IBM PowerNP network processor: Hardware, software, and applications

J. R. Allen, Jr.
B. M. Bass
C. Basso
R. H. Boivie
J. L. Calvignac
G. T. Davis
L. Frelechoux
M. Heddes
A. Herkersdorf
A. Kind
J. F. Logan
M. Peyravian
M. A. Rinaldi
R. K. Sabhikhi
M. S. Siegel
M. Waldvogel

Deep packet processing is migrating to the edges of service provider networks to simplify and speed up core functions. On the other hand, the cores of such networks are migrating to the switching of high-speed traffic aggregates. As a result, more services will have to be performed at the edges, on behalf of both the core and the end users. Associated network equipment will therefore require high flexibility to support evolving high-level services as well as extraordinary performance to deal with the high packet rates. Whereas, in the past, network equipment was based either on general-purpose processors (GPPs) or application-specific integrated circuits (ASICs), favoring flexibility over speed or vice versa, the network processor approach achieves both flexibility and performance. The key advantage of network processors is that hardware-level performance is complemented by flexible software architecture. This paper provides an overview of the IBM PowerNP™ NP4GS3 network processor and how it addresses these issues. Its hardware and software design characteristics and its comprehensive base operating software make it well suited for a wide range of networking applications.

Introduction

The convergence of telecommunications and computer networking into next-generation networks poses challenging demands for high performance and flexibility. Because of the ever-increasing number of connected end users and end devices, link speeds in the core will probably exceed 40 Gb/s in the next few years. At the same time, forwarding intelligence will migrate to the edges of service provider networks to simplify and speed up core functions.[1] Since high-speed traffic aggregates will be switched in the core, more services will be required at the edge. In addition, more sophisticated end user services lead to further demands on edge devices, calling for high flexibility to support evolving high-level services as well as performance to deal with associated high packet rates. Whereas, in the past, network products were based either on GPPs or ASICs, favoring flexibility over speed or vice versa, the network processor approach achieves both flexibility and performance.

[1] The term *edge* denotes the point at which traffic from multiple customer premises enters the service provider network to begin its journey toward the network core. Core devices aggregate and move traffic from many edge devices.

Current rapid developments in network protocols and applications push the demands for routers and other network devices far beyond doing destination address lookups to determine the output port to which the packet should be sent. Network devices must inspect deeper into the packet to achieve content-based forwarding; perform protocol termination and gateway functionality for server offloading and load balancing; and require support for higher-layer protocols. Traditional hardware design, in which ASICs are used to perform the bulk of processing load, is not suited for the complex operations required and the new and evolving protocols that must be processed. Offloading the entire packet processing to a GPP, not designed for packet handling, causes additional difficulties. Recently, field-programmable gate arrays (FPGAs) have been used. They allow processing to be offloaded to dedicated hardware without having to undergo the expensive and lengthy design cycles commonly associated with ASICs. While FPGAs are now large enough to accommodate the gates needed for handling simple protocols, multiple and complex protocols are still out of reach. This is further intensified

0018-8646/03/$5.00 © 2003 IBM

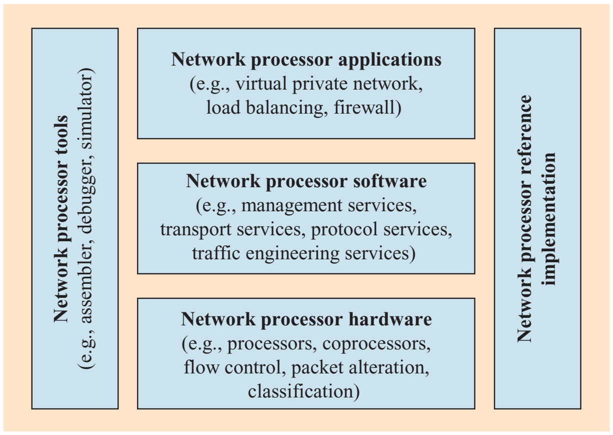
Network processor tools
(e.g., assembler, debugger, simulator)
Network processor applications
(e.g., virtual private network, load balancing, firewall)
Network processor software
(e.g., management services, transport services, protocol services, traffic engineering services)
Network processor hardware
(e.g., processors, coprocessors, flow control, packet alteration, classification)
Network processor reference implementation

Figure 1

Components of a network processor platform.

by their relatively slow clock speeds and long on-chip routing delays, which rule out FPGAs for complex applications.

Typical network processors have a set of programmable processors designed to efficiently execute an instruction set specifically designed for packet processing and forwarding. Overall performance is further enhanced with the inclusion of specialized coprocessors (e.g., for table lookup or checksum computation) and enhancements to the data flow supporting necessary packet modifications. However, not only is the instruction set customized for packet processing and forwarding; the entire design of the network processor, including execution environment, memory, hardware accelerators, and bus architecture, is optimized for high-performance packet handling.

The key advantage of network processors is that hardware-level performance is complemented by horizontally layered software architecture. On the lowest layer, the forwarding instruction set together with the overall system architecture determines the programming model. At that layer, compilation tools may help to abstract some of the specifics of the hardware layer by providing support for high-level programming language syntax and packet handling libraries [1]. The interface to the next layer is typically implemented by an interprocess-communication protocol so that control path functionality can be executed on a control point (CP), which provides extended and high-level control functions through a traditional GPP. With a defined application programming interface (API) at this layer, a traditional software engineering approach for the implementation of network services can be followed. By providing an additional software layer and API which spans more than one network node, a highly programmable and flexible network can be implemented. These layers are shown in **Figure 1** as hardware, software, and applications, respectively, and are supported by tools and a reference implementation.

Flexibility through ease of programmability at line speed is demanded by continuing increases in the number of approaches to networking [2–4]:

- Scalability for traffic engineering, quality of service (QoS), and the integration of wireless networks in a unified packet-based next-generation network requires traffic differentiation and aggregation. These functions are based on information in packet headers at various protocol layers. The higher up the protocol stack the information originates, the higher the semantic content, and the more challenging is the demand for flexibility and performance in the data path.
- The adoption of the Internet by businesses, governments, and other institutions has increased the importance of security functions (e.g., encryption, authentication, firewalling, and intrusion detection).
- Large investments in legacy networks have forced network providers to require a seamless migration strategy from existing circuit-switched networks to next-generation networks. Infrastructures must be capable of incremental modification of functionalities.
- Networking equipment should be easy to adapt to emerging standards, since the pace of the introduction of new standards is accelerating.
- Network equipment vendors see the need of service providers for flexible service differentiation and increased time-to-market pressure.

This paper provides an overview of the IBM PowerNP* NP4GS3[2] network processor platform, containing the components of Figure 1, and how it addresses those needs. The specific hardware and software design characteristics and the comprehensive base operating software of this network processor make it a complete solution for a wide range of applications. Because of its associated advanced development and testing tools combined with extensive software and reference implementations, rapid prototyping and development of new high-performance applications are significantly easier than with either GPPs or ASICs.

System architecture

From a system architecture viewpoint, network processors can be divided into two general models: the *run-to-*

[2] In this paper the abbreviated term *PowerNP* is used to designate the IBM PowerNP NP4GS3, which is a high-end member of the IBM network processor family.

completion (RTC) and *pipeline* models, as shown in **Figure 2**. The RTC model provides a simple programming approach which allows the programmer to see a single thread that can access the entire instruction memory space and all of the shared resources such as control memory, tables, policers, and counters. The model is based on the symmetric multiprocessor (SMP) architecture, in which multiple CPUs share the same memory [5]. The CPUs are used as a pool of processing resources, all executing simultaneously, either processing data or in an idle mode waiting for work. The PowerNP architecture is based on the RTC model.

In the pipeline model, each pipeline CPU is optimized to handle a certain category of tasks and instructions. The application program is partitioned among pipeline stages [6]. A weakness in the pipeline model is the necessity of evenly distributing the work at each segment of the pipeline. When the work is not properly distributed, the flow of work through the pipeline is disrupted. For example, if one segment is over-allocated, that segment of the pipeline stalls preceding segments and starves successive segments.

Even when processing is identical for every packet, the code path must be partitioned according to the number of pipeline stages required. Of course, code cannot always be partitioned ideally, leading to unused processor cycles in some pipeline stages. Additional processor cycles are required to pass packet context from one stage to the next. Perhaps a more significant challenge of a pipelined programming model is in dealing with changes, since a relatively minor code change may require a programmer to start from scratch with code partitioning. The RTC programming model avoids the problems associated with pipelined designs by allowing the complete functionality to reside within a single contiguous program flow.

Figure 3 shows the high-level architecture of the PowerNP—a high-end member of the IBM network processor family which integrates medium-access controls (MACs), switch interface, processors, search engines, traffic management, and an embedded IBM PowerPC* processor which provides design flexibility for applications. The PowerNP has the following main components: embedded processor complex (EPC), data flow (DF), scheduler, MACs, and coprocessors.

The EPC processors work with coprocessors to provide high-performance execution of the application software and the PowerNP-related management software. The coprocessors provide hardware-assist functions for performing common operations such as table searches and packet alterations. To provide for additional processing capabilities, there is an interface for attachment of external coprocessors such as content-addressable memories (CAMs). The DF serves as the primary data path for receiving and transmitting network traffic. It provides an interface to multiple large data memories for buffering data traffic as it flows through the network processor. The scheduler enhances the QoS functions provided by the PowerNP. It allows traffic flows to be scheduled individually per their assigned QoS class for differentiated services. The MACs provide network interfaces for Ethernet and packet over SONET (POS).

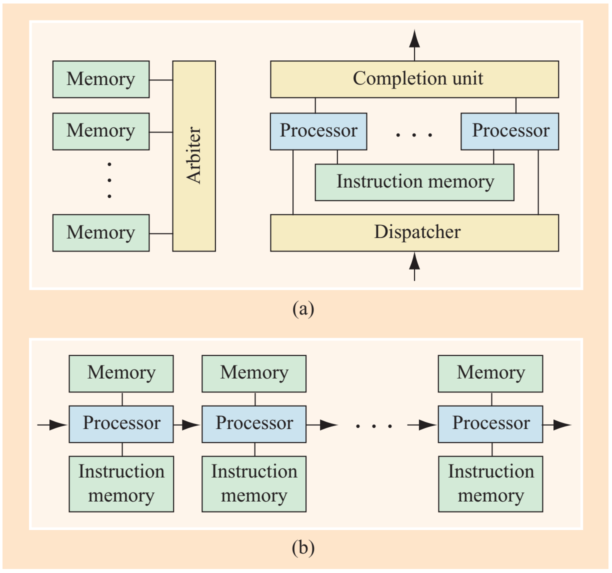

Figure 2

Network processor architectural models: (a) run-to-completion (RTC) model; (b) pipeline model.

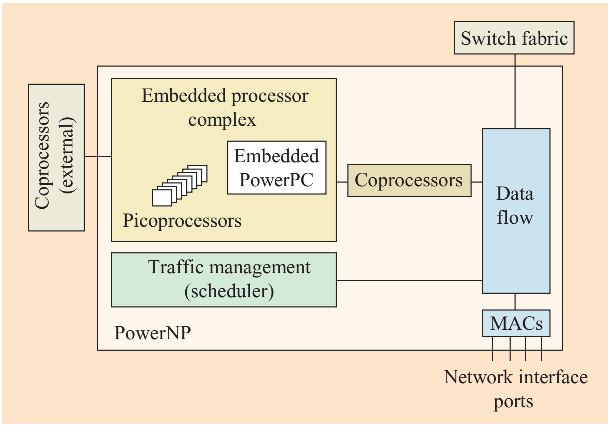

Figure 3

PowerNP high-level architecture.

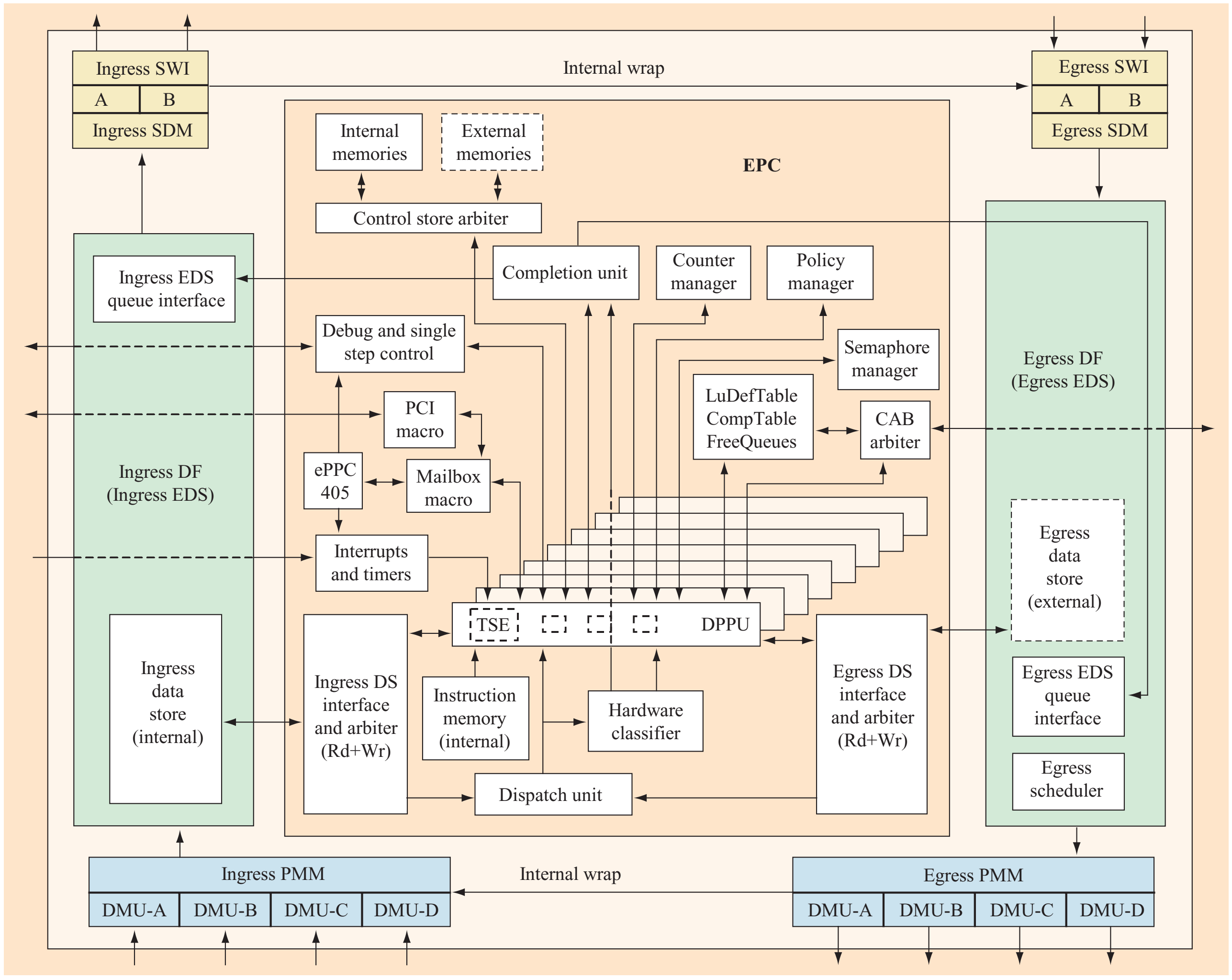

Figure 4

PowerNP functional block diagram.

Functional blocks

Figure 4 shows the main functional blocks that make up the PowerNP architecture. In the following sections we discuss each functional block within the PowerNP.

Physical MAC multiplexer

The physical MAC multiplexer (PMM) moves data between physical layer devices and the PowerNP. The PMM interfaces with the external ports of the network processor in the ingress PMM and egress PMM directions. The PMM includes four data mover units (DMUs), labeled A, B, C, and D. Each of the four DMUs can be independently configured as an Ethernet MAC or a POS interface. The PMM keeps a set of performance statistics on a per-port basis in either mode. Each DMU moves data at 1 Gb/s in both the ingress and the egress directions. There is also an "internal wrap" link that enables traffic generated by the egress side of the PowerNP to move to the ingress side without going out of the chip.

When a DMU is configured for Ethernet, it can support either one port of 1 Gigabit Ethernet or ten ports of Fast Ethernet (10/100 Mb/s). To support 1 Gigabit Ethernet, a DMU can be configured as either a gigabit media-independent interface (GMII) or a ten-bit interface (TBI). To support Fast Ethernet, a DMU can be configured as a serial media-independent interface (SMII) supporting ten Ethernet ports. Operation at 10 or 100 Mb/s is determined by the PowerNP independently for each port.

When a DMU is configured for POS mode, it can support both clear-channel and channelized optical carrier (OC) interfaces. A DMU supports the following types and speeds of POS framers: OC-3c, OC-12, OC-12c, OC-48, and OC-48c.[3] To provide an OC-48 link, all four DMUs are attached to a single framer, with each DMU providing four OC-3c channels or one OC-12c channel to the framer. To provide an OC-48 clear channel (OC-48c) link, DMU A is configured to attach to a 32-bit framer and the other three DMUs are disabled, providing only interface pins for the data path.

Switch interface

The switch interface (SWI) supports two high-speed data-aligned synchronous link (DASL)[4] interfaces, labeled A and B, supporting standalone operation (wrap), dual-mode operation (two PowerNPs interconnected), or connection to an external switch fabric. Each DASL link provides up to 4 Gb/s of bandwidth. The DASL links A and B can be used in parallel, with one acting as the primary switch interface and the other as an alternate switch interface for increased system availability. The DASL interface is frequency-synchronous, which removes the need for asynchronous interfaces that introduce additional interface latency. The ingress SWI side sends data to the switch fabric, and the egress SWI side receives data from the switch fabric. The DASL interface enables up to 64 network processors to be interconnected using an external switch fabric.

The ingress switch data mover (SDM) is the logical interface between the ingress enqueuer/dequeuer/scheduler (EDS) packet data flow, also designated as the ingress DF, and the switch fabric cell data flow. The ingress SDM segments the packets into 64-byte switch cells and passes the cells to the ingress SWI. The egress SDM is the logical interface between the switch fabric cell data flow and the packet data flow of the egress EDS, also designated as the egress DF. The egress DF reassembles the switch fabric cells back into packets. There is also an "internal wrap" link which enables traffic generated by the ingress side of the PowerNP to move to the egress side without going out of the chip.

Data flow and traffic management

The ingress DF interfaces with the ingress PMM, the EPC, and the SWI. Packets that have been received on the ingress PMM are passed to the ingress DF. The ingress DF collects the packet data in its internal data store (DS) memory. When it has received sufficient data (i.e., the packet header), the ingress DF enqueues the data to the EPC for processing. Once the EPC processes the packet, it provides forwarding and QoS information to the ingress DF. The ingress DF then invokes a hardware-configured flow-control mechanism and then either discards the packet or places it in a queue to await transmission. The ingress DF schedules all packets that cross the ingress SWI. After it selects a packet, the ingress DF passes the packet to the ingress SWI.

The ingress DF invokes flow control when packet data enters the network processor. When the ingress DS is sufficiently congested, the flow-control actions discard packets. The traffic-management software uses the information about the congestion state of the DF, the rate at which packets arrive, the current status of the DS, and the current status of target blades to compute *transmit probabilities* for various flows. The ingress DF has hardware-assisted flow control which uses the software-computed transmit probabilities along with tail drop congestion indicators to determine whether a forwarding or discard action should be taken.

The egress DF interfaces with the egress SWI, the EPC, and the egress PMM. Packets that have been received on the egress SWI are passed to the egress DF. The egress DF collects the packet data in its external DS memory. The egress DF enqueues the packet to the EPC for processing. Once the EPC processes the packet, it provides forwarding and QoS information to the egress DF. The egress DF then enqueues the packet either to the egress scheduler, when enabled, or to a target port queue for transmission to the egress PMM. The egress DF invokes a hardware-assisted flow-control mechanism, like the ingress DF, when packet data enters the network processor. When the egress DS is sufficiently congested, the flow-control actions discard packets.

The egress scheduler provides traffic-shaping functions for the network processor on the egress side. It addresses functions that enable QoS mechanisms required by applications such as the Internet protocol (IP)-differentiated services (DiffServ), multiprotocol label switching (MPLS), traffic engineering, and virtual private networks (VPNs). The scheduler manages bandwidth on a per-packet basis by determining the bandwidth required by a packet (i.e., the number of bytes to be transmitted) and comparing this against the bandwidth permitted by the configuration of the packet flow queue. The bandwidth used by a first packet determines when the scheduler will permit the transmission of a subsequent packet of a flow queue. The scheduler supports traffic shaping for 2K flow queues.

Embedded processor complex

The embedded processor complex (EPC) performs all processing functions for the PowerNP. It provides and controls the programmability of the network processor. In general, the EPC accepts data for processing from both the ingress and egress DFs. The EPC, under

[3] The transmission rate of OC-n is $n \times 51.84$ Mb/s. For example, OC-12 runs at 622.08 Mb/s.
[4] Other switch interfaces, such as CSIX, can currently be supported via an interposer chip. On-chip support for CSIX will be provided in a future version of the network processor.

software control and with hardware-assisted coprocessors, determines what forwarding action is to be taken on the data. The data may be forwarded to its final destination or may be discarded.

Within the EPC, eight dyadic protocol processor units (DPPUs) containing processors, coprocessors, and hardware accelerators support functions such as packet parsing and classification, high-speed pattern search, and internal chip management.

Each DPPU consists of two programmable core language processors (CLPs), eight shared coprocessors, one coprocessor data bus, one coprocessor command bus, and a 4KB shared memory pool—1 KB per thread. Each CLP supports two threads. Although there are 32 independent threads, each CLP can execute the instructions of only one of its threads at a time, so at any instant up to 16 threads are executing simultaneously. Most coprocessors perform specialized functions and operate concurrently with one another and with the CLPs. The 16 CLPs run at 133 MHz, providing 2128 MIPS aggregate processing capability.

Each CLP is a 32-bit "picoprocessor" (i.e., a scaled-down RISC processor) consisting of sixteen 32-bit or thirty-two 16-bit general-purpose registers (GPRs) per thread and a one-cycle ALU supporting an instruction set that includes

- Binary addition and subtraction.
- Bit-wise logical AND, OR, and NOT.
- Compare.
- Count leading zeros.
- Shift left and right logical.
- Shift right arithmetic.
- Rotate left and right.
- Bit-manipulation commands: set, clear, test, and flip.
- GPR transfer of halfword-to-halfword, word-to-word, and halfword-to-word with and without sign extensions.

All ALU instructions support "predicated execution." Predicated execution can eliminate hard-to-predict branches by translating them into predicate definitions, which do not require prediction. For short branches, predicated execution provides a performance improvement over traditional test-and-branch.

A thread has a unique set of general-purpose, scalar, and array registers. A thread shares execution resources in the CLP with another thread and execution resources in the coprocessors with three other threads. Five types of threads are supported:

- *General data handler (GDH)*
  Seven DPPUs contain the GDH threads for a total of 28 GDH threads. GDHs are used for forwarding packets.
- *Guided frame handler (GFH)*
  There is one GFH thread available in the EPC. A guided packet (or frame) can be processed only by the GFH thread, but the GFH can be configured to process data packets like a GDH thread. Guided packets are used for transporting control plane information. The GFH executes guided-packet-related code, runs the code related to network processor management, and exchanges control information with a CP function or a remote network processor. When there is no such task to perform and the option is enabled, the GFH may execute packet-forwarding-related code.
- *General table handler (GTH)*
  The GTH thread executes tree-management commands. It performs actions including hardware assist to perform tree inserts, tree deletes, and tree aging. Other threads can also execute tree-insert and -delete operations, achieving about half a million inserts/deletes per second. The GTH can process data packets like a GDH when there are no tree-management functions to perform.
- *General PowerPC handler request (GPH-Req)*
  There is one GPH-Req thread available in the EPC. The GPH-Req thread processes packets targeted for the embedded PowerPC 405. The GPH-Req thread moves data targeted for the PowerPC to the PowerPC mailbox (i.e., a memory area) and then notifies the PowerPC that it has data to process.
- *General PowerPC handler response (GPH-Resp)*
  There is one GPH-Resp thread available in the EPC. The GPH-Resp thread processes responses from the embedded PowerPC. Work for this thread is dispatched due to an interrupt initiated by the PowerPC and does not use the dispatch unit memory. All of the information used by this thread is found in the embedded PowerPC mailbox.

One DPPU contains the GFH, GTH, and PowerPC threads, and the other seven DPPUs contain the GDH threads.

The interrupts and timers component of the EPC supports four interrupt vectors. Each interrupt can be configured to initiate a dispatch to one of the threads for processing. The network processor also has four timers that can be used to generate periodic interrupts.

The instruction memory consists of eight embedded RAMs that are loaded during initialization and contain the code for forwarding packets and managing the system. It is designed so as to be able to feed all 16 picoprocessors simultaneously. Its size is 128 KB (i.e., 32K word instructions).

The control store arbiter (CSA) controls access to the control store (CS), which allocates memory bandwidth among the threads of all DPPUs. The CS is shared among the tree search engine (TSE) coprocessors, and the code can access the CS directly through commands to the TSE coprocessor. The TSE coprocessor also accesses the CS during tree searches.

The TSE coprocessor provides hardware search operations for full match (FM) trees, longest prefix match (LPM) trees, and software-managed trees (SMTs). Software initializes and maintains trees. Leaves can be inserted into and removed from FM and LPM trees without a CP intervention, permitting scalable configurations with the CP control when needed. The FM trees provide a mechanism for searching tables with fixed-size patterns, such as layer-2 Ethernet Unicast MAC tables, which use fixed six-byte address patterns. Searches of FM trees are efficient because FM trees benefit from hash functions. The TSE offers multiple fixed hash functions that provide very low collision rates. The LPM trees provide a mechanism for searching tables with variable-length patterns or prefixes, such as layer-3 IP forwarding tables, where IP addresses can be full-match host addresses or prefixes for network addresses. The SMT trees provide a mechanism for creating trees that follow a CP-defined search algorithm, such as an IP quintuple filtering table containing IP source address, IP destination address, source port, destination port, and protocol ID. In contrast to FM and LPM, SMT allows leaves to specify ranges (for instance, that a source port must be in the range of 100 . . . 110).

LuDefTable, CompTable, and FreeQueues are tables and queues for use by the TSE coprocessor. The lookup definition table (LuDefTable), an internal memory structure that contains 128 entries to define 128 trees, is the main structure that manages the CS. The table indicates in which memory (i.e., external DDR-SDRAM, SRAM, or internal RAM) trees exist, whether caching is enabled, key and leaf sizes, and the type of search to be performed.

The dispatch unit dequeues packet information from the ingress DF and egress DF queues. After dequeue, the dispatch unit reads part of the packet from the ingress or egress DS and places it in an internal RAM inside the dispatch unit. As soon as a thread becomes idle, the dispatch unit places the packet in the shared memory pool and passes the packet control information to the thread for processing. The dispatch unit also handles timers and interrupts by dispatching the work required for these to an available thread.

The hardware classifier (HC) parses packet data that is dispatched to a thread. The classification results are used to precondition the state of a thread by initializing the general-purpose and coprocessor scalar registers of the thread and a starting instruction address (SIA) for the CLP. Classification results indicate the type of layer-2 encapsulation, as well as some information about the layer-3 packet. Some recognizable layer-2 encapsulations include point-to-point protocol (PPP) and virtual local-area-network (VLAN) tagging. Reportable layer-3 information includes IP protocol, five programmable network protocols, the detection of IP option fields, user datagram protocol (UDP), and transmission control protocol (TCP).

Each thread has read and write access to the ingress and egress DS through a DS coprocessor. The ingress and egress DS interface and arbiters are for controlling accesses to the DS, since only one thread at a time can access either DS.

Each thread has access to the control access bus (CAB) arbiter, which permits access to all memory and registers in the network processor. The CAB arbiter arbitrates among the threads for access to the CAB.

The debug and single-step control component enables the GFH thread or CABwatch to control each thread on the device for debugging purposes. For example, the CAB can be used by the GFH thread to run a selected thread in single-step execution mode.

The policy manager is a hardware-assist component of the EPC that performs policy management on up to 1K ingress flows. It supports the “single-rate three-color-marker” and “two-rate three-color-marker” algorithms which can be operated in colorblind or color-aware mode. The algorithms are specified in [7, 8].

The counter manager is a hardware-assist component used by the EPC to manage counters defined by the code for statistics, flow control, and policy management. The counter manager is responsible for counter updates, reads, clears, and writes.

The semaphore manager assists in controlling access to shared resources, such as tables and control structures, through the use of semaphores. It grants semaphores either in dispatch order (“ordered semaphores”) or in request order (“unordered semaphores”).

The completion unit (CU) performs two functions:

- It provides the interfaces between the EPC and the ingress and egress DFs. Each DF performs an enqueue action whereby a packet address, together with appropriate parameters, is queued in a transmission queue or a dispatch unit queue.
- The CU guarantees packet sequence. Since multiple threads can process packets belonging to the same flow, the CU ensures that all packets are enqueued in the ingress or egress transmission queues in the proper order.

Sequence of events of a packet traversing a PowerNP network processor

This section describes the sequence of events that occur from the time a packet is received by a PowerNP processor until it is transmitted out. There are too many data flow

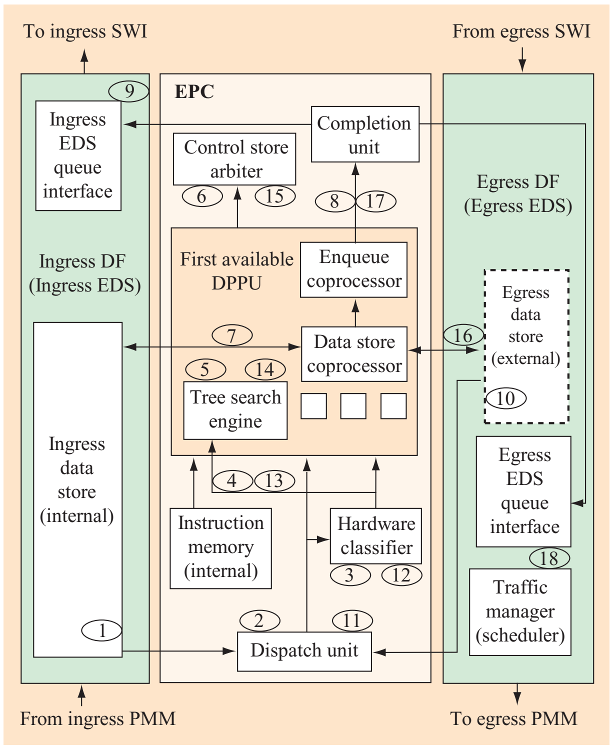

Figure 5

Sequence of events for a packet traversing a PowerNP network processor.

routes and possibilities to fully describe here. However, data generally moves through the network processor (**Figure 5**) as shown in the following sections.

Ingress side

The ingress PMM receives a packet from an external physical layer device and forwards it to the ingress DF:

1. The ingress DF enqueues the packet to the EPC.
2. The dispatch unit fetches a portion of the packet and sends it to the next available thread.
3. Simultaneously, the HC determines the starting instruction address, parses different packet formats (for example, bridged and IP), and forwards the results to the thread.
4. The code examines the information from the HC and may examine the data further; it assembles search keys and launches the TSE.
5. The TSE performs table searches, using search algorithms based on the format of the downloaded tables.
6. The CSA allocates CS memory bandwidth among the protocol processors.
7. Packet data moves into the memory buffer of the DS coprocessor.
   - Forwarding and packet alteration information is identified by the results of the search.
   - The ingress DF can insert or overlay VLAN tags on the packet (hardware-assisted packet alteration), or the code can allocate or remove buffers to allow alteration of the packet.
8. The enqueue coprocessor builds the necessary information to enqueue the packet to the SWI and provides it to the CU, which guarantees the packet order as the data moves from the 32 threads of the DPPUs to the ingress DF queues.
9. The packet is enqueued to the ingress DF:
   - The ingress DF forwards the packet to the ingress scheduler.
   - The ingress scheduler selects the packet for transmission to the ingress SWI. The entire packet is not sent at once; the ingress scheduler sends it a cell at a time.
   - With the help of the ingress DF, the ingress switch data mover (I-SDM) segments the packets from the switch interface queues into 64-byte cells and inserts cell header and packet header bytes as they are transmitted to the SWI.

The ingress SWI forwards the packet to a switch fabric, to another PowerNP processor, or to the egress SWI of the device.

Egress side

The egress SWI receives a packet from a switch fabric, from another PowerNP processor, or from the ingress SWI of the device. The egress SWI forwards the packet to the egress DF:

10. The egress DF enqueues the packet to the EPC.
11. The dispatch unit fetches a portion of the packet and sends it to the next available thread.
12. Simultaneously, the HC determines the starting instruction address, parses different packet formats (e.g., bridged and IP), and forwards the results to the thread.
13. The code examines the information from the HC and may examine the data further; it assembles search keys and launches the TSE.
14. The TSE performs table searches, using search algorithms based on the format of the downloaded tables.
15. The CSA allocates CS memory bandwidth among the protocol processors.
16. Forwarding and packet-alteration information is identified by the results of the search. The PowerNP

provides two packet-alteration techniques: hardware-assisted packet alteration and flexible packet alteration:

- In hardware-assisted packet alteration, commands are passed to the egress DF hardware during enqueueing. These commands can, for example, update the TTL field in an IP header, generate packet CRC, or overlay an existing layer-2 wrapper with a new one.
- In flexible packet alteration, the code allocates additional buffers, and the DS coprocessor places data in these buffers. The additional buffers allow prepending of data to a received packet and bypassing part of the received data when transmitting. This is useful for packet fragmentation when an IP header and MAC header must be prepended to received data in order to form a packet fragment of the correct size.

17. The enqueue coprocessor develops the necessary information to enqueue the packet to the egress DF and provides it to the CU, which guarantees the packet order as the data moves from the 32 threads of the DPPUs to the egress DF queues.
18. The packet is enqueued to the egress DF:
    - The packet is forwarded to the egress scheduler if enabled.
    - The egress scheduler selects the packet for transmission to a target port queue.
    - If the scheduler is not enabled, the DF forwards the packet directly to a target queue.
    - The egress DF selects packets for transmission from the target port queue and moves their data to the egress PMM.

The egress PMM sends the packet to an external physical layer device.

System software architecture

The PowerNP system software architecture is defined around the concept of partitioning control and data planes, as shown in **Figure 6**. This is consistent with industry and standard directions, for example, the Network Processing Forum (NPF) and the Forces group of the Internet Engineering Task Force (IETF) organization. The non-performance-critical functions of the control plane run on a GPP, while the performance-critical data plane functions run on the PowerNP processing elements (i.e., CLPs). In order to optimize instruction memory use, the data plane is partitioned so that only the performance-critical paths run on the PowerNP processor. For example, there is no need to use the instruction memory on the network processor for IP options processing, given that options are associated with only a small percentage of IP packets. The software architecture and programming model describes the data plane functions and APIs, the control plane functions and APIs, and the communication model between these components. The software architecture supports a distributed system model including multiple PowerNP processors and GPPs.

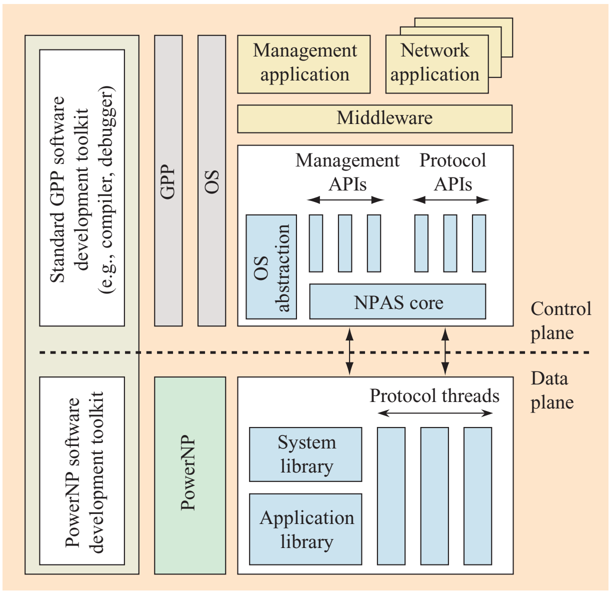

Figure 6

PowerNP system software architecture.

Data plane

The data plane is structured as two major components:

- *System library*
  The system library provides basic functions such as memory management, debugging, and tree services. These functions provide a hardware abstraction layer that can be used either from the control plane, using a message-passing interface, or from the data plane software, using API calls.
- *Forwarding software*
  The forwarding software is responsible for performing high-speed packet processing running the data plane steady-state portion of networking applications (such as the IP packet-forwarding protocols).

These components plus the overall software design help a programmer to develop the PowerNP networking applications quickly. The data plane programming model is based on

- *Run-to-completion (as described earlier)*

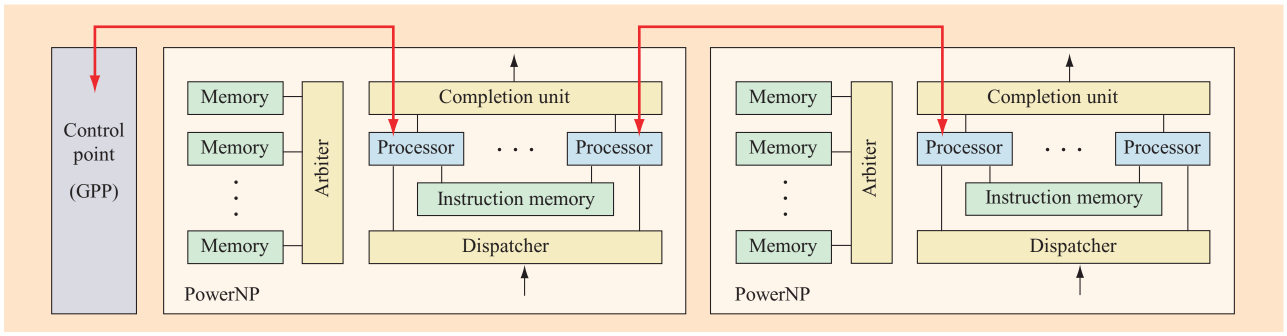

Figure 7

PowerNP message-based control model.

- *Message-based control (MBC), as shown in* ***Figure 7***
  The MBC model provides scalability in configuration, supporting multiple network processors and control processors, and achieves a scalable system architecture. MBC supports transparency in terms of the location of the CP and transparency in terms of the connection media (PCI and Ethernet, etc.). MBC also provides a seamless interface between CP-to-PowerNP processors and between PowerNP-to-PowerNP processors.
- *Distributed control processing*
  The architecture supports a distributed control processing (DCP) model in which any thread in the PowerNP can execute performance-critical control tasks, as opposed to a single control-processing unit in the PowerNP processor.

Control plane

The control plane networking applications interface with the data plane functions using the network processor application services (NPAS) layer that exposes two types of APIs:

- *Protocol services API (such as IPv4 and MPLS)*
  This set of services handles hardware-independent protocol objects.
- *Management services API (such as the PowerNP initialization and debug)*
  This set of services manages hardware resources.

The control plane software architecture and APIs play a pivotal role in the integration of the PowerNP in a communication system. The control plane software provides a way to manage the PowerNP, and it also provides a set of APIs which can be used to support generic control plane protocol stacks. This support is provided without imposing any constraints on the software environment (i.e., operating system and development tools).

Network processor application services structure (NPAS)

The NPAS is written using an operating system (OS) independent layer so that it can run on any OS. It provides a rich set of APIs that can be used to develop networking applications quickly. Additionally, it provides the basis (such as standard application binding, initialization services, resource management, and error logging) for developing customized APIs, and is composed of four major components:

- *Transport services*
  This set of services provides a unified layer for exchanging control and data packets between the CP and the PowerNP.
- *Message services*
  This set of services provides a library of control messages that enables the CP to exchange commands with the network processor. The network processor interprets the commands and takes appropriate actions.
- *Management services*
  This set of services is used to manage and to abstract the network processor hardware resources, such as the trees or the flow queues.
- *Protocol services*
  This set of services is used to manage the protocol objects, such as IPv4 forwarding entries or PPP interface.

The NPAS is provided in the form of a library. The various APIs are implemented in C language, and the invocation of an API is done by a function call. Since communication with the network processor is performed with guided packets (i.e., frames), the API software model is *asynchronous*, which means that an API call is not blocked while waiting for a response from the network processor. A registration mechanism provides the binding between any custom application and the NPAS to send and receive any kind of data through the APIs.

Most of the APIs provide the support for unicast and multicast delivery of the commands and data packets to the network processor(s). Because the multicast delivery utilizes the capabilities of the network processor and/or the one provided by the switch connecting multiple network processors, this feature dramatically improves the performance of the CP. This feature is particularly important when an application is starting, as it usually wants to populate many network processors with the same information.

The look and feel of the APIs is consistent inside the NPAS. Even more significant, however, all APIs have the same behavior. This allows a programmer to use the same mechanisms to interface with all APIs, whether the APIs are related to protocol or resource management services. One can even create customized APIs inside the NPAS by simply following the same rules and taking any existing API as an example.

The NPAS can easily be ported to any operating system and can be used with any application because it is designed to run on any OS and to connect to the network processor via any transport mechanism. It has two components which help to provide this ease of portability and integration:

- *System services*
  This set of services provides an interface between the NPAS and OS. The system services provide a thin layer which maps some primitives defined to handle resources such as memory, buffers, and timers into the corresponding OS calls.
- *Physical transport services*
  This set of services handles the transmission and reception of packets on the actual media. The physical transport services provide a consistent interface to the transport services, regardless of the physical transport mechanism used to communicate with the network processor.

Software development toolkit

The PowerNP software development toolkit provides a set of tightly integrated development tools which address each phase of the software development process. The toolkit includes the following:

- Assembler/linker (NPAsm).
- System simulator (NPSim).
- Graphical debugger (NPScope).
- Test-case generator (NPTest).
- Software performance profiler (NPProfile).

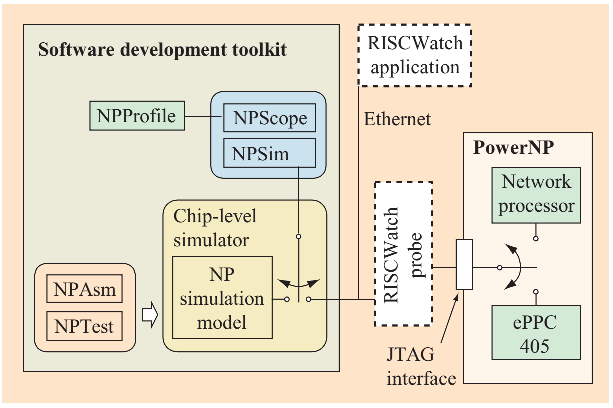

Figure 8

PowerNP software development kit.

All of these tools are written in C++ and are tightly coupled with Tcl/Tk scripting language, which provides scripting and graphical interfaces. The toolkit runs on a wide range of OS platforms. **Figure 8** shows the relationships among the various software tools as well as the actual hardware. The tools are designed to run either on top of a chip-level simulation model or on top of the actual hardware. Connection to the hardware is provided via the industry-standard RISCWatch interface, which also provides access to the on-chip PowerPC processor.

NPScope allows the developer to monitor, execute, and debug picocode written for the PowerNP. NPScope provides the developer the same graphical user interface and capabilities on both the simulation model and the hardware. It provides

- A comprehensive internal view of the PowerNP that includes the registers, memories, buffers, and queues.
- Debugger facilities such as packet injection, viewing of register or memory locations, single step execution, breakpoints, and saving and restoring of a simulation session.

NPSim provides a Tcl/Tk-based scripting interface to the simulation model which allows a developer to simulate the PowerNP chip. Through the use of Tcl/Tk scripts, developers can interact with the simulation model and perform a wide range of functions in support of packet traffic generation and analysis. Through NPSim, the developer can connect the simulation model to a remote process via a standard socket interface. The remote process can then send Ethernet frames or special simulator messages to the simulation model. The remote process can also receive egress data from the simulation model through the same socket interface.

NPProfile provides the developer with the capability to analyze the performance of picocode. NPProfile analyzes simulation event information contained in a message log file produced by NPScope to accumulate relevant data regarding the performance of picocode execution. It then displays the information in graphical charts. The following is a partial list of some of the information available from NPProfile:

- Summary of the average and cumulative number of cycles executed for thread dispatches for each thread.
- Histogram of the number of threads stalled for *n* cycles.
- Coprocessor stall statistics on a per-dispatch basis for the specified thread.

NPTest provides developers with the environment needed for the initial testing of picocode and for test-case regression. It provides a Tcl/Tk scripting interface that allows developers to construct control packets or data packets intended to test the forwarding picocode functions, to load tables with known entries, and to set the network processor to a known state. This information can be saved as Tcl/Tk scripts and then retrieved for input into the simulator.

NPAsm is the assembler for creating images designed to execute on the PowerNP. It generates files used to execute picocode on the chip-level simulation model or the PowerNP, as well as files that picocode programmers can use for debugging.

The chip-level simulation model is a software framework that provides an accurate simulation of the functions of the PowerNP. The model can "execute" (emulate) more than 100 000 picocode instructions per second, allowing quick testing of code under various conditions. It enables a developer to model the flow of packets through a PowerNP-based system, including connections to a CP and to a switch fabric. The simulation model also allows a developer to debug individual pieces of picocode by designing comprehensive test cases and examining execution of source code line by line.

The RISCWatch probe connects a host computer to the PowerNP JTAG interface through Ethernet connections. The same RISCWatch probe can simultaneously debug both PowerNP, with NPScope, and its embedded PowerPC 405 processor, with the RISCWatch application.

Networking applications support

The power and flexibility of the PowerNP is useful in supporting a wide range of existing and emerging applications. A number of networking applications have been implemented on the PowerNP chip, and the following sections discuss two of them.

Small group multicast

Small group multicast (SGM) [9] is a new approach to IP multicast that makes multicast practical for applications such as IP telephony, videoconferencing, and multimedia "e-meetings." Like today's multicast schemes, SGM sends at most one copy of any given packet across any network link, thus minimizing the use of network bandwidth. However, SGM (also known as explicit multicast, or Xcast [10] at the IETF) provides an important complement to today's multicast schemes. Whereas existing multicast schemes can support very large multicast groups, they have difficulty supporting very large numbers of distinct multicast groups. SGM, by contrast, can support very large numbers (actually an unlimited number) of small groups. It does this by eliminating the "per-flow" state information and "per-flow" signaling of traditional multicast protocols. Just as routers can support huge numbers of TCP connections or UDP flows today, since they do not need to track or do any signaling for these individual flows, routers can similarly support huge numbers of simultaneous SGM flows.

SGM avoids the use of per-flow state information and per-flow signaling by including an explicit list of destinations in a new packet header. A router performs a route table lookup to determine the "next hop" for each of the destinations. The router partitions the destinations on the basis of the next hop and sends a packet to each of the next hops that includes for each next hop the list of destinations that are reachable through that next hop. This requires more processing than traditional multicast packet forwarding but this extra cost is well worth paying, since SGM supports unlimited numbers of small groups and thus makes multicast practical for applications such as IP telephony, videoconferencing, multimedia e-meetings, and other applications for which today's multicast schemes are not practical.

Of course, the "extra processing" required by SGM cannot be performed by a "hardwired" ASIC that was developed for standard IP processing prior to the development of SGM, and a general-purpose processor cannot easily perform the SGM extra processing. But for a network processor such as the PowerNP, SGM processing is a simple matter, since the PowerNP provides

- Route table lookup engines (i.e., TSEs) that can be used to look up the next hop for each of the destinations in an SGM packet.
- Checksum engines that can be used to recompute header checksums when SGM packets are forwarded to one or more next hops.
- Programmability that allows the above functions to be combined in nontraditional ways.
- An array of picoprocessors providing an abundance of processing power.

Thus, the PowerNP makes SGM practical and enables a variety of applications that are not practical today. A preliminary version of SGM has been implemented on the PowerNP chip.

GPRS tunneling protocol

General packet radio service (GPRS), a set of protocols for converging the mobile data with the IP packet data, presents new challenges to the equipment manufacturers of GPRS support nodes (GSNs) as the bandwidth available to mobile terminals increases significantly with wireless technology advances. The routing challenges introduced by the mobility of terminals are supported by tunneling packets between the GSN of a wireless network provider, similarly to the mobile IP scheme [11]. Therefore, in addition to the common functions performed by an IP router, such as routing table lookup and packet forwarding, a GSN has to encapsulate or decapsulate IP packets as part of the stateful GPRS tunneling protocol (GTP) [12]. A GTP context containing information pertaining to the mobile terminal must be stored for each terminal registered in the network. A GSN must therefore not only alter and forward IP packets at media speed, but also store and retrieve millions of GTP contexts.

We consider the support of GTP as a typical networking application that requires a high memory/bandwidth product and deeper packet processing than the common packet forwarding of an IP router. The PowerNP, with its massive processing power, large memory, and programmability, is ideally suited for this type of application. We have identified four data-path functions necessary to provide GTP support on the network processor:

- GTP encapsulation and decapsulation.
- Traffic-volume recording for billing.
- Packet reordering based on the GTP sequence numbers.
- Flow mirroring for legal interception.

The encapsulation process requires the retrieval of a GTP context based on the IP address of the packet being encapsulated and the construction of the GTP header using information contained in the context. Traffic counters associated with the context are incremented to account for the data transmission. A header chain composed of the GTP, a UDP, and an IP header is then prepended to the packet.

The decapsulation process requires the retrieval of the GTP context from the IP address of the inner IP header. The content of the GTP header is then checked against the content of the context to prevent any "spoofing" of the data contained in the GTP header (e.g., setting a fake IP address in the datagram). Traffic counters associated with the context are incremented to account for the data transmission. The outer header chain composed of the IP, UDP, and GTP headers is stripped, and normal IP forwarding is applied to the decapsulated packet.

Packet reordering based on the GTP sequence number requires the temporary storage of misordered packets in a per-context associated reordering queue. Packets are stored in the egress data store of the PowerNP to make use of the large memory available. Packets belonging to the same flow (i.e., associated with the same GTP context) are daisy-chained using pointers to form an ordered queue. As a packet is already stored in the egress data store when the EPC starts a thread to process it, the packet does not have to be copied but has only to be inserted by reference at the right place in the reordering queue. Semaphores are used to ensure the consistency when manipulating the reordering queue structures.

Flow mirroring for legal interception requires that the packets of some flows be mirrored and sent to a legal interception system. The multicast support on the PowerNP provides the basic functions to send multiple copies of the same frame to different ports.

Our GTP design makes extensive use of the hardware coprocessors available in the network processor. GTP contexts are organized as a tree, and the TSE coprocessor is used to retrieve the GTP context associated with an incoming packet. Counters for traffic accounting are incremented using the counter manager coprocessor. Header prepending and stripping use the packet alteration hardware assist. Consistency in the assignment and verification of GTP sequence numbers, and in operations on the reordering queues, is ensured by using the semaphore coprocessor. Generic services for table, memory, and counter management are already available in the NPAS. Thus, the management of the GTP lookup, the GTP context table, and the GTP counter tables by the CP is greatly simplified by making use of these services.

A demonstrator of the GTP encapsulation and decapsulation functions and the traffic-volume recording has been designed, developed, and tested during a three-month internship by a group of four students with no prior knowledge of the PowerNP architecture and development environment. A prototype of the GTP frame-reordering function is currently under development.

Further advanced applications, such as network address translation (NAT), are under consideration as complementary functions to the GPRS extensions. Integrated support of these two services can help networking-equipment manufacturers develop high-performance integrated-gateway GPRS service nodes.

Performance

The PowerNP picoprocessors provide 2128 MIPS of aggregate processing capability. However, this figure alone

Table 1 PowerNP cycle budgets for Ethernet and POS.

Maximum wire speed (Gb/s)	*Type of packet/ Minimum packet size* (bytes)	*Maximum packets per second possible for minimum size*	*Maximum TSE cycles per packet available* (cycles)	*Maximum code cycles available per packet* (cycles)	*Assumed dead time between packets*
3	Ethernet/64	4.5 million	948	474	20 bytes (preamble)
2.4 (OC-48)	PPP POS/48	6.1 million	700	350	1 byte (flag byte)
2.4 (OC-48)	PPP POS/51	5.88 million	725	362	1 byte (flag byte)

does not represent the total processing capability of the PowerNP, since it encompasses a large number of specialized coprocessors that are used in conjunction with picoprocessors for packet processing. This makes providing a single performance figure for the PowerNP a challenging problem. Given this, we discuss two performance measures: the raw cycle budget per packet for POS and Ethernet, and available headroom over typical IP packet forwarding.

Table 1 summarizes the PowerNP cycle budgets for both Ethernet and POS applications. Packet rates for minimum packet sizes are also listed. Packet rates not only determine how often a particular forwarding path must be executed, but also relate directly to several data-flow throughput limitations—for example, the rate at which the device can dispatch new packets to the EPC for processing. Note that only half of the TSE cycles can be used to execute picocode[5] if they are not used for tree-search activity. This limitation is due to the dual threads on each processor sharing the available machine cycles. Also note from Table 1 that Ethernet has a significantly larger cycle budget because of larger minimum packet size and the 20-byte interpacket gap, making more complex forwarding paths achievable at media speed for Ethernet connections. Two packet rates are quoted for POS applications. To sustain media speed with 48-byte packets, 6.1 million packets per second, the egress DS must run with a 10-clock cycle data store access window. The lower rate of 5.88 million packets per second corresponds to a DS running with an 11-clock cycle access window.

The cycle budgets quoted in Table 1 do not correlate directly with instruction path lengths, since some instructions require multiple cycles to execute. Similarly, some cycles are not usable for instruction execution (i.e., both threads waiting for search result). To quantify expected performance for specific applications, associated code paths are profiled in terms of memory accesses, coprocessor use, program flow (i.e., branch instructions), and overlap of coprocessor operations with instructions or other coprocessor operations. Database structures used for tree searches and other table accesses must also be profiled. Once a code path is profiled, an event-driven simulation model is used to project performance for the corresponding application. The following code paths from the PowerNP software package achieve OC-48 line speed at minimum packet size:

- Border gateway protocol (BGP) layer-3 routing.
- BGP layer-3 routing with behavior aggregate (BA) lookup for QoS.
- MPLS forwarding.

A significant advantage of network processors, as compared to a fixed-function design, is the availability of headroom[6] that can be applied to enhance functionality. Several different perspectives of headroom are illustrated in **Figure 9**. Of the code paths described in the previous paragraph, the BGP layer-3 routing with BA lookup comes closest to fully consuming the PowerNP processor resources, as illustrated in the third column in the figure. Removing the BA lookup could potentially make room for alternate functionality, as shown in the second column in the figure.

It is important to recognize that the PowerNP software package goes significantly beyond basic IP forwarding in terms of both enhanced features and robust error checking. A more basic IP forwarding path has been written with half the code path length, resulting in a corresponding increase in headroom, as illustrated in the first column of the figure. The first three columns of the figure illustrate headroom within the context of media speed and minimum packet size. The fourth column illustrates the impact of using average packet size instead of minimum packet size to define headroom. The average packet size may be as much as ten times the minimum

[5] Picocode is the picoprocessor assembly language program.

[6] The term *headroom* denotes the processing power beyond that required by the application.

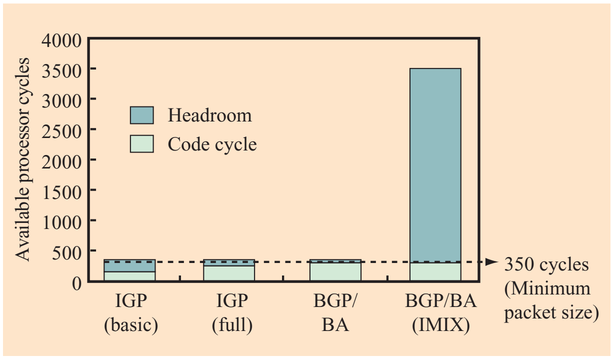

Figure 9

PowerNP processing cycle headroom.

packet size (i.e., 476 bytes vs. 48 bytes),[7] resulting in ten times the number of available processing cycles per packet. Although these excess cycles for basic functions may not be available during benchmarking tests, there are many examples of enhanced functions that are required only within the context of real Internet traffic.

Summary

As the demand on network edge equipment increases to provide more services on behalf of the core and the end user, the role of flexible and programmable network processors becomes more critical. In this paper, we have discussed the challenges and demands posed by next-generation networks and have described how network processors can address these issues by performing highly sophisticated packet processing at line speed. We have presented an overview of the IBM PowerNP NP4GS3, a programmable, high-performance network processor. Its hardware and software design characteristics make it an ideal component for a wide range of networking applications. Its run-to-completion model supports a simple programming model and a scalable system architecture, which provide abundant functionality and headroom at line speed. Because of the availability of associated advanced development and simulation tools, combined with extensive reference implementations, rapid prototyping and development of new high-performance applications are significantly easier than with either GPPs or ASICs.

Acknowledgments

We gratefully acknowledge the significant contributions of a large number of our colleagues at IBM who, through years of research, design, and development, have helped to create and document the PowerNP network processor. The processor would not have been possible without the appreciable efforts and contributions by those colleagues.

References

1. J. Wagner and R. Leupers, "C Compiler Design for a Network Processor," *IEEE Trans. Computer-Aided Design Int. Circuits & Syst.* **20,** No. 11, 1302–1308 (November 2001).
2. T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, "Building a Robust Software Based Router Using Network Processors," *Oper. Syst. Rev.* **35,** No. 5, 216–229 (December 2001).
3. P. Gupta and N. McKeown, "Algorithms for Packet Classification," *IEEE Network* **15,** No. 2, 24–32 (April 2001).
4. D. Herity, "Network Processor Programming," *Embedded Syst. Program.* **14,** No. 8, 33–52 (August 2001).
5. E. Strohmaier, J. J. Dongarra, and H. W. Meuer, "Marketplace of High-Performance Computing," *Parallel Comput.* **25,** No. 13, 1517–1544 (December 1999).
6. T. Werner and V. Akella, "Asynchronous Processor Survey," *Computer* **30,** No. 11, 67–76 (November 1997).
7. J. Heinanen and R. Guerin, "A Single Rate Three Color Marker," *Internet Engineering Task Force (IETF) Request for Comments (RFC) 2697*, September 1999; see *http://www.ietf.org/*.
8. J. Heinanen and R. Guerin, "A Two Rate Three Color Marker," *IETF RFC 2698*, September 1999; see *http://www.ietf.org/*.
9. R. Boivie, N. Feldman, and C. Metz, "Small Group Multicast: A New Solution for Multicasting on the Internet," *IEEE Internet Comput.* **4,** No. 3, 75–79 (May 2000).
10. R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, and O. Paridaens, "Explicit Multicast (Xcast) Basic Specification," IETF Internet Draft, October 2001; see *http://www.ietf.org/*.
11. C. Perkins, "IP Mobility Support," *IETF RFC 2002*, October 1996; see *http://www.ietf.org/*.
12. Third Generation Partnership Project (3GPP), GPRS Tunneling Protocol (GTP), *TS 29.060 Version 3.2.2*, December 1999; see *http://www.3gpp.org/*.

Received February 11, 2002; accepted for publication September 3, 2002

[7] The "Internet mix" (IMIX) packet size is defined in *www.lightreading.com/*.

James R. Allen, Jr. *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (jallenjr@us.ibm.com).* Mr. Allen received a B.S. degree in electrical engineering from North Carolina State University in 1981. He joined IBM that same year and worked on the development of IBM communication controllers. He is currently a Senior Engineer working on network processor architecture. Mr. Allen holds several patents in the areas of networking and network processors. His professional interests include networking, network processors, wireless communications, and network security.

Brian M. Bass *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (bass@us.ibm.com).* Mr. Bass is a Senior Engineer, working on network processing. He received an M.S. degree in electrical engineering from Clemson University in 1984. He is one of the original architects for the IBM PowerNP family of network processors and has worked on their architecture, design, and laboratory "bringup." Mr. Bass has applied for more than 45 patents in the network processing and communication area.

Claude Basso *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (cbasso@us.ibm.com).* Mr. Basso is a Distinguished Engineer, in charge of software architecture development for IBM network processors. He received an engineering degree in computer science from the Grenoble Polytechnic Institute, France. Since joining IBM in 1985, he has been involved in the definition and development of many products including communication controllers and ATM switches. Mr. Basso holds more than 40 patents in the networking field.

Richard H. Boivie *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (rhboivie@us.ibm.com).* Dr. Boivie manages the Advanced Network Technologies Department at the Thomas J. Watson Research Center. He received B.S. and M.S. degrees in electrical engineering from Rensselaer Polytechnic Institute and a Ph.D. in computer science from the State University of New York at Stony Brook. Among other things, he was the technical lead and then manager of the group in IBM that developed the hardware and software for the NSFnet—the principal backbone of the Internet from 1988 to 1995 and the foundation for today's Internet. His interests include operating systems, networking, multicast, security, and cryptography.

Jean L. Calvignac *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (calvi@us.ibm.com).* Mr. Calvignac is an IBM Fellow, currently responsible for the system design of IBM network processors. In 1998, at the IBM Research Triangle Park Laboratory, he and his team initiated the IBM network processor activities. He had previously been responsible for system design of the ATM switching products, which he initiated with his team in 1992 at the IBM La Gaude Laboratory in France. Before then, he had held different management and technical leader positions for architecture and development of the IBM communication controller products at the IBM La Gaude Laboratory. Mr. Calvignac joined IBM in 1971 as a development engineer in telephone switching products. He received an engineering degree in 1969 from the Grenoble Polytechnic Institute, France. He holds more than 120 patents in the field of communication and networking and has published more than 90 papers or contributions for standards. Mr. Calvignac is a Fellow Member of the IEE (in Europe) and a Senior Member of the IEEE.

Gordon T. Davis *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (gtdavis@us.ibm.com).* Mr. Davis is a Senior Technical Staff Member, leading the Network Processor Performance team. He received a B.E.E. degree in electrical engineering from the Georgia Institute of Technology in 1971, and an M.S. degree in electrical engineering from U.C.L.A. in 1974. His expertise includes networking, network processors, performance, digital signal processing, and telecommunications. Mr. Davis is an author of three papers, more than 80 publications in the IBM Technical Disclosure Bulletin, 30 patents, and 48 pending patent applications.

Laurent Frelechoux *IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (frl@zurich.ibm.com).* Mr. Frelechoux is a researcher at the Zurich Research Laboratory. He received an M.S. degree in computer science from the Swiss Federal Institute of Technology of Lausanne, Switzerland, in 1996. His interests include networking and network processors, with a focus on mobile and wireless networking.

Marco Heddes *TranSwitch Corporation, 3 Enterprise Drive, Shelton, Connecticut 06484 (mheddes@onexco.com).* At the time this paper was written, Dr. Heddes was a Network Processor Architect with the IBM Microelectronics Division at Research Triangle Park, North Carolina. He received a Ph.D. degree in electrical engineering from the Eindhoven University of Technology, The Netherlands, in 1995. He joined the IBM Zurich Research Laboratory in Switzerland in 1990, working on switching technologies and VLSI design. In 1998, he transferred to the IBM Research Triangle Park Laboratory, where he made significant contributions to the development of IBM network processors. Dr. Heddes is an author of more than 60 patent applications in the switching and networking areas.

Andreas Herkersdorf *IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (anh@zurich.ibm.com).* Dr. Herkersdorf received a Dipl.-Ing. degree in electrical engineering from the Technical University of Munich, Germany, in 1987, and a Ph.D. degree in electrical engineering from the Swiss Federal Institute of Technology (ETH) in Zurich, Switzerland, in 1991. Since 1988, he has been with the Zurich Research Laboratory, where he currently manages the Network Processor Hardware group. Dr. Herkersdorf's areas of interests are high-speed communication networks and systems, and VLSI design methodologies.

Andreas Kind *IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (ank@zurich.ibm.com).* Dr. Kind is a Research Staff Member at the IBM Zurich Research Laboratory. He received a Ph.D. degree from the University of Bath (UK). Dr. Kind joined the

Network Processor Software group at the Zurich Research Laboratory after working on next-generation converged service routers at the C&C Research Laboratories of NEC Europe. Since 1995, he has worked on open programmable networks.

Joe F. Logan *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (jflogan@us.ibm.com).* Mr. Logan is a Senior Engineer, working on a team responsible for system design and architecture of IBM network processors. He received a B.S. degree in electrical engineering from Clemson University in 1983. In 1984 he joined IBM, where he has worked on the development of numerous components used in networking adapters, bridges, routers, and switches. He was part of the team that designed the first token-ring local area network adapter announced by IBM in 1985, and has designed components supporting other networking protocols such as Ethernet, FDDI, and ATM. His interests include networking, network processors, and high-speed bus architectures. Mr. Logan has received an IBM Outstanding Technical Achievement Award and an IBM Division Award; he is an author of more than ten patents.

Mohammad Peyravian *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (peyravn@us.ibm.com).* Dr. Peyravian is a Network Processor Architect. He received a Ph.D. degree in electrical engineering from the Georgia Institute of Technology in 1992. His interests include networking, network processors, cryptography, security, and wireless telecommunications. Dr. Peyravian is an author of more than 40 journal and conference papers, and more than 30 patents.

Mark A. Rinaldi *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (nalds@us.ibm.com).* Mr. Rinaldi is a Hardware/Software Engineer, currently working on software development tools for IBM network processors. He received an M.S.E.E. degree in electrical engineering from Rensselaer Polytechnic Institute in 1974. His interests include hardware and software architecture, software simulation, and language compilation.

Ravi K. Sabhikhi *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (sravi@us.ibm.com).* Mr. Sabhikhi is a marketing manager in the Network Processor Architecture group of the IBM Microelectronics Division. He received an M.S. degree in computer science and an M.S. degree in business management, both from North Carolina State University. Mr. Sabhikhi has more than 25 years of experience in the networking industry as a developer, architect, and manager.

Michael S. Siegel *IBM Microlectronics Division, Research Triangle Park, North Carolina 27709 (siegelm@us.ibm.com).* Mr. Siegel is a Senior Technical Staff Member. He has been a network processor architect since 1997, providing major contributions to the development of the IBM PowerNP NP4GS3 architecture. Previously, he worked on the development of token ring switches, vector processing, and I/O channel development for IBM mainframes, and he was a coeditor of the 802.5 DTR token ring standard. In 1977 Mr. Siegel joined the IBM Large Systems Division in Poughkeepsie, New York, after receiving a B.S.E.E. degree from Rensselaer Polytechnic Institute. He holds patents in the fields of local area networks, network processors, vector processor design, and scalar processor design, and is a coauthor of "Understanding Token Ring Protocols and Standards," published in 1998.

Marcel Waldvogel *IBM Research Division, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland (mwl@zurich.ibm.com).* Dr. Waldvogel is a Research Staff Member at the IBM Zurich Research Laboratory. He received a Ph.D. degree in electrical engineering from the Swiss Federal Institute of Technology (ETH), Zurich, in 2000. Before joining IBM, he was an Assistant Professor in Computer Science at Washington University in St. Louis. His research interests include algorithms and protocols for high-speed networking and data dissemination.