A Network Processor Performance and Design Model with Benchmark Parameterization

Mark A. Franklin and Tilman Wolf
Departments of Computer Science and Electrical Engineering
Washington University in St. Louis, MO, USA
{jbf,wolf}@ccrc.wustl.edu

Abstract

Network processors have become central elements in the design of modern routers. With higher line speeds and more demanding functional requirements, design of network processors has increased in difficulty. In this paper an analytic performance model for a multiprocessor-based single chip network processor is presented and used to aid in the design process. The model contains a number of workload parameters that have been obtained from a benchmark that reflects typical processing characteristics associated with packet processing. The system parameters include the number of processors per chip, the processor multithreading level, the sizes of on-chip data and instruction caches, and the number of required off-chip memory channels. Using the presented model, an optimal design can be obtained that maximizes the overall processing power per chip area. The presented results give an understanding of how to configure a network processor optimally and what effect changes in system parameters have on the overall performance.

1 Introduction

Two design considerations are important for contemporary network processors (NPs). One is the flexibility to adapt to new functional requirements; another is the ability to provide scalable performance in response to increasing line rates. Such requirements can lead to a host of potential architectures as can be seen when examining the multitude of commercial designs available.

While commercial designs have a number of common characteristics, we focus on three of these. First, to deal with high line rates, NPs employ parallel processors. Network workloads inherently lend themselves to high levels of parallelism due to independence between different packet flows. Second, to reduce the latency effects of off-chip instruction and data access, the processors employ on-chip caches and multithreading techniques. Third, various state information (e.g., routing tables) and packet data is stored in separate off-chip memories, which require an on-chip memory interface.

In this paper, we develop a performance model to quantify the design alternatives associated with these three architectural elements and optimize the design to maximize overall processing power per area. This represents a starting point for developing a coherent approach and theory of NP architecture design.

The idealized single-chip NP architecture that is used in our work is shown in Figure 1. It contains a number of identical multithreaded general-purpose processors, 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 achieving speedup by exploiting parallelism. Thus, after assignment of a flow to a processor, all packets of the same flow are routed to the same processor. More detail on the architecture can be found in [7]. Packet scheduling issues in this environment are considered in [6].

2 The Performance Model

The system parameters used in the performance model are listed in Table 1. The entire system has m clusters with n RISC processors in each cluster.

Each cluster has a single memory interface with an area (in units of mm^2) of $s(mchl)$ and the entire chip has a single I/O interface with an area of $s(io)$. Each processor has an area of $s(p_j, k)$ and has its own instruction and data caches of size c_i and c_d bytes and chip areas of $s(c_i)$ and $s(c_d)$ respectively.

Each cache is 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. This means that if one thread stalls on a memory miss, another thread can immediately start processing with no cycle delay. The processor is a typical RISC processor that ideally executes one instruction per cycle when no hazards are present. We also assume that the on-chip SRAM cache can be accessed in a single cycle.

The goal of our work is to find the "optimal" configuration of a network processor for a given workload. Optimal, in this context, means obtaining the maximum processing power per chip area. In the remainder of the paper we develop analytic expressions for the processing power, IPS, (Instructions Per Second) and the area, $area$, associated with a given architecture configuration (e.g., number of processors, sizes of caches, etc.). From these expressions we can obtain $IPS/area$ and find its maximum as a function of the various configuration parameters, thus developing an "optimal" architecture. In the remainder of this section, we discuss obtaining IPS and $area$ in terms of system and workload characteristics.

2.1 Processing Performance

For a single processor, processing power can be expressed as the product of the processor's utilization, ρ_p, and its clock rate, clk_p. The processing power of the entire 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:

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

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

$$IPS = m \cdot n \cdot \rho_p \cdot clk_p. \quad (2)$$

A key question is how to determine the utilization of the processors. In the extreme case where there are a large number of threads per processor, 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. Our goal is to find the optimal configuration of these parameters in terms of processing power per chip area. Thus, we need to develop a cost function for different configurations that reflect the required chip area.

2.2 Chip Area

The on-chip area equation for an NP configuration in our general architecture is:

$$area_{NP} = s(io) + \sum_{j=1}^{m}(s(mchl)+ \sum_{k=1}^{n}(s(p_{j,k}, t) + s(c_{i_{j,k}}) + s(c_{d_{j,k}}))). \quad (3)$$

This is the summation over all the system component areas shown in Figure 1. With identical processor configurations, this can be simplified to:

$$area_{NP} = s(io) + m \cdot (s(mchl)+ n \cdot (s(p, t) + s(c_i) + s(c_d))). \quad (4)$$

The processor size, $s(p, t)$, depends on the number of hardware threads and is therefore expressed as $s(p, t)$, a function of t. We model the processor size in terms of two components. The first component, size $s(p_{basis})$, is independent of the number of supported threads. It represents the basic processor logic (e.g., ALU, pipeline control, branch prediction, etc.). The second component, size $s(p_{thread})$, relates to logic associated with a thread (e.g., thread context registers, associated logic, etc.). This thread component is modeled as increasing linearly with the number of threads, t. While this might be optimistic for large numbers of threads, it is a reasonable assumption for the relatively small number of threads considered here. Thus, the processor size is:

$$s(p, t) = s(p_{basis}) + t \cdot s(p_{thread}). \quad (5)$$

The size of a memory or I/O bus also consists of a basis area plus the on-chip area of the pin drivers and pads. The total size depends on the width of the bus:

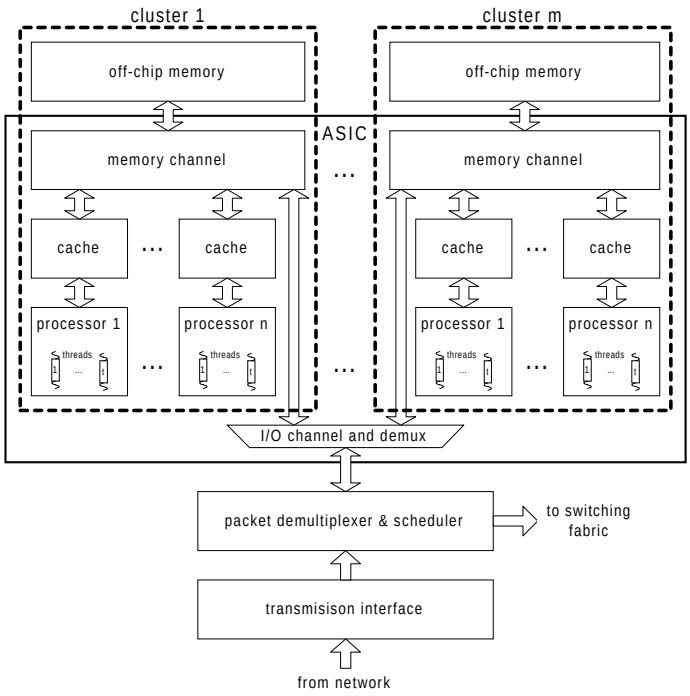

Figure 1. Overall Network Processor Architecture

Component	Symbol	Description
processor	clk_p	processor clock frequency
	t	number of simultaneous 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 rate 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
	$s(x)$	actual size of component x, with $x \in \{ASIC, p, c_i, c_d, io, mchl\}$

Table 1. System Parameters.

$$s(mchl) = s(mchl_{basis}) + width_{mchl} \cdot s(mchl_{pin}). \quad (6)$$

The number of pins depends on the bus clock, clk_{mchl}, and the required bus bandwidth, bw_{mchl}. Thus:

$$s(mchl) = s(mchl_{basis}) + \left\lceil \frac{bw_{mchl}}{clk_{mchl}} \right\rceil \cdot s(mchl_{pin}). \quad (7)$$

with the equivalent equation being used for the I/O channel.

Equation 4 and the subsequent Equations 5-7 define the space of available architecture configurations (e.g., n, m, t, etc.) and determine the network processor chip area. However, before this can be used in the evaluation of the overall performance metric $IPS/area_{NP}$, the processor utilization must be determined so that IPS from Equation 2 can be evaluated. In particular, the processor utilization, ρ_p, depends on the performance of the memory system.

2.3 Memory System

The performance of the network processor is determined by the utilization of the individual processing engines. A processor is fully utilized as long as memory misses do not cause a processor stall. Other stalls due to hazards, such as branch misprediction, are not considered here since, with modern processor and compiler designs, they generally have a relatively small effect compared to the effects of cache misses. Using the model proposed and verified by Agarwal [1], 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:

$$\rho_p(t) = 1 - \frac{1}{\sum_{i=0}^{t} \left(\frac{1}{p_{miss} \cdot \tau_{mem}} \right)^i \frac{t!}{(t-i)!}}. \quad (8)$$

To illustrate the overall trend in this equation, we can simplify Equation 8 by ignoring the second and higher order terms of the summation. Thus:

$$\rho_p(t) = \frac{t}{(t + \tau_{mem} \cdot p_{miss})}. \quad (9)$$

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 $\lim_{t\to\infty} \rho_p(t) = 1$. While it is desirable to run processors at high utilization there is an area cost with this as indicated in Equation 5. This impacts overall performance since the added processor area due to more thread contexts leads to less area available for caches and thus can lead to higher miss rates. On the other hand, more threads can also help mask cache misses and thus can be beneficial. Thus, there is a design tradeoff here, which can be understood more fully only after expressions for the memory access time, τ_{mem}, and the cache miss rate, p_{miss}, are obtained.

2.3.1 Off-Chip Memory Access

We assume the memory channel implements a FIFO service order on the memory requests in such a way that they can be interleaved in a split transaction fashion. 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 represented in terms of numbers of processor clock cycles):

$$\tau_{mem} = \tau_Q + \tau_{DRAM} + \tau_{transmit}. \quad (10)$$

The DRAM access time and the cache line transmission time are straightforward to determine. The queuing time, however, depends on the load on the memory channel, which depends on the number of processors that share the memory channel, the number of threads per processor, and the cache miss rates. This system component can be simply modeled as a single server queuing system with n processors that generate requests. The request distribution can be modeled as geometrically distributed random variables (as suggested in [1]). Based on the average cache miss rate of a thread (see Equation 14 below), the parameter of the geometric random variable is p_{miss}. The number of requests per processor is limited to t, which corresponds to the situation where all the processor threads are stalled and the processor is idle until a memory request is served. The service time for the memory

channel is taken to be deterministic with parameter $1/\tau_{transmit}$.

This model can be slightly modified to make it more suitable for the analytical evaluation. Instead of considering n processor sources each providing up to t requests, we model the system as a single finite source having up to $n \cdot t$ requests. Since each of the n sources generates requests at a mean rate p_{miss}, the single source model generates requests at a rate $n \cdot p_{miss}$.

Assuming an exponential distribution rather than a geometric and ignoring the limit of $n \cdot t$ customers, the queuing system can be approximated by a M/D/1 queuing system. The request rate is $\lambda = n \cdot p_{miss}$ and the deterministic service rate is $\mu = 1/\tau_{transmit}$.

The M/D/1 model is a reasonable approximation to the real system, which has a finite source population. Figure 2 shows the average queue length for the simulated real finite source system and the analytic result for the M/D/1 system. The number of threads in this example is $t = 8$, the number of processors is $n = 4$, and the service time $\tau_{transmit} = 40$. The M/D/1 model has no constraint on the maximum number of requests and therefore reaches a much larger queue length for very high loads (i.e., $> 90\%$). For a more typical load of $\rho_{mchl} = 0.5 \ldots 0.9$, the difference between M/D/1 and the other models is relatively small. Furthermore, below 50% load the queue length is small enough for both models to have relatively little effect on the overall performance. Therefore, we will use the M/D/1 model for an approximation of the queuing time τ_Q.

The bus access time, τ_Q, is then given by the queuing time of the M/D/1 system, which is

$$\tau_Q = \frac{\rho_{mchl}^2}{2(1-\rho_{mchl})} \cdot \frac{linesize}{width_{mchl}} \cdot \frac{clk_p}{clk_{mchl}}. \quad (11)$$

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

$$\tau_{transmit} = \frac{linesize}{width_{mchl}} \cdot \frac{clk_p}{clk_{mchl}}, \quad (12)$$

we can substitute in Equation 10 to obtain the memory access time:

$$\tau_{mem} = \tau_{DRAM} + \left(1 + \frac{\rho_{mchl}^2}{2(1-\rho_{mchl})}\right) \cdot \frac{linesize}{width_{mchl}} \cdot \frac{clk_p}{clk_{mchl}}. \quad (13)$$

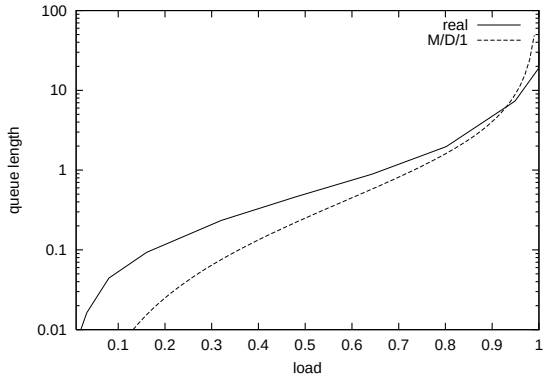

Figure 2. Comparison of Average Memory Queue Length for Different Queuing Models.

2.3.2 On-Chip Cache

The remaining component needed to evaluate the utilization expression (Equation 8) is the cache miss rate p_{miss}. For a simple RISC style load-store processor running application a, the miss probability is given as [4]:

$$p_{miss,a} = mi_{c,a} + (f_{load_a} + f_{store_a}) \cdot md_{c,a}, \quad (14)$$

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 multi-threading.

Cache pollution from multi-threading 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. In steady-state, each thread can use $\frac{1}{t}$ 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, we expect the cache sizes to be smaller than the working set size due to chip area constraints, which leads to equal sharing of the available cache between threads. Thus, the effective cache size that is available to a thread is:

$$c_{i,eff} = \frac{c_i}{t}, \quad c_{d,eff} = \frac{c_d}{t}. \quad (15)$$

The application characteristics that are necessary for evaluating Equation 14 are derived from a communications benchmark that is discussed in Section 3.

2.4 Memory and I/O Channels

The expression for miss rate, p_{miss}, (Equation 14) and for total memory access time, τ_{mem}, (Equation 10) can now be substituted into Equation 8 to obtain processor utilization. In order to do this, we need to fix the memory channel load, ρ_{mchl}, because τ_Q depends on ρ_{mchl}. Thus, with the memory channel load given, we can determine the utilization of a single processor. This gives us the memory bandwidth, $bw_{mchl,1}$, required by a single processor:

$$bw_{mchl,1} = \rho_p \cdot clk_p \cdot linesize \cdot (mi_c + (f_{load} + f_{store}) \cdot md_c \cdot (1 + dirty_c)). \quad (16)$$

In this equation, we have to consider the case where a dirty cache line needs to be written back to memory. The probability of the dirty bit being set on a cache line is $dirty_c$. In Equation 14, considering dirty cache lines was not necessary, since a write-back does not stall the processor. In practice, the write-back only increases the required memory bandwidth slightly and Equation 16 can be approximated by

$$bw^*_{mchl,1} = \rho_p \cdot clk_p \cdot linesize \cdot p_{miss}. \quad (17)$$

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

$$n = \left\lfloor \frac{width_{mchl} \cdot clk_{mchl} \cdot \rho_{mchl}}{bw_{mchl,1}} \right\rfloor. \quad (18)$$

This gives us a complete cluster configuration for all ranges of cache sizes and thread contexts. Finally, we need to determine the bandwidth that is required for the I/O channel. The I/O channel is used to send packets to the processing engines and back out. Thus, each packet traverses the I/O channel twice. From Equation 21 (below), we get a relation between the number of instructions executed in processing a packet and the size of the packet. This "complexity" is a parameter that is characteristic for each application. The I/O channel is operated at a load of ρ_{IO}; thus, the I/O channel bandwidth for the entire network processor is:

$$bw_{IO} = 2 \cdot \frac{IPS}{compl \cdot \rho_{IO}}. \quad (19)$$

Finally, the network processor is limited in the number of pins that the package can have. As a rough estimate, we add the number of pins required by the I/O and memory channels, which depends on their respective width, to the control pins for the network processor:

$$pins_{NP} = pins_{IO} + m \cdot pins_{mchl} + pins_{control}. \quad (20)$$

We can see later that, for our basic architecture, the number of pins that can be supported do not pose a practical limit on the network processor.

2.5 Optimization

With the performance and area of the network processor expressed in terms of cache configurations, application characteristics, and memory channel load, we can find the maximum $IPS/area$. Since the optimization space is discrete (other than the memory channel load) and relatively small, we can do this by exhaustive search.

3 Workload & System Characteristics

Before we can optimize the network processor configuration, we have to define the workload and system parameters.

3.1 Network Processor Workload

To properly evaluate and design network processors it is necessary to specify a workload that is typical of that environment. This has been done in the development of the benchmark CommBench [5]. Applications for CommBench were selected to include a balance between header-processing applications (HPA) and payload-processing applications (PPA). HPAs process only packet headers that generally makes them computationally less demanding than PPAs that process all of the data in a packet. A list of the applications is given in Table 2.

For each application, the following properties have been measured experimentally: computational complexity, load and store instruction frequencies, instruction cache and data cache miss rate, and dirty

HPA	PPA
Deficit round robin	CAST encryption
IP header fragmentation	JPEG transcoding
Radix tree routing	Reed-Solomon FEC
TCP filtering	ZIP compression

Table 2. Benchmark Applications.

bit probability. The complexity of an application can be obtained by measuring the number of instructions that are required to process a packet of a certain length (for header-processing applications, we assumed 64 byte packets):

$$compl = \frac{instructions\ executed}{packet\ size} \qquad (21)$$

The cache properties of the benchmark applications were also measured to obtain $mi_{c,a}$, $md_{c,a}$, and $dirty_{c,a}$. This was done with a processor and cache simulator (Shade [2] and Dinero [3]) and cache sizes ranging from $1kB$ to $1024kB$. A 2-way associative write-back cache with a linesize of 32 bytes was simulated. The cache miss rates were obtained 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.

We aggregate the application parameters from CommBench into two workloads that we consider for the evaluation of our analysis:

- Workload A: Header-processing applications.
- Workload B: Payload-processing applications.

These workloads are such that there is an equal distribution of processing requirements over all applications within each workload. Table 3 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). Header processing typically only reads few header fields and does simple lookup and comparison operations. The aggregate cache miss rates for instruction and data cache are shown in Figure 3. The x-axis corresponds to the effective cache size available to a thread as given in Equation 15. Both workloads achieve instruction miss rates below 0.5% for cache sizes of $8kB$ or more. The data cache miss

Workload W	$compl_W$	$f_{load,W}$	$f_{store,W}$
A - HPA	9.1	0.2319	0.0650
B - PPA	249	0.1691	0.0595

Table 3. Computational Complexity and Load and Store Frequencies of Workloads.

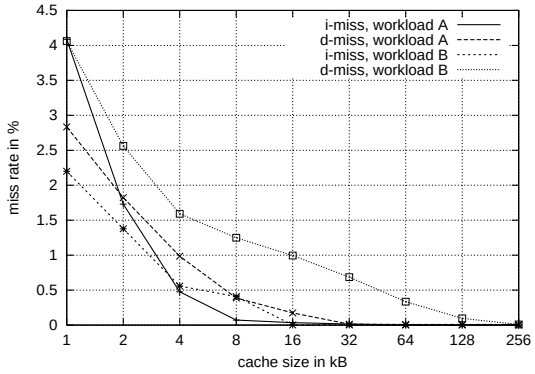

Figure 3. Aggregate Cache Performance of Workloads.

rate for workload A also drops below 0.5% for $8kB$. For workload B, though, the data cache miss rate only drops below 1% for $32kB$ or larger caches.

3.2 System Parameters

The system parameters for the network processor are listed in Table 4. The values for the on-chip area of different components are approximate for .18μm CMOS technology. It should be noted that exact values are hard to obtain from industrial sources. The performance model can of course be used with more accurate parameter sets.

4 Design Results

This section presents and discusses the optimization results and performance trends for various system parameters.

4.1 Optimal Configuration

Table 5 shows the overall best configuration for both workloads. There are several important points that can be seen from this table:

Parameter	Value(s)
clk_p	200 MHz ... 800 MHz
t	1 ... 16
c_i	1 kB ... 1024 kB
c_d	1 kB ... 1024 kB
$linesize$	32 byte
τ_{DRAM}	60 ns
$width_{mchl}$	16 bit ... 64 bit
ρ_{mchl}	0 ... 1
$width_{io}$	up to 72 bit
ρ_{io}	0.75
$clk_{mchl},\ clk_{io}$	200 MHz
$s(p_{basis})$	1 mm^2
$s(p_{thread})$	0.25 mm^2
$s(c_i),\ s(c_d)$	0.10 mm^2 per kB
$s(mchl_{basis}),\ s(io_{basis})$	10 mm^2
$s(mchl_{pin}),\ s(io_{pin})$	0.25 mm^2
$s(ASIC)$	up to 400 mm^2

Table 4. System Parameters for Optimization.

- The optimal number of threads in both cases is $t = 2$, which indicates that it is not necessary to have a large number of threads to obtain good performance.

- The cache sizes are in the range of 16kB to 32kB, which yields an effective cache size of 8kB to 16kB per thread. These values correspond to knees in the i-miss curves in Figure 3. Note that for the d-cache of workload B a small cache size gives better results since there is no clear knee in the curve that makes a larger cache pay off.

- Both configurations use the fastest processor because there is no cost in the model associated with higher clock rates. Also the widest memory channel is used, because it amortizes the basis cost $s(mchl_{basis})$ over a wider channel.

- The number of processors per cluster, n, is 31 and 20. This is relatively high, because a wider memory channel with more processors sharing it amortizes the basis cost better. When limiting the width of the memory channel to smaller sizes (e.g., 48 bit), the same configuration as in Table 5 with a smaller n (e.g., 24) and a larger m (e.g., 3) is the overall best. The $IPS/area$ value for this configuration is slightly lower (e.g., 173 MIPS/mm^2).

- The number of clusters per system is 2 or 3, which is limited by the overall chip area and the I/O channel width. With smaller memory channels and smaller n the number of possible cluster increases.

- The width of the I/O channel is much higher for workload A, because the processing complexity is much smaller for header-processing applications. Therefore data moves more quickly into and out of the network processor. For payload processing, the data remains on the processor for a longer time. Thus, the width of the I/O channel is smaller.

- The overall processing power for both workloads is about the same (although workload B uses more chip area). Due to the lower complexity of header processing, this translates into a much larger throughput for workload A.

Note that these results are optimistic and do not account for certain factors. For example, packet classification is assumed to be done off-chip and no resources are consumed in maintaining and managing memories and routing tables.

4.2 Performance Trends

The optimal configurations of the network processor are very specific to a particular workload. To get more general results, we now look at the impact of different system parameters on the overall performance by varying them. Unless noted otherwise, parameters are fixed to: $t = 2$, $clk_p = 800$ MHz, $m = 1$, $c_i = 16$ kB, $c_d = 16$ kB, $width_{mchl} = 64$ bit, and workload A. Note that these parameters correspond to the optimal configurations for workload A shown in Table 5. Also, ρ_{mchl} is chosen to be such that it yields the maximum performance. When using the term "performance," we mean $IPS/area$ (not IPS). Some of the configurations discussed below exceed the limits on total chip area, width of the I/O channel, and pin count. They are still shown as they might become feasible in the future.

Parameter	workload A	Workload B
clk_p	800 MHz	800 MHz
t	2	2
m	2	3
c_i	16 kB	32 kB
c_d	16 kB	16 kB
$width_{mchl}$	64 bit	64 bit
ρ_{mchl}	0.91	0.89
p_{miss}	0.187%	0.286%
τ_{mem}	137.6	121.6
ρ_p	0.974	0.957
n	31	20
$width_{io}$	71	3
$pins_{NP}$	199+$pins_{control}$	195+$pins_{control}$
IPS	48324 MIPS	45934 MIPS
$area$	272 mm^2	322 mm^2
$IPS/area$	178 MIPS/mm^2	142 MIPS/mm^2

Table 5. Optimal Configurations.

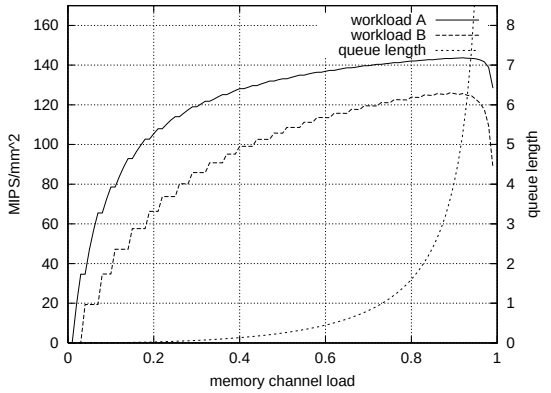

Figure 4. Performance Depending on Memory Channel Load.

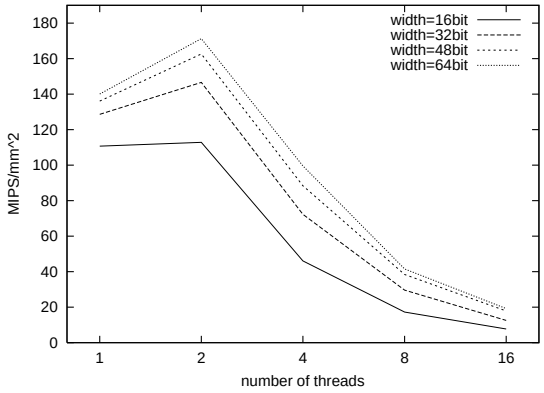

Figure 5. Performance Depending on Memory Channel Width and Number of Threads.

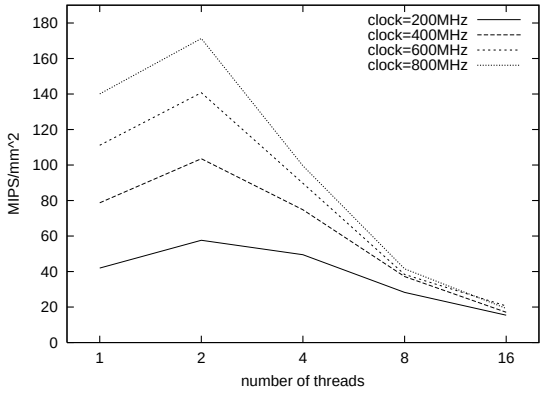

Figure 6. Performance Depending on Processor Clock Rate and Number of Threads.

4.2.1 Memory Channel

One critical parameter for the memory channel performance is the load, ρ_{mchl}. Figure 4 shows the performance of the network processor depending on the chosen load. It also shows the queue length given by the M/D/1 queuing model. For high loads the queuing time is so high that it impacts the performance of the processors. For most configurations the best load is about $\rho_{mchl} = 0.9$.

The width of the memory channel also affects the performance of the network processor. Figure 5 shows that for one thread the memory channel performance does not impact the overall performance, because the system is mostly limited by τ_{DRAM} and τ_Q. For two or more threads, a four-fold increase in memory channel bandwidth (from 16 bit to 64 bit) yields up to twice the performance.

4.2.2 Processor

The processor can be configured in terms of clock rate and the number of thread contexts. Figure 6 shows the performance gains for higher clock rates over different numbers of threads. For one or two threads, the performance increases practically linear with clock speed. For larger numbers of threads, the

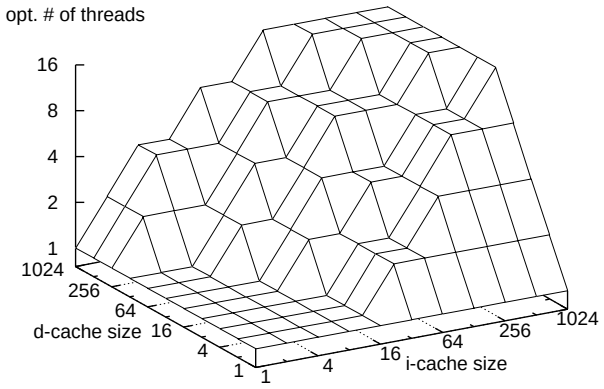

Figure 7. Optimal Number of Threads for Cache Configuration.

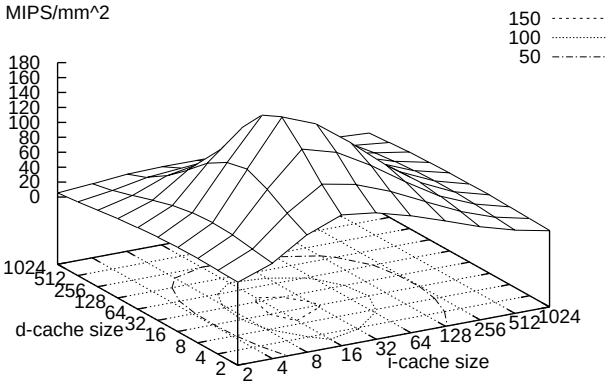

Figure 8. Performance Depending on Cache Configuration (Workload A).

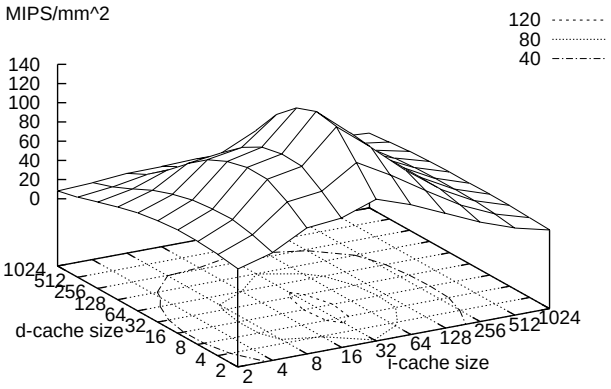

Figure 9. Performance Depending on Cache Configuration (Workload B).

amount of available cache per thread is less, which leads to more cache misses and possible memory stalls. Thus, the increase in performance is limited by off-chip memory accesses that cause processor stalls.

The performance impact of the number of available thread contexts can also been seen in Figures 5 and 6. In both graphs, the optimal number of threads is two. For larger number of threads, there are two factors that limit their benefits. One is the higher cache miss rate due to memory pollution. The other is the additional area cost for the thread context.

To illustrate the impact of the cache pollution, Figure 7 shows the optimal number of threads for a given i-cache and d-cache configuration. This shows that if larger caches were available, more threads could be used for optimal performance. This indicates that with advances in on-chip memory technology, it can be expected that the number of threads in a processing engine will increase in the future.

4.2.3 Cache Memory

The size of on-chip caches is also an important configuration parameter. Since on-chip SRAM is expensive in terms of area cost, the amount of memory should be minimized, while still maintaining good cache hit rates to allow efficient execution of applications. Figures 8 and 9 show the performance of different cache configurations for both workloads. The performance is low for small caches due to high miss rates. It is also low for very large caches, since much chip area is used. The optimum for workload A lies at $c_i = 16$ kB and $c_d = 16$ kB. The optimum for workload B is at $c_i = 32$ kB and $c_d = 16$ kB. With $t = 2$, each thread uses effectively half of the available cache.

Another observation is that the performance is relatively sensitive to deviations from the optimal i-cache size. The d-cache size is less sensitive, but still has much impact on the overall performance. This leads to the conclusion that it is important to configure the memory system of network processors for the particular workload.

4.2.4 Chip Area Usage

Finally, to give a rough idea on how the chip area of a network processor is used, we evaluate what fraction of the total area is used for the processor (including thread contexts), the cache, and the memory and I/O channels. Figure 10 shows the fraction of processor area versus the fraction of cache area. The remaining fraction (to add up to 1) is the memory and I/O channel area. The top 1% ($= 531$) of all configurations are shown. Thus, the processor area typically makes up for 25-40% of the chip area. The cache area accounts for 20-60% and the memory and I/O channel area for 20-60%. The centroid lies at 34% for processors, 38% for cache, and 27% for memory and I/O channel.

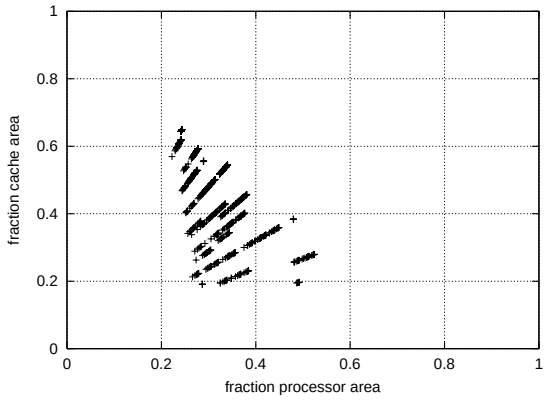

Figure 10. Chip Area Distribution for Top 1% of Configurations.

4.3 Summary of Results

The above results of our performance model can be used to extract a few general design guidelines for network processors:

- The cache configuration has a big impact on the overall performance, which is sensitive to the workload.
- Two to four hardware contexts for threads is optimal. With large on-chip caches, more threads perform better.
- Higher processor clock rates and memory channel bandwidths are directly related to performance improvements for four or less threads.
- The chip area is roughly evenly split between processors, caches, and memory interfaces.

These results are somewhat dependent on the particular workload and systems parameter that are used. The main contribution of this work are not the design results per se but the performance model that can be used with other workloads and systems parameters.

5 Conclusions

The network processor model and the associated performance expressions represent an attempt at developing a coherent approach to designing NPs. The approach is driven by the requirements of applications, the constraints associated with technology, and the selection of design alternatives within a general architecture. While the presented architecture is relatively simple, it could handle line rates on the order of Gbps given current ASIC technology.

Several extensions to the model can be pursued to account for trends in commercial network processor design:

- Specialized co-processors and instruction sets: Certain time consuming networking tasks occur frequently. To deal with such tasks, it is often worthwhile to develop customized logic for use with specialized processor instructions or co-processors.
- More sophisticated memory management: In the current model, every packet is transmitted to a processing engine and back. For header-processing applications only the header of the packet is needed. Many commercial NPs store part of the packet (i.e., the header) in faster on-chip SRAM and the payload in off-chip DRAM.
- Trace-driven simulation instead of mean-value analysis: The proposed model provides a first step towards understanding design tradeoffs. Using actual packet and instruction traces for a system simulation is the next step towards more accurate results.

These extensions are current work in progress. We believe that this work will help formalize NP design and provide a method for fast and accurate exploration of the vast network processor design space.

References

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

[2] R. F. Cmelik and D. Keppel. Shade: A fast instruction-set simulator for execution profiling. In *Proc. of ACM SIGMETRICS*, Nashville, TN, May 1994.

[3] J. Edler and M. D. Hill. *Dinero IV Trace-Driven Uniprocessor Cache Simulator*, 1998. http://www.cs.wisc.edu/˜markhill/DineroIV/.

[4] J. L. Hennessy and D. A. Patterson. *Computer Architecture – A Quantitative Approach*. Morgan Kaufmann Publishers, Inc., San Mateo, CA, second edition, 1995.

[5] T. Wolf and M. A. Franklin. CommBench - a telecommunications benchmark for network processors. In *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 154–162, Austin, TX, Apr. 2000.

[6] T. Wolf and M. A. Franklin. Locality-aware predictive scheduling for network processors. In *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 152–159, Tucson, AZ, Nov. 2001.

[7] T. Wolf and J. S. Turner. Design issues for high performance active routers. *IEEE Journal on Selected Areas of Communication*, 19(3):404–409, Mar. 2001.