Design Considerations for Network Processor Operating Systems *

Tilman Wolf, Ning Weng, and Chia-Hui Tai
Dept. of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA, USA
{wolf,nweng,ctai}@ecs.umass.edu

ABSTRACT

Network processors (NPs) promise a flexible, programmable packet processing infrastructure for network systems. To make full use of the capabilities of network processors, it is imperative to provide the ability to dynamically adapt to changing traffic patterns and to provide run-time support in the form of a network processor operating system. The differences to existing operating systems and the main challenges lie in the multiprocessor nature of NPs, their on-chip resources constraints, and the real-time processing requirements. In this paper, we explore the key design tradeoffs that need to be considered when designing a network processor operating system. In particular, we explore the performance impact of (1) application analysis for partitioning, (2) network traffic characterization, (3) workload mapping, and (4) run-time adaptation. We present and discuss qualitative and quantitative results in the context of a particular application analysis and mapping framework, but the observations and conclusions are generally applicable to any run-time environment for network processors.

Categories and Subject Descriptors: D.4.1 [Operating Systems]: Multiprocessing;

General Terms: Design.

Keywords: Network processors, application partitioning, application mapping.

1. INTRODUCTION

The success of the Internet as a communication medium is driving research in the areas of sensor networks, overlay networks, ubiquitous computing, grid computing, and storage area networks. This trend expands the functionality of networks to include increasingly diverse and heterogeneous end-systems, protocols, and services. Even in today's Internet, routers perform a large amount of processing in the *data path*. Examples are firewalling , network address translation (NAT), web switching, IP traceback, TCP/IP offloading for high-performance storage servers, and encryption for virtual private networks (VPN). Many of these functions are performed in access and edge networks, which exhibit the most diversity of systems and required network functions. With the broadening scope of networking it can be expected that this trend will continue and more complex processing of packets *inside* the network will become necessary [2, 4].

The processing infrastructure for these various packet processing tasks can be implemented in a number of ways. Well-defined, high-speed tasks are often implemented on application-specific integrated circuits (ASICs). Tasks that are not well-defined or possibly change over time need to be implemented on a more flexible platform that supports the ability to be reprogrammed. Network processors (NPs) have been developed for this purpose.

The performance demands of increasing link speeds and the need for flexibility require that these network processors are implemented as multiprocessor systems. This makes the programming of such devices difficult, as the overall performance depends on the fine-tuned interaction of different system components (processors, memory interfaces, shared data structures, etc.). The main problem is handling the complexity of various, interacting NP system components. To achieve the necessary processing performance to support multi-gigabit links, network processors are implemented as system-on-a-chip multiprocessors. This involves multiple multithreaded processing engines, different types of on- and off-chip memory, and a number of special-purpose co-processors. On conventional workstation or server systems these complexities are hidden by the operating system or do not express themselves as drastically due to their uniprocessor architecture. To simplify this process, a number of domain-specific programming languages and optimizing compilers are currently being developed [5, 17, 19]. These approaches aim at optimizing a single application (i.e., router functionality) statically for the underlying hardware. In current network processors, most performance critical tasks are implemented and fine-tuned in assembly (e.g., to balance the processing times in each step of a software pipeline). As a result, slight changes in the functionality can have drastic performance impacts which require re-tuning. Due to the necessary fine-tuning of individual applications, it is very difficult to integrate and dynamically change multiple packet processing functions on a single network processor. However, network processing is inherently a dynamic process. The main motivation for implementing packet processing functions in a network processor (rather than in a faster, more power-efficient custom logic device) is the need to change the functionality over time. Changing traffic patterns, new network services and protocols, new algorithms for flow classification, and changing defenses against denial of service attacks present the dynamic background that a programmable router needs to accommodate. This requires that the router (1) can implement multiple packet processing appli-

*This research was supported in part by National Science Foundation grant CNS-0447873.

ANCS'05, October 26–28, 2005, Princeton, New Jersey, USA.

cations at the same time, (2) can quickly add and remove processing functions from its workload, and (3) can ensure efficient operation under all circumstances. In particular, the management of various system resources is important to avoid performance degradation from resource bottlenecks.

The complexity of NP multiprocessor architectures has limited the use of existing operating system concepts in this domain. The few existing approaches to providing run-time support are still aimed at single packet processing applications that are optimized offline [19, 12] or consider network processing applications as a single monolithic entity [9]. In this paper, we explore a variety of design issues for a run-time environment that supports several concurrent network processing applications and allows dynamic reconfiguration of the workload on a multiprocessor system. The key design considerations that are addressed fall into four broad categories: (1) application partitioning, (2) traffic characterization, (3) run-time mapping and adaptation, and (4)system constraints.

Research efforts in the network processor domain have long been influenced by particular system designs. In order to separate the general design issues of operating systems from a particular system, we separate qualitative observations and quantitative results. First, we explore the above design issues and discuss the *qualitative* tradeoffs between available design choices. This analysis provides an important understanding of interactions between system tradeoffs. Second, we provide *quantitative* results that illustrate the tradeoffs discussed above in the context of actual NP applications. For this, we leverage NP application partitioning and mapping techniques and analytic performance modeling results that were developed in our prior work [15, 20].

In Section 2, we discuss related work. Section 3 discusses the key characteristics of network processor operating systems and how they differ from conventional workstation operating systems. Section 4 discusses qualitative design tradeoffs and Section 5 quantifies these observations in the context of our particular experimental system. Section 6 combines the results and discusses their implications on network processor operating system design. Section 7 concludes this paper.

2. RELATED WORK

Commercial examples of network processors are numerous [3, 6, 7, 10]. A network processor is typically implemented as a single-chip multiprocessor with high-performance I/O components, which is optimized for packet processing. In particular, NPs provide a more suitable architecture to handle these workloads than conventional workstation or server processors. The need for a specialized architecture is due to the uniqueness of the workload of NPs, which is dominated by many small tasks and high-bandwidth I/O operations [21]. Another alternative for implementing packet processing functions is programmable logic devices, e.g., field-programmable gate arrays (FPGAs), which are more suitable for some of the data flow style processing functions [1, 18]. The design considerations for run-time support in FPGA-based systems are very similar to those of conventional NPs and thus the results of our work can be expected to be equally applicable to this domain.

In order to achieve the necessary performance of ever-increasing line speeds and increasingly complex packet processing functions, both NPs and FPGAs exploit the parallelism that is inherent in the workload of these systems. In general, packets can be processed in parallel as long as they do not belong to the same flow. Processing functions *within* a packet can also be parallelized to decrease packet delay. This leads to NP systems with numerous parallel processing and co-processing engines. To program such a system, several domain-specific programming languages have been developed. Intel jointly with the Shangri-La project at UT Austin has developed Baker [5]. The MESCAL project at UC Berkeley has developed NP-Click [17], which is based on the Click modular router [8].

In the NEPAL project [12], Memik et al. propose a run-time system which controls the execution of applications on a network processor. Applications are separated into modules at the task level using the NEPAL API and modules are mapped to execution cores dynamically. There is an extra level of translation during which the application code is converted to modules. The analysis of run-time techniques is limited to a couple of heuristic allocation algorithms. Teja [19] is a commercial programming environment for the Intel IXP family of network processors. While they provide a thin network processing operating system (NPOS), both Teja and NEPAL are designed to simplify the programming process of a single application and aim at code-reuse across platforms. The ability to quickly adapt to multiple applications on the same network processor system is not supported.

Even though there has not been much work on the mechanisms for dynamically managing multiple applications, there has been work on the algorithms for adapting and scheduling network processors to save power. Kokku et al. have explored run-time environment design issues [9] similar to the ones we present here, but do not considers partitioned applications that are distributed over several processors of the network processing system. Instead, it is assumed that an entire application is mapped to a single processor core.

3. NETWORK PROCESSOR OPERATING SYSTEM

The term "operating system" is most commonly used in the context of workstation computers. The responsibilities of such an operating system are to manage hardware resources and isolate users from each other. The optimization target is commonly to minimize the execution time of a single task (the one the user is currently working on). It is important to note that the goals of an operating system for network processors are very different. On a network processor, all applications are controlled by the same administrative entity and optimization aims at maximizing overall system throughput.

3.1 Differences to Workstation Operating Systems

The following list details the differences between network processor operating systems and conventional operating systems:

- **Separation Between Control and Data Path.** This separation refers to the processor context, not the networking context. To achieve high throughput on NPs, several studies have shown that it is more economical to implement a larger number of simpler processing engines than fewer, more powerful ones. Such simple processors do not have the capability to run complex control tasks on top of packet processing tasks. In today's NP designs, control is implemented on a separate control processor. Due to this separation between "classes" of processors, it is necessary to have a more explicit control structure than one would have in a conventional operating system.

- **Limited Interactivity.** Users do not directly interact with an NP or its operating system. At most, applications are installed and removed occasionally. This does not mean that a user could not change configurations on an NP (e.g., update rules for a firewall application), but the key variable in this

system are the traffic patterns that determine what processing needs to happen.

- **Regularity and Simplicity of Applications.** One dominating aspect of network processing is that data path processing is performed on individual packets. This means that packet processing tasks are typically limited in complexity due to the real-time constraints imposed by continuously arriving packets. As a result, the processing demands are low (few hundred to several thousand instructions [16]). Additionally, the execution path within an application is the same in a large number of cases and only slightly different for the other cases. Therefore it is feasible to analyze packet processing applications in detail to find good processor mappings.

- **Processing Dominates Resource Management.** Conventional operating systems need to implement a number of different functions: processor scheduling, memory management, application isolation, abstraction of hardware resources, etc. In network processor systems, these challenges are dominated by managing processing resources. The diversity of hardware resources is limited and many are controlled directly by the application. Also, memory is usually allocated statically to ensure deterministic run-time behavior. This might change in the future as network applications become more complex and network processor operating systems become more similar to conventional operating systems. In this work, we focus on processing aspects of operating system functionality.

- **Non-Existence of User-Space/Kernel-Space Separation.** All functions on a network processor are controlled by the same administrative entity. There is no clear separation between user-space and kernel-space in the traditional sense. Instead, functionality is divided betwen control and data path. As a result, traditional protection mechanisms are typically not implemented in network processor operating systems.

3.2 Design Considerations

Due to these numerous and significant differences between what is conventionally thought of as an operating system and what is necessary for a network processor, we believe it is important to explore some of the fundamental design issues that one encounters in the context of network processor operating systems.

The questions that arise from such an exploration are:

- How should applications be dynamically installed on and removed from the network processor?

- How can applications be partitioned and mapped to utilize the underlying multiprocessor hardware?

- How should the operating system adapt to changes in network traffic that require different application configurations?

- What should be the frequency and level of adaptation?

In order to explore operating system design aspects concretely, we assume a general operational approach as shown in Figure 1. There are four basic steps that are necessary for run-time support of network processor systems: application analysis, traffic characterization, workload mapping, and adaptation. There is a fundamental question what should be done offline (e.g., during application development) and what should (and can realistically) be done during run-time.

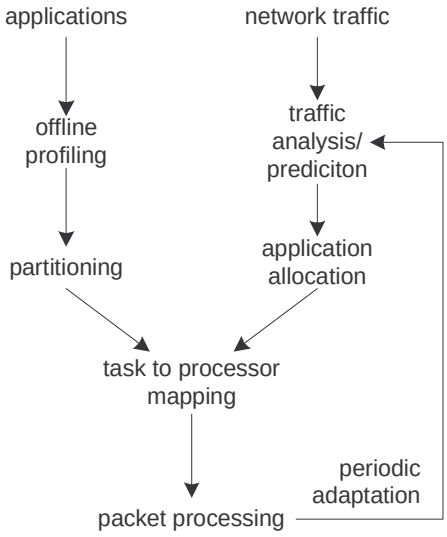

Figure 1: Processing and Traffic Analysis in an Network Processor Operating System.

Application analysis is necessary to analyze the processing requirements of the application and to be able to partition the application. Partitioning allows the distribution of different subtasks onto different processing elements to fully utilize the resources on the NP. The simplicity and repetitiveness of network processing applications [21, 16] allows for a detailed analysis of the application. The profiling process is shown as an offline component. With the limited processing resources on current NP architectures, this analysis cannot be done online. Typically, such an analysis can be performed in the context of the application development environment for the NP applications. The partitioning can be performed in different ways and the level of granularity at which it should be performed is discussed in more detail below.

Network traffic characterization is another important aspect of run-time support for network processor systems. Depending on the requirements of current network traffic, different applications dominate the processing. The dynamically changing workload is the main reason why run-time support is necessary. In order to achieve a good allocation of processing resources, it is necessary to know what processing is necessary for packets currently in the system (or that will be processed in the near future). The result of traffic analysis is an application allocation, which describes the ratio of processing required by each application that is available on the system.

Workload mapping is the process that assigns processing tasks to actual processing engines. This assignment is based on the application allocation and the application partitioning derived in the previous two steps. Mapping can be performed in a number of different ways and depends on the particular system architecture, application development environment, and operational principles of a system. The goals of mapping are to achieve high system throughput and efficient resource utilization.

The *adaptation* step illustrates the need to reconfigure the network processor system to match the processing requirements that are dictated by the traffic workload. During adaptation, the application allocation is changed according to the new traffic requirements. Then, the mapping step modifies the allocation of tasks to processors to match the new allocation.

4. QUALITATIVE TRADEOFFS

With each of these four components of a network processor operating system discussed in the previous section, there are a number of qualitative design tradeoffs. We discuss the different design choices in this section and then provide quantitative results in Section 5. Since the quantitative results are highly dependent on a particular system, we have separated the discussion of tradeoffs to preserve its general applicability.

4.1 Application Partitioning

Network processor applications rarely consist of a single monolithic piece of code. More commonly, NP applications are split into several subtasks. For example, on the Intel IXP2400, input processing is separated from forwarding and from output processing. This partitioning makes application development somewhat easier and changes to the application easier to implement. Also, it allows for exploiting parallelism and pipelining to fully utilize the multiprocessor infrastructure. How can an operating system support this application partitioning?

4.1.1 Manual Partitioning

Manual application partitioning is the most common approach to determining a suitable separation of tasks and a mapping of tasks to processors. Using simulation environments, programmers can implement a certain partitioning and obtain performance results. By manually adapting the partitioning, the bottlenecks can be removed and the performance can be fine-tuned. This approach is very time consuming and requires a detailed understanding of the application and the NP hardware. From an operating system perspective, manually partitioned applications limit the amount of dynamic support. It is generally not possible to adapt to changing traffic conditions.

4.1.2 Automated Partitioning

More recently, several approaches to automated partitioning of applications have been investigated. The automatic mapping aspect discussed below is related to this. An auto-partitioning compiler has been developed by Intel in [5]. Ramaswamy et al. have developed a profiling-based instruction clustering algorithm that derives a directed acyclic graph representation of NP applications [15]. The granularity of the partitioning can be adapted as needed. Plishker et al. take an approach were applications are described in a domain-specific language and then distributed onto processing elements [14].

4.1.3 Design Choices

One key question is what granularity of application partitioning is most suitable. The spectrum of choices ranges from monolithic applications to extremely fine-grained instruction (or basic block) allocations to processing resources. The design tradeoffs are:

- **Monolithic Applications.** If the application is not partitioned, it can only be allocated to a single processing engine. This significantly limits how the application workload can be adapted to network traffic requirements. Also, it may cause performance bottlenecks in pipelined systems (e.g., multiple sequential applications per packet), where the pipeline speed is determined by the maximum stage time. Finally, as application size continues to grow, monolithic applications do not allow for a scalable distribution of processing and may conflict with instruction store limitations.
- **Very Fine-Grained Partitioning.** The extreme opposite case of a monolithic application is a partitioning where each instruction is mapped individually to a processing resource. This approach provides more flexibility, but also generates more overhead for managing the installation and control of the application. It also increases the complexity of the mapping problem, because a large number of nodes have to be mapped and the space of possible solutions grows significantly with the number of mapping choices.
- **Balanced partitioning.** Ideally, we would like to find a balanced partitioning that allows efficient distribution of processing tasks, but keeps the complexity of the mapping problem at bay.

In the quantitative section below, we show that an optimally balanced partitioning exists. This emphasizes the importance of considering application partitioning for network processors. The assumption that applications are a monolithic structure that needs to be mapped to a single processing engine is not sufficient.

4.2 Traffic Characterization

Traffic characterization is an important input to determining a suitable allocation of different applications on the network processor. Heavily used applications typically need to be replicated multiple times to provide sufficient performance (e.g., multiple parallel IP forwarding applications). When considering run-time support for workloads, it is important to be able to analyze network traffic to estimate and possibly predict a suitable application allocation.

4.2.1 Static Traffic Model

The simplest case of traffic characterization is a static traffic model. This is the most commonly used model since it does not require any online changes in the system. The assumption is that traffic requirements remain the same over the entire run time of the system. Short term variations are compensated by buffering (thus increasing the packet delay) or overprovisioning, where additional resources are allocated to each type of application (thus increasing the total required hardware resources).

4.2.2 Batch Model

In the batch-processing approach, a certain number of packets are buffered and their processing requirements analyzed before a workload allocation decision is made. This buffering allows an accurate characterization of the requirements. Since the allocation or reallocation of tasks cannot be done arbitrarily fast, it is necessary to maintain a particular processor mapping for a certain amount of time. This period might be only a few milliseconds, but given high link rates, it is equivalent to at least several dozen or hundred packets. If batch processing is used, then all the packets in a batch need to be buffered. Since this increases the overall delay that is experienced by a packet, predictive batch processing might be more suitable.

4.2.3 Predictive Batch Model

In many cases, network traffic exhibits a certain amount of temporal locality. This can be exploited to derive a traffic requirement estimation for a batch by observing a smaller window within the batch ("sampling"). Based on the observation, the total processing requirements for the batch can be extrapolated. For small samples, only a few packets need to be buffered to determine the allocation of a batch. With larger samples, the accuracy increases, but so does the processing delay.

4.2.4 Design Choices

In order to determine the allocation of applications to the network processor system, traffic characterization can be performed according to the approaches described above. The tradeoffs are:

- **Choice of Batch Size.** The batch size determines how many packets are processed given the same allocation of processing resources to applications. The smaller the batch size, the more frequently this allocation can be adapted to changes in traffic. However, with smaller batch sizes, less time is available between reconfigurations for the operating system to determine how to adapt and to find a good task mapping.
- **Choice of Sample Size.** In order to minimize the delay for packets, the sample should be chosen to be small. This reduces the accuracy and thus there is a limit to how small it can be.
- **Granularity of Application Allocation.** From the traffic models, an application allocation is derived that assigns a fraction of the overall processing requirements to each application. For large batch and sample sizes, this allocation can be determined with high precision. In contrast, most network processor systems only support a very coarse allocation of applications (e.g., number of times a certain application is replicated on the multiprocessor). Therefore, the granularity of application allocation can be kept at the same level of precision as the network processor can support. This permits the usage of smaller samples.

4.3 Run-Time Mapping and Adaptation

The mapping of application tasks to processing elements is performed through a mapping algorithm. We explore the design tradeoffs without requiring that a particular algorithm be used. However, we assume two properties of the mapping algorithm:

- The mapping algorithm can yield incrementally better results as the run-time increases. This implies that the operating system designer could choose the run time of the algorithm and the quality of the resulting mapping.
- The mapping algorithm can be employed on a partially configured system. This means that some applications can be removed and others can be added without changing the mapping of applications that are not affected.

The resulting design choices address the frequency and level of (partial) mapping.

4.3.1 Static Mapping

Static mapping goes hand-in-hand with static partitioning and a static traffic model. In this case, processing tasks are allocated to processors offline and no changes are made during run-time.

4.3.2 Complete Dynamic Mapping

Complete mapping refers to a mapping solution where the entire workload is mapped from scratch. The mapping algorithm can place processing tasks on the entire architecture without any initial constraint. This typically leads to a good solution that approximates the theoretical optimum for increasing processing times.

4.3.3 Partial Dynamic Mapping

Partial mapping assumes that some part of the workload is already mapped to the NP system. The mapping algorithm only needs to map a few applications to the remaining processing resources. This approach is more restrictive than complete dynamic mapping because much of the system resources are already assigned to applications. The incremental nature of this approach poses the risk that the mapping algorithm "gets stuck" in a local minimum. Nevertheless, the processing cost for the partial mapping is less than mapping the entire workload.

4.3.4 Mapping Effort

Another interesting parameter for dynamic mapping is how often to reprogram the system. Ideally, we want to reconfigure application allocation with every packet to guarantee the best system utilization and high performance while traffic is varying. However, there is a cost associated with mapping and remapping. Apart from the cost for uploading new instructions to each processor, determining the new mapping requires processing power and computation time. It is important to keep the reprogramming frequency at a low enough rate where sufficient processing time is available to find good mapping results.

4.3.5 Design Choices

Design choices for mapping determine how often, how much, and with how much effort to perform complete or partial mapping.

- **Choice of Dynamic Adaptation Frequency and Mapping Effort.** In order to adapt to changing traffic conditions, the network processor operating system needs to change the application allocation and thus the mapping. The rate at which this happens determines the maximum time that is available for determining a new mapping. The lower the adaptation rate, the more time can be spent on finding a better mapping solution. As the adaptation rate increases, the quality of the derived mapping solution decreases.
- **Choice of Partial Mapping and Mapping Effort.** Changes in traffic conditions may only affect a few applications. In order to be able to adapt quickly with low mapping cost, an operating system designer may choose to only map a small part of the overall allocation. The benefit of this is the ability to adapt quickly, but the amount of traffic variation that can be supported is limited to the fraction of the NP that is remapped.
- **Balance of Partial and Complete Mapping.** Repeated partial mapping causes mapping solutions to deteriorate. In order to avoid this, complete mapping steps should be performed periodically. The more frequently this happens, the less likely the system will go into an inefficient state. However, this also increases the overall mapping effort.

The quantitative results show that partial mapping can provide quicker adaptation to changes in traffic, but that the deterioration of the mapping result can have significant impact on the performance.

4.4 Constraints

A network processor has a number of system constraints that are not considered in the above discussion. These constraints can play a major role when making design decisions.

4.4.1 Instruction Store Limitations

Most network processor systems are severely limited in the amount of instruction store that is available for each processing engine. This is due to the relatively high chip area cost of memory compared to processing logic. This limitation is the reason that not all applications can be installed on all processing engines at all time. Therefore mapping changes are quite costly as they require uploading of new instructions to every processor.

4.4.2 Overprovisioning

A network processing system needs to be capable of processing any packet that is transmitted on the network. In the context of a network processor operating system, this means that processing resources should be available for all applications. If this is not the

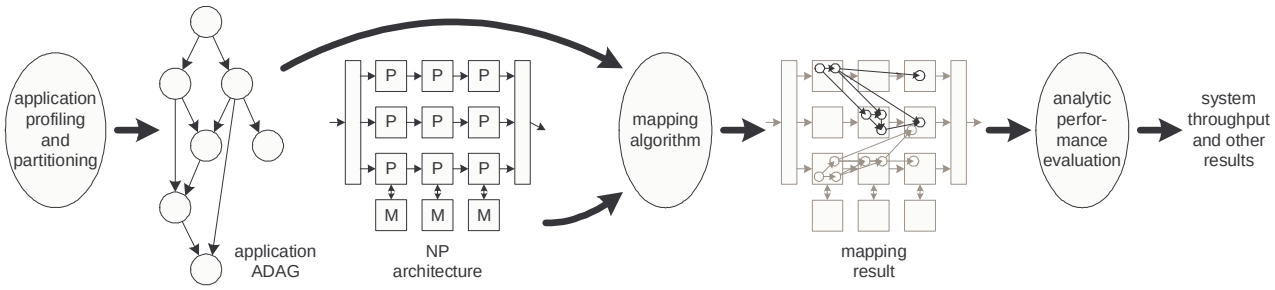

Figure 2: Application Analysis, Mapping, and Performance Evaluation Process. Typically, multiple ADAGs are mapped to the NP architecture to reflect the workload mix that can be processed by the network processor.

case, packets need to be delayed until the next adaptation cycle or processed in the slow path of the router. One way to avoid this delay is to overprovision the system and increase the application allocation to more than 100%. It can be ensured that even applications that are not expected to see any traffic in the upcoming batch can be installed just in case.

5. QUANTITATIVE RESULTS

In this section, we support the qualitative observations of the previous sections with quantitative results. This helps illustrating which trends have a large impact and which trends have a small impact on the system performance.

In order to derive quantitative results, we use a particular system baseline. Of course, there are big differences between different network processor systems and results on another system would look somewhat different. It is therefore more important to consider the trends that can be observed in our results (e.g., does an optimum exist?) than individual data points (e.g., where exactly is the optimum?).

5.1 Baseline System

The metric that we are interested in is the throughput of a network processor system given a certain workload. In order to derive this information, we need to implement some of the functions that are necessary for network operating systems. In particular, we need to consider realistic network processing applications, their partitioning, and the mapping of processing tasks to processing engines. In order to explore the design space that we have described, it is not sufficient to only consider a handful of partitioning and mapping solutions. We choose therefore to use an analytic modeling approach instead of simulation. With the automated partitioning, mapping, and performance modeling environment provided in [15] and [20], we can evaluate a large number of possible application partitioning, mapping results, etc. This provides a first-order understanding of the quantitative tradeoffs. In the process, several ancillary metrics (e.g., cost for deriving a certain quality mapping) can be obtained.

The process of obtaining performance results is shown in Figure 2. We briefly describe the three key components, application partitioning and representation, the mapping algorithms, and the analytic performance model, to provide a basis for the understanding of the results below.

5.1.1 Application Representation

A network processing application needs to be represented in a way that it can be easily mapped to multiple parallel or pipelined processing elements. This requires a representation that exhibits the application parallelism while also ensuring that data and control dependencies are considered. We use an annotated directed acyclic graph (ADAG) to represent the dynamic execution profile of applications.

The ADAG is derived from dynamic profiling of the application by determining data and control dependencies between individual instructions. Using a clustering heuristic that minimizes the communication overhead, instructions are aggregated to larger nodes in the graph. Each node is annotated with information on the total number of instructions and memory accesses that need to be executed when processing the node.

5.1.2 Mapping Algorithm

Once we have the application represented as an ADAG, the next step is to map the ADAG onto a NP topology. The goal of the mapping is to assign processing tasks (i.e., ADAG nodes) to processing elements and generate a schedule that achieves the maximum system throughput. This assignment is not easy because the mapping process needs to consider the dependencies within an ADAG and ensure that a correct processing of packets is possible. Further, Malloy et al. have shown that producing an optimal schedule for a system that includes both execution and communication cost is NP-complete, even if there are only two processing elements [11]. Therefore we need to develop a heuristic to find an approximate solution.

Our heuristic solution to the mapping problem is based on "randomized mapping." The key idea is to randomly choose a valid mapping and evaluate its performance. By repeating this process a large number of times and picking the best solution that has been found over all iterations, it is possible to achieve a good approximation to the global optimum. With the randomized approach any possible solution is considered and chosen with a small, but nonzero probability. This technique has been proposed and successfully used in different application domains [13]. The mapping is performed for multiple, possibly different, ADAGs. The mixture of ADAGs represents the allocation of applications to the NP architecture.

5.1.3 Analytic Performance Model

In order to evaluate the throughput performance of a given solution, we use an analytic performance model that considers processing, inter-processor communication, memory contention, and pipeline synchronization effects. After mapping the application ADAGs to the network processor topology, we know exactly the workload for each processing element. This information includes the total number of instructions executed, the number of memory accesses, and the amount of communication between stages. The model needs to take this into account as well as contention for shared resources (memory channels and communication intercon-

Table 1: Baseline Configuration for Quantitative Results.

System parameter	Baseline configuration
NP pipeline stages	4
PEs per stage	4
Total number of PEs	16
Memory interfaces per stage	2
Memory access time (in cycles)	10
Number of ADAGs	20
Nodes per ADAG	8

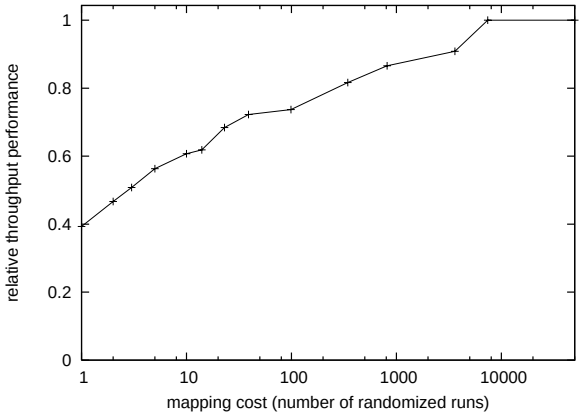

Figure 3: System Performance Compared to Mapping Cost.

nects). We are particularly interested in the maximum latency of each pipelined processing stage since that determines the overall system speed. The number of ADAGs that are mapped to an architecture determines how many packets are processed during any given stage time. After specifying the several system parameters, the throughput of the system for a given mapping can be expressed.

5.1.4 System Configuration and Workload

The system architecture considered in the above model can be configured to represent any regular NP architecture with a specified number of parallel processing engines (PEs) and pipeline stages. We have chosen one single baseline system with a fixed configuration to explore the operating system issues (see Table 1). A separate question is how these change for different architecture configurations. This design space exploration is currently not addressed in our work. The applications that are considered for this system are radix-tree-based IP-lookup and hash-based flow classification.

5.2 Processing Task Mapping

5.2.1 Metrics

Mapping takes a certain amount of processing. This is expressed as c. The performance that is achieved by the mapping is expressed as t, the throughput of the system. Due to the NP-completeness of the mapping problem, finding the overall optimal solution is infeasible. What is really desirable in a system is to obtain a "good enough" solution by running the approximation algorithm for a large amount of time.

5.2.2 Results

Figure 3 shows the increasing quality of the mapping result as more processing effort is dedicated to the mapping process. The mapping cost is the time that it takes to calculate the solution (expressed as the number of randomized mapping iterations). We do not consider the additional overhead for stopping, reprogramming, and restarting processing engines, which occurs in real systems. Since the maximum system performance is unknown, we use a mapping result with very high cost as the baseline.

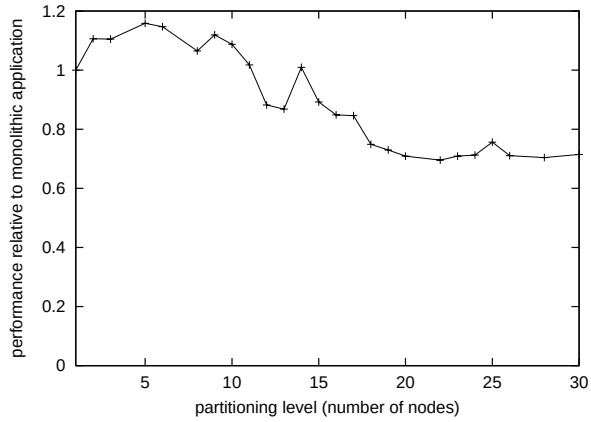

Figure 4: Performance for Different Levels of Application Partitioning. The overall mapping effort is limited to 10,000 total node mapping attempts.

5.3 Partitioning

The goal of application partitioning is to study the impact of the partition granularity on the performance.

5.3.1 Metrics

We consider the number of tasks (or nodes in the ADAG), n, into which the application is partitioned. The maximum n is different for each application, but for this evaluation we only consider values of n that are much smaller than this maximum. If the partitioning is balanced, then the size of each subtask is approximately $\frac{1}{n}$ of the application size.

5.3.2 Results

First, we explore the tradeoff between system throughput and partitioning granularity. Finer granularity promises better performance if the permissible mapping effort is unbounded. When considering the realities of a network processor operating system, the mapping effort is bounded by the batch size and adaptation frequency. Figure 4 shows the throughput of the baseline system with different levels of application granularity relative to a monolithic implementation, where the entire application is executed on a single processor. The mapping effort is fixed and it can be observed that the best performance is achieved for n=5. The monolithic application performs worse because it does not permit an even distribution of processing tasks on the multiprocessor system. Larger values of n also degrade the performance because the mapping problem becomes more complex and finding a good mapping in a limited amount of time becomes more difficult.

5.4 Traffic Characterization

To illustrate the need for traffic characterization and run-time adaptation, we have performed network measurement experiments.

5.4.1 Metrics

As described above, the predictive batch model assumes that traffic is processed in batches (with batch size b) and the applica-

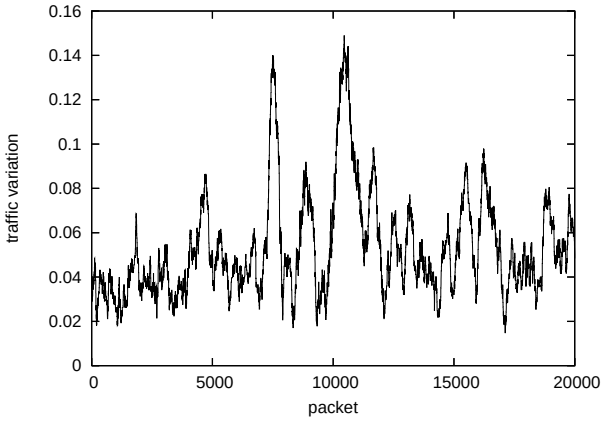

Figure 5: Traffic Variation Over a Sequence of Packets.

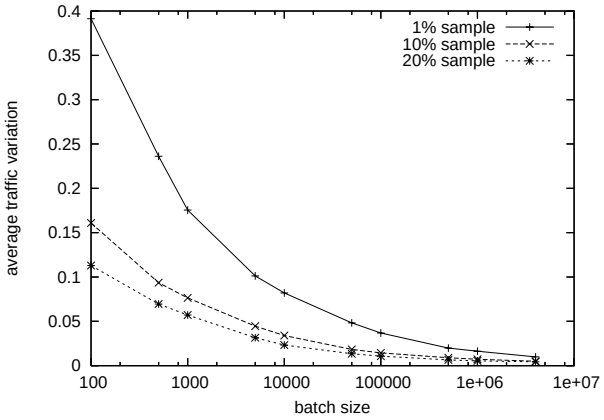

Figure 6: Traffic Variation Compared to Batch Size with Different Levels of Sampling. The sample size is set to a percentage of the batch size.

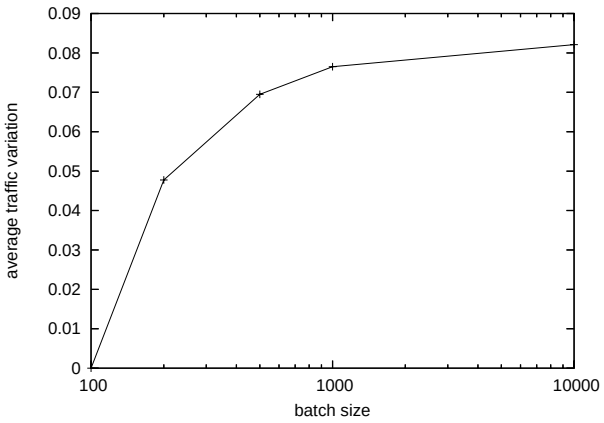

Figure 7: Traffic Variation Compared to Batch Size with Fixed Sample Size. The sample size is fixed to l=100.

tion allocation is based on a sample (size l). We can then describe the traffic variation v based on two metrics, $e_{i,j}(a)$ and $p_{i,j}(a)$. Metric e reflects the estimated number of packets requiring application a and metric p is the actual number of packets requiring this application in packet interval $[i \ldots j)$:

$$v_i(l,b) = \frac{1}{b} \cdot \sum_a \max(p_{i,i+b}(a) - \frac{b}{l} e_{i,i+l}(a), 0). \qquad (1)$$

For example, if the traffic matches exactly the estimated allocation, then all packets "matched up" and the traffic variation is v=0. If half the packet of a batch are different from what was expected (e.g., all packets require a single application instead of an estimated 50/50 split between two application), then the traffic variation is v =0.5.

5.4.2 Results

To get realistic data on what traffic variation looks like in a network, we obtained measurement data from our institution's Gigabit internet access link. We collected 4,235,403 packets and classified them by layer 7 applications (using the classification rules of the Ethereal tool). There are a total of 175 categories, but over 98% of the traffic falls into the top five categories.

Figure 5 shows traffic variation over a sequence of packets. The parameters are b=10,000 and l=100 and the variation is computed as a sliding window. While this is only a small sample of the overall measurement that we have performed, it does reflect the overall trends correctly. In most cases, the variation is around v=0.04 with a few spikes of up to v=0.15.

Of course, the observed variation depends on the quality of the estimate (i.e., size of l relative to b) and the batch size b. To show this relationship, Figure 6 shows the average variation observed throughout the entire trace for different batch sizes and different sample percentages $\frac{l}{b}$. With small samples and small batch sizes, the average variation is very high (v=0.4). The peak variation in this case can reach v=1. The larger the sample percentage, the better are the estimates and thus the lower is the traffic variation. As the batch size increases, the temporary variations within the network traffic are "smoothed out" and less average variation is observed. This does not mean that the static allocation approach (b=∞) is necessarily ideal. What the figure does not show is the delay that would be caused by smaller scale traffic variation nor the overprovisioning that would be necessary to support all possible traffic conditions. When the amount of sampling is fixed, then larger batch sizes cause higher traffic variations as shown in Figure 7. These results illustrate the importance of run-time adaptation because traffic does change significantly on short time scales.

5.5 Adaptation

Adaptation during run-time is guided by the variation of traffic. Complete and partial mapping is performed during each adaptation process.

5.5.1 Metrics

The metrics that are interesting for adaptation are the system throughput in comparison to a baseline configuration. The key input parameters are the frequency of adaptation and the amount of partial mapping (i.e., the fraction of NP resources to which new applications can ge allocated). For our experiment, we consider the adaptation frequency to be the same as the batch size.

5.5.2 Results

Figure 8 shows the degradation effect of repeated partial mapping. The figure shows that repeated removals and additions of applications cause system performance to quickly drop and then stabilize at a suboptimal state of about 80% of the peak performance. There is little variation between different levels of partial adaptation; the stabilizing value is driven more by the amount of effort

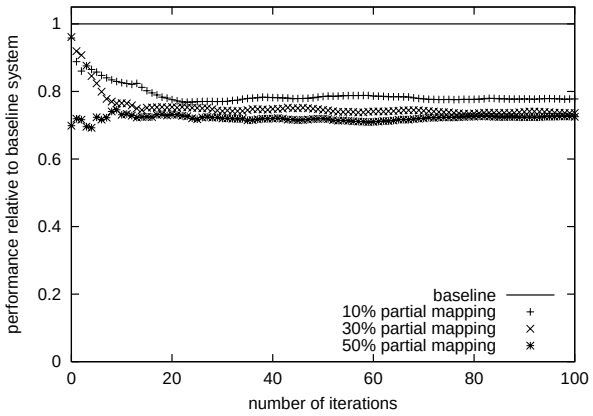

Figure 8: Performance Degradation Due to Repeated Partial Mapping. The baseline case is a complete mapping.

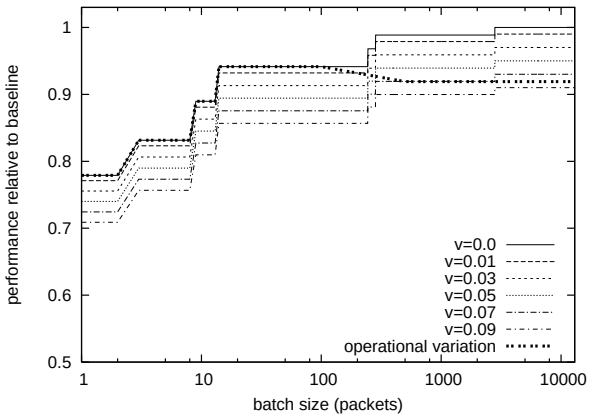

Figure 9: Performance for Different Batch Sizes under Traffic Variation. The total mapping effort is fixed and the baseline case has infinite batch size with no traffic variation.

that is dedicated to the partial mapping. This result clearly shows that partial mapping can very quickly lead to suboptimal configurations. When designing a NP operating system, it should be taken into account that complete mapping is occasionally necessary.

The adaptation frequency (i.e., the batch size) poses a tradeoff between the adaptiveness of the system to changing traffic and the quality of the mapping result that can be derived. Figure 9 shows the relative performance of configurations with different batch sizes and traffic variation. With increasing batch size, the performance of the system increases, but as the traffic variation increases, the performance decreases.

The line marked as "operational variation" shows the maximum system performance with real traffic variation and a chosen batch size (with fixed sample size of l=100 as shown in Figure 7). As the batch size increases, the traffic variation does so, too. The resulting deviation from the processing allocation causes the performance to drop. The trend of this curve shows that there clearly is an optimal batch size. This further supports the argument that careful design considerations are necessary when choosing such system parameters.

6. NP OPERATING SYSTEM DESIGN SCENARIOS

In the previous two sections, we have discussed a number of design considerations and explored their qualitative and quantitative dependencies. To summarize some of these observations and to identify concrete results, we present three example designs for network processor operating systems. For each scenario, we discuss the performance aspects based on our observations above.

6.1 Scenario 1: Static Configuration

This scenario assumes that all analysis, mapping, and allocation operations are performed offline. Once the network processor is configured, no adaptation is performed. This scenario represents many of today's network processor systems that are programmed and optimized for a single application scenario. The design and performance considerations are:

- **Simplicity of System.** Clearly, as static handling of all mapping and allocation issues simplifies the implementation of the system. There is no need for run-time control.
- **Limited Run-Time Flexibility.** The static approach does not allow for any changes during run-time. Any change in the workload configuration requires the use of a software development tool to reconfigure the entire system. As network processors become more integral components of networks, this approach will become less feasible.
- **Performance Degradation under Traffic Variation.** From Figure 5, we can see that traffic requirements do change over time. This leads to performance degradation or the need for significant overprovisioning.

6.2 Scenario 2: Predetermined Configurations

In this scenario, we assume that the NP system can be configured to one of multiple predetermined workload configurations. Each configuration is statically mapped in an offline process. During run-time, the system can adapt to any one of these configurations. The design and performance considerations are:

- **Offline Mapping.** While the system needs to monitor traffic variation, it does not need to perform mapping computations. This limits the complexity of the operating system.
- **Limited Adaptability.** The number of configurations that can be precomputed is limited. If traffic varies outside the estimated bounds, no further adaptation is possible. Within the bounds of estimated traffic, it is unlikely that actual traffic completely matches a predetermined configuration. Thus, a certain level of performance degradation can be expected.
- **Better Quality Mapping Results.** Due the availability of arbitrary amounts of computational power for offline mapping, the quality of mapping results (see Figure 3) can be better than for online mapping.

6.3 Scenario 3: Fully Dynamic Configuration

In the fully dynamic scenario, mapping is performed online and the system adaptation is performed to match the traffic variation. This case utilizes sampling to estimate the processing requirements for each batch. The design and performance considerations are:

- **Complete Adaptability.** A fully dynamic system can adapt to any traffic variations – even configurations that could not

be predicted when programming the system. This is clearly the most important functional benefit of this scenario.

- **Limited Mapping Quality.** Due to the online nature of the mapping process, only limited amounts of processing time are available (very large batch sizes decrease performance – see Figure 9). Thus, the quality of the mapping results is less than that of the other two scenarios.

- **Lower Overprovisioning Overhead.** Due to the ability to adapt to changing traffic conditions, a fully dynamic scenario can provide high throughput performance with less processing resources.

In summary, each of the three approaches has its benefits (simplicity vs. high quality mapping vs. lower system requirements). Nevertheless, the ability to completely adapt at run-time is key in future network processor systems.

7. CONCLUSION

We have presented an extensive qualitative discussion of design issues related to operating system design for network processors. To illustrate the design considerations, we have provided quantitative results that highlight performance tradeoffs between various design parameters. Finally, we have explored three different operating system designs and their benefits and drawbacks.

We believe that this study provides an important basis for design and implementation of future operating systems for network processors. Understanding the presented tradeoffs will guide operating systems designers to consider the relevant interactions between applications, network traffic, and the underlying hardware. This will bring us closer to realizing network processors as easy-to-use components of network systems.

8. REFERENCES

[1] BAKER, Z., AND PRASANNA, V. K. A methodology for synthesis of efficient intrusion detection systems on FPGAs. In *Proc. of the IEEE Conference on Field-Programmable Custom Computing Machines* (Napa, CA, Apr. 2004).

[2] ELSON, J., AND CERPA, A. Internet content adaptation protocol (ICAP). RFC 3507, Network Working Group, Apr. 2003.

[3] EZCHIP TECHNOLOGIES LTD. *NP-1 10-Gigabit 7-Layer Network Processor*. Yokneam, Israel, 2002. http://www.ezchip.com/html/pr_np-1.html.

[4] GAVRILOVSKA, A., SCHWAN, K., NORDSTROM, O., AND SEIFU, H. Network processors as building blocks in overlay networks. In *Proc. of Hot Interconnects* (Stanford, CA, Aug. 2003), ACM, pp. 83–88.

[5] GOGLIN, S. D., HOOPER, D., KUMAR, A., AND YAVATKAR, R. Advanced software framework, tools, and languages for the IXP family. *Intel Technology Journal 7*, 4 (Nov. 2004), 64–76.

[6] IBM CORPORATION. *IBM Power Network Processors*, 2000. http://www.chips.ibm.com/products/wired/communications/network_processors.html.

[7] INTEL CORPORATION. *Intel Second Generation Network Processor*, 2002. http://www.intel.com/design/network/products/npfamily/ixp2400.htm.

[8] KOHLER, E., MORRIS, R., CHEN, B., JANNOTTI, J., AND KAASHOEK, M. F. The Click modular router. *ACM Transactions on Computer Systems 18*, 3 (Aug. 2000), 263–297.

[9] KOKKU, R., RICHÉ, T., KUNZE, A., MUDIGONDA, J., JASON, J., AND VIN, H. A case for run-time adaptation in packet processing systems. In *Proc. of the 2nd Workshop on Hot Topics in Networks (HOTNETS-II)* (Cambridge, MA, Nov. 2003).

[10] LUCENT TECHNOLOGIES INC. *PayloadPlus^{TM} Fast Pattern Processor*, Apr. 2000. http://www.agere.com/support/non-nda/docs/FPPProductBrief.pdf.

[11] MALLOY, B. A., LLOYD, E. L., AND SOUFFA, M. L. Scheduling DAG's for asynchronous multiprocessor execution. *IEEE Transactions on Parallel and Distributed Systems 5*, 5 (May 1994), 498–508.

[12] MEMIK, G., AND MANGIONE-SMITH, W. H. NEPAL: A framework for efficiently structuring applications for network processors. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)* (Anaheim, CA, Feb. 2003).

[13] MOTWANI, R., AND RAGHAVAN, P. *Randomized Algorithms*. Cambridge University Press, New York, NY, 1995.

[14] PLISHKER, W., RAVINDRAN, K., SHAH, N., AND KEUTZER, K. Automated task allocation for network processors. In *Proc. of Network System Design Conference* (Oct. 2004), pp. 235–245.

[15] RAMASWAMY, R., WENG, N., AND WOLF, T. Application analysis and resource mapping for heterogeneous network processor architectures. In *Proc. of Third Workshop on Network Processors and Applications (NP-3) in conjunction with Tenth International Symposium on High Performance Computer Architecture (HPCA-10)* (Madrid, Spain, Feb. 2004), pp. 103–119.

[16] RAMASWAMY, R., WENG, N., AND WOLF, T. Analysis of network processing workloads. In *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)* (Austin, TX, Mar. 2005), pp. 226–235.

[17] SHAH, N., PLISHKER, W., AND KEUTZER, K. NP-Click: A programming model for the intel IXP1200. In *Proc. of Second Network Processor Workshop (NP-2) in conjunction with Ninth International Symposium on High Performance Computer Architecture (HPCA-9)* (Anaheim, CA, Feb. 2003), pp. 100–111.

[18] TAYLOR, D. E., TURNER, J. S., LOCKWOOD, J. W., AND HORTA, E. L. Dynamic hardware plugins: Exploiting reconfigurable hardware for high-performance programmable routers. *Computer Networks 38*, 3 (Feb. 2002), 295–310.

[19] TEJA TECHNOLOGIES. *TejaNP Datasheet*, 2003. http://www.teja.com.

[20] WENG, N., AND WOLF, T. Profiling and mapping of parallel workloads on network processors. In *Proc. of The 20th Annual ACM Symposium on Applied Computing (SAC)* (Santa Fe, NM, Mar. 2005), pp. 890–896.

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