PRO3:
A Hybrid NPU Architecture

THE PRO3 REDUCES THE OVERHEAD INCURRED BY COMMON "BRUTE-FORCE" ARCHITECTURES BY USING THE LEAST-REQUIRED HARDWARE RESOURCES FOR CERTAIN COMMON WELL-DEFINED TASKS.

Ioannis Papaefstathiou
Ellemedia Technologies
Technical University of Crete

Stylianos Perissakis
Theofanis G. Orphanoudakis
Nikos A. Nikolaou
George Kornaros
Nicholas A. Zervos
Ellemedia Technologies

George Konstantoulakis
InAccess Networks

Dionisios N. Pnevmatikatos
Technical University of Crete

Kyriakos Vlachos
University of Patras

• • • • • • As the telecommunications industry recovers from the severe downturn of recent years, data traffic continues to exhibit a rate of increase that outpaces advances in VLSI technology. Therefore, lowering overall system cost at network processing nodes and maximizing network utilization—hence revenues—remain extremely important objectives. To address these issues, new semiconductor devices called network processing units (NPUs) have emerged. They are optimized to provide programmable processing of protocol data units (PDUs) in networks with diverse requirements while efficiently supporting current and emerging protocols and services. NPUs promise to deliver an ASIC's speed with a CPU's programmability, thus augmenting the capacity and features of network nodes that forward and manipulate data traffic.

An NPU often accommodates hardwired components that have only a few hardware configuration options, have no software programmability support, and are dedicated to well-defined tasks such as framing, cyclic redundancy code and checksum checking and calculation. Furthermore, a number of NPUs incorporate reduced-instruction-set-computing (RISC) microengines that provide specialized operations for network processing, such as bit field manipulation, and extensive software programmability features. Because NPUs play an important role in today's heterogeneous networks, efficient support of the entire service-offering path, with a focus on providing acceptable quality of service (QoS), is critical.

Our Programmable Protocol Processor (PRO3) system is a hybrid approach to the demanding network processing puzzle comprehensively described by Comer.[1] PRO3 provides the necessary processing power for network processing with minimal hardware complexity by following a more delicate approach than that of most NPUs. That is, the architecture uses special hardware modules for specific computation-intensive tasks; special-purpose processors with low hardware complexity, optimized for certain functions; and high-performance, general-purpose CPU cores only when absolutely needed. The special-purpose processors and dedicated hardware blocks handle most of the computation-intensive and real-time protocol functions, and a pair of on-chip general-purpose processors optimized for fast context switching handle the remaining functions, including higher-layer protocols. PRO3 combines wire-speed processing up to the network layer with best-effort processing for higher-layer protocols. Finally, an on- or off-chip control processor handles computation that is not on the fast path, such as control-plane or exception processing. These components are interconnected very efficiently.

The resulting system on a chip achieves high performance in terms of processing capacity and number of supported flows, providing additional functionality for differentiated QoS and traffic shaping. Operating at 200 MHz, PRO3 supports complex packet

Published by the IEEE Computer Society

Related work

Many companies have developed high-end NPUs, using proprietary architectures with various programming models. Most commercial, high-end NPUs fall into two categories: those using a large number of simple RISC CPUs and those using a few high-end, special-purpose CPUs optimized for processing network streams. The first category includes solutions from big semiconductor companies, such as

- Intel's IXP family, which uses from 8 to 16 very simple microprocessors, together with a standard general purpose processor;[1]
- Freescale's C-5, which uses 16 simple, general-purpose CPUs and five very simple special-purpose coprocessors;[2] and
- Cisco's Toaster family, which uses 16 simple microcontrollers.[2]

These NPUs claim to handle data rates from 2.5 to 10 Gbps, but the actual bandwidth they can service depends heavily on the application.[2] Thus, for complex applications, their performance degrades dramatically. Moreover, their performance is often limited by their intercommunication overhead and limited off-chip memory bandwidth.

The second category includes

- Broadcom's BCM1250 with two two-way, superscalar, ultra-high-speed CPUs that incorporate special-purpose processing units;[3]
- Clearwater's CNP810SP, with an eight-thread, 10-issue CPU that can simultaneously process eight packets and includes special-purpose processing units;[4]
- EZChip's NP1 0 with a number of very simple processing units, specialized for ultra-high speed memory lookups;[2] and
- an academic device that uses reconfigurable, complicated processors, presented by Memik, Memik, and Mangione-Smith.[5]

Although these NPUs provide higher processing power for some complex network protocols, they lack the first category's parallelism. Therefore, their performance for processing large numbers of independent flows is very often lower in terms of bandwidth serviced than that of the first category.

HiFn's (formely IBM's) PowerNP is probably the only device that follows an approach similar to PRO3's. The PowerNP integrates many dedicated hardware modules for queue handling, memory management, traffic scheduling, and so forth.[6] However, it differs markedly from our architecture in that its dedicated hardware modules consist of general-purpose processors augmented for specific tasks, whereas PRO3 uses hardware modules that are designed and optimized for executing *only* the specific tasks. Moreover, PRO3 performs protocol processing using special-purpose programmable processing units designed especially for packet field manipulation, whereas the PowerNP uses general-purpose CPUs.

References

1. S. Lakshmanamurthy et al., "Network Processor Performance Analysis Methodology," *Intel Technology J.*, Aug. 2002, vol. 6, no. 3.
2. D.E. Comer, *Network Systems Design Using Network Processors*, Prentice Hall, 2003.
3. Broadcom Corporation, BCM1250 Integrated 64-Bit MIPS Multiprocessor, http://www.broadcom.com/products/product.php?product_id=BCM1250&category_id=27.
4. Clearwater Networks CNP810SP Simultaneous Multithreading (SMT) Core, http://www.zytek.com/~melvin/clearwater.html.
5. G. Memik, S.O. Memik, and W.H. Mangione-Smith, "Design and Analysis of a Layer Seven Network Processor Accelerator Using Reconfigurable Logic," *Proc. 10th Annual IEEE Symp. Field-Programmable Custom Computing Machines* (FCCM 02), IEEE Press, 2002, pp. 131-140.
6. J. Allen et al., "PowerNP Network Processor Hardware, Software and Applications," *IBM Journal of Research and Development,* vol. 47, no. 2/3, March/May 2003, pp. 177-194.

processing at up to 2.5 Gbps. The PRO3 architecture's main innovations are the use of special-purpose, low-complexity programmable processors and sophisticated interface hardware modules (memory managers and schedulers), as well as the efficient microarchitecture of hardware blocks that implement common functions and reduce software workload. (The "Related work" sidebar summarizes other NPUs.)

PRO3 architecture

The initial NPU approach, focused on high-speed switching and routing systems, usually results in multi-chip solutions comprised of dedicated classifiers, traffic managers, and switch fabrics. In contrast, PRO3 targets systems that require increased packet manipulation power for many concurrent connections at very high link rates. Typical systems with these requirements include traffic concentrators supporting enhanced per-flow services, such as security systems performing packet filtering and protocol-aware connection tracking; signaling controllers; and traffic-policing, metering, and statistics-collecting systems. PRO3's main goal is to accelerate particular bottlenecks in processing communication protocols.

Through detailed protocol analysis and profiling, we developed a taxonomy of protocol-processing requirements for which most existing approaches face certain problems. Among all the protocols we examined, we found a common set of frequently used functions that generic RISC processors cannot execute efficiently. We classify these functions as follows:

- bit- and byte-level operations for header parsing and modification;
- efficient memory management functions, requiring specialized memory controllers for maximum memory throughput, and requiring intelligent operations for efficiently handling structures accessed by software;
- complex task- and traffic-scheduling algorithms to support QoS;
- more frequent context switching than in desktop processors (in the worst case, once for every packet arrival); and
- robust on-chip interconnecton of various processing modules without introducing bottlenecks.

PRO3 addresses these critical issues by including special-purpose CPUs for the header's field extraction and modification, a highly sophisticated memory management hardware block, and two highly efficient hardware scheduling modules to facilitate fair and balanced packet processing and to control data streams generated by internal modules. Moreover, PRO3 uses two general-purpose processors optimized for fast context switching to handle parts of the protocol processing that the special-purpose processors and hardware modules cannot efficiently execute.

Figure 1 illustrates the PRO3 architecture. PRO3 classifies PDUs into up to 512,000 flows. The preprocessor integrates a special-purpose processor that performs header parsing, checksum calculation, and classification,[2] based on a user-defined (through microcode) set of the header's fields, and generates a flow ID, obtained by the external ternary content-addressable memory (CAM). The special-purpose processor assigns to each received packet a flow ID that specifies in which queue, handled by the data memory manager (DMM), the packet belongs. The flow ID also specifies the internal processing unit needed and the piece of software that the unit should execute. From the flow ID, the task scheduler (TSC) identifies the appropriate destination for further processing—that is, the reprogrammable pipeline modules (RPMs), a general-purpose CPU (either the internal RISC core control CPU or an external host CPU), or the output if the flow is configured in that way.

As Figure 1 illustrates, when a PDU arrives at PRO3, it goes to the DMM; when the flow ID is available, the PDU is segmented and stored in the respective queue, and the DMM notifies the TSC. At a certain time, the TSC commands the DMM to send the PDU for processing. At that point, the processing stage starts. The TSC defines the processing unit that will process the PDU, as well as the part of the PDU to be sent for processing (usually the first few segments containing protocol headers), taking into account the need for load balancing and efficiently controlling the RPMs' pipeline.

After processing the PDU, PRO3 can modify the PDU and append it to another queue to undergo further processing by a higher-layer protocol. This is the case when PRO3 implements a protocol stack and a user PDU must be processed by several protocol layers. It is the current processing task's responsibility to assign a new flow ID, on the basis of which PRO3 continues the processing. The outgoing-traffic scheduler (TRS) handles transmission, the final PDU-processing stage.

All PRO3 modules communicate over a 12.8-Gbps-bandwidth internal bus, coordinated by a central arbiter. Bus transactions transfer a segment of up to 64 bytes accompanied by a 64-bit control word that indicates the transfer source, destination, and other control information. Modules can exchange control messages using a control word without a segment payload. A transaction's request/grant phase is pipelined with the data transfer phase, to avoid idle bus cycles.

PRO3 building blocks

The PRO3 building blocks consist of the packet preprocessor, the data memory management unit, the scheduler modules, and the reprogrammable pipeline module. The first three blocks are optimized for handling a very large number of independent queues while the last one executes very efficiently the majority of today's network protocol processing. Moreover, the first three blocks can support

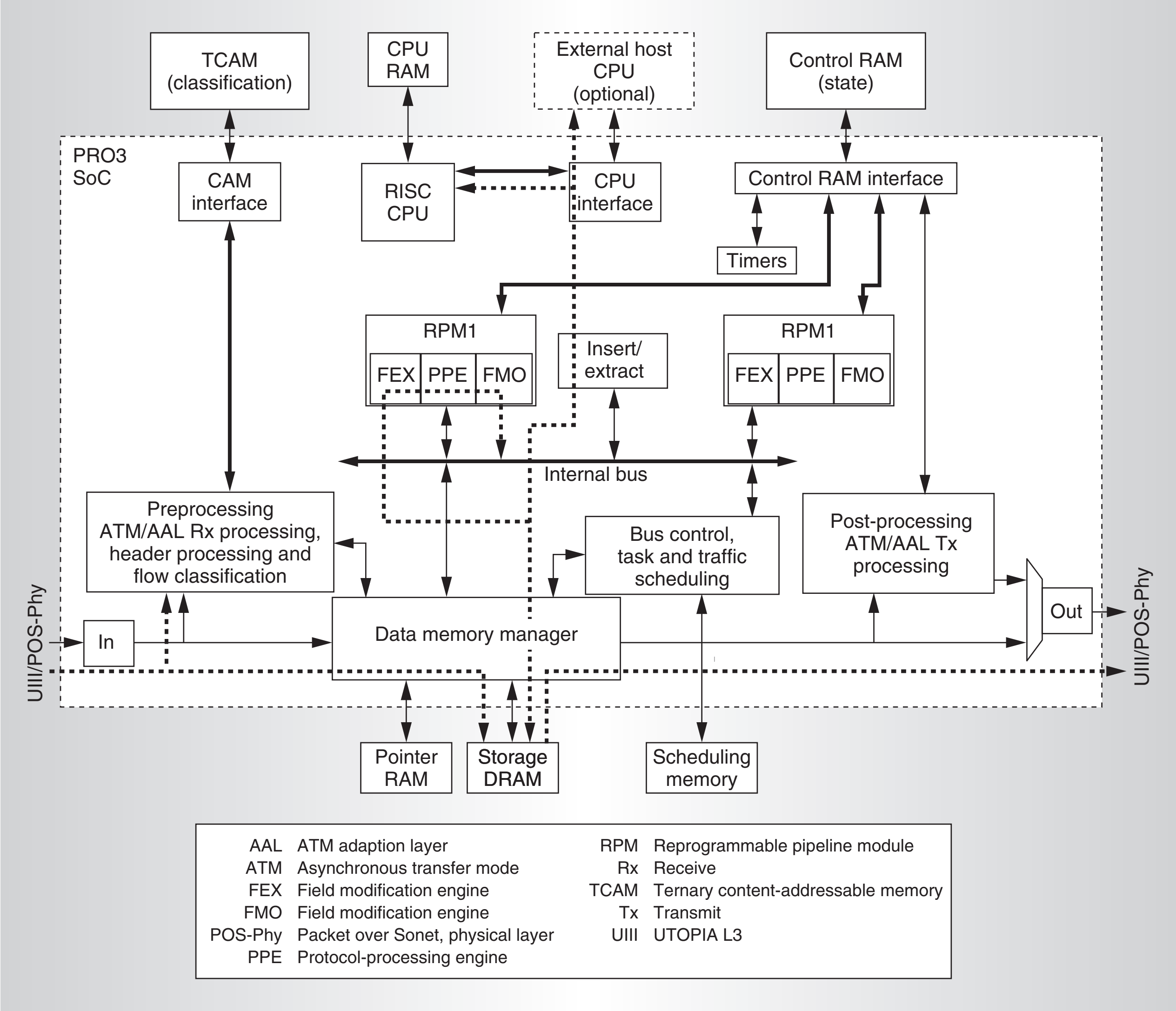
TCAM (classification)
CPU RAM
External host CPU (optional)
Control RAM (state)
PRO3 SoC
CAM interface
RISC CPU
CPU interface
Control RAM interface
Timers
RPM1
FEX
PPE
FMO
Insert/ extract
RPM1
FEX
PPE
FMO
Internal bus
Preprocessing ATM/AAL Rx processing, header processing and flow classification
Bus control, task and traffic scheduling
Post-processing ATM/AAL Tx processing
UIII/POS-Phy
In
Data memory manager
Out
UIII/POS-Phy
Pointer RAM
Storage DRAM
Scheduling memory
AAL ATM adaption layer
ATM Asynchronous transfer mode
FEX Field modification engine
FMO Field modification engine
POS-Phy Packet over Sonet, physical layer
PPE Protocol-processing engine
RPM Reprogrammable pipeline module
Rx Receive
TCAM Ternary content-addressable memory
Tx Transmit
UIII UTOPIA L3

Figure 1. PRO3 architecture and packet-processing paths (dashed lines).

traffic processing at a constant rate of 2.5 Gbps under any conditions. The supported network rate of the reprogrammable pipeline module heavily depends on the application running on it.

Packet preprocessor

The packet classifier in the ingress exploits the features of the programmable field-processing engine, which has the same features as the protocol-processing engine, while still is a different entity and the external TCAM with 144-bit-wide classification key entries. This block is responsible for lookup of address or other fields, employed by most network applications.

Data memory management unit

The DMM implements per-flow queuing for up to 512,000 flows, enabling efficient queue handling, variable-length packet storage, and access to specific packet segments by the processing engines. The DMM's main functions are storing incoming traffic, retrieving packet data, and forwarding that data to either the processing units or the output interface. It operates on both fixed-length cells and variable-length packets. To achieve efficient memory management, the DMM partitions incoming data items into fixed-size segments of 64 bytes each. It supports two incoming and two outgoing ports of 2.5-Gbps bandwidth each—one port for receiving traffic from the network (input), one for transmitting traffic to the network (output), and a bidirectional port for receiving and sending traffic from and to the internal bus. The unit uses double-data-rate (DDR) DRAM for data storage and zero bus turnaround (ZBT) SRAM for segment and packet pointers. Thus, all data structure manipulations occur in parallel with data transfers, keeping DRAM accesses to a minimum. Although hardware implementation of packet pointers may reduce performance on small packets (40 to 64 bytes) by increasing pointer RAM accesses, it reduces software overhead and ultimately increases performance for multisegment packets, which are much more common in practice.

To support an aggregate throughput of 10 Gbps, we optimized the DMM's free list organization and memory access ordering.

Efficient free list organization. The DRAM provides high throughput and capacity at the cost of high latency and throughput limitations caused by bank conflicts. Hence, we gave special care to buffer allocation and deallocation. Using two separate free lists reduces memory accesses during buffer releasing (deleting or dequeuing a large packet requires a fixed number of accesses to pointer memory).[3] Our highly efficient allocation-deallocation scheme reduces the number of DRAM bank conflicts by 70 percent during writes and 46 percent during reads.[3]

Memory access reordering. The execution of an incoming operation might send read and write commands to the data memory to update corresponding data structures, thus causing bank conflicts. By effectively reordering the read and write accesses (and keeping a small list of accesses waiting to be serviced), we reduced overall latency caused by DRAM conflicts and increased system performance. This reordering achieves a 30 percent reduction in mean access latency.[3]

Table 1 shows the 18 commands provided by the DMM to support the diverse protocol-processing requirements of a device that handles queues. The latency of these commands varies from 4 to 13 clock cycles. The complete instruction set's mean latency, when running a large number of real networking applications, such as Internet Protocol (IP) forwarding, DiffServ metering, firewalling, and multi-protocol encapsulation, is slightly more than seven clock cycles.

As described in detail elsewhere,[3] the DMM meets its performance requirements (10 Gbps equally shared by its four ports), even under worst-case traffic. In any case, the DMM is idle at least 25 percent of the time and uses at most 75 percent of the available 13-Gbps sustained DRAM throughput and 55 percent of the available SRAM bandwidth. Therefore, the DMM can service an overall throughput even higher than 10 Gbps.

Scheduler modules

The requirement for processing-task scheduling emerges when the RPM engines process packets at a rate lower than the packet arrival rate. Hence, PDUs must be stored and scheduled for processing after the target processing block has become available. For this purpose, we use an internal scheduler, the task scheduler. The TSC resembles the process-scheduling functionality found in operating systems. It maintains priority queues to schedule the forwarding of packets for processing according to a configurable priority per flow or per QoS class.[4] The network administrator sets up the priorities. In the outgoing path, cells or packets processed or generated by PRO3 must comply with specific traffic profiles. Thus, we use a traffic scheduler (TRS) to shape transmitted traffic according to traffic management specifications. Figure 2 (on p. 26) shows the organization of the scheduler blocks. Scheduling is based on command and status messages issued by the DMM, the CPU, and the classifier (preprocessor). The Service and Redirect block routes these commands to the appropriate scheduling queue.

The TSC supports 32 hierarchical scheduling queues, which load the PRO3 processing resources (the two RPMs, the on-chip RISC, and the external CPU) in a weighted-fair-queuing order.[4] Multiple flows are multiplexed round-robin in one scheduling queue. The TSC treats the first queue with strictly highest priority over the others. It can either treat the remaining 31 scheduling queues with one priority or organize them into two sets with different priorities. Scheduling queues of the same priority are served in a weighted-round-robin (WRR) fashion. The TSC uses a connection table, where it stores per-flow state information, as well as connection-specific parameters

Table 1. DMM commands (*N* is the number of segments in a packet).

Segment commands	**Clock cycles**	**Input parameters/ Return values**	**Action**
Enqueue_Segment	10	Flow_id, Segment/None	Enqueue single segment at end of a queue
Read_Segment	5	Flow_id/Segment	Copy data of a single segment from head or tail of specified queue (without deleting it)
Read_N_Segments	5 + *N*	Flow_id, N/N Segments	Read *N* segments from head or tail of specified queue (without deleting it)
Dequeue_N_Segments	13 + *N*	Flow_id, N/N Segments	Dequeue *N* segments from head of specified queue
Overwrite_Segment	5	Flow_id, segment/None	Overwrite first or last segment of a queue with new segment
Overwrite_Segment_length	7	Flow_id, New_segment_ length/None	Overwrite length attribute of last segment of flow id's queue
Dequeue_Segment	10	Flow_id/Segment	Dequeue single segment at beginning of flow id's queue
Ignore_Segment	4	None/None	Ignore or delete single segment received from input
Ignore&Overwrite_Segment_length	7	Flow_id, New_segment_length/ None	Ignore or delete received segment (from input) and overwrite length of packet at tail of queue
Overwrite_Segment_length&Append	11	Src_Flow_id, Dst_flow_id, New_segment_length/ None	Overwrite length at head of Source queue and append segment at Destination queue
Overwrite_Segment&Append	11	Src_Flow_id, Dst_flow_id, Segment/None	Overwrite head of Source queue with new segment and append this segment at tail of Destination queue
Packet commands			
Enqueue_Packet (after segmentation) at tail of queue	9 x *N* + 3	Flow_id, Packet/None	Enqueue a single packet received from input
Read_Packet queue (without deleting it)	4 x *N* + 2	Flow_id/Packet	Copy data of a packet from head of flow id's
Dequeue_Packet queue	8 x *N* + 7	Flow_id/Packet	Dequeue a single packet from head of flow id's
Append_Packet	11	Src_Flow_id, Dst_Flow_id/None	Append a single packet from head of Source queue to tail of Destination queue
Ignore_Packet	4	None/None	Ignore or delete packet received from input
Ignore&Delete_Packet	9	None/None	Ignore last segment (received from input) and delete packet from tail of queue
Delete_Packet	9	None/None	Delete packet received from input

such as the flow's priority weight. Flows that map to the same scheduling queue are grouped in a linked list. Overall, the TSC and the TRS share the same memory, which contains 512,000 traffic descriptors corresponding to the number of supported flows. The TSC supports up to 2.5 Gbps while introducing a mean latency of 217 ns per packet.[4]

The TRS performs WRR and shaped-virtual-clock scheduling of outgoing asynchronous transfer mode (ATM) cells or IP packets. Each flow's peak rate *PR* is compiled to minimum transmission interval *MTI* through the following equation, where the predefined data segment size *PDS* is 53 bytes for ATM, while for IP it coincides with the segment size supported by the DMM (64 bytes) and *Slot* represents the time required for the transmis-

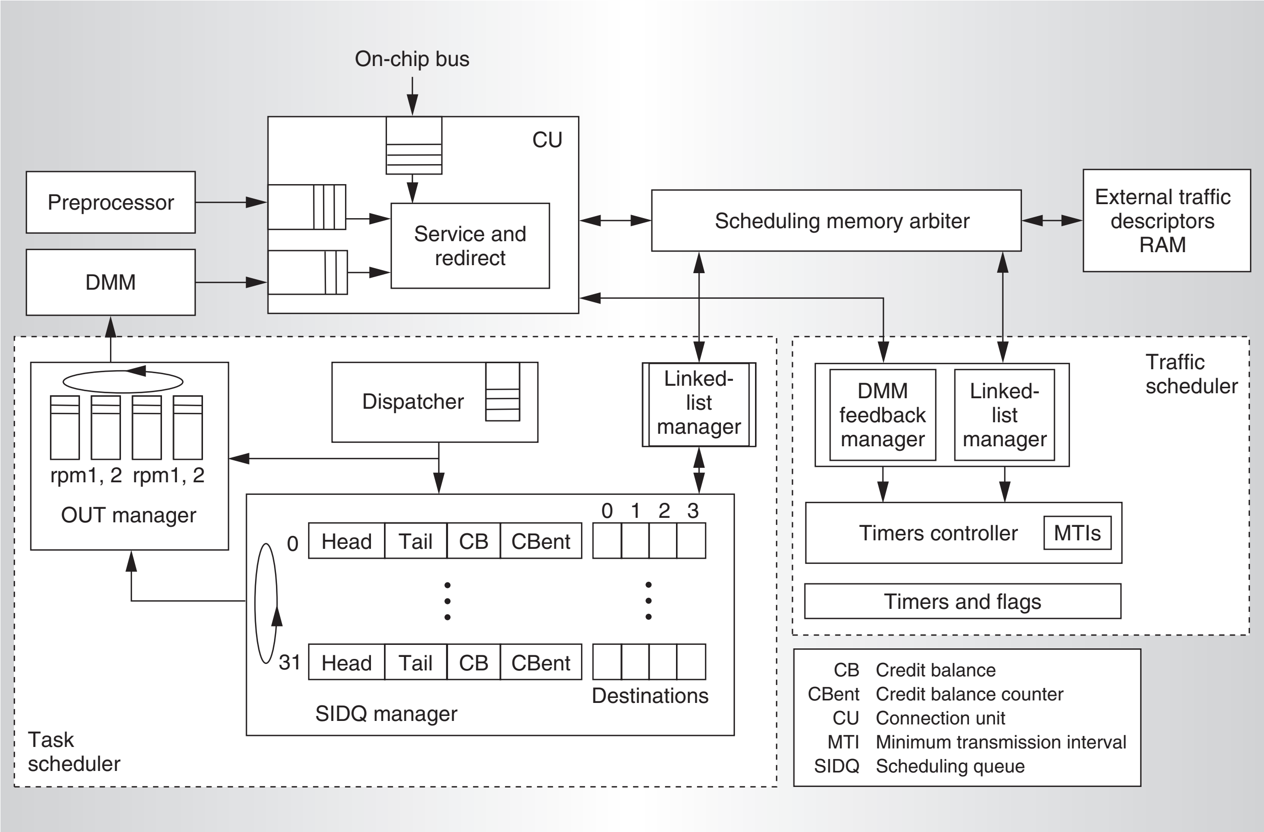

Figure 2. Scheduler organization.

sion of an ATM cell, in the ATM case, or for the transmission of the Minimum packet size, in the case that variable size packets are transmitted:

$$PR = PDS/((MTI + 1) * slot)$$

The TRS implements 32 scheduling queues in total, each associated with the appropriate group of flows and corresponding to a certain peak rate, and it uses the same data structures as the TSC. For a line rate of 2.5 Gbps, 32 discrete rates are adequate,[5] and the rate resolution becomes finer as peak rate decreases, as the following equation shows:

$$Granularity = PDS/((MTI + 1)^2 *slot)$$

The network administrator configures the 32 rates during system initialization by loading specific configuration registers; he also assigns flows to rate groups during flow setup by properly initializing the scheduling memory. The TRS achieves per-flow shaping for ATM connections and aggregate per-group shaping for IP flows.

Reprogrammable pipeline module

The RPM is the PRO3's main protocol-processing element, depicted in Figure 3. The RPM's core is the PPE, which has a modified Hyperstone E1-32XS RISC-CPU core (http://www.hyperstone.com) and external control logic. The main processor modification is the partitioning of registers into two sets. While the processor core accesses one set, the PPE control logic can access the other, to place packet information or retrieve updated data directly into or out of the register file. Processor switches register sets and informs the external logic, via a special instruction, every time it has finished processing a packet. Similarly, direct I/O can be performed through the processor's dual-ported on-chip data memory. By overlapping I/O operations with packet processing, we manage to hide the latency of these operations and substantially accelerate network processing.

The PPE is surrounded by the field extrac-

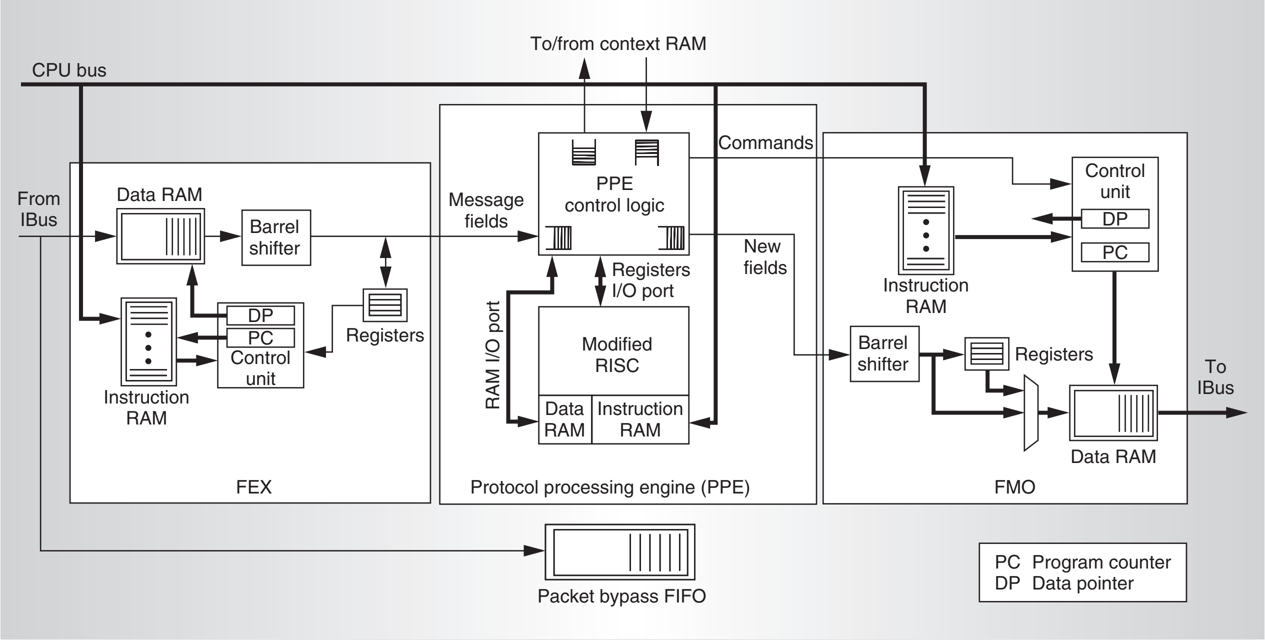

Figure 3. RPM architecture.

tion and field modification engines. The FEX parses packet headers and directly loads the required protocol fields to the PPE for processing. The FMO performs packet construction or reconstruction and header modification. All form a powerful three-stage pipeline that constitutes the PRO3's software-processing heart.

We optimized both the FEX and the FMO for bit and byte processing.[6] Figure 3 shows their basic architectures. The FEX instruction set includes nine basic instructions and four commands (instructions without arguments), which operate on data stored in a first-in, first-out (FIFO) buffer of 32-bit words. The FEX instruction set supports variable-length field extraction, backward and forward movement in the data FIFO, conditional branches, and addition. FEX conditional branches are based on extracted fields and header parsing. The FMO instruction set includes 16 basic instructions and six commands, which support both field extraction and insertion/modification. Instructions can be combined and executed in parallel with any command, reducing code size and increasing performance.

The FEX and the FMO use only four and five generic registers, respectively, plus a special-purpose register and a data pointer (DP). They execute up to 2,000-instruction firmware, which is sufficient for common network applications. Both components process data with a maximum throughput of 6.4 Gbps at 200 MHz.[6] The sustained throughput might be lower because some instructions need more cycles, and because one 32-bit word can contain several fields that should be extracted or inserted separately. The average cycle-per-instruction ratio in executing real-world network applications was 1.6 for the FEX and 1.7 for the FMO.[6]

Development tools and implementation details

To facilitate software development and configuration for the PRO3, we built a development suite that configures registers, initializes memory locations, and loads software modules into the PRO3. Figure 4 shows this tool's overall architecture. The user interacts with the tool through the configuration GUI (CG). For each PRO3 hardware module, the CG contains a separate page on which the user configures or develops code for that module.

The tool's core is the configuration library (CL), which maintains an internal register map of the chip. The CL implements and exports a set of functions, which the CG uses to set and get the values of configuration registers and internal memory locations. Additionally, the

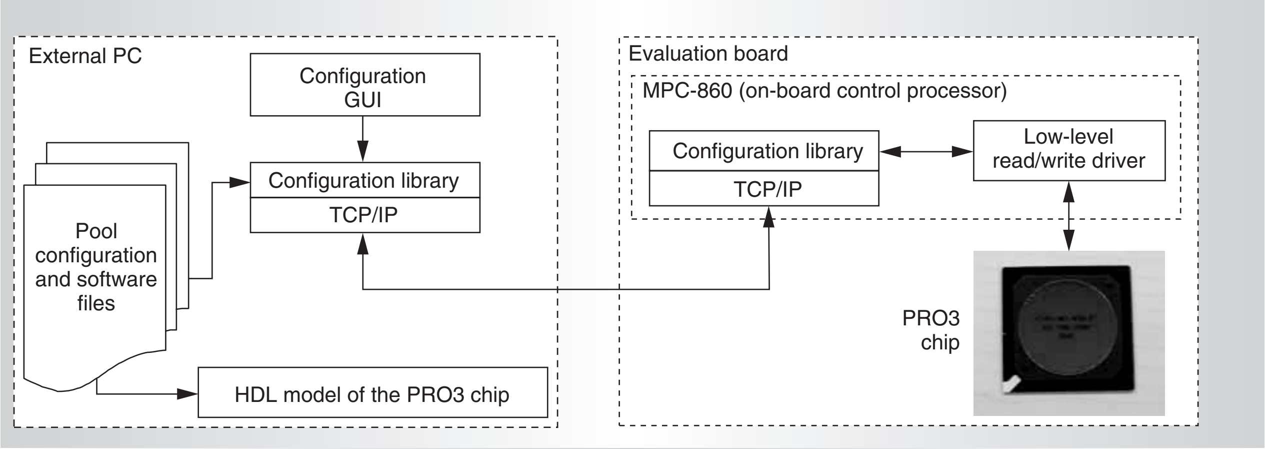
External PC
Configuration GUI
Configuration library
TCP/IP
Pool configuration and software files
HDL model of the PRO3 chip
Evaluation board
MPC-860 (on-board control processor)
Configuration library
TCP/IP
Low-level read/write driver
PRO3 chip

Figure 4. PRO3 software development and configuration tool.

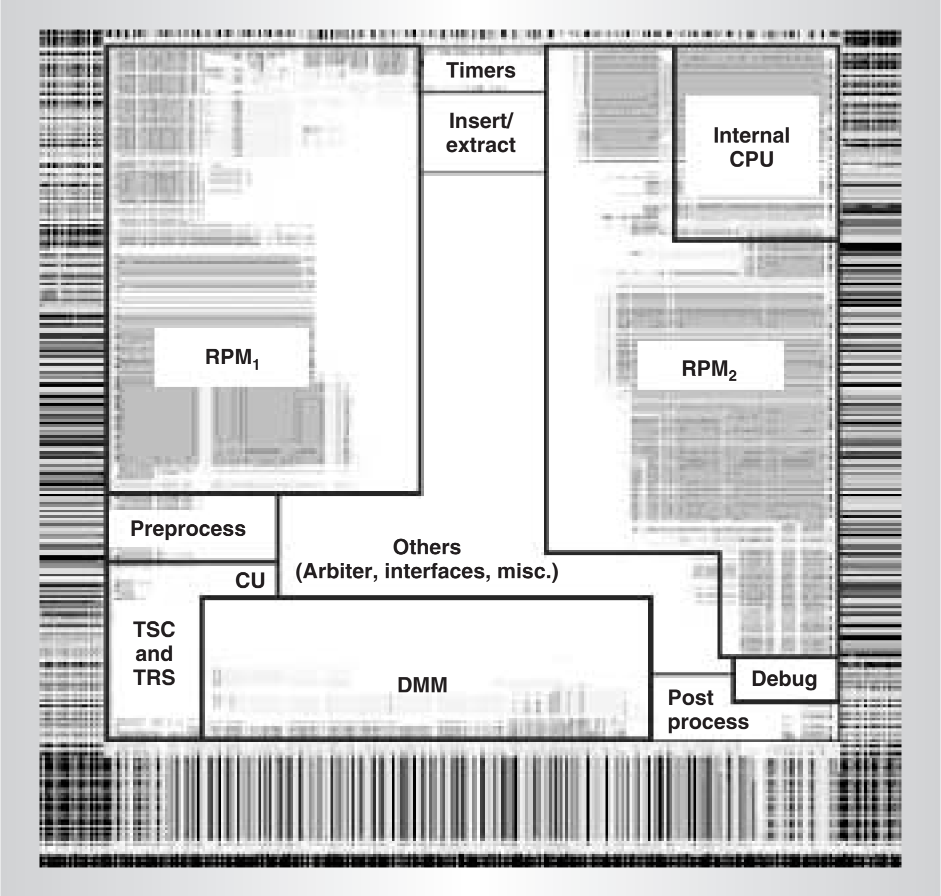

Figure 5. PRO3 layout.

CL reads, parses, and compiles the configuration files and software programs for all processing units (mainly the FEX, FMO, and PPE). Finally, the CL maintains a Transmission Control Protocol (TCP)-based session with the configuration server running on the on-chip or on-board control processor. As a result, configuration commands initiated by this tool arrive at the CS, which in turn uses the software API offered by the low-level read/write driver to access the PRO3's memories and registers.

An innovative and useful characteristic of the tool is that it can configure either the actual PRO3 chip residing on the development board or a hardware description language model or netlist. This feature considerably reduced the time spent testing the fabricated chip and facilitated the collection of measurements.

The PRO3 is fabricated in United Microelectronics' 0.18-μm, 1.8-V process. It occupies a 10 × 10 mm^2 die and is pad limited. Figure 5 shows the chip's layout. The core occupies a central 7.3 × 7.3 mm^2 area, and the periphery holds 912 pads arranged in a staggered dual-row pattern.

Table 2 lists the PRO3 prototype's main characteristics. Note that the sum of the individual cores' areas is only 36.95 mm^2, but because the design was pad limited, we expanded the layout to fill a 53 mm^2 area to limit wiring congestion and improve overall performance. Table 3 shows the complexity of all blocks. Evidently, the most complicated custom core is the DMM because it supports a complete instruction set for manipulating queues. However, the DMM is still less expensive than similar memory management systems.[3] It is also worth noting that the two custom 32-bit processing engines (the FEX and the FMO) cover a silicon area similar to that of an 8051-compatible, 8-bit microcontroller core,[7] but, as we will show, they are more efficient for certain applications than some state-of-the-art 32-bit processors commonly found in NPUs.

Table 2. PRO3 characteristics.

Feature	Description
Process	0.18-μm, six metal
Supplies	1.8 V (core), 3.3 V (I/O)
Area	100 mm^2 (53 mm^2 core)
I/Os	912 (690 signals, 222 supplies)
Package	1,096-I/O fine-line BGA
On-chip memory	1 Mbit (mainly dual-ported RAMs)
Logic gates	865,000
Transistors	12 million

Table 3. Complexity of PRO3 cores.

Hardware block	No. of logic gates (thousands)	Area (mm2)
DMM	152.9	5.77
TSC	47.8	0.86
TRS	53.3	0.86
CU	9.7	0.23
PPE (each)	90.3	6.8
FMO (each)	19.8	2.24
FEX (each)	8.3	1.16
Timers	7.9	0.13
Arbiter	5.8	0.08
Debugger	7.1	0.58
Internal CPU	59.2	2.56
Packet preprocessor	56.3	1.38
Packet postprocessor	27.4	0.52
Insert/Extract	15.6	0.57
Host interface	9.5	0.12
Context RAM interface	18.9	0.71
Miscellaneous	62.8	2.08
Total	**865**	**36.95**

Performance measurements

We evaluated the PRO3's performance in terms of the processor's sustainable data throughput, measured in packets per second (PPS). We derived the detailed results presented here from extensive simulations of the PRO3's RTL model (which we also verified against the actual chip) executing two representative NPU applications: TCP stateful inspection with network address translation (NAT) and User Datagram Protocol (UDP) stateful inspection (without NAT). We developed complete firmware for these applications using the tools described earlier and executed them in typical IP traffic.

The maximum packet arrival rate from the 2.5-Gbps input link is about 6.75 million PPS, taking into account the minimum IP packet size (40 bytes) plus at least 6 more overhead bytes introduced by the off-chip PRO3's incoming interface. The PRO3's dedicated hardware units, such as the schedulers and the DMM, perform fixed functions and can support 2.5 Gbps even when the input stream consists of minimum-size packets. The internal control CPU is used primarily for system configuration, initialization, and handling exceptional cases, so its impact on application performance is limited. In general, the time needed for PDU processing depends on the performance of the RPMs, which, given their three-stage pipelined architecture, is limited by the part that has the heaviest workload. PDU processing varies for different applications and protocols. In our experiments, the mean latency a PDU suffers from the time its first bit enters the PRO3 until that bit is transmitted on the output link is about 4.8 μs (or 960 clock cycles). This number does not include queuing delay due to QoS support, which can be very high for a low-priority packet when the PRO3 is heavily loaded with high-priority traffic. In those 4.8 μs, the PRO3 receives about 12 Kbits of traffic, arriving at at 2.5 Gbps. But the DMM acts as an elastic input buffer (and supports several Mbytes of memory), ensuring that the latency introduced by the various processing components does not adversely affect the processor's *mean* throughput.

Performance results

Table 4 displays the number of executed instructions and the corresponding latency in clock cycles for the FEX, FMO, and PPE when

Table 4. Performance comparison: PRO3 versus IXP2400 (KPPS: thousand packets per second).

	TCP/NAT application			UDP application (no NAT)		
Processor	**Instructions**	**Clock cycles**	**Throughput (KPPS)**	**Instructions**	**Clock cycles**	**Throughput (KPPS)**
PRO3						
FEX	58	94		31	53	
FMO	70	112		59	93	
PPE	108	113		19	20	
RPM (slowest stage)		113	1,770		93	2,150
PRO3 (2 RPMs)		3,540		4,300		
IXP2400						
One microengine (FEX program)	227	297		132 (together with PPE program)	198	
One microengine (FMO program)	272	326		159	231	
One microengine (PPE program)	170	230		N/A	N/A	
Microengine complex (slowest engine)		326 (3 engines)	1,840 (3 engines)		231 (2 engines)	2,608 (2 engines)
IXP (8 microengines)		4,571		9,567		

executing the IP firewall applications. Clearly, in the TCP/NAT application, the pipeline stages are very well balanced and the slowest one is the PPE. Moreover, in the UDP application, the slowest stage is the FMO, whereas the PPE stage has many idle cycles because the RPMs are optimized for more complex protocol processing. To compare the measured RPM performance to that of state-of-the-art NPUs, we ported the IP/NAT filter application to the IXP2400 using the Intel development platform and tools. We used three of the IXP2400's microengines, executing the equivalent of the FEX, FMO, and PPE programs. We connected the microengines in a three-stage pipeline because this configuration, dividing the workload almost equally among the three engines, produces the best results for the IXP as well as the PRO3. The UDP case doesn't require address translation and recalculation, and the optimal configuration proved to be two IXP microengines forming a two-stage pipeline.

We obtained the instruction counts shown in Table 4 for the IXP2400 by applying the highest compiler optimizations for performance and handcrafting the produced code, further increasing execution speed. The throughput numbers also include the intercommunication and scheduling overheads of executing these applications, which are unavoidable in the IXP but are handled by the TSC in the PRO3. As the table shows, the two RPMs support up to 3,540 KPPS in total in the TCP/NAT case, while the entire IXP system supports up to 4,571 KPPS. We measured the aggregate IXP performance with eight microengines; one executes both the FEX and PPE code while the other seven are running a single program. Similarly, in the UDP case, the two RPMs handle 4,300 KPPS in total, whereas the entire IXP2400 supports up to 9,567 KPPS. These results show the presented architecture's advantages in such applications, since, by utilizing two RPM engines operating at 200 MHz, the PRO3 can achieve throughput comparable to that of eight optimized IXP microengines operating at 600 MHz. The optimization stems from the efficiency of the FEX and FMO instruction sets for certain protocol-processing tasks, for which the IXP's microengines need about three times more instructions and clock cycles. Additionally, the PPE's low instruction count and low latency show the efficiency of the innovative direct-register-access I/O scheme.

To directly compare the throughput results with those provided for the IXP family, we attempted to estimate the PRO3's equivalent

MIPS (million instructions per second), which is the only metric for the IXP available in the literature. The IXP2400 provides 4,800 MIPS according to its datasheet, and as Table 4 shows, it executes the TCP/NAT application at a maximum rate of 4,571 KPPS. Since the PRO3 RPMs support up to 3,540 KPPS when executing the same application, we can express their equivalent protocol-processing capacity as (3,540/4,571) × 4,800 or about 3,700 IXP2400 MIPS. The RPMs' efficiency is also demonstrated by the fact that the two RPMs, on their own, are three times more powerful in certain network applications than the entire IXP1200 commodity NPU, even though they have about the same complexity.[6]

In the preceding computation, we took only the filtering and NAT processing into account. To extend our computation to the scheduling modules' equivalent processing power, we implemented the WRR scheduling algorithm on the IXP2400 (in software) and measured the MIPS needed to achieve the required throughput. According to our results and to Lakshmanamurthy et al.,[8] the IXP2400 needs about 900 MIPS for WRR of 512,000 flows at a rate of 4.8 MPPS, or 1,800 MIPS to schedule both packet processing and transmission at that same rate. In contrast, the PRO3 task and traffic schedulers perform this function without consuming RPM processing resources at a rate of 6.75 MPPS, providing an equivalent processing power of 2,530 IXP2400 MIPS. This processing power should be added to that of the RPMs (and that of the DMM and the preprocessor which are, though very difficultly, expressed in MIPS) in order to calculate the overall power of the PRO3. The PRO3 architecture's performance gains are also apparent when we compare them with the PowerNP family. According to Allen et al., the PowerNP4GS3 (also manufactured in a 0.18-μm CMOS technology) supports up to 2.5 Gbps or 6.1 MPPS of incoming TCP/IP traffic, consisting of minimum-size packets, when executing simple IPv4 forwarding.[9] Moreover, according to Allen et al., to achieve this throughput, the PowerNP4GS3 requires from 50 to 80 percent of its total processing power (provided by 16 "picoprocessors"), depending on the complexity of the application's extensive error-checking functions and on the required robustness. We implemented the basic IPv4 packet-forwarding functionality in PRO3. The two RPMs support the maximum packet input rate of 6.75 MPPS, while consuming about 80 percent of their processing power. The RPM bottleneck in this application is the FMO, whereas the PPE is utilized less than 50 percent of the time, leaving significant processing power for additional packet processing. Finally, the PowerNP's additional coprocessors for enqueuing, classification, checksumming, policing, and scheduling acceleration are similar, according to their specifications, to the PRO3's preprocessor/classifier module, the DMM engine, and the scheduling modules, which support packet classification, queuing, and QoS functions at 6.75 MPPS. In the new 0.13-μm PowerNP4GX, which operates at 500 MHz, the PowerNP coprocessors aggregately provide the equivalent of 8,250 MIPS when processing traffic at a rate of 6 MPPS.[10]

Summarizing our comparisons with state-of-the-art NPUs, for the real network applications examined, the PRO3 architecture is more efficient than other common NPU architectures. The PRO3 provides higher processing power for certain QoS-aware applications than the IXP2400, is more efficient, for a certain application, than the PowerNP4GS3, and has comparable functionality with the PowerNP4GX. Strengthening the advantages of its architectural approach, the PRO3 is implemented in 0.18-μm technology and operates at 200 MHz, whereas the IXP2400 is implemented in 0.13-micron technology and operates at 600 MHz. Finally, the PRO3 is about five times smaller (12 million transistors) than the IXP2400 (60 million transistors), showing the efficiency of the PRO3's approach in terms of the hardware resources needed.

Scaling

To scale the PRO3 architecture to higher rates, such as 10 Gbps, we must increase both the clock rate and the processing parallelism. Shrinking the PRO3 design to the next technology generation would allow twice as much logic capacity, and it is reasonable to expect a clock rate of 400 MHz, for a fourfold net increase in nominal performance. The obvious obstacle to those end-to-end performance gains is the increasing memory bandwidth required. We expect to overcome this obstacle in two ways: first, using higher-performance

RAM architectures such as quad-data-rate (QDR) SRAM and DDR-2 DRAM or fast-cycle RAM (FCRAM); and second, by implementing caching schemes. Since the DMM handles most of the memory management in PRO3, we can add a caching scheme to it, which would be transparent to the rest of the system. In addition, caching can be effective for the most heavily utilized external RAM, the control and context RAM, because access bandwidth to the flow context increases proportionally to the packet arrival rate, but storage requirements do not. Therefore, caching the context of high-bandwidth flows can relieve off-chip bandwidth significantly, without excessive on-chip storage requirements.

The PRO3 network processor is based on a novel architecture, which demonstrated high performance when executing real-world applications. Our experience with this architecture, summarized in the experiments presented here, suggests that tight coupling of the processing elements and hardware support in specific areas, such as queue management and packet scheduling, can yield more efficient platforms than the alternative approach of combining large numbers of simple processing units on a single chip. PRO3 has been developed for the target application domain of core network processing, supporting link speeds of up to 2.5 Gbits/sec. Applying the same design principles, we are currently extending our design approach to the area of lower speed access networks, where we expect to see similar benefits. MICRO

Acknowledgments

We performed this work in the framework of the Information Society Technologies IST-1999-11449 PRO3 project, cofunded by the European Union. We acknowledge contributions from Lucent Technologies, Hyperstone AG, IMEC, the National Technical University of Athens, and Ellemedia Technologies.

References

1. D.E. Comer, *Network Systems Design Using Network Processors*, Prentice Hall, 2003.
2. P. Gupta and N. McKeown, "Packet Classification on Multiple Fields," *Computer Communication Rev.*, vol. 29, no. 4, Sept. 1999, pp. 147-160.
3. G. Kornaros et al., "A Fully Programmable Memory Management System Supporting Queue Handling at Multi-Gigabit Rates," *Proc. 40th Design Automation Conf.* (DAC 03), ACM Press, 2003, pp. 54-59.
4. I. Papaefstathiou, et al., "An Innovative Scheduling Scheme for High-Speed Network Processors," *Proc. 2003 IEEE Int'l Symp. Circuits and Systems* (ISCAS 03), vol. 2, IEEE Press, 2003, pp. 93-96.
5. D.C. Stephens, J.C.R. Bennett, and H. Zhang, "Implementing Scheduling Algorithms in High-Speed Networks," *IEEE J. Selected Areas in Communications*, vol. 17, no. 6, June 1999, pp. 1145-1158.
6. I. Papaefstathiou et al., "Packet Processing Acceleration with a 3-Stage Programmable Pipeline Engine," *IEEE Communication Letters*, vol. 8, no. 4, Mar. 2004, pp. 183-185.
7. B. Finch and W. Miller, "A New Reference Design Development Environment for JPEG 2000 Applications," System-on-Chip and ASIC Design Conf. (DesignCon 03), 2003, http://www.cast-inc.com/info/pr/materials/cast_JPEG2K-paper_DCon03.pdf.
8. S. Lakshmanamurthy et al., "Network Processor Performance Analysis Methodology," *Intel Technology J.*, vol. 6, no. 3, Aug. 2002.
9. J. Allen et al., "PowerNP Network Processor Hardware, Software and Applications," *IBM J. of Systems and Development,* vol. 47, nos. 2/3, March/May 2003, pp. 177-194.
10. IBM PowerNP4GX Product Brief, http://www.bellmicro.com/vendorshowcase/ibmmicro/downloads/NP4GX.pdf.

Ioannis Papaefstathiou has recently been elected an assistant professor at the Technical University of Crete, Greece. He is also a technical consultant at Ellemedia Technologies, Athens, and a research associate at the Institute of Computer Science, Foundation for Research and Technology-Hellas (ICS-FORTH). His research interests include architectures for network processors and specific-purpose networking systems. Papaefstathiou has a BSc from the University of Crete, an MSc from Harvard University, and a PhD in computer science from the University of Cambridge.

Stylianos Perissakis designs data transmission and network-processing ICs at Ellemedia Technologies. His research interests include full- and semicustom IC design with emphasis on reconfigurable and embedded DRAM architectures. Perissakis has a diploma in electrical and computer engineering from the National Technical University of Athens and MSc and PhD degrees in computer science from the University of California, Berkeley.

Theofanis G. Orphanoudakis leads R&D activities in broadband networking components at Ellemedia Technologies. His research interests include network processors, performance evaluation of broadband access networks, and resource management. Orphanoudakis has a Dipl-Ing in electrical and computer engineering and a PhD in telecommunications, both from the National Technical University of Athens. He is a member of the IEEE and the Technical Chamber of Greece.

Nikos A. Nikolaou is a Technical Manager at Ellemedia Technologies. His research interests mainly include Communication protocols and QoS. Nikolaou has a diploma in computer engineering and information science from the University of Patras and a PhD in electrical and computer engineering from the National Technical University of Athens. He is a member of the IEEE and the Technical Chamber of Greece.

George Kornaros is an acting technical manager of the Broadband Networking Components Group at Ellemedia Technologies. His research interests include high-speed network processors and communication architectures, and VLSI design. Kornaros has a BS in computer engineering from Patras University, Patras, Greece, and an MS in computer science from the University of Crete. He is a member of the Technical Chamber of Greece.

Nicholas A. Zervos is managing director of Ellemedia Technologies. His research interests include exploratory development of core network technologies in home networking, wireline/wireless systems, and broadband networking components. Zervos has a diploma in electrical and mechanical engineering from the National Technical University of Athens, an MSc in systems and computing science form Carleton University, Ottawa, and a PhD in electrical engineering from the University of Toronto. He is a member of the IEEE.

George Konstantoulakis is cofounder and CTO of InAccess Networks. His research interests include residential networks and high-speed networking devices. Konstantoulakis has a diploma in electrical and computer engineering and a PhD in telecommunications, both from the National Technical University of Athens. He has initiated and managed the PRO3 project. He is a member of the IEEE and the Technical Chamber of Greece.

Dionisios N. Pnevmatikatos is an associate professor of electrical and computer engineering at the Technical University of Crete and a research associate at ICS-FORTH. His research interests include the architecture and design of computer systems, networking systems, and system software. Pnevmatikatos has a BS in computer science from the University of Crete and an MSc and a PhD, both in computer science, from the University of Wisconsin-Madison. He is a member of the IEEE.

Kyriakos Vlachos is a professor in the Computer Engineering and Informatics Department of the University of Patras. Earlier, he was a member of the technical staff of Bell Laboratories, Lucent Technologies, the Netherlands, where he participated in the work described in this article. His research interests include high-speed communication systems, network processors, optical packet switching, and optical labeling techniques. Vlachos has a Dipl-Ing and a PhD, both in electrical and computer engineering, from the National Technical University of Athens. He is a member of the IEEE.

Direct questions and comments about this article to Ioannis Papaefstathiou, Ellemedia Technologies, 223, Syggrou Av. 17121, Athens, Greece; ygp@ics.forth.gr.

For further information on this or any other computing topic, visit our Digital