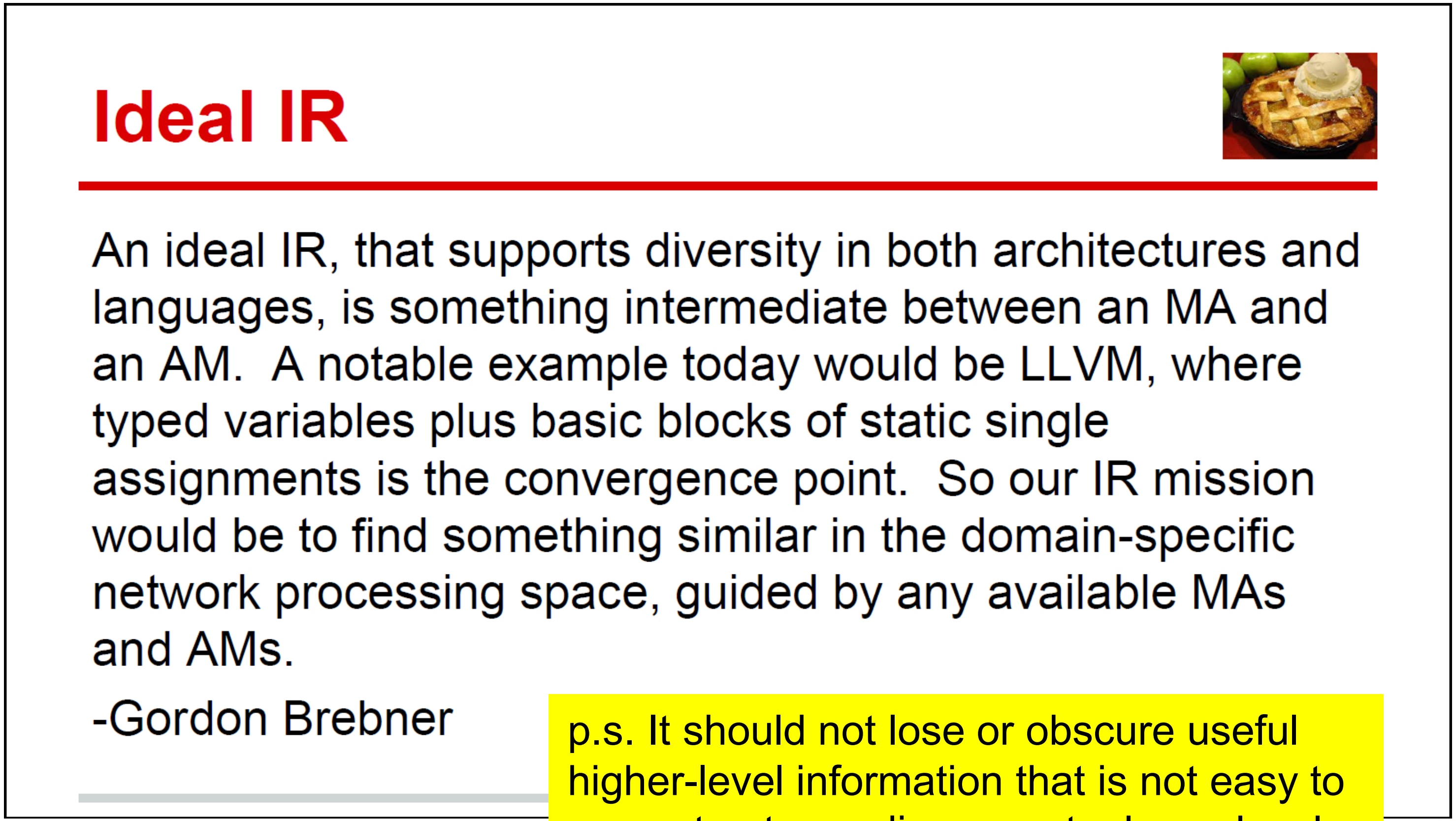
IR for PIF: Towards a useful intermediary

Gordon Brebner

Xilinx

October 30, 2014

Beyond Motherhood and Apple Pie



Ideal IR

An ideal IR, that supports diversity in both architectures and languages, is something intermediate between an MA and an AM. A notable example today would be LLVM, where typed variables plus basic blocks of static single assignments is the convergence point. So our IR mission would be to find something similar in the domain-specific network processing space, guided by any available MAs and AMs.

-Gordon Brebner

p.s. It should not lose or obscure useful higher-level information that is not easy to reconstruct or rediscover at a lower level

Source:
PIF Update
Dan Talyco
October 9, 2014

Glossary:

MA =
Machine Abstraction
(of hardware)

AM =
Abstract Machine
(for software)

- This presentation: coming from the MA direction

Packet Processing Technologies

- Numerous classifications, for example:

CAB (used for classifying OF extensions):

- Existing ASIC/ASSP
- Future ASIC/ASSP or current NPU
- Programmable pipeline NPU
- Software only

CAB (used for general perspective):

- Legacy ASIC/ASSP
- Pipelined NPU
- Multicore NPU

Yatish Kumar draft collection:

- Dataflow
- Highly-pipelined NPU
- Run-to-completion NPU
- Systolic array (e.g. RMT)
- Distributed switch

Pongrácz *et al* (HotSDN13):

- CPU
- NPU
- Programmable pipeline
- Switch chip

- How does one abstract from all that?

Underlying Hardware 1: Architectures and Programmability

- Control flow centric stereotype:
  - Random packet access
  - Run to completion
  - Generalized execution units and datapaths
  - High programmability through instructions
- Data flow centric stereotype:
  - Linear packet access
  - Pipelined
  - Specialized function units and datapaths
  - Varied programmability through configurations
- … and hybrids of the above

Question:
Is there an IR that can be mapped efficiently to both models, without an excessive number of optional features?

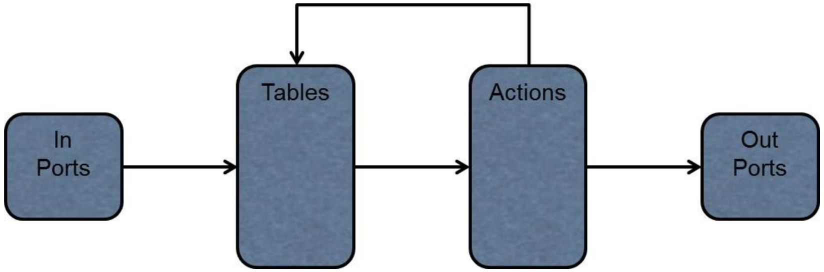
Underlying Hardware 2:
Chips, Boards, Racks, and Software

- The “switching chip” is just a virtual concept (= the abstract forwarding model)
- Hardware:
  - Chips for switching, packet processing, lookups, storage
  - Boards with multiple chips
  - Racks with line cards and switch fabrics
- Software:
  - Software-only switching or packet processing
  - Software with hardware acceleration
  - Firmware for hardware configuration

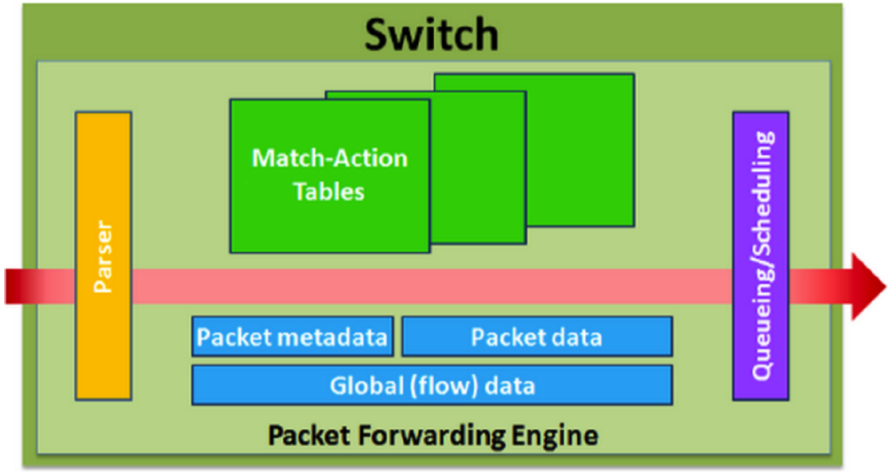
Question:
Is there an IR that can be mapped efficiently to all targets, without an excessive number of optional features?

Abstract Forwarding Model: Continuum of Granularity

Haoyu Song proposal, October 2014



OF-PI strawman proposal, July 2014



- Finer-grain end:
  - Functions programmed using (e.g. POF-FIS) actions

Question:
Is packet processing better represented by simple instructions operating on packets, or by larger components acting on packets?

- Coarser-grain end:
  - Built-in functions programmed with custom configuration

Match-Action Table (MAT)
Key Characteristic of OpenFlow

- Anchor point for any IR
- Provides (ideally) constant-time key-value mapping
- Input: packet data and context data - *how provided?*
- Output: actions on packet and context - *how executed?*

- Doesn’t force either control flow or data flow model
- Doesn’t force any particular MAT hardware or software
- Doesn’t force fine-grain or coarse-grain actions
- So it’s a good part of the abstract model
- … and its programming model is well-understood

“Action Execution Block” (AEB)

- Proposed second key component of any IR
- Updates packet and context
- Input: packet and context, actions
- Output: packet(s) and context, match data

- Modular approach, with different AEB constructions:
  - Directly programmed engine using primitive instructions (e.g. POF-FIS)
  - Well-known standard function (e.g. parser, traffic manager) configured using custom programming information
  - Built hierarchically from inter-connected AEBs and MATs with packet/context data flow and action control flow
  - Open-ended white box (or black box?) components (e.g. Ben M-C AFs)

Relationship to AIR

- Processor
  - AEB
- Processor layout
  - Top-level AEB built out of (pipeline of) smaller AEBs
- Parser
  - Example of a standard AEB, with custom programming via parse tree
- Ingress or egress pipeline
  - Example of mid-level AEB built out of pipeline of smaller AEBs and MATs

Comments

- Summary:
  1. MAT = flow-programmable brain
  2. AEB = provider-programmable organ or limb
- Re. Slide 4: Architectures and Programmability
  - Packet data flow between AEBs: OF tradition, and hardware friendly
  - Modular approach: AEB accommodates different in-AEB processing styles
  - Programmability of AEBs, and dynamic issue of actions to AEBs
- Re. Slide 5: Chips, Boards, Racks, and Software
  - Hierarchy of AEBs: friendly to distributed hardware components
  - Packet/context/action flow between AEBs: also friendly to distributed hardware
  - Modular approach: accommodates separate hardware and software components
- Re. Slide 6: Continuum of Granularity
  - Allows some AEBs to be fine-grain – but doesn't insist on it
  - Allows coarse-grain AEBs for some well-known functions – hardware friendly
  - Allows for different granularities for actions

“Open Hardware” Prototyping: Sub-Project Proposal

- Current software commitment from Gordon:
  - Once IR proposal is stable enough …
  - … write back-end compiler from IR to FPGA (with SDNet soft architecture)
  - One of a few prototypes (others e.g. to software switch, multicore NPU, …)
- Possible extension to include “open hardware” sub-project:
  - FPGA+SDNet allows creation of packet processor architecture instances
  - Enables simultaneous architecture adaptation to meet the IR, alongside the backend compiler algorithms to map the IR
  - Validate hardware friendliness of IR proposal by checking IR against hardware technologies, for example:
    - Native FPGA
    - Future ASIC/ASSP
    - Pipelined NPU
- Any volunteers to help?