Evaluating Network Processors Using NetBench

GOKHAN MEMIK
Northwestern University
and
WILLIAM H. MANGIONE-SMITH
University of California, Los Angeles

The Network Processor market is one of the fastest growing segments of the microprocessor industry today. In spite of this increasing market importance, there does not exist a common framework to compare the performance of different Network Processor designs. Our primary goal in this study is to fill this gap by creating the NetBench benchmarking suite. NetBench is designed to represent Network Processor workloads. It contains 11 programs that form 18 different applications. The programs are selected from all levels of packet processing: Small, low-level code fragments as well as large application-level programs are included in the suite. These applications are representative of the Network Processor applications in the market. Using the SimpleScalar simulator to model an ARM processor, we study these programs in detail and compare key characteristics, such as instructions per cycle, instruction distribution, cache behavior, and branch prediction accuracy with the programs from MediaBench. Using statistical analysis, we show that the simulation results for the programs in NetBench have significantly different characteristics than programs in MediaBench. Finally, we present performance measurements from Intel IXP1200 Network Processor to show how NetBench can be utilized.

Categories and Subject Descriptors: C.4 [**Computer Systems Organization**]: Performance of Systems—*Measurement techniques*; C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems—*Real-time and embedded systems*; I.6 [**Computing Methodologies**]: Simulation and Modeling—*Simulation Support Systems*

General Terms: Design, Measurement, Performance, Standardization

Additional Key Words and Phrases: Embedded systems, benchmarking, network processors

1. INTRODUCTION

Emerging applications in the networking field demand increasingly higher network bandwidths. In addition, new applications and protocols require the

Authors' addresses: Gokhan Memik, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60208; email: memik@ece.northwestern.edu; William H. Mangione-Smith, Department of Electrical Engineering, University of California, Los Angeles, Los Angeles, CA 90095; email: billms@ee.ucla.edu.

network to do more than just deliver packets. Instead, these applications have requirements, such as quality of service guarantees, secure transmission of data, and intelligent/dynamic routing and switching. These operations require high processing power. This set of features, coupled with the higher network link speeds, puts a heavy demand on the network processing elements.

Traditionally, embedded processors in networks are either custom-designed ASIC chips or variations of general-purpose processors. Both schemes have their advantages and disadvantages. ASIC chips have better performance, but they have higher manufacturing costs and lack the flexibility of programmable processors. If there is a change in the protocol or application, it is hard to reflect the change in the ASIC design. General-purpose processors, on the other hand, are not optimized for networking applications and, hence, do not provide satisfactory performance for most of the applications.

Network Processors (NPUs) eliminate the drawbacks of general-purpose processors and ASIC designs by combining the flexibility of general-purpose programmable processors and performance of ASIC chips.

Soon after their introduction [MMC Networks], the NPU market became one of the fastest growing segments of the microprocessor industry. In the last 2 years, more than 40 new vendors have announced their NPU architectures [e.g., Improv Inc.; XStream Corp.]. Although these processors aim at the same application domains, they vary widely in their architectural designs. Hence, there is a tremendous need to evaluate the performances of these different designs.

A designer of a product should know the type of applications, based on marketing requirements, for which the processor is optimized. Similarly, customers benefit from benchmarks by selecting the product that gives the best performance for the applications they consider important (when benchmarks are aligned with commercial workloads). In spite of the rapid increase in use of NPUs, there still does not exist a common framework or methodology for evaluating them. Our goal in this study is to fill this gap. Specifically, our contributions in this paper are

- Creating a benchmarking suite by defining a set of applications that are common for NPUs;
- Investigating several characteristics of these networking applications to understand their nature;
- Comparing the characteristics of these applications with the applications from MediaBench [Lee et al. 1997a];
- Reporting results for several different cache and branch prediction configurations using an accurate StrongARM simulator [Burger and Austin 1997] to guide designers in the selection of architectural parameters;
- Demonstrating how NetBench can be utilized by providing a performance measurement of Intel IXP1200 Network Processor [Halfhill 1999], a representative NPU product currently available in the market.

This paper is organized as follows. In the next section, we summarize architectural characteristics of NPUs. Section 3 discusses the related work. In

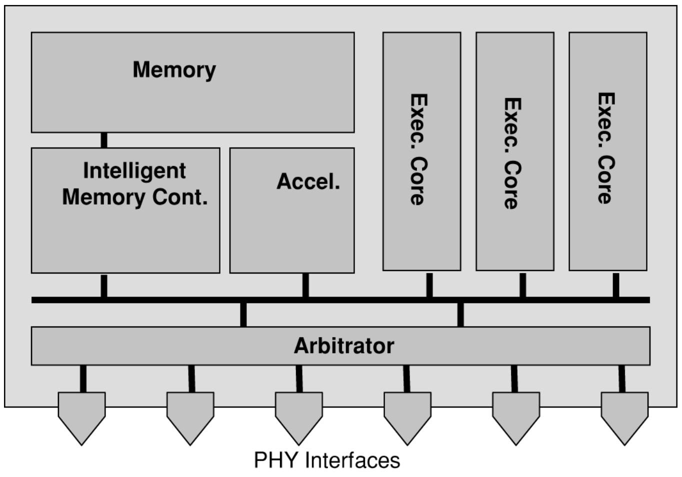

Fig. 1. A generic NPU design.

Section 4, we present the applications in NetBench. Applications in NetBench are compared with the MediaBench applications in Section 5. In Section 5.3, we present experimental results for Intel IXP1200 simulations. Section 6 concludes the paper with a summary.

2. NETWORK PROCESSOR CHARACTERISTICS

This section discusses important characteristics of NPUs, such as on-chip caches used in the processors, and techniques for hiding memory latency.

NPUs vary significantly in their design methodologies. Designs span from single-core superscalar processors (Broadcom SiByte) to system-on-chip designs containing more than 40 execution cores (EZChip). Their major design methodologies can be grouped into three categories: VLIW-based processors [Improv Inc.], SMT-based processors [XStream Corp.] and chip multiprocessor systems [C-Port Corp.; Halfhill 1999]. Most of the processors contain multiple execution cores to take advantage of the data parallelism that exists in many networking applications. In addition, most of the NPUs modify RISC-like ISAs with instructions that efficiently perform operations required by networking applications. In addition, they employ special-purpose elements to improve the execution efficiency. These elements are either packet-oriented memory controllers or accelerators that perform certain operations (e.g., table lookup) that occur frequently in the applications and are not efficient to implement in the execution cores.

Figure 1 presents a generic NPU design. The processor contains a set of execution cores, accelerators, and on-chip secondary memory (level-2 cache). The processors communicate through a global shared bus. Each execution core is employed with local level-1 instruction and data caches. The execution cores can be very simple, such as an ALU enhanced with local registers (e.g., EZChip), or they can be complex, such as a modified MIPS core (e.g., PMC-Sierra RM9000).

The most common property among different NPUs is multithreading. Almost all the NPUs available in the market today employ a variation of a multithreading technique (e.g., Clearwater CNP810SP, Intel IXP, IBM Rainer, MMC nP7510, Motorola C-5). Clearwater CNP810SP can execute instructions from eight different threads. Intel IXP family processors, on the other hand, execute instructions from a single thread, but have hardware support for single-cycle thread switching between four active threads.

The size of the level 1 instruction and data caches employed in the execution cores also varies among different designs, but most of the designs employ caches of 4 to 16 KB. For example, Intel IXP 2800 has a 4 KB instruction store, while IBM Rainier and Lexra NetVortex have 8 KB and 16 KB level-1 instruction caches, respectively.

3. RELATED WORK

NPUs are a class of programmable IC's based on SOC (system-on-chip) technology that implements communication-specific functions more efficiently than general-purpose processors. Crowley et al. [2000] evaluate different design mechanisms for NPU. They measure the performance of a VLIW-based, an SMT-based, a fine-grain multithreaded multiprocessor, and a single-chip multiprocessor. For their study, they use a subset of applications that are in NetBench. These applications are, however, not available to the public.

Benchmarks play a major role in any product design process. SPEC [Standard Performance Evaluation Council] benchmarks have been well accepted and used by several processor manufacturers and researchers to measure the effectiveness of their design. Other fields have useful benchmarking suites designed for the specific application domain: TPC [Transaction Processing Council] for database systems, and SPLASH [Woo 1995] for parallel machine architectures.

The need for a benchmarking suite in the NPU area has been pointed out by several researchers. Nemirovsky [2000] discusses the requirements and challenges of a benchmarking suite for NPUs. He defines a set of metrics to be used with any benchmarking suite and draws the guidelines for defining a benchmark. Currently, three benchmarking suites contain applications that might be used in NPUs. EEMBC has a benchmarking suite designed for embedded processors [EEMBC], which contains three networking applications. However, these applications are control-plane tasks (such as a shortest path algorithm) and do not form a basis for measuring the effectiveness of NPUs. MediaBench [Lee et al. 1997a] also contains security and communication applications that might be used by some of the NPUs. These applications perform translations between different data formats and, therefore, are not representative of most NPU applications. CommBench [Wolf and Franklin 2000] is designed for telecommunications NPUs. It contains four header-processing applications, which effectively represent tasks related to traditional Ipv4 routing, and four payload applications. The payload applications are jpeg, cast (CAST-128 block cipher algorithm), reed (Reed-Solomon Forward Error-Correction algorithm), and zip (Lempel-Ziv (LZ77) compression algorithm). Although these applications are representative of telecommunications NPUs, the selected applications are limited to this type of NPUs and do not represent applications employed by majority of NPUs. Similarly, the Network Processor Forum (NPF) established a set of applications that is used as a benchmarking suite [Network Processor Forum]. However, all four applications are routing-related applications.

4. NETBENCH PROGRAMS

In this section, we present the applications in NetBench. Any benchmarking suite should be representative of the applications in the domain the benchmark is designed for. This was the most important criterion in our selection of the applications.

NPU applications contain a large variety of tasks from traditional routing and switching tasks to much more complicated applications containing intelligent routing and switching decisions. Therefore, any benchmarking suite attempting to represent the applications on NPUs should consider all levels of a networking application. Instead of using the traditional 7-level OSI model for categorizing the applications, we have used a three-level categorization. These levels are:

- Low- or micro-level routines containing operations nearest to the link or operations that are part of more complex tasks;
- Routing-level applications, which are similar to traditional IP level routing and related tasks; and
- Application-level programs, which have to parse the packet header and sometimes a portion of the payload and make intelligent decisions about the destination of the packet.

This categorization is performed by considering the complexity of the application, instead of the specific task it is performing. Hence, it is a better categorization for the designers of the NPUs than the seven-layer categorization. For example, as we will show in Section 5, applications parsing the packet data have different characteristics than the applications that only parse header information regardless of the task they are performing. Note that these three categories cover all the levels of a traditional seven-level reference model and, hence, present an inclusive characterization of all networking applications. In the following, we list the applications in NetBench according to the category they belong:

4.1 Micro-Level Programs

CRC: The CRC-32 checksum calculates a checksum based on a cyclic redundancy check as described in ISO 3309 [International Organization for Standardization 1984]. CRC-32 is used in Ethernet and ATM Adaptation Layer 5 (AAL-5) checksum calculation. The code is available in the public domain [Cell-Relay].

TL: TL is the table lookup routine common to all routing processes. We have used radix-tree routing table, which was used in several UNIX systems. The code segment is from FreeBSD operating system [FreeBSD Project].

4.2 Routing-Level Programs

These programs make a decision depending on the source or destination IP address of the packet.

ROUTE: Route implements IPv4 routing according to RFC 2644 [Senie]. When a router receives a packet, it has to decide the next network hop. Route implements the table lookup along with internet checksum (for the header). It makes the necessary changes in the header (for example, the Time-To-Live value), fragments the packet if necessary, and forwards it. The code is from the FreeBSD operating system [FreeBSD Project].

DRR: Deficit-round robin (DRR) scheduling [Shreedhar and Varghese 1995] is a scheduling method implemented in several switches today. In DRR, all the connections through the router have separate queues. Using these queues, the router tries to accomplish a fair scheduling by allowing same amount of data to be passed from each queue. The implementation is based on the algorithm by Shreedhar and Varghese [1995].

IPCHAINS: IPCHAINS is a firewall application that checks the IP source of each of the incoming packet and decides either to pass the packet through the firewall (accept), to deny the packet (deny), to modify it (masq), or to reject the packet and send information to the sender (reject). The decision is based on rules given by the user. The implementation is from Rustcorp Inc. [Russell].

NAT: Network Address Translation (NAT) is a common method for IP address management. NAT operates on a router, usually connecting two networks, and translates the private (not globally unique) addresses in the internal network into legal addresses before packets are forwarded onto the public network. Hence, for any departing packet, the source IP on the packet should be changed. Similarly, the destination address on any incoming packet should also be modified. The program accomplishing this task uses several routines from FreeBSD operating system [FreeBSD Project].

4.3 Application-Level Programs

These programs are the most time-consuming applications in NetBench because of their processing requirements.

DH: Diffie-Hellman (DH) is a common public key encryption/decryption mechanism. It is the security protocol employed in several Virtual Private Networks (VPNs). The implementation is from RSA Data Security, Inc.

MD5: Message Digest algorithm (MD5) creates a signature for each outgoing packet, which is checked at the destination [Rivest 1992]. The signature is cryptographically secure, hence, if the received packet does not match the signature, then the receiver will assume that the packet is unreliable and discard it. The implementation is from RSA Data Security, Inc.

SNORT: Snort is an open-source network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis and content searching/matching in order to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, and CGI attacks [Roesch]. It uses a user-defined rule set that defines actions for each packet. We have used the default configuration file (snort.conf) that contains 886 rules for the snort-nids application. The logging mode (snort-l) stores information about the packets.

SSL: SSL (secure sockets layer) is the secure transmission package used in several UNIX systems. SSL is used by applications, such as ssh [Barrett and Silverman 2001] and sftp [OpenBSD Project], which perform secure communication over insecure public networks. The ssl implementation we used is from the OpenSSL Project. The application interface is modified to perform three different computations with varying strength: weak performs rc4-40 encryption without any digest, medium performs DSA authentication, followed by blowfish encryption and md5 digest for each packet, and strong performs RSA authentication followed by 168-bit key 3-DES and SHA digest for each packet.

URL: URL implements URL-based destination switching, which is a commonly used content-based load balancing mechanism. In URL-based switching, all the incoming packets to a switch are parsed and forwarded according to URL. For example, all image requests might be sent to an image server. This application increases the utility of specialized servers in a server farm. The implementation is based on the description from PMC-Sierra.

4.4 Discussion

NetBench contains 11 applications implemented in C or C++. Many of the available NPUs in the market today have corresponding compilers for high-level languages such as C. For such processors, the implementations can be automatically mapped into the processor. We recommend the usage of the applications in such a framework to establish a fair comparison of the systems. Nevertheless, many NPUs do not provide such compilers and the applications have to be manually coded. Even in systems with compiler support, either the output of the compiler should be optimized or library routines should be used to perform activation of special hardware structures (e.g., table lookup engines). The NetBench applications do not employ such special calls. However, the implementations can be easily modified to perform the necessary operations. First, the user should locate the segments of the application that perform the specific task. Once such locations are identified, the code can be modified to perform the necessary operation in the special structure. For example, consider the route application: the table lookup is performed in the rn_search procedure. If the NPU employs a table lookup engine, this procedure can be modified to activate the engine instead of performing the radix-tree lookup. Although modifications to the code could be done, the state of the code cannot be changed. For example, the user can change the method for storing the routing table. However, the routing table has to include the same information after the modification. In addition, the same sequence of operations have to be performed.

5. PROGRAM CHARACTERISTICS

In this section, we compare several characteristics of NetBench applications with MediaBench [Lee et al. 1997a] applications. MediaBench is designed for multimedia and communication systems, which are, in many ways, similar to NPUs. We have selected MediaBench to compare against NetBench, because of this similarity of the target processor architectures. Although these applications are intuitively similar, we show that the applications for these architectures are significantly different, thus validating the need for a separate benchmarking suite for NPUs.

In Section 5.1, we explain the simulation environment and the applications from MediaBench. Section 5.2 summarizes the experimental results. Section 5.3 presents the results for the Intel IXP1200 simulations.

5.1 Simulation Environment

In order to compare NetBench and MediaBench applications, we have performed several simulations on the SimpleScalar/ARM simulator [Burger and Austin 1997]. SimpleScalar is a cycle-accurate simulator that is capable of simulating a variety of processors. Although it provides detailed information about instruction count and execution time, we had to modify the simulator to gather information about the executed instructions. We simulate a processor model that represents a StrongARM SA-110. We modified the model to represent a common execution core in NPUs. First, we reduced the instruction decode and issue rates to 1. We also changed the cache configurations. The simulated processor has 4 KB direct-mapped level-1 instruction and data caches and a 512 KB unified level-2 cache.

Most of the NetBench applications use IP header traces as input. We have used the traces from Columbia University available in the public domain [The NLANR Project]. In the experiments, the first 10000 packets are read by the applications. All the applications use this trace except the DH and snort programs. DH generates and communicates Diffie–Hellman key pairs and, hence, does not need any packet trace. Snort, on the other hand, uses defcon-8 trace generated by the DEFCON during a capture-the-flag event. This trace also has malicious packets that should be detected by the snort system. The routing table sizes for Drr, Nat, Route, and Tl is set to 128 for the base applications. For the large versions of these applications, the routing table size is set to 1024 entries. The input data sets along with the application codes can be obtained from the NetBench website [Crowley et al. 2000]. The NetBench applications and the arguments used to execute them are summarized in Table I, which also presents important characteristics of the applications such as number of instructions executed and number of cache accesses.

We have simulated 15 programs from MediaBench to make the comparison. The programs, the data sets we have used, and important characteristics of the applications are listed in Table II.

5.2 Experimental Results

In this section, we compare the instruction-level parallelism (ILP), dynamic instruction distribution, branch prediction accuracy, cache hit/miss behavior, and unique data rate of NetBench applications with MediaBench applications. These are the key architectural characteristics of an application and, hence, are used to differentiate between application sets. Each of them is studied in the following subsections.

Table I. NetBench Applications and Their Properties[a]

Appl.	Argument	# Inst [M]	# Cycle [M]	# IL1 Acc [M]	# DL1 Acc [M]	# L2 Acc [M]	bpred Rate [%]
crc	crc 10000	145.8	262.0	219.0	59.8	0.6	0.1
dh	dh 5 64	778.3	1663.1	1009.1	364.7	38.4	21.6
drr	drr 128 10000	12.9	33.5	22.8	7.9	1.1	8.1
drr-l	drr 1024 10000	34.7	80.2	60.1	23.3	5.0	5.6
ipchains	ipchains 10 10000	61.7	160.2	103.9	26.2	3.6	26.1
md5	md5 10000	209.1	474.7	296.8	73.2	11.0	19.6
nat	nat 128 10000	11.4	26.7	17.3	5.6	1.2	17.9
nat-l	nat 1024 10000	33.2	74.2	55.0	21.1	5.1	8.5
rou	route 128 10000	14.2	32.0	23.3	7.1	0.9	11.1
rou-l	route 1024 10000	36.8	81.7	62.6	22.8	5.0	7.6
snort-l	snort −r defcon −n 10000 −dev −l./log −b	343.0	925.6	515.0	132.2	33.4	36.7
snort-n	snort −r defcon −n 10000 −v −l./log −c sn.cnf	545.9	1654.1	893.7	219.7	56.2	28.0
ssl-m	openssl NetBench medium 10000	2718.2	5367.0	3260.1	989.7	142.8	35.7
ssl-s	openssl NetBench strong 10000	3616.1	8727.5	4453.4	1383.3	426.8	36.1
ssl-w	opensll NetBench weak 10000	329.0	832.1	441.1	152.0	31.8	50.8
tl	tl 128 10000	6.9	15.7	11.8	3.9	0.7	5.9
tl-l	tl 1024 10000	30.3	67.1	52.2	19.9	4.7	5.1
url	url small_inputs 10000	497.0	956.7	768.9	249.1	10.0	32.8
Mean		523.6	1190.8	681.5	209.0	43.2	19.9

[a] *Arguments* are the execution arguments, # *inst* is the number of instructions executed, # *cycle* is the no. of cycles required, # *il1 (dl1) acc* is the no. of accesses to the level-1 instruction (data) cache, # *l2 acc* is the level-2 cache accesses, and *bpred rate* is the branch prediction rate for not taken strategy.

5.2.1 *Instruction Level Parallelism.* The first characteristic we explore is the instruction-level parallelism (ILP) measured in instructions per cycle (IPC). It is well known that the networking applications tend to exhibit data-level parallelism, because, in most cases, the packets are independent of each other and, hence, can be processed in parallel. However, dependency between the instructions that process the same data is not studied in detail. We first study this characteristic in a realistic environment. Figure 2 presents the instructions per cycle values for NetBench and MediaBench applications. The average IPC value of NetBench applications is 0.437, which is 21.5% lower than the average of MediaBench applications. A statistical analysis of the results is presented in Appendix A, where we show that the NetBench applications have, indeed, a lower IPC value using a 99% confidence interval. This result indicates the complexity in achieving high instruction-level parallelism in networking applications and motivates usage of data-level parallelism in the NPUs.

5.2.2 *Instruction Distributions.* In these simulations, we have counted the number of instruction types in the NetBench and MediaBench applications. The results are summarized in Figure 3. The figure presents the number of instructions executed from each of the major instruction categories. The two benchmarking suites differ in almost all instruction categories, but we concentrate on the load/store and conditional branch operations, because they are

Table II. MediaBench Applications and Their Properties[a]

Appl.	Argument	# Inst [M]	# Cycle [M]	# IL1 Acc [M]	# DL1 Acc [M]	# L2 Acc [M]	bpred Rate [%]
adp-c	rawcaudio < clinton.pcm > out.adpcm	8.1	10.2	9.3	1.1	0.0	49.6
adp-d	rawdaudio < clinton.adpcm > out.pcm	6.5	8.5	7.7	1.1	0.0	49.9
epic-e	epic test_image.pgm –b 25	60.7	108.5	89.2	18.9	0.3	45.3
epic-u	unepic test_image.pgm.E	10.1	19.0	14.8	1.9	0.2	28.0
g72-e	decode -4 –l -f clinton.g721	369.2	753.5	510.4	118.3	13.4	47.7
g72-d	encode -4 –l -f clinton.pcm	386.9	785.9	536.8	125.0	13.7	46.7
gsm-t	toast –fpl clinton.pcm	294.4	436.5	333.8	92.4	1.7	11.7
gsm-u	untoast –fpl clinton.pcm.gsm	102.7	162.3	141.0	19.8	0.4	32.6
jpg-c	cjpeg –dct int –progressive –opt testimg.ppm	16.0	32.3	23.3	5.8	0.5	14.5
jpg-d	djpeg –dct int –ppm –opt testimg.jpg	4.2	7.6	5.3	1.8	0.1	25.4
mpg-e	mpeg2encode options.par out.m2v	158.3	342.1	252.1	57.6	5.6	34.5
mpg-d	mpeg2decode –bmei16v2.m2v –r –f –o0 rec	1032.3	1830.8	1293.9	354.2	23.8	20.5
pegwit-e	pegwit –e my.pub pgtest.plain pegwit.enc	19.1	35.4	23.9	6.6	0.8	47.4
pegwit-d	pegwit –d pgtest.enc pegwit.dec	38.9	72.9	48.3	13.3	1.7	46.1
rasta	rasta –A –J –S 8000 –n12 –f weights.dat	16.2	42.2	23.8	7.4	1.2	31.3
Mean		168.2	309.8	220.9	55.0	4.2	35.4

[a]See Table I for explanation of abbrevation.

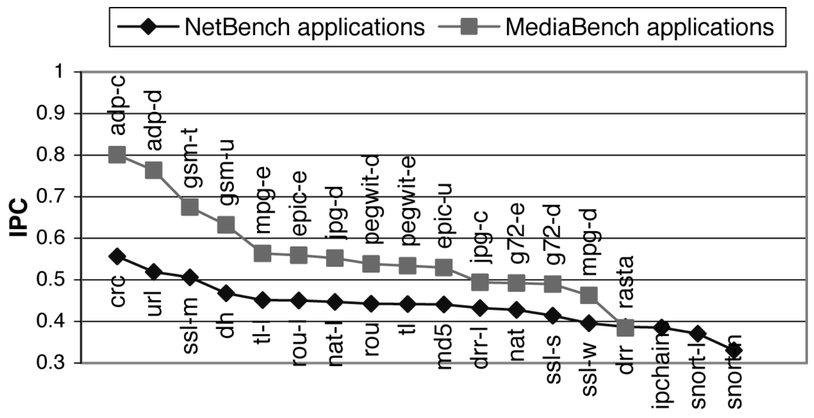

Fig. 2. Instructions per cycle (IPC) values for the NetBench and MediaBench applications measuring instruction level parallelism (ILP).

more important than other instructions in determining the nature of an application and its performance. On average, NetBench applications have a higher load/store frequency (34.4 versus 29.8%). This shows the data-intensive nature of these applications. In addition, the NetBench applications have a higher conditional branch instruction percentage (14.0 versus 8.9%). These two properties cause the lower IPC values observed in NetBench applications. A statistical analysis, similar to the study for the IPC values, shows that with a 95%

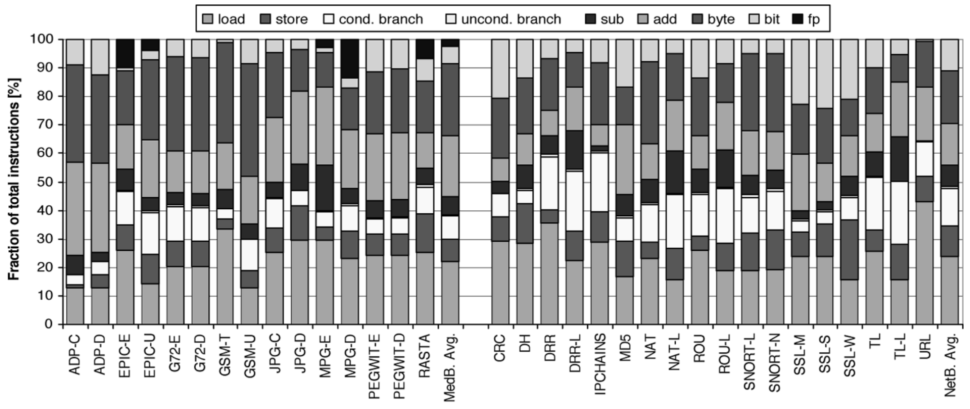

Fig. 3. Distribution of dynamic instructions for MediaBench and NetBench applications (byte includes arithmetic operations, such as rotate as well as data movement and comparison operations, such as mov; bit is for logical operations, such as xor; fp includes floating-point operations).

confidence interval NetBench applications have higher load/store instruction frequency. In addition, we can claim with 99% confidence interval that NetBench applications have more frequent branch instructions. In addition, NetBench applications contain significantly more bit-wise operations (e.g., shift, rotate instructions) than the MediaBench applications and no floating-point operations.

5.2.3 *Branch Prediction Accuracy.* Branch prediction has not been extensively studied in the context of NPUs. This is partly because of the relatively small branch misprediction penalty in NPUs (execution cores usually have a shallow pipeline, making the branch misprediction penalty low) and partly due to the area complexity of designing efficient branch prediction mechanisms. The branch predictor we have used in the base experiments assumes that all the branches are "not taken" (i.e., the instructions following the branch is scheduled as if the branch condition will not be met) similar to the StrongARM processors [Intel Corp.]. Note that, "not taken" is easier to implement than taken, because, in case of a taken strategy, the destination address has to be calculated early in the pipeline (or should be predicted) complicating the instruction fetch unit. The branch misprediction penalty is set to two cycles in all the experiments. To see the effects of branch prediction, we have simulated three additional branch prediction mechanisms: two-level [Yeh and Patt 1992], bimodal [Lee et al. 1997b], and combined (combining these two approaches). The bimodal predictor uses a 2-KB branch target buffer and the two-level branch predictor uses a 2-KB first level buffer and a 4-KB, 4-way associative level-2 buffer.

Figure 4 reports the address prediction accuracy for different mechanisms. All the branch predictors have high prediction rates for both MediaBench and NetBench applications showing that the branches are well behaved. The branch predictors have better performance for NetBench applications, correctly predicting the next address with up to 99.79% success rates. A statistical analysis

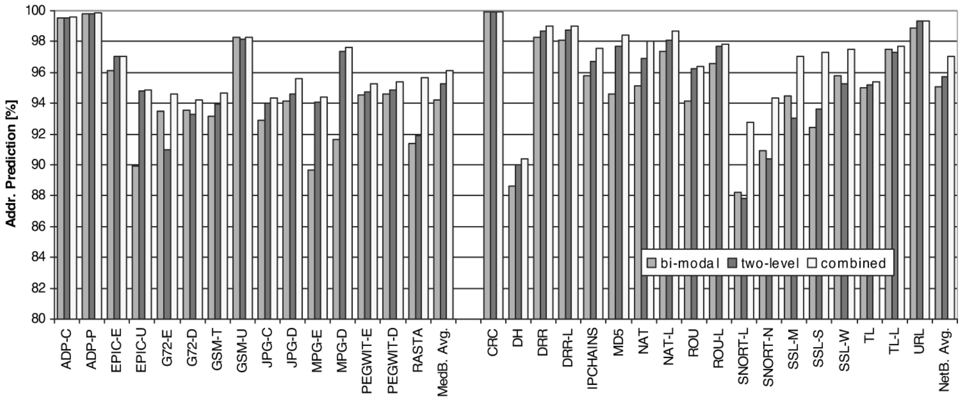

Fig. 4. Branch prediction ratios (address prediction) for different mechanisms.

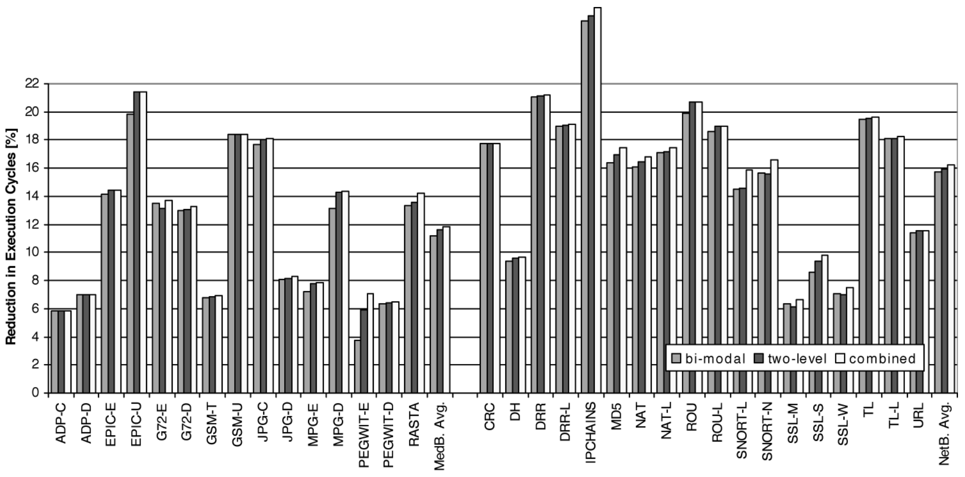

Fig. 5. Effect of branch prediction mechanisms on the execution cycles.

shows that the branch address misprediction for all prediction techniques is smaller for NetBench applications with 90% confidence interval.

The reduction in the execution cycles for different branch prediction mechanisms are presented in Figure 5. All mechanisms are successful in improving the execution efficiency. In spite of the small misprediction penalty (two cycles), even the relatively simple bimodal prediction mechanism reduces the execution time by 15.7%, on average. These results motivate the usage of branch prediction mechanisms in the NPUs.

5.2.4 *Cache Behavior.* Another characteristic we have examined is the cache behavior. The architectural values for the cache sizes in the base processor were explained in Section 5.5.1. We have performed additional simulations for larger level-1 data and instruction caches. The miss ratios for the different data and instruction cache sizes are presented in Figures 6 and 7, respectively. Each bar in the figures represents the miss ratio for all the cache sizes simulated. As

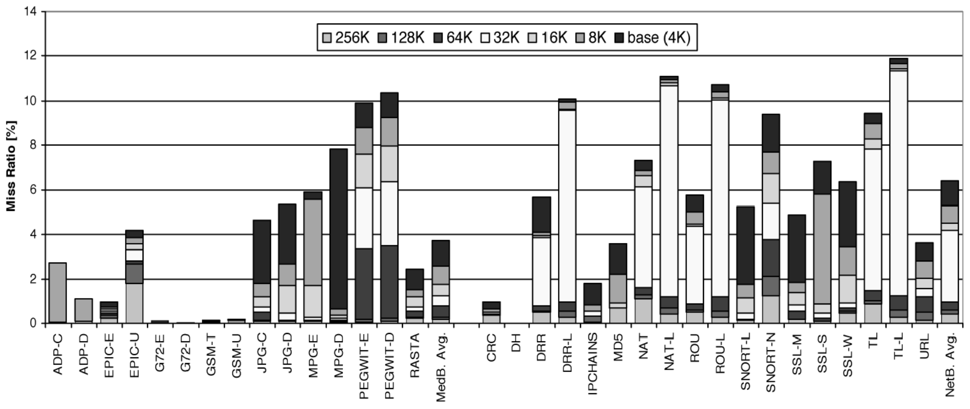

Fig. 6. Level-1 data cache miss ratios for different sizes of the cache.

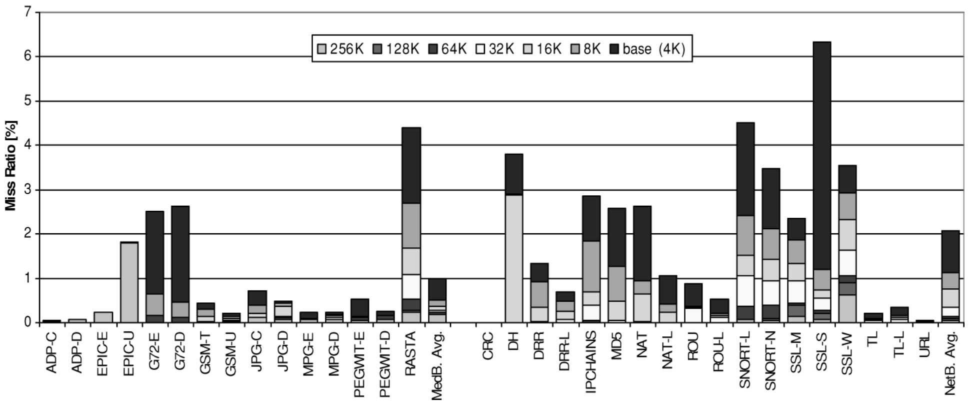

Fig. 7. Level-1 instruction cache miss ratios for different sizes of the cache.

the cache size is increased, the miss ratio naturally decreases. Therefore, the top of each bar corresponds to the miss ratio of the base architecture. The following portions of each bar correspond to increasing sizes of caches. For example, for the SSL-S application, the base architecture has 7.29%, the architecture with 8 KB L1 data cache has 5.79%, and the architecture with 16 KB L1 data cache has 0.88% data cache miss ratios. For data caches, we see that increasing the cache size from 16 to 32 KB has a significant effect (on average the miss ratio drops from 4.2 to 1.0%). For the instruction caches, on the other hand, increasing the cache size above 32 KB has almost no effect for NetBench applications: the 32 KB instruction cache results in 0.34% miss ratio, whereas the 256 KB cache results in 0.06% miss ratio. A statistical analysis shows that with 95% confidence interval, the base cache miss ratios are different for NetBench and MediaBench applications for both data and instruction cache misses.

The effect of different data cache sizes on the execution cycles are presented in Figure 8. The results reveal that data cache sizes have relatively small effect on the execution efficiency of NetBench applications: the 256 KB data cache reduces the execution cycles by 3.2% on average. The results for the instruction

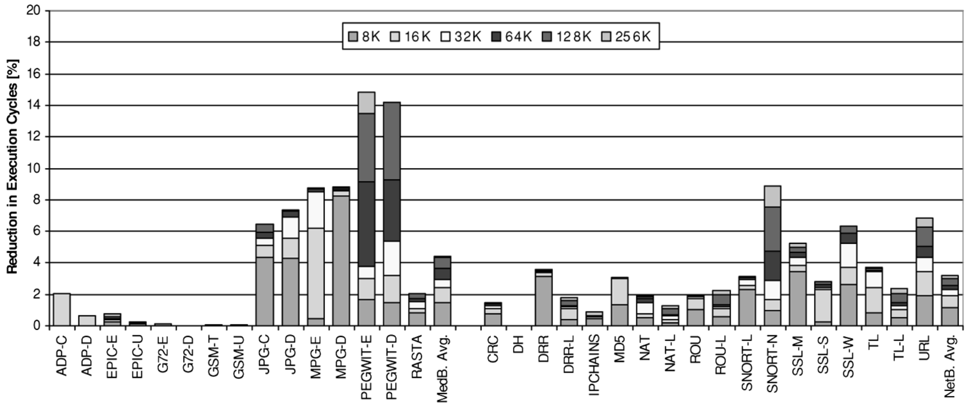

Fig. 8. Effect of increasing level-1 data cache size on the execution cycles.

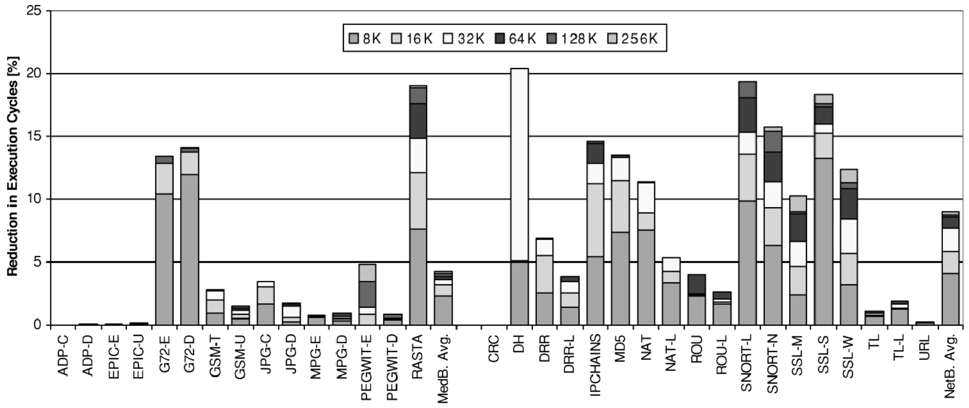

Fig. 9. Effect of increasing level-1 instruction cache size on the execution cycles.

caches are summarized in Figure 9. Increasing the instruction cache size affects the NetBench applications more than it affects the MediaBench applications. In addition, an increase in the instruction cache size results in better execution efficiency than an increase in the data cache size.

5.2.5 *Unique Data Rate* . The final property that we wish to discuss is the unique data rate. Unique data rate is defined as the unique memory locations accessed by the processor for each instruction executed. Figure 10 presents the results. It plots the amount of unique memory locations accessed by the application divided by the number of instructions executed. This indicates the streaming nature of the applications. In other words, if this ratio is high, that means the application accesses large amount of data that is processed with small number of instructions. The results reveal that NetBench applications have three times larger unique data rates. On average, every instruction accesses 0.03 bytes[1] of new data for NetBench applications and

[1]Assuming that each load/store accesses 4 bytes, on average, one in every 130 instructions in NetBench applications is a load/store to an address that has not been accessed before.

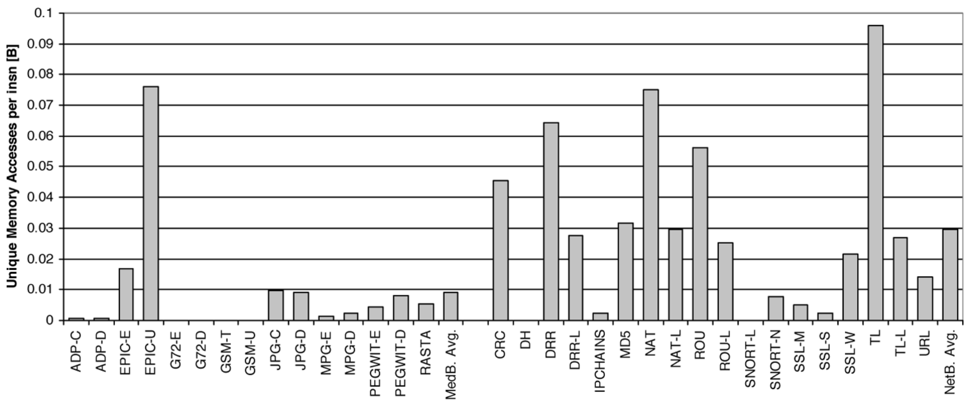

Fig. 10. Unique data rate measured in unique memory locations accessed per instruction executed in MediaBench and NetBench applications.

0.01 bytes for MediaBench applications. A statistical analysis shows that with 99% confidence interval, NetBench applications have a higher unique data rate.

5.2.6 *Discussion.* In the previous sections, we studied important characteristics of NetBench and MediaBench applications. In all these categories, NetBench applications had significantly different values than the MediaBench applications. This shows the need for a separate application set for the NPUs. One important property of NetBench applications is their data-intensive nature. As seen in the load/store instruction ratios, the NetBench applications make high number of memory accesses. They also contain more conditional branch operations, reducing the effectiveness of traditional instruction-level parallelism techniques.

5.3 Intel IXP1200 Performance Measurements

In this section, we give an example of how to utilize NetBench by presenting experimental results with the Intel IXP1200 network processor [Halfhill 1999]. We have used the Intel IXP simulator to perform these simulations. The next section explains the Intel IXP1200 architecture, while the following section summarizes the results.

5.3.1 *Architecture.* Intel IXP1200 processor is one of the most commonly used NPUs. It is a highly integrated, hybrid data processor that delivers high performance parallel processing power and flexibility to a wide variety of networking, communications, and other data-intensive applications. IXP1200 combines the StrongARM microprocessor with six 32-bit RISC data engines having hardware multithread support that provide 1 giga-operations per second with 200-MHz clock speed. The microengines possess the power to perform tasks that previously required high-speed ASICs.

5.3.2 *Simulation Results.* In order to simulate the IXP1200, we have converted codes from NetBench applications to Intel IXP Microcode. Because of the

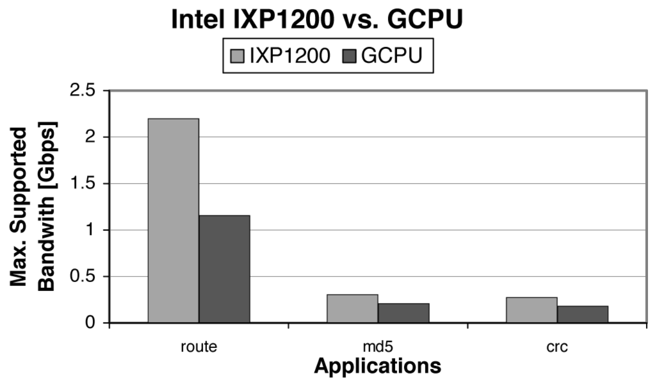

Fig. 11. Maximum supported bandwidth for Intel IXP1200 and a general-purpose processor similar to Intel Pentium III.

complexity of programming the IXP1200, we have selected three representative programs, one from each category: crc from micro-level programs, route from IP-level programs, and md5 from application-level programs. These codes are manually converted into the IXP microcode. To make a fair comparison, we also hand-optimized the applications used for simulating the general-purpose processor. In all the simulations, all the microengines are used (six microengines) and they execute the same application.

We compare the performance of IXP1200 with a general-purpose processor (GCPU), similar to Intel Pentium III, having a 1-GHz clock speed. We wanted to make sure that the simulated GCPU has more transistors than the Intel IXP1200 so that the comparison is fair: Intel IXP1200 has 6.5 million transistors [Halfhill 1999], an Intel Pentium III processor similar to the simulated processor that can run a 1-GHz clock has 28.1 million transistors [Intel Corp. 2000].

To gather information about the general-purpose processor, we used SimpleScalar simulator [Burger and Austin 1997]. Figure 11 summarizes the results. The figure gives the maximum amount of traffic the processor can handle. This value is calculated by finding the total number of bytes manipulated in the program and dividing this value to the simulated time required to execute the program. An example calculation is given in Appendix B. Figure 11 illustrates the power of the IXP processor. Although the simulated IXP processor had a clock speed of 200 MHz, it outperformed the GCPU in all programs: by 51% for crc, by 44% for md5, and by 80% for route.

The results also show how NetBench can be utilized. It shows that the IXP is more suitable for route than it is for MD5, because the relative performance improvement over GCPU is much higher with the route application. In addition, it gives the maximum amount of traffic the IXP1200 can handle for a given application. A customer can decide whether this supported bandwidth meets his/her requirements.

6. CONCLUSION

In this paper, we introduced a benchmarking suite for NPUs. In spite of the increase in demand and supply for NPUs, there still does not exist a common framework for evaluating them. Many designers still use benchmarks designed for other purposes, such as MediaBench and SPEC2000. We have shown that the applications for NPUs are significantly different from the applications for Media Processors; hence, a specific benchmarking suite is a necessity. In addition, we have presented simulation results for different cache sizes and branch prediction mechanisms indicating the bottlenecks and improvement opportunities in a representative RISC core executing the NetBench applications. We have also presented a performance study of a popular NPU showing how NetBench can be utilized.

APPENDIX A

To statistically show that the result sets for the IPC values are different we test the hypotheses

$$\mathtt{H}_0 : \mu_1 - \mu_2 \geq 0 \text{ and } \mathtt{H}_1 : \mu_1 - \mu_2 < 0$$

where μ_1 is the population mean of the IPC from MediaBench applications and μ_2 is the mean of the IPC from NetBench applications. The above hypotheses are best tested using a one-tailed test with the sampling distribution of (avg. x_1 – avg. x_2) i.e., we accept H_0 if,

$$\text{avg.}x_1 - \text{avg.}x_2 \geq t_{\alpha/2} * S_{\text{avg.x1}-\text{avg.x2}}$$

where $t_{\alpha/2}$ is the value based on t distribution with (n-1) degrees of freedom, avg. x_1 is the mean IPC of the MediaBench applications, and avg. x_2 is the mean IPC of the NetBench applications. $S_{\text{avg.x1}-\text{avg.x2}}$ is calculated by,

$$\begin{aligned} S_{\text{avg.x1}-\text{avg.x2}} &= \sqrt{\left(\left(s_1^2/n_1\right) + \left(s_2^2/n_2\right)\right)} \\ &= \sqrt{\left(\left(s_1^2/n_1\right) + \left(s_2^2/n_2\right)\right)} \\ &\approx \sqrt{((4.5 * 10^{-2}/15) + (2.9 * 10^{-3}/18))} \\ &= 5.6 * 10^{-2} \end{aligned}$$

Using the values from Figure 2 and a confidence interval of 99%, we see that

$$\begin{aligned} \text{avg.}x_1 - \text{avg.}x_2 = 0.119 &\geq 2.947 * 5.6 * 10^{-2} \\ &\approx 0.092 \end{aligned}$$

Therefore, with a 99% confidence interval we conclude that the NetBench applications have a lower IPC value.

APPENDIX B

Assuming the application processed 100 packets, with an average size of 100 bytes, the total amount of traffic handled is

100 (packets) $*$ 100 (bytes/packet) $*$ 8 (bits/byte) = 80,000 bits

We obtain the number of cycles it took to complete the application from the simulator. Assuming an execution of 100,000 cycles with a clock speed of 1 GHz, this corresponds to

100,000 (cycles)/1,000,000,000 (cycles/sec) = 0.1 ms
Then, the maximum bandwidth supported equals
80,000 bits/0.1 ms = 800 megabits/s

REFERENCES

BARRETT, D. AND SILVERMAN, R. 2001. *SSH: The Secure Shell, The Definitive Guide.* O'Reilly Publishers, Sebastopol, CA.

BURGER, D. AND AUSTIN, T. 1997. *The SimpleScalar Tool Set, Version 2.0.* University of Wisconsin, Technical report.

CELL-RELAY. *CRC-32 Calculation, Test Cases and HEC Tutorial.* http://cell.onecall.net/cell-relay/publications/software/.

C-PORT CORP. *C-5 Digital Communications Processor Product Brief.* http://www.cportcorp.com/products/pdf/c5brief.pdf.

CROWLEY, P., FIUCZYNSKI, M. E., BAER, J. L., AND BERSHAD, B. N. 2000. Characterizing Processor Architectures for Programmable Network Interfaces. In *Proceedings of International Symposium on Supercomputing*, Santa Fe, NM.

EEMBC. *An Industry Standard Benchmark.* http://www.eembc.org/Benchmark/networking.asp

FREEBSD PROJECT. *FreeBSD Operating System.* http://freebsd.org

HALFHILL, T. R. 1999. Intel Network Processor Targets Routers. *Microprocessor Report 13–12*, 1–26.

IMPROV INC. *The Jazz PSA platform.* http://www.improvsys.com/Products/Jazz.

INTEL CORP. 2000. *Intel Pentium III Processor and Intel 815E Performance Brief.*

INTEL CORP. *SA-110 Microprocessor Technical Reference Manual.* ftp://download.intel.com/design/strong/applnots/27819401.pdf.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. 1984. *ISO Information Processing Systems—Data Communication High-Level Data Link Control Procedure—Frame Structure.*

LEE, C., POTKONJAK, M., AND MANGIONE-SMITH, W. H. 1997a. MediaBench: A Tool for Evaluating and Synthesizing Multimedia and Communications Systems. In *Proceedings of International Symposium on Microarchitecture*, Research Triangle Park, NC, December. 330–335.

LEE, C.-C., CHEN, I.-C. K., AND MUDGE, T. N. 1997b. The bi-mode branch predictor. In *Proceedings of International Symposium on Microarchitecture,* Research Triangle Park, NC, December. 4–13.

MEMIK, G. AND MANGIONE-SMITH, W. H. *The NetBench Web Site,* http://istanbul.icsl.ucla.edu/NetBench.

MMC NETWORKS. *Leading the Network Processor Revolution,* http://www.mmcnet.com/Solutions.

NEMIROVSKY, A. 2000. *Towards Characterizing Network Processors: Needs and Challenges*, XStream Logic Inc.

NETWORK PROCESSOR FORUM. *Network Processor Forum,* http://www.npforum.org.

PMC-SIERRA INC. *URL-based Switching,* PMC-2002232, http://www.pmcsierra.com.

RIVEST, R. 1992. *The MD5 Message-Digest Algorithm.*

ROESCH, M. *The Open Source Network Intrusion Detection System Web Site,* http://www.snort.org.

RSA DATA SECURITY. *RSA Security Downloads,* http://www.rsasecurity.com/download.

RUSSELL, P. *IPCHAINS version 1.3.10,* http://netfilter.filewatcher.org/ipchains.

SENIE, D. Changing the Default for Directed Broadcasts in Routers. *Request for Comment (RFC) 2644.*

SHREEDHAR, M. AND VARGHESE, G. 1995. Efficient Fair Queuing using Deficit Round Robin. *In Proceedings of SIGCOMM'95*, Cambridge, MA, Aug/Sep.

STANDARD PERFORMANCE EVALUATION COUNCIL. *Spec CPU2000: Performance Evaluation in the New Millennium, Version 1.1.*

THE NLANR PROJECT. *NLANR Network Traffic Packet Header Traces,* http://moat.nlanr.net/Traces.

THE OPENBSD PROJECT. *Manual pages: sftp(1),* http://www.openbsd.org.

THE OPENSSL PROJECT. *The Open Source Toolkit for SSL/TSL,* http://www.openssl.org.

TRANSACTION PROCESSING COUNCIL. *TPC Benchmarks,* http://www.tpc.org.

WOLF, T. AND FRANKLIN, M. 2000. CommBench—A Telecommunication Benchmark for Network Processors. *In Proceedings of IEEE International Symposium on Performance Analysis of Systems and Software,* Austin/TX, (April).

WOO, S. E. A. 1995. The SPLASH-2 Programs: Characterization and Methodological Considerations. *In Proceedings of International Symposium on Computer Architecture*, Santa Margherita Ligure, Italy, June. 24–36.

XSTREAM CORP. *XStream Logic Packet Processing Core,* http://www.xstreamlogic.com/architectural_files.

YEH, T. H. AND PATT, Y. 1992. Alternative implementations of two-level adaptive branch prediction. *In Proceedings of International Symposium on Computer Architecture*, Queensland, Australia, May. 124–134.

Received January 2002; revised April 2004; accepted May 2005