Packet Manipulator Processor: A RISC-V VLIW core for networking applications

Salvatore Pontarelli, Marco Bonola, Marco Spaziani Brunella, Giuseppe Bianchi

Speaker: Salvatore Pontarelli
Introduction

Network softwarization is seen as the optimal solution to design next generation network infrastructures, services and applications:

- protocol independence,
- short deployment time,
- easiness in upgrading network devices,
- no need for legacy support,
- smart forwarding,
- efficient energy consumption

[and much more...]
How we can do it at high-speed?

→ x86 servers: (dual xeon workstation) + Intel DPDK + fd.io[1] < 100 Gbps

→ network processors: hard to design/program, complex parallel/pipelined model [2] 100-200 Gbps

→ FPGA: HDL design, slow development/debug time, hard to upgrade 100-200 Gbps

→ programmable forwarding dataplanes: ad hoc programmable parsing, packet classification (header matching)[3][4]. No/small programmability on packet «manipulation»: encapsulate/decapsulate, checksum computation, header update > 5 Tbps

[1] https://fd.io/
programmable forwarding dataplanes

Multiple $\mu$-instr = multiple stages... (requirement: 1 clock per $\mu$-instr!)
- upper bound on #no $\mu$-instr ? (32 stages)
- is mix of match and $\mu$-instr what we really need?
programmable forwarding dataplanes + PMP
programmable forwarding dataplanes + PMP
Action Programmability

• Design a small, efficient CPU for packet manipulation.
  • Differs from NPU since still use the forwarding dataplanes for protocol parsing, packet classification

• Postpone to the output queues (or to the switch output port) the action to apply to the packet. → deploy the PMP at the output port.

• Requirement: process packets at 10/40 Gbps.
Possible programmable actions

• Inband packet reply
  • Generate a packet from scratch / from a template?

• Custom tunneling
  • «program» your encapsulation mechanism?
    • Opposed to selecting available tunneling mechanisms

• NAT/PNAT
  - header update/modification
PMP characteristics

• Instruction memory: execute small routines (<8Kinstr.)
• Data memory size:
  • store few packets (8KB)
  • store small variables (counters, constant, packet templates, etc.)
• Data memory:
  • unaligned, wide data bus(currently std 32 bits. → 64/128 bits)
  • dual/multiple port RAM to increase throughput → 1 port from fwd dataplane
• Flat memory, no cache hierarchy (scratchpad?)
• Limited number of instructions: (e.g. no multipliers)
• New instructions can increase throughput
PMP throughput

- 10 Gpbs $\rightarrow$ 14.88 Mpps $\rightarrow$ 67 cc @ 1GHz
- Solution 1:
  - Multi core CPU
    - simplified parallel processing.
    - reordering issues
- Solution 2:
  - VLIW CPU
    - well-known issues: data dependency, complex compilers, etc. Limited effect on PMP: small routine footprint, multiple memory accesses
PMP implementation

- Static 8-issue VLIW RISC-V core with RV32I instruction set
- 32bit dataplane (to be upgraded)
- 32x32bit general purpose registers
- VHDL described
- Branch prediction
- Lane forwarding
- Unaligned output memory handling
- Synthetized FPGA prototype
- Standard AXI4 wrapper for integration with Xilinx IP cores
- Implemented in a NetFPGA based programmable dataplane
Very small HW footprint on a NetFPGA SUME, equipped with a Virtex 7 FPGA

Fully synthesized at 250MHz
Acknowledgement

This work has been partly funded by the EU commission in the context of the 5G-PICTURE project, Grant Agreement 762057
Q&A