Controlling Network Processors by using Packet-processing Cores

Yasusi Kanada
Central Research Laboratory, Hitachi, Ltd.
Totsuka-ku Yoshida-cho 292, Yokohama 244-0817, Japan
Yasusi.Kanada.yq@hitachi.com

***Abstract* – A network processor (NP) usually contains multiple packet processing cores (PPCs) and a control processing core (CPC), and the synchronization and communication between CPC and PPCs, which is required for controlling an NP, is very complex. To reduce the complexity, a method for controlling packet processing in NPs by using PPCs is proposed. By means of this method, complex control messages are partially processed and divided into simplified control packets by a CPU outside the NP chip, and these packets are sent to a control-processing PPC. The control-processing PPC controls data-processing PPCs by using data-exchange mechanisms, such as a shared memory or an on-chip network, which are more uniform and simpler than those between a CPC and PPCs. This control method is applied to a virtual-link control-processing task and packet-processing tasks in a network node with a virtualization function. Both tasks are described by a hardware-independent high-level language called "Phonepl," and communication between the PPCs is programmed following normal and uniform shared-memory semantics. As a result, programming the control-processing task and porting the program become much easier.**

***Keywords* – Network processors, Multi core, Control processing, Packet processing, Network virtualization.**

I. INTRODUCTION

A network-processor LSI (Larg-Scale Integration) usually has many packet-processing cores (PPCs) because today's wire rate, namely, 10, 40, or 100 Gbps, is too fast for a single core; that is, a single core cannot process all the packets. To reduce overhead, PPCs have a RISC (Reduced Instruction-Set Computer) core architecture and are used without running an operating system (OS); that is, it operates as in "bare metal" (or "bare bone") mode. Programs used in PPCs are simplified because they are not suited for the complicated processing that requires an OS. In contrast, because control processing such as routing is complicated, a more complex core, i.e., a control processing core (CPC), is built on the same chip and used for controlling PPCs. For example, IXP network processors [Gog 03] developed by Intel have 16 or more PPCs and one StrongARM or XScale CPC.

There are three problems concerning synchronization and communication between PPCs and the CPC that controls the PPCs: the methods for synchronization and communication are complicated, hardware- and vendor-dependent, and not portable. Because a CPC controls PPCs by sending and receiving messages, synchronization and communication between them is required. These problems occur during this process. In addition, standard-based methods for synchronization and communication between a CPC and PPCs are difficult to use.

The above-described problems are caused by the difference between "slow-path" and "fast-path" processing, which require different software and hardware. As for the difference between software, the existence or non-existence of an OS makes the difference. A CPC can support abstract and flexible methods for synchronization and communication, but, usually, PPCs only support specialized and restricted methods, which do not easily match standardized methods. In addition, software libraries for synchronization and communication between PPCs are usually hardware- and vendor-dependent. As for the difference between hardware, hardware design of a CPC and PPCs makes the difference. That is, PPC hardware is optimized for high performance and bare-metal usage. This design imposes restrictions on usage of synchronization and communication mechanisms. In contrast, CPC hardware is optimized for OS-based environments.

In the present study, to solve the above-described problems, a method for controlling packet processing in network processors by using PPCs is proposed. By means of this method, PPCs are used for both data processing and control processing and; that is, one or more PPCs are allocated to control, instead of a CPC, and other PPCs are allocated to data processing. Complex control messages are partially processed and divided into simplified control packets by a CPU outside the network processor chip, and they are sent to the control-processing PPC. This PPC controls data-processing PPCs by using data-exchange mechanisms, such as a shared memory or an on-chip network, which are uniform and simpler than data-exchange mechanisms between a CPC and PPCs.

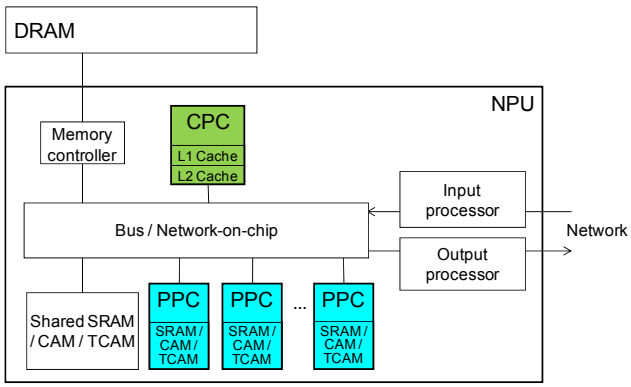
The rest of this paper is organized as follows. Section II describes the methodology used for solving the above-described problems. Section III describes an application of the proposed control method, and Section IV describes a prototype implementation and evaluation of the method. Section V concludes the paper.

II. METHODOLOGY

The methodology used for solving the above-mentioned problems is described in the following.

A. Outline

A typical design of a network processor is shown in **Figure 1**. A network processor, i.e., an LSI chip, contains a CPC and multiple PPCs. Certain types of network processors have two or more types of specialized PPCs. The system (board) has a dynamic random-access low-



DRAM: Dynamic Random Access Memory, CPC: Control Processing Core, SRAM: Static Random Access Memory, CAM: Content Addressable Memory, TCAM: Ternary Content Addressable Memory, PPC: Packet Processing Core

Figure 1. Typical structure of network processor

speed memory (DRAM), which can hold hundreds or thousands of packets, and high-speed memories such as static RAM (SRAM), content-addressing memory (CAM), or ternary CAM (TCAM), which can hold a limited number of packets. The design shown in the figure is not the only possible one. For example, in other types of network processors, the architecture of the network that connects the PPCs and the CPC may be different from that shown in the figure.

A method for controlling packet processing in network processors by using PPCs (mainly by software), which is proposed in this paper, is compared with a conventional method using **Figure 2**. The same NP hardware as conventional methods is used in the proposed method. However, it can be used in a different way; that is, PPCs are used for both data processing and control processing. In a conventional control method (Figure 2(a)), the CPC controls the PPCs. However, by means of the proposed method (Figure 2(b)), one or more PPCs are used for control processing instead of the CPC. The control-processing PPC receives packets that contain control information. It receives the control packets as data-plane packets; that is, it receives them by the same method as that by which data-processing PPCs receive data packets. The control-processing PPC receives control packets from

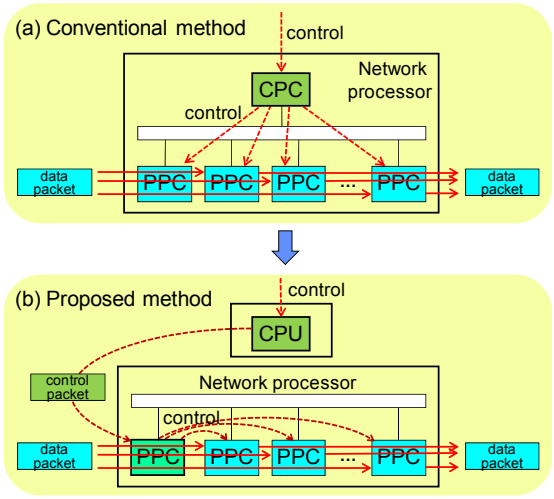


Figure 2. Conventional and proposed methods for controlling a network processor

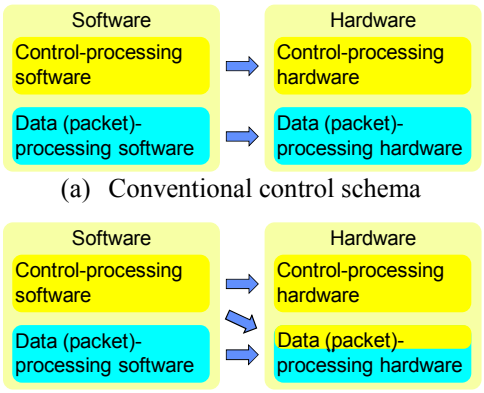


Figure 3. Comparison of conventional and proposed control schema

a CPU outside the network processor chip. Complex control messages received by the CPU are partially processed and divided into simplified control packets by the CPU, and they are sent to the control-processing PPC. No third type of methods is known to the authors.

Figure 3 diagrams the control schema of this method and compares it to the conventional schema. In the conventional schema, control-processing software works on control-processing hardware, and data (packet) processing software works on data processing hardware. However, the proposed method can change the border; that is, part of control processing works on data processing hardware. This method enables a more flexible task division and more effective hardware use.

B. Problems to be solved

The proposed method solves the problems between a CPC and PPCs, i.e., complexity of interaction, hardware- and vendor-dependence, non-portability, and non-applicability of standards. The first and most important problem is interaction, i.e., synchronization and communication, between the control-processing and data-processing tasks. A program on the controlling PPC can control data-processing PPCs using data-exchange mechanisms, which are uniform and simpler than those between a CPC and PPCs because the communicating cores are uniform. This issue will be explained more in Section II.C.

The second problem is control-message simplification. Because control messages to be received by the network processor are complex, it is difficult to handle messages by a bare-metal controlling PPC. The messages are thus divided into simpler control packets before sending them to the control-processing PPC. However, they can usually be translated into a collection of simpler messages by a CPU outside the network processor chip. This issue will be explained more in Section II.D.

The third problem is core allocation. The control task may be statically allocated to one or more PPCs or it may be dynamically allocated to one or more PPCs. This issue is briefly explained below.

In static allocation, instead of the CPC, one or more PPCs are used only for control processing, and other PPCs are used only for packet processing. The control-processing PPC receives packets that contain control information. Static allocation requires that the data-packet

scheduler, which is implemented by hardware in bare-metal processors, distinguishes data and control packets.

In the case of dynamic allocation, if the controlling task can be allocated to PPCs that may process data packets, the PPCs must distinguish data and control packets, usually by software, when they arrive. Dynamic allocation is focused on hereafter.

Both in static and dynamic allocations, control and data-processing loads on processor cores can be balanced. Although network processors usually have a load-balancing function for packet processing, they usually do not have one between data and control processing. The proposed method can provide this function. The ratio of data processing to control processing may be changed. If the load can be estimated statically, it can be balanced statically by static allocation. If it can be estimated dynamically, it can be balanced by dynamic allocation.

C. Control-processing and data-processing tasks and interaction between them

To solve the first problem, i.e., interaction complexity between control and data processing, in the proposed method, a program on the controlling PPC controls data-processing PPCs using data-exchange mechanisms, such as a shared memory or an on-chip network. These data-exchange mechanisms are more uniform and simpler than those between a CPC and PPCs.

In addition, in the method described here, PPCs are allocated dynamically; that is, each PPC is used for both control and data processing. A control-processing PPC controls data-processing PPCs, including itself.

The flow chart of the controlling and data-processing in each PPC is shown in **Figure 4**. When a packet arrives at the internal network interface, which is connected to the control CPU, the PPC first distinguishes control and data packets by testing the tag that the packet contains. If the packet is a data packet, it is processed by the packet-processing procedure for internal-to-external packets. If it is a control packet, it is processed by the control-processing procedure. Although this test causes an increase in processing time, it is negligibly small. When a packet arrives the external-network interface, which is connected to the external network (i.e., a network between network nodes), the packet is assumed to be a data packet (i.e., no control CPU exists in the external network) and processed by the packet-processing procedure for external-to-internal packets.

The control-processing and data-processing programs can be written following the same semantics, thereby making description of interactions easier. That is, they can be written using the same language or different languages can be used for writing these program modules.

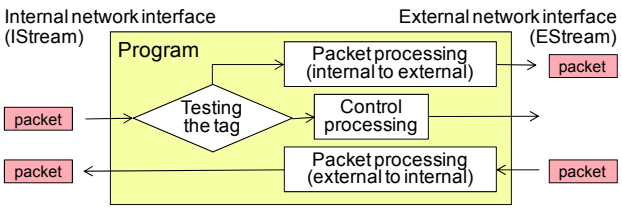


Figure 4. Outline of control- and data-processing by using PPCs

However, if the same language or languages with common semantics are used, the interactions between the control-processing and data-processing PPCs follow the same semantics. Description of synchronization and communication thus becomes easier. In addition, if the language is portable, both control-processing and data-processing programs can be ported to another network processor at once because they can be written in the same language and in the same program.

D. Division and simplification of control messages

To solve the second problem, i.e., that control messages are too complex to be process in PPCs, the messages are divided into simpler control packets by a CPU outside the network processor chip in the proposed method. The control messages may be written in a complex form such as SOAP [Mit 03] or XML-RPC [XML], and the messages may be iteratively or recursively structured. However, if the command language is properly defined, the complex messages can usually be translated to simpler messages. As for the proposed method, therefore, they are partially evaluated by the external CPU and divided into simplified control packets. The control packets do not have complex structures such as iteration or recursion, and they consist of fixed-length fields that can be processed by simple methods.

This message conversion conceptually consists of two functions, shown in **Figure 5**. The first function is message division, and the second function is translation into control packets.

The message-division function divides a complex message into a sequence of unit operations. The operations are still abstract, and the length of each operation may be variable. If these operations are used in a PPC, the PPC requires a parser that analyzes the operation, and the parsing requires computational resources; therefore, the second function is required. It translates each operation into fixed-format control packets. The control packets, which do not have complex structures such as iteration or recursion and consist of fixed-length fields that can be processed by simple methods, are sent to the control-processing PPC.

When a complex message is divided, a structured message is basically formed by using the following three types of structures:

- *Concatenation*: Multiple operations are executed sequentially or in parallel.
- *Selection*: An operation is selected from multiple operations by a conditional expression (a Boolean expression) and executed.
- *Repetition*: An operation is repeatedly executed while the value of a conditional expression is true.

To unroll a complex message, the conditional expression must be evaluated by the CPU. If it cannot always be evaluated before the operations are executed in the PPC, the message cannot be divided.

III. APPLICATION

The proposed control method can be applied to a virtual-link management function. This function is implemented in a virtualization node (VNode) [Nak 12][Kan 12] for

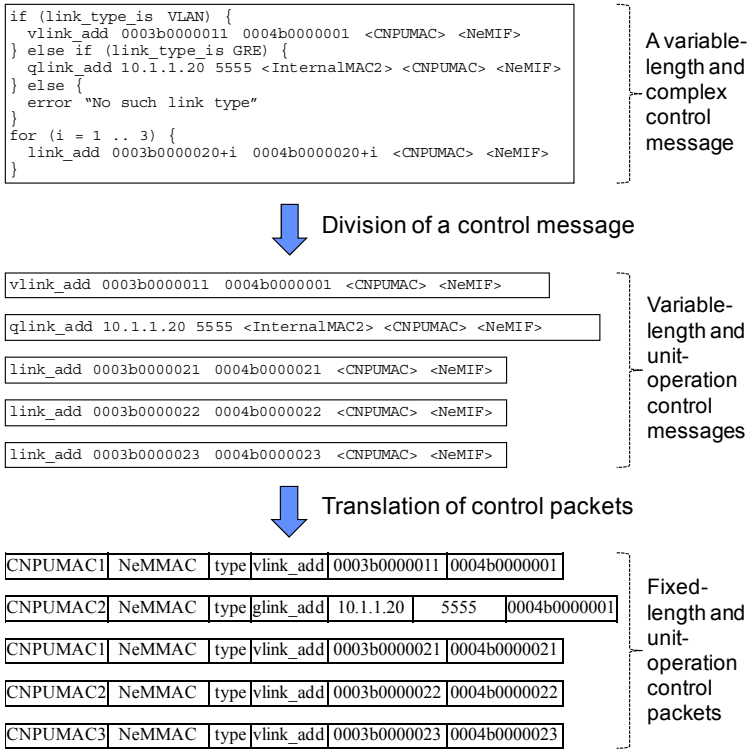


Figure 5. Division and simplification of a control message

network-virtualization platforms.

A. Introduction to VNode platform

Nakao et al. [Nak 10][Nak 12] have developed a VNode architecture and VNode platform, which consists of VNodes and management servers, and a high-performance fully functional virtualization testbed. The goal of these works is to develop an environment in which multiple slices (virtual networks) with independently and arbitrarily designed and programmed new-generation-network functions run concurrently, but are logically isolated, on a physical network.

The VNode can be extended to add a new type of virtual link, i.e., a "VLAN link sliver," to a slice. The original VNode only had generic routing encapsulation (GRE) [Far 00]-based virtual links (called "GRE link slivers"). A VLAN-based virtual link can be added by using the proposed control method.

B. Virtual-link management for a VNode platform

On a VNode platform, a virtual link between VNodes is created by negotiating link parameters, i.e., a link identifier, such as GRE key or VLAN identifier (ID), and end-point addresses, such as IP addresses or MAC addresses (see **Figure 6**). A substrate link such as a GRE tunnel or a VLAN path is configured using these parameters.

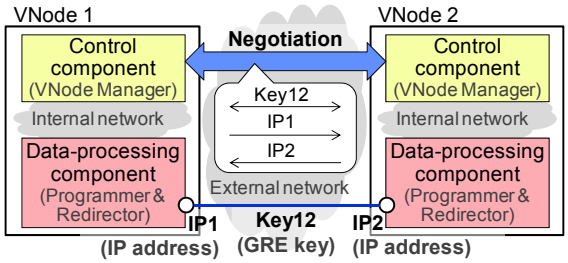


Figure 6. Negotiation for creating a virtual link

To satisfy the "clean virtualization" criteria [Kan 12], which include requirements for avoiding interference between the external networks between VNodes and the internal network of a VNode, the protocols and addressees of these networks are independently defined. A VLAN is therefore used for the internal network in the current version of a VNode, but outside the VNode, a VLAN or any other type of network can be used. Currently, a VNode platform supports only GRE-based representation in the external network.

This network-separation design requires address and property translation on the border of a VNode (see **Figure 7**) [Kan 12]. If the external representation is GRE/IP and the internal representation is VLAN, the IP addresses in the incoming packets is converted to corresponding MAC addresses, and the link properties (i.e., GRE keys) in the incoming packets must be converted to the corresponding properties (i.e., VLAN IDs) if necessary. The reverse conversion is required for outgoing packets. Even if both the external and the internal representations are VLANs, this translation is still required for the separation; the MAC addresses and the VLAN ID in packets must be translated.

C. Virtual-link creation using plug-ins

In addition to built-in GRE link slivers, a new type of virtual link can be added to VNodes by using the proposed control method. For example, a type of VLAN-based virtual link can be implemented. As shown in Figure 7, a network processor (in the networking component called a "redirector") and a control CPU (in the controlling components) are plugged into each VNode by using the plug-in architecture for a VNode [Kan 13b]. The control CPU translates a link addition, a link deletion, or another type of message into a control packet and sends it to a control PPC in the network processor.

When a virtual link is created, the creation message contains the following three virtual-link parameters.

- *Internal address* is the internal MAC address of the local end-point of the virtual link. It is used for the destination MAC address used for converting and

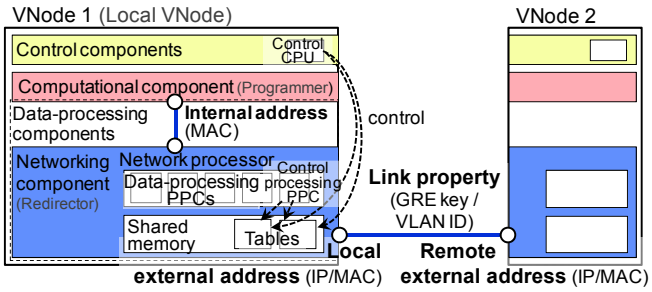


Figure 7. External/internal addresses and translation mechanism for data packets

sending incoming packets by the network processor. It is also used for the source MAC address when outgoing packets are received and converted.

- *Local external address* is the external MAC address of the local end-point of the virtual link. It is used for the destination MAC address when a packet is sent from the remote VNode. It is also used for the source MAC address when a packet is sent from the local VNode.
- *Remote external address* is the external MAC address of the remote end-point of the virtual link. It is used for the destination MAC address when a packet is sent from the local VNode. It is also used for the source MAC address when a packet is sent from the remote VNode.

If multiple VLAN IDs are used for virtual links, there must be a fourth parameter: VLAN ID (a link property).

These parameters are stored in two tables in a memory block shared among the PPCs (see Figure 7). Two tables are required because there are two types of keys: the local (or remote) external MAC address for incoming packets and the internal MAC address for outgoing packets.

The data-processing PPC replaces the MAC addresses. When the PPC receives an external packet, it matches the destination (or source) MAC addresses with the external MAC addresses stored in the first table. If entries that match the MAC addresses are found, the destination MAC address is replaced by the corresponding internal MAC addresses and processed in the VNode. (A constant source-address is used in the current implementation.) If no entries match, the packet is routed to another node on the platform. When the PPC receives an internal packet, it matches the source and destination MAC addresses with the internal MAC addresses stored in the second table. If entries that match the MAC addresses are found, the MAC addresses are replaced by the corresponding external MAC addresses and sent to the external network, i.e., another node on the platform.

A virtual link can be deleted by another type of message, i.e., a deletion message. When the control PPC receives a deletion message, it removes an entry from each table. Because this sequence is similar to a creation sequence, the detail is omitted here. Although the control plug-in should support other types of virtual-link control messages, they are out of scope of this paper.

IV. Implementation and Evaluation

A type of VLAN-based virtual link was implemented by using the proposed control method. Two sets of VNodes are used for this implementation. A control CPU and a network processor are plugged into each VNode by using the plug-in interface [Kan 13b]. A PC with a control CPU and a Cavium Octeon® network processor [Cav 10] board, called WANic-56512 (developed by General Electric Company), is used. The data (processing) plug-in was developed by using an open, hardware-independent, and high-level portable programming language for network processors, called "Phonepl" (which means "portable high-level and high-performance open network processing language" and was previously called CSP) [Kan 13a]. The control CPU receives input messages through a command-line interface (CLI). The syntax of a creation message is given as follows.

```
000 import IStream; // Internal stream
001 import EStream; // External stream

002 class ControlAndDataProcessing {
     ...
003   public ControlAndDataProcessing(
                NetStream iport > itoe,
004             NetStream eport > etoi) {
005     // Initialization
006   }

007   void processControl(Packet i) {
                                  // Process a control packet
008     // Control-packet processing
009   }

010   void itoe(Packet i) { // Process an i-to-e data packet
011     int tag = i....;
012     if (tag == ControlPacketTagValue) {
013       processControl(i);
014     } else {
015       // Data-packet processing (internal to external)
016     }
017   }

018   void etoi(Packet i) { // Process an e-to-i data packet
019     // Data-packet processing (external to internal)
020   }

021   void main() {
022     new ControlAndDataProcessing(new IStream(),
                                     new EStream());
023   }
024 }
```

Figure 8. Phonepl program structure for control- and data-processing by using PPCs

`link add` LMAC RMAC IMAC Additional_Parameters

LMAC is the local external MAC address, RMAC is the remote external MAC address, and IMAC is the internal MAC address. Several additional parameters are required for minor controls. The syntax of a deletion message is similar to that of a creation message.

The control CPU translates messages to a control packet (without message division). A bare Ethernet packet is used for control packets because a complex packet-processing library, such as IP, UDP, or TCP, is not available on the network processor. Because no specific mechanism for improving reliability is used, the same control packets are sent repeatedly.

Both control-processing and data-processing tasks processed by PPCs are described by Phonepl, and the communication between them is programmed following normal and uniform shared-memory semantics. This makes programming the control-processing task and porting a program much easier.

The outline of a program for control processing and data processing by using PPCs is described in **Figure 8**. This program defines class **`ControlAndDataProcessing`**. It contains two bidirectional packet streams: **`IStream`** and **`EStream`** (lines 001-002). **`IStream`** is a stream for the VNode-internal network, and **`EStream`** is a stream for the external network. When **`ControlAndDataProcessing`** is initialized, function **`main()`** (line 021) is executed. This function generates an instance (a singleton) of **`ControlAndDataProcessing`**. Two packet streams are generated and passed as arguments of **`ControlAndDataProcessing`**. They are assigned to physical interfaces outside of this program.

In the constructor of **`ControlAndDataProcessing`** (lines 003-006), the parameter declarations specify that input packets to the first parameter, **`iport`**, are sent to

Table 1. Comparison of packet-header handling implementations

	Data (packet) processing		Control processing		Interface (memory set-up) between D/C	
	Program Length	Description Language	Program Length	Description Language	Program Length	Description Language
Control by PPC (proposed method)	26	Phonepl	21	Phonepl	30	Phonepl
			230	C (Linux)		
Conventional method	160	C (bare metal)	200	C (Linux)	80	C (bare metal)

method **itoe**, and input packets to the second parameter, **eport**, are sent to method **etoi**. These methods receive one packet at a time and process packets in parallel using multiple PPCs (multiple threads). Method **itoe** (line 010) handles an outgoing data/control packet that comes from **IStream**. If the packet is a control packet, it is processed by another method called **processControl** (line 007) by using the same PPC as **itoe** uses. Method **etoi** (line 018) handles a packet that comes from **EStream**.

The control packets are processed by a PPC in the Octeon, and the parameters stored in control packets are stored in shared variables to control data-processing PPCs. Shared variables are supported by Phonepl and implemented using a shared memory block that is supported by the software of Octeon. The programmer, therefore, does not need to pay special attention to communication between the control-processing and data-processing PPCs. In this program, these PPCs do not need to be synchronized.

Table 1 compares the implementations of the proposed method and a conventional method; it compares the program lengths (in lines) used for the virtual-link implementation (including a program for the control CPU) and an estimated set of programs described by Octeon C program. Because no complete implementation for the latter exists, an estimated set is used. Although the total number of lines for the control processing is longer for the proposed method, the programming is easier because it uses normal packet-processing mechanism instead of using proprietary hardware and software.

Performance of both control processing and data processing should be evaluated. However, a unit operation, i.e., a link creation or deletion, is estimated to be 3 μs or less. It is thus difficult to be measured. In contrast, data processing can be measured. Successful IP communication between the virtual nodes connected by the VLAN virtual-link was confirmed by a **ping** command; although virtual links in VNodes can transmit arbitrary format packets, IP was used because it requires only two commands (i.e., **ifconfig** and **ping**) built into the virtual node. The performance of the whole prototype implementation was not measured, but the throughput of the data plug-in was measured to be 9 Gbps or more when the packet size was 900 bytes or larger.

V. Concluding Remarks

A method for controlling packet processing in network processors by using packet-processing cores (PPCs) is proposed. By means of this method, complex control messages are divided into simplified control packets by a CPU outside the network processor chip, and the control packets are sent to a control-processing PPC. The control-processing PPC controls data-processing PPCs by using interaction mechanisms, such as a shared memory or an on-chip network, which are more uniform and simpler than similar mechanisms between a CPC and PPCs.

This method was applied to a virtual-link control-processing task and packet-processing tasks in a network node with a virtualization function. Both tasks are described by a hardware-independent high-level language called "Phonepl," which was implemented for Octeon, and the communication between them is programmed following normal and uniform shared-memory semantics. As a result, programming the control-processing and high-performance data-processing tasks and program porting between different types of network processors become much easier.

Future work includes application of the proposed method to other types of network processors.

Acknowledgments

The author thanks Michitaka Okuno from Hitachi for his useful comments on the method for controlling network processors by using PPCs. The author also thanks Yasushi Kasugai, Kei Shiraishi, Takanori Ariyoshi, and Takeshi Ishikura from Hitachi for implementing the plug-in interfaces in the redirector. Part of the research results described in this paper is an outcome of the Advanced Network Virtualization Platform (Project A) funded by the National Institute of Information and Communications Technology (NICT).

References

[Cav 10] "OCTEON Programmer's Guide, The Fundamentals", Cavium Networks, 2010, http://university.caviumnetworks.com/downloads/-Mini_version_of_Prog_Guide_EDU_July_2010.pdf

[Far 00] Farinacci, D., Li, T., Hanks, S., Meyer, D., and Traina, P., "Generic Routing Encapsulation (GRE)", RFC 2784, IETF, March 2000.

[Gog 03] Goglin, S. D., Hooper, D., Kumar, A, and Yavatkar, R., "Advanced Software Framework, Tools, and Languages for the IXP Family", *Intel Technology Journal*, Vol. 7, No. 4, pp. 64–76, 2003.

[Kan 12] Kanada, Y., Shiraishi, K., and Nakao, A., "Network-Virtualization Nodes that Support Mutually Independent Development and Evolution of Components", *IEEE International Conference on Communication Systems* (*ICCS 2012*), November 2012.

[Kan 13a] Kanada, Y., "Open, High-level, and Portable Programming Environment for Network Processors", *IEICE 7th Meeting of Network Virtualization SIG*, July 2013 (in Japanese).

[Kan 13b] Kanada, Y., "A Node Plug-in Architecture for Evolving Network Virtualization Nodes", *IEEE Workshop on Software Defined Networks for Future Networks and Services* (*SDN4FNS*), November 2013.

[Mit 03] Mitra, N., and Lafon, Y., "SOAP version 1.2 part 0: Primer", W3C Recommendation 24 (2003): 12.

[Nak 10] Nakao, A., "Virtual Node Project — Virtualization Technology for Building New-Generation Networks", *NICT News*, No. 393, pp. 1–6, Jun 2010.

[Nak 12] Nakao, A., "VNode: A Deeply Programmable Network Testbed Through Network Virtualization", *3rd IEICE Technical Committee on Network Virtualization*, March 2012, http://www.ieice.org/~nv/05-nv20120302-nakao.pdf

[XML] XML-RPC Home Page, http://www.xmlrpc.com/