Computer-Aided Design,* pp. 39-42, Nov. 2001.

[23] B.K. Lee and L.K. John, "NpBench: A Benchmark Suite for Control Plane and Data Plane Applications for Network Processors," *Proc. IEEE Int'l Conf. Computer Design (ICCD '03),* pp. 226-233, Oct. 2003.

[24] "Embedded Microprocessor Benchmark Consortium," http://www.eembc.org, 2006.

[25] A. Agarwal, "Performance Tradeoffs in Multithreaded Processors," *IEEE Trans. Parallel and Distributed Systems,* vol. 3, no. 5, pp. 525-539, Sept. 1992.

[26] J.L. Hennessy and D.A. Patterson, *Computer Architecture—A Quantitative Approach,* second ed., San Mateo, Calif.: Morgan Kaufmann Publishers, Inc., 1995.

[27] D. Burger and T. Austin, "The SimpleScalar Tool Set Version 2.0," *Computer Architecture News,* vol. 25, no. 3, pp. 13-25, June 1997.

[28] G. Reinman and N.P. Jouppi, "CACTI 2.0: An Integrated Cache Timing and Power Model," Technical Report WRL Research Report 2000/7, Palo Alto, Calif.: Western Research Laboratory, Feb. 2000.

[29] M.K. Gowan, L.L. Biro, and D.B. Jackson, "Power Considerations in the Design of the Alpha 21264 Microprocessor," *Proc. 35th Design Automation Conf.,* pp. 726-731, June 1998.

[30] M. Shreedhar and G. Varghese, "Efficient Fair Queuing Using Deficit Round Robin," *Proc. ACM SIGCOMM '95,* pp. 231-242, Aug. 1995.

[31] S. McCanne and V. Jacobson, "The BSD Packet Filter: A New Architecture for User-Level Packet Capture," *Proc. USENIX Technical Conf.,* pp. 259-270, Jan. 1993.

[32] C. Adams, "The CAST-128 Encryption Algorithm," RFC 2144, Network Working Group, May 1997.

[33] J. Ziv and A. Lempel, "A Universal Algorithm for Sequential Data Compression," *IEEE Trans. Information Theory,* vol. 23, no. 3, pp. 337-342, May 1977.

[34] T.R.N. Rao and E. Fujiwara, *Error-Control Coding for Computer Systems.* Englewood Cliffs, N.J.: Prentice Hall, 1989.

[35] G.K. Wallace, "The JPEG Still Picture Compression Standard," *Comm. ACM,* vol. 34, no. 4, pp. 30-44, Apr. 1991.

Tilman Wolf received the Diplom in informatics from the University of Stuttgart, Germany, in 1998. He also received the MS degree in computer science in 1998, the MS degree in computer engineering in 2000, and the DSc degree in computer science in 2002, all from Washington University in St. Louis. Currently, he is an assistant professor in the Department of Electrical and Computer Engineering at the University of Massachusetts, Amherst. He is engaged in research and teaching in the areas of computer networks, computer architecture, and embedded systems. His research interests include the design and implementation of programmable routers, network processors, and their application to next-generation network systems. Dr. Wolf is a member of the IEEE and the ACM. He has been active as a program committee member and organizing committee member of several professional conferences, including IEEE INFOCOM and ACM SIGCOMM. He is currently serving as treasurer for the ACM SIGCOMM Society.

Mark A. Franklin received the BA, BSEE, and MSEE degrees from Columbia University, and the PhD degree in electrical engineering from Carnegie-Mellon University. He is currently a professor in the Department of Computer Science and Engineering at Washington University in St. Louis and holds the Hugo F. and Ina Champ Urbauer Chair in engineering. He founded and is former director of the Computer and Communications Research Center. Dr. Franklin is a fellow of the IEEE and the IEEE Computer Society, and a member of the ACM. He has been chair of the IEEE TCCA (Technical Committee on Computer Architecture) and vice-chair of the ACM SIGARCH (Special Interest Group on Computer Architecture). He is one of the founders of the Workshop on Network Processors and the Symposium on Architectures For Networking and Communications Systems. His research areas include computer and systems architecture, chip multiprocessors, embedded processor design, and systems performance evaluation.