Chapter 7

Workloads for Programmable Network Interfaces

Patrick Crowley, Marc E. Fiuczynski, Jean-Loup Baer and Brian Bershad
Department of Computer Science & Engineering, University of Washington

Key words: programmable network interfaces, superscalar microprocessor, multithreaded microprocessor, simultaneous multithreaded microprocessor, networking applications

Abstract: Network equipment vendors are increasingly incorporating a programmable microprocessor on network interfaces to meet the performance and functionality requirements of present and emerging applications in parallel with market demand. This study identifies some properties of programmable network interface (PNI) workloads and their execution characteristics on modern high-performance microprocessor architectures including aggressive superscalar, fine-grain multithreaded, and simultaneous multithreaded architectures.

1. INTRODUCTION

As networks have evolved and expanded technologically, their role and importance have also increased considerably. The role of the network interface (NI), which allows a computer system to exchange messages with other systems connected to the network, has grown accordingly. To meet the functionality and performance requirements of present and emerging network applications there has been a renewed trend by vendors to use *programmable* microprocessors on *network interfaces* (PNI) that can be customized with domain-specific software. This trend is enabled by significant cost/performance improvements in both embedded microprocessor and memory technology.

Network interfaces are used both at the “edge” of the network in host systems, and in the “middle” of the network in switches or routers. The latter have used microprocessors on the NI for some time now to improve scalability and overall system performance. Despite this moderate acceptance, the utility of using a programmable microprocessor on a NI for host systems has been debated for over a decade. For the sake of brevity in this study we will assert that vendors *are* developing powerful PNIs[1], which allow them to add software with unique, value-added features to their products well into the development cycle or after those products have been shipped. In contrast, vendors using traditional ASIC – or fixed-function – technology must lock down feature sets long before a product release date. Consequently, the use of PNIs enables networking equipment vendors to develop differentiated, cutting-edge products in parallel with market demand.

We observe that the processors used in PNIs are architecturally similar to the processors found in workstations, often lagging only a few generations behind. Consequently, one can expect that today’s high-performance processor technology may be incorporated into tomorrow’s programmable network interfaces. However, the architectural design of processors for the PC/workstation market is driven by a well-defined application workload (e.g., SPEC, databases, and office productivity applications). In contrast, the application workload for programmable NIs is different and not yet tied to a relatively small set of processor instruction-set architectures as in the PC/workstation market.

In this chapter, we explore the properties of PNI workloads and their execution characteristics on modern high-performance architectures. Our contributions are as follows. First, we describe and characterize a representative suite of PNI applications, and discuss how these applications form a workload in the context of a programmable NI. In particular, we observe that there exists a high degree of packet-level parallelism in the PNI application workload. Second, we present a preliminary evaluation of these benchmarks on a contemporary high-performance architecture as well as two types of next-generation multithreaded processors. Specifically, we investigate a range of modern processor configurations to discover the computational throughput necessary to sustain application-specific packet processing at gigabit network rates.

The chapter is organized as follows. Section 2 describes the workloads we consider for programmable network interfaces. Section 3 discusses our

[1] Recent PNIs for host systems include 3COM’s R990, Unisys’ IntelliLAN, Alteon’s Tigon2, and many more. Similarly, various LAN/WAN equipment vendors and startup companies are incorporating PNIs. The PNIs use powerful, embedded microprocessors, such as ARM, MIPS, and PowerPC running at clock speeds of 233-500MHz.

simulation methodology and experimental results. The chapter concludes with final comments in Section 4.

2. WORKLOADS FOR PROGRAMMABLE NETWORK INTERFACES

The primary purpose of a NI is to move messages efficiently between the system and the network media. The NI's basic unit of work is thus a message. With processing capacity improvements vastly outpacing link rate improvements, it is possible to perform some amount of application-specific processing on a per packet basis. Consequently, a PNI is no longer just a passive device in the computer system that transfers bytes to and from the network; rather, it is a specialized communications processing facility that is itself software-driven.

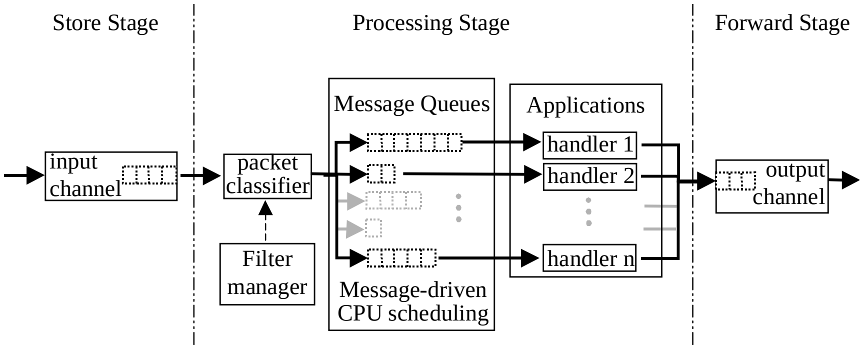

Figure 1. The Store-Process-Forward stages of messages flowing through a programmable NI

The workload for a PNI can be decomposed into two main components: 1) *application-specific packet processing routines* and 2) an *execution environment* that drives the overall operation of the PNI and invokes the aforementioned routines. The rest of this section first presents a high-level description of the PNI execution environment, and then identifies and characterizes applications for PNIs.

2.1 PNI Execution Environment

The execution environment for a PNI can be characterized as having a basic fundamental store-process-forward structure. The *store* and *forward* stages simply transfer message data into and out of the NI's buffer memory. The *process* stage invokes application-specific handlers based on some matching criteria applied to the message. *Figure 1* illustrates this structure as a pipeline of stages.

As messages arrive on the input channel, as depicted in *Figure 1*, they are first *stored* in the NI's buffer memory. Messages are then classified and dispatched to some application-specific handler function to be *processed*. Finally, the messages may then be *forwarded* on to an output channel. To achieve high throughput and low latency it is important that messages are pipelined through these stages. Thus, the goal is to sustain three forms of concurrent operation: message reception from the input channel, message processing on the NI, and message transmission to the output channel. It is the task of the system software to schedule these concurrent operations such that each pipeline stage is occupied. Further, we believe that for the majority of applications these messages may be processed independently, and therefore there exists a high degree of packet-level parallelism on a PNI that may be exploited with the appropriate processor architecture.

The next subsection describes several example applications that would run in the context of a PNI's execution environment.

2.2 Applications

There are three main application domains for programmable NIs, which can be roughly categorized into *Server* NI software, *Web Switching* software [1], and *Active Networking* software [4][6]. The first two are product niches of their own right and are driving the development of programmable NIs. The latter is a research area that is pursuing the ability to dynamically deploy innovative network services rapidly and securely. With computation becoming less expensive, other NI resident applications are emerging as well.

In the context of these application domains we have identified a set of application-specific packet processing (ASPP) routines that we intend to use as workloads in the context of a PNI execution environment. Table 1 briefly describes a representative set of present and emerging ASPPs in ascending order of per packet processing requirements.

Table 1. A representative set of applications used in both LAN/WAN networking equipment and host network adapters listed in ascending order of per packet processing requirements.

Applications	Description
Packet Classification/Filtering	Claim/forward/drop decisions, statistics gathering, and firewalling.
IP Packet Forwarding	Forward IP packets based on routing information.
Network Address Translation	Translate between globally routable and private IP packets. Useful for IP masquerading, virtual web server, etc.
TCP connection management	Traffic shaping within the network to reduce congestion.
TCP/IP	Offload TCP/IP processing from Internet/Web servers.
Web Switching	Web load balancing and proxy cache monitoring.
Virtual Private Network (VPN) IP Security (IPSec)	Encryption (DES) and Authentication (MD5)
Data Transcoding	Converting a multimedia data stream from one format to another within the network.
Duplicate Data Suppression	Reduce superfluous duplicate data transmission over high cost links.

The applications in *Table 1* can be divided into two categories according to the amount of the packet that is processed. The first six applications process a limited amount of data within the protocol headers of the packet, and their processing requirements are independent of the packet's overall size. However, the applications tend to maintain state tables in complex data structures that need to be searched or accessed on a per packet basis. In contrast, the last three applications (those beneath the dashed line) compute over all of the data contained in a packet, and therefore require a significant amount of processing capacity to process packets at the network link rate.

An important observation is that for these workloads, the processing of one packet is largely, and usually entirely, independent of the processing of any other packet. Consequently, packets may be processed in parallel. This packet-level parallelism, like other forms of parallelism, can be exploited to achieve significant performance gains.

The execution time devoted to each of these applications on a per packet basis needs to be short in order to sustain message throughput at high link rates (e.g., 100Mbps - 10Gbps). The typical run length of these applications is on the order of hundreds of nanoseconds to tens of microseconds. We envision that PNIs in switches/routers and host adapters will support some combination of these applications concurrently. Consequently, we believe that it is the combined behavior of these applications running in the previously described execution environment that yields an interesting and compelling workload at both the system software and architectural level.

3. EVALUATION

In order to examine each benchmark and investigate architectural alternatives, we use cycle-accurate processor simulations. Via simulation, we are able to gather many descriptive statistics for each benchmark, as well as examine performance for a variety of experimental architectures. In the interest of space, we consider two applications as workloads: one that processes only a portion of a packet, and another that performs computation over the entire packet.

First, we consider the packet classification routine ip4lookup. This application performs address-based lookups in a tree data structure and is a component of conventional layer-3 switches and routers. In test runs of ip4lookup we perform 160,000 lookups into a routing table of 1000 entries. This workload is representative of a large corporate (or campus) intranet. Next, we consider MD5, a message digest application used to uniquely identify messages. MD5 computes a unique signature over the data in each packet and is used for authentication. In test runs of MD5 we digest 100 1500 byte messages.

3.1 Simulation Environment

Our simulation infrastructure provides a cycle accurate simulation of a modern high-performance, out-of-order, superscalar microprocessor core and memory system. By varying the microprocessor and memory system configurations, we can investigate a number of architectural alternatives. The simulator runs on the Alpha/OSF platform and simulates a superset of the Alpha instruction set. Modern superscalar processors attempt to dynamically find and exploit the maximum amount of instruction-level parallelism (ILP) within a single thread of execution.

Our infrastructure also supports both fine-grained (FGMT) and simultaneous multithreading (SMT) [5]. The multithreading support extends the core out-of-order, superscalar microprocessor by adding support for multiple hardware thread contexts.

In an FGMT architecture, instructions may be fetched and issued from a different thread of execution each cycle. The FGMT architecture attempts to find and exploit all available ILP within a thread of execution, but can also mitigate the effects of a stalled thread, a thread with little ILP, by quickly switching to another thread. In this way, ideally, overall system throughput is increased. The architecture simulated here chooses threads to issue in round-robin fashion.

In an SMT architecture, instructions are fetched and issued from multiple threads each cycle. As a result, overall instruction throughput can be increased according to the amount of ILP available *within* each thread and the amount of thread-level parallelism (TLP) available *between* threads. The SMT architectures we simulate in this study fetch eight instructions each cycle from a maximum of two threads.

Unless noted otherwise our simulated microprocessors have the following cache configuration: separate 32K instruction and data L1 caches, a 512K combined L2 cache. Most simulated latencies and processor structures closely model those found in an Alpha 21264 out-of-order processor.

3.2 Experiments and Preliminary Results

The basic question we wish to answer with our experiments is as follows: "What are the appropriate processor and memory features needed to support application-specific packet processing at the network link rate?" We investigate this by asking the following questions for the application-workload described in Section 2:

- **Sensitivity to cache organization:** by varying cache parameters within the simulated architecture (i.e., size and associativity), we can determine if these workloads are compute or memory bound. Also, we gain a sense of how the lack of locality of message data streaming through the PNI affects applications with a high degree of data locality.
- **Degree of ILP:** by varying the number of functional units available within a single thread context, we gain a sense of the available ILP present in the workload.
- **Degree of TLP:** by varying the number of thread contexts and fixing the number of functional units, we can gauge the degree of thread-level parallelism in the workload.

3.3 Cache Sensitivity

The instruction usage of a program describes that program in broad strokes. By considering the relative importance, i.e., usage, of loads, stores, control flow instructions, etc., for a given application, insight may be gained as to the degree to which that application will rely on the memory system for overall performance. A general description of the instruction set usage for these benchmarks is reported in Table 2. An examination of the instruction and data cache accuracy for ip4lookup and MD5 suggests that these applications are compute bound as both the instruction and data working sets fit within L1 caches of size 32K and greater (miss rates less than 1%). While none of our applications tax the memory hierarchy in isolation, it is likely that cache accuracy and organization will be important when more than one program is running concurrently. As noted below, we will consider this fact as we evaluate the simultaneous execution of these workloads.

Table 2. General Instruction Set Characteristics.

Application	Insts Executed per Message	Loads/Stores (%)	Ctrl Flow (%)	Other (%)
ip4lookup	120	18.6	12.1	69.3
MD5	23K	10.3	0.6	89.1

We have previously observed that these compute bound PNI workloads can exhibit a high degree of packet-level parallelism. We will now consider the degree to which ILP and TLP are able to exploit packet-level parallelism.

3.4 ILP and TLP Characteristics

3.4.1 Investigating ILP

Aggressive superscalar microprocessors are adept at discovering ILP in programs. It is unlikely, however, that exploiting ILP alone will permit a microprocessor to capitalize on the larger-grain packet-level parallelism available in these workloads. As an experiment, we varied the number of functional units by adding one at a time on five simulated superscalar machines with clock rates ranging from 100MHz to 500MHz. We then measured the performance of these machines on each benchmark individually. The results of these experiments for runs of ip4lookup and MD5 are found in *Figure 2*. The black horizontal lines indicate the bandwidth requirements for the indicated link rates.

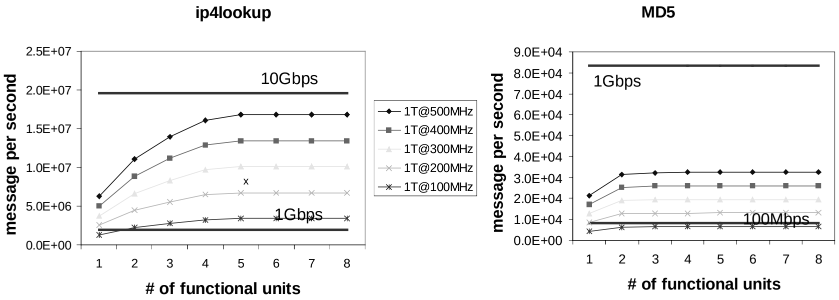

Figure 2. Performance of ip4lookup and MD5 ASPPs on a superscalar microprocessor (message size = 1500 bytes)

There are a number of observations to be made.

- The performance of these compute-bound programs scales linearly with clock rate.
- For ip4lookup, performance continues to improve sub-linearly as the first five sets of additional functional units are added. The performance of MD5 stagnates after two additional sets of functional units. This indicates that ip4lookup has nearly three times as much available ILP than MD5.
- Nearly every processor configuration is capable of executing ip4lookup fast enough to sustain a 1 Gbps link rate, while none can sustain ip4lookup at 10 Gbps.
- At least a 200MHz processor with 1 integer and 1 load/store functional unit are needed to support MD5 at a line rate of 100Mbps. None of these architectures can support MD5 at 1Gbps.

While increasing the number of functional units does improve performance up to a point for these workloads, it does not achieve the linear speedup available with packet-level parallelism. These results suggest, therefore, that ILP, and a microprocessor aggressively exploiting it exclusively, is insufficient to capitalize on packet-level parallelism.

3.4.2 Investigating TLP

We have seen that an aggressive superscalar processor exploiting ILP is unable to capitalize on the large-grain packet-level parallelism available in these workloads. It is conceivable that the fine-grain thread-level parallelism exploited by the FGMT and SMT architectures will be able to do so. The fundamental observation is that each hardware context could simultaneously process independent packets. To investigate this conjecture, we varied the number of thread contexts for each flavor of multithreaded processor and measured the performance of our PNI workloads independently. As each context was added, one additional functional unit was added to support the available thread-level parallelism. The simulated FGMT results are found in *Figure 3*.

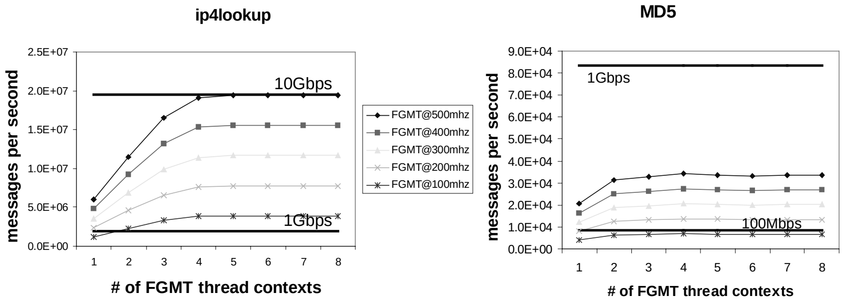

Figure 3. Performance of ip4lookup and MD5 ASPPs on a fine-grained multithreaded (FGMT) microprocessor (message size = 1500 bytes).

Regarding the FGMT architecture, a number of observations are possible.

- The performance of these compute-bound programs scales linearly with clock rate, as expected.
- ip4lookup performance improves relative to the superscalar processor, and MD5 is unchanged.
- The line rates supported by FGMT are essentially those supported by the superscalar processor.

The reason why ip4lookup improves slightly with the FGMT architecture, and MD5 does not, is not immediately obvious. Further investigation reveals that ip4lookup has a greater percentage of short latency computations that benefit from rapid context switches whereas MD5 has mostly long latency operations that require more than a few cycles to complete. Still, ip4lookup does not experience a significant speedup with FGMT. Since only a single thread issues each cycle and these applications have limited ILP, most of the machine resources (i.e., issue slots) go unused each cycle when there are more functional units available than can be used by a single thread of execution (5 for ip4lookup and 2 for MD5).

The results from the SMT simulations are given in *Figure 4*, and we make the following observations.

- The performance of these compute-bound programs scales linearly with clock rate *and the number of thread contexts*.
- Most configurations can support ip4lookup at 1Gbps, and thirteen can sustain a 10Gbps link.

- Most configurations can support MD5 at 100Mbps, and eleven can sustain a 1Gbps link.

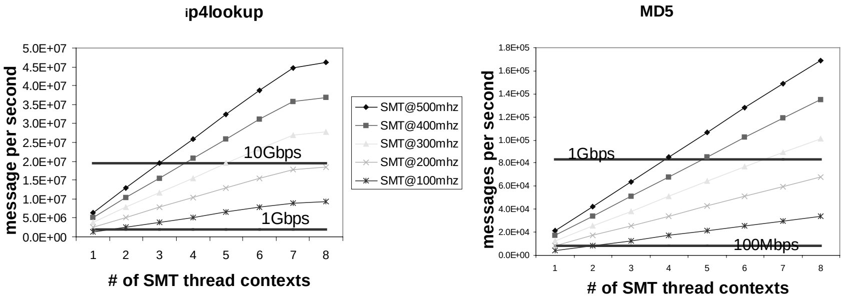

Figure 4. Performance of ip4lookup and MD5 ASPPs on a simultaneously multithreaded (SMT) microprocessor (message size = 1500 bytes).

Also note that these results suggest, for example, that a 500MHz SMT with four thread contexts has roughly the same MD5 performance as 300MHz superscalar processor with seven thread contexts. These tradeoffs are particularly important for SMT, where high clock rates are generally at odds with size and complexity [5]. The most relevant high-level conclusion to be drawn from these preliminary results is that thread-level parallelism, combined with the ability of SMT to keep all machine resources utilized by issuing from multiple threads each cycle, is capable of exploiting the packet-level parallelism of these workloads.

3.5 Additional Experiments

Our future work will include the following.

- **Chip Multiprocessor Performance:** One would expect a chip multiprocessor to be able to exploit packet level parallelism in a fashion similar to SMT. Work is currently underway to investigate this architecture.
- **Multiprogramming workloads:** The results presented here were gathered executing each application in isolation. While scenarios exist in which an entire machine is dedicated to a single application, a more common scenario includes machines whose normal workload includes several concurrent applications. We are currently porting the SPINE programmable NI operating system [2, 3] to our simulation

environment to evaluate such “multiprogramming” workloads based on the applications described in Section 2.

- **Sensitivity to memory contention:** by varying the memory access latency, memory bandwidth, and employing memory interleaving, we can gain a sense of how the DMA hardware components (e.g., Ethernet and PCI DMA engines) and the CPU contend for memory.

4. CONCLUSION

In this study we have considered workloads for programmable network interfaces. We observed that these workloads exhibit a high degree of packet-level parallelism in which each message, the basic unit of work, may be processed independently. Our preliminary simulation results suggest that aggressive superscalar and fine-grained multithreaded (FGMT) processors are incapable of exploiting this inherent packet-level parallelism. However, our results do suggest that the fine-grain thread-level parallelism supported by simultaneous multithreaded architectures can exploit packet-level parallelism. As a result, it is possible to achieve performance improvements linear with the number of hardware thread contexts, and thereby enable application-specific processing at the link rates of gigabit networks.

ACKNOWLEDGEMENTS

We would like to thank David Wetherall and Neil Spring for providing us with their duplicate data suppression system source code and Alec Wolman for providing similar access to his TCP connection management software. Finally, particular thanks to Sujay Parekh and Josh Redstone for contributing their SMT expertise.

REFERENCES

[1] Alteon. WEBWORKING: Networking with the Web in mind. Alteon WebSystems, White Paper: www.alteon.com/products/white_papers/webworking, San Jose, California, May 3rd 1999.

[2] M.E. Fiuczynski, R.P. Martin, B.N. Bershad, and D.E. Culler. SPINE: An Operating System for Intelligent Network Adapters. University of Washington, TR 98-08-01, Seattle, August 1998.

[3] M.E. Fiuczynski, R.P. Martin, T. Oha, and B.N. Bershad. SPINE - A Safe Programmable and Integrated Network Environment. *Proceedings of the Eight ACM SIGOPS European Workshop*, pp. 8. Sintra, Portugal, September 1998.

[4] J.M. Smith, K.L. Calvert, S.L. Murphy, H.K. Orman, and L.L. Peterson. Activating Networks: A Progress Report. IEEE Computer Magazine vol. 32, no. 4, pp. 3-41, April 1999.

[5] D. Tullsen, S. Eggers, and H. Levy. Simultaneous Multithreading: Maximizing On-Chip Parallelism. *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pp. 392-403. Santa Margherita Ligure, Italy, June 1995.

[6] D.J. Wetherall, U. Legedza, and J. Guttag. Introducing New Internet Services: Why and How. IEEE Network Magazine , July/August 1998.