High-level Portable Programming Language for Optimized Memory Use of Network Processors

Yasusi Kanada
Hitachi, Ltd., Central Research Laboratory
1. Introduction

Network-processor (NP) programming is important for high-performance networking and in-network processing.
- NPs enable quick development of arbitrary protocols and functions.

Problems of NP programming:
- **No portability** because of hardware/vendor-dependent proprietary tools are required.
- **Difficulty in development** because of required specialized skill and knowledge (which are not widely available).

Focus: Programmers must distinguish SRAM and DRAM to achieve 10-100 Gbps wire-rate packet processing.
- Whole packet must be stored in DRAM.
- Operations on data in DRAM may disable wire-rate processing.
- NPs might not have cache, or cache mishit may cause serious problems in NPs.

Means for solving these problems:
- **Phonepl**: an open, high-level, and portable programming language.
- **Four packet-data representations**, which programmers can be mostly unaware of.
2. Phonepl language design

► Syntax and semantics are close to Java.
  • Java/C++ programmers can use Phonepl easily.

► Packets are immutable byte-strings.
  • Basic operations on packets are substring, concatenation, etc.
  • Non-IP protocols can be programmed easily.
  • Immutability enables data sharing among (input/output) packets.

► Byte strings and packets are different types of objects.
  • They are handled by quite different methods.

► Assumptions on packet and string data
  • Whole packet is stored only in DRAM, but the header is in SRAM (cached).
  • Whole byte string is cached (in SRAM and maybe stored in DRAM).

► Packet and byte string operations
  • substring and subpacket operations
    • subpacket generates a packet from part of a packet.
    • substring generates a byte string from part of a packet or string.
  • Concatenation operations
    • concat generates a byte string from byte strings.
    • “new Packet” generates a packet from byte strings and a packet.
    • No methods for concatenating two or more packets.
3. Phonepl program example

Simple MAC header addition/removal

```java
001 import NetStream1;
002 import NetStream2;
003 class AddRemMAC {
004     NetStream out1;
005     NetStream out2;
006     public AddRemMAC(NetStream port1 > process1,
007                NetStream port2 > process2) {
008         out1 = port1;
009         out2 = port2;
010     }
011     void process1(Packet i) { // Port 1 to 2 (no VLAN -> no VLAN)
012         Packet o = new Packet(i.substring(0,14),i); // MAC header of original packet
013         out2.put(o);
014     }
015     void process2(Packet i) { // Port 2 to 1 (no VLAN -> no VLAN)
016         Packet o = i.subpacket(14); // remove MAC header (no VLAN)
017         out1.put(o);
018     }
019     void main() {
020         new AddRemMAC(new NetStream1(), new NetStream2());
021     }
022 }
```

Object that process packets (singleton) is created.
4. Implementation method

A Phonepl implementation selects four packet representations statically or dynamically.

Four representations of packet type

- **Cached**: whole packet is in SRAM.
  - No copy is assumed to be in DRAM.
- **Mixed**: whole packet is in DRAM and packet header is in SRAM.
  - Header size is variable.
- **Gathered**: packet consists of multiple fragments.
  - Represented by an array or list of fragments.
- **Uncached**: whole packet is in DRAM
  - No copy is assumed to be in SRAM.

Outline of packet data structure:

Four representations
5. Prototyping and Evaluation

► Prototype
  • Phonepl was implemented for Cavium Octeon® NP.

► Evaluation using two Phonepl programs
  • Prototype was evaluated by using a program for MAC header addition/deletion and a pass-through program.
  • An Octeon board with these programs was connected to each node in VNode Infrastructure (a network-virtualization infrastructure).
    • NP Board: GE WANic 56512

► Evaluation result
  • The throughput was over 7.5 Gbps (close to 10-Gbps wire-rate).
6. Conclusion

To make NP programming easier, Phonepl language and a method for implementing Phonepl are proposed.
- Programmers can use SRAM and DRAM appropriately without distinguishing them.
- Four representations of packet type and usage of them were proposed.

The evaluation result shows Phonepl for Octeon NP enables high throughput (close to 10-Gbps wire-rate) in simple packet processing.
Appendix: Detailed packet data structures

(1) Cached packet

(2) Mixed packet

(3) Gathered packet

(4) Uncached packet
Appendix: Detailed usage of four representations

► Mixed representation is required.
  • In packet processing, only packet headers are usually processed.
  • In such cases, it is better to store only packet headers to SRAM and tails to DRAM.

► Gathered representation is required.
  • This representation is useful when generating a packet by concatenating multiple data in DRAM or SRAM.
  • If trying to collect whole data into contiguous area in DRAM, DRAM to DRAM copy, which requires much time, is required.

► State transition between four representations