Controling Network Processors by using Packet-processing Cores

Yasusi Kanada
Hitachi, Ltd., Japan
Introduction

- Network processors (NPs) are used for customizable and high-performance networking.

- NPs usually contain two different types of cores.
  - Packet processing core (PPC)
  - Control processing core (CPC)

- Problems of NP programming
  - Synchronization and communication of PPCs and CPC
  - Hardware- and vendor-dependence of NP software
  - Lack of portability of NP software

- A method for solving these problems is proposed in this study.
There are various (proprietary) NP architectures.
- NDA is required to develop NP programs.

They may be summarized to ...

**NP Architecture**

- **Memory controller**
  - DRAM

- **Internal bus/network-on-chip**
  - Shared SRAM / CAM / TCAM
  - CPC
    - L1 Cache
    - L2 Cache
  - PPC
    - SRAM / CAM / TCAM
  - PPC
    - SRAM / CAM / TCAM
  - PPC
    - SRAM / CAM / TCAM

- **NPU**
  - Input processor
  - Output processor

- **External network**
Proposal: to control PPCs by a PPC

► PPCs are conventionally controlled by a CPC.
  ■ This control method causes complexity and the problems because of proprietary hardware and software between CPC and PPCs.
  ■ The complexity comes from the architectural differences between CPC and PPCs.
    • E.g., CPC has virtual memory, but PPCs does not.
    • E.g., CPC runs OS, but PPCs are bare-bone (i.e., OS-less).

► To simplify the control, a method for controlling PPCs by using a PPC is proposed.
Comparison of Conventional and Proposed Methods

Conventional methods

- Control (by a proprietary method) flows through the CPC and network processor.
- Data packets flow through PPCs.

Proposed method

- Control (by a widely-used method) is translated by a CPU.
- Data packets and control packets flow through PPCs.

Network processor
How to Solve Problems by Proposed Method

► 1. Communication and synchronization in PPC
► 2. Control message simplification in CPU
► 3. Core allocation of PPCs
Uniform and simpler communication and synchronization (C&S) hardware can be used.
- C&S hardware between PPCs, such as shared memory, are simpler than that between PPC and CPC.

C&S can be programmed in a simpler and hardware- and vendor-independent method.
- A high-level language “Phonepl” is being developed for this purpose.
**Issue 2: Control Message Simplification in CPU**

```c
if (link_type_is VLAN) {
    vlink_add 0003b0000011 0004b0000001 <CNPUMAC> <NeMIF>
} else if (link_type_is GRE) {
    qlink_add 10.1.1.20 5555 <InternalMAC2> <CNPUMAC> <NeMIF>
} else {
    error "No such link type"
}
for (i = 1 .. 3) {
    link_add 0003b0000020+i 0004b00000020+i <CNPUMAC> <NeMIF>
}
```

### Division of a control message

- `vlink_add 0003b0000011 0004b0000001 <CNPUMAC> <NeMIF>`
- `qlink_add 10.1.1.20 5555 <InternalMAC2> <CNPUMAC> <NeMIF>`
- `link_add 0003b0000020 0004b0000020 <CNPUMAC> <NeMIF>`
- `link_add 0003b0000021 0004b0000021 <CNPUMAC> <NeMIF>`
- `link_add 0003b0000022 0004b0000022 <CNPUMAC> <NeMIF>`
- `link_add 0003b0000023 0004b0000023 <CNPUMAC> <NeMIF>`

### Translation into control packets

- **CNPUMAC1**: NeMMAC type `vlink_add 0003b0000011 0004b0000001`
- **CNPUMAC2**: NeMMAC type `qlink_add 10.1.1.20 5555`
- **CNPUMAC3**: NeMMAC type `link_add 0003b0000020 0004b0000020`
- **CNPUMAC4**: NeMMAC type `link_add 0003b0000021 0004b0000021`
- **CNPUMAC5**: NeMMAC type `link_add 0003b0000022 0004b0000022`
- **CNPUMAC6**: NeMMAC type `link_add 0003b0000023 0004b0000023`
Issue 3: Core Allocation of PPCs

- Cores may be allocated statically or dynamically.
- Proposed method is advantageous in both.
- In static allocation, load-balancing is enabled.
  - **Conventional method**: no load balancing
    - Control tasks: \(c_1, c_2, \ldots, c_n\)
    - Data processing tasks: \(t_1, t_2, \ldots, t_m\)
  - **Proposed method**: static load balancing
    - Control tasks: \(c_1, c_2, \ldots, c_n\)
    - Data processing tasks: \(t_1, t_2, \ldots, t_m\)
- Dynamic allocation is enabled.
  - **Proposed method**: dynamic load balancing
    - Control tasks: \(c_1, c_2, \ldots, c_n\)
    - Data processing tasks: \(t_1, t_2, \ldots, t_m\)
Application: Creating a New Type of Virtual Links

A network node with network virtualization function, which is called VNode, has been developed.

NPs are used for adding new functions to a VNode.

By using NP-based plug-ins and the proposed method, a new type of virtual link is created and managed.

- Built-in virtual-link creation/management mechanism is extended.

VNode 1 (Local VNode)

VNode 2
Implementation and Evaluation

► Implementation
- Cavium Octeon NPs were used for data processing (packet header conversions).
- Data and control processing tasks were programmed for the PPCs by Phonepl (a high-level language).

► Comparison of proposed and conventional methods

<table>
<thead>
<tr>
<th></th>
<th>Data (packet) processing</th>
<th>Control processing</th>
<th>Interface (memory set-up) between D/C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Program</td>
<td>Description Language</td>
<td>Program</td>
</tr>
<tr>
<td>Control by PPC (proposed method)</td>
<td>26</td>
<td>Phonepl</td>
<td>21</td>
</tr>
<tr>
<td>Conventional method</td>
<td>160</td>
<td>C (bare metal)</td>
<td>200</td>
</tr>
</tbody>
</table>
Conclusion

► A method for controlling packet processing in NPs by using PPCs was proposed.

► This method makes
  ■ synchronization and communication tasks and programming control/data-processing tasks easier and hardware/vendor-independent.
  ■ porting between different types of NPs much easier.

► Future work includes application of the proposed method to other types of NPs.
### Appendix: Comparison

#### Conventional control schema

<table>
<thead>
<tr>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control-processing software</td>
<td>Control-processing hardware</td>
</tr>
<tr>
<td>Data (packet)-processing software</td>
<td>Data (packet)-processing hardware</td>
</tr>
</tbody>
</table>

#### Proposed control schema

<table>
<thead>
<tr>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control-processing software</td>
<td>Control-processing hardware</td>
</tr>
<tr>
<td>Data (packet)-processing software</td>
<td>Data (packet)-processing hardware</td>
</tr>
</tbody>
</table>
PPCs are dynamically allocated.

Data/control packets are processed in the following.

- Program for PPC
- Testing tag
- Control processing
- Packet processing (internal to external)
- Packet processing (external to internal)
- Internal network interface
- External network interface
“Phonepl!” language is used for high-level NP programming.

Packet and control processing are not separated, but they can be separated.

```
000 import IStream;  // Internal stream
001 import EStream;  // External stream
002 class ControlAndDataProcessing {
...
003  public ControlAndDataProcessing(
            NetStream import > itoe,
            NetStream export > etoi) {
004      // Initialization
005  }
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 }
```