DSA for Networking
Offload Infrastructure in Linux

Domain specific accelerators (DSA) are specialized hardware components that accelerate certain workloads within a specific domain or application

General motivation and benefits:

https://cacm.acm.org/research/domain-specific-hardware-accelerators

Scope of DSA in networking domain:

Manifestations and use cases, system design and APIs,
SW/HW integration, (logical) HW interfaces, programming DSA
Mundane (csum offload) to the ornate (Falcon)
Close (1m for AI/ML rack) to far ($2.25X10^8$ km to Mars)
Goals: Performance, power, cost, or all of the abve

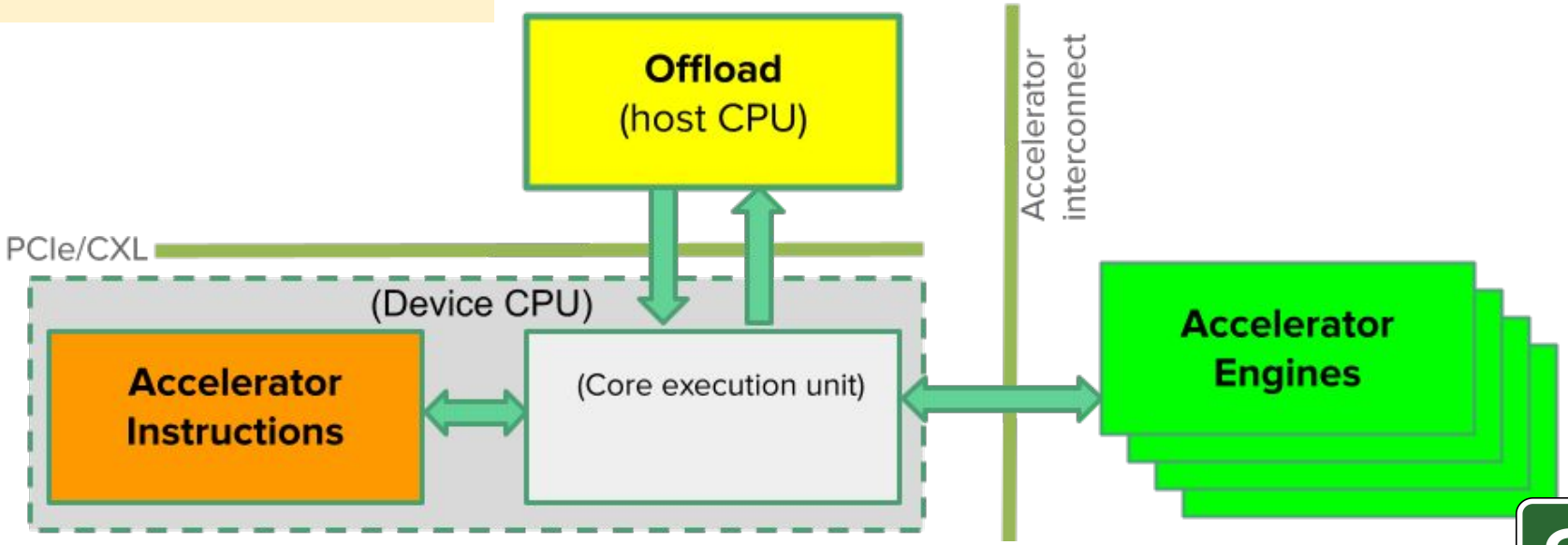
What makes networking different?

High speed Ethernet is the only asynchronously driven (by surprise receive traffic) high speed I/O device

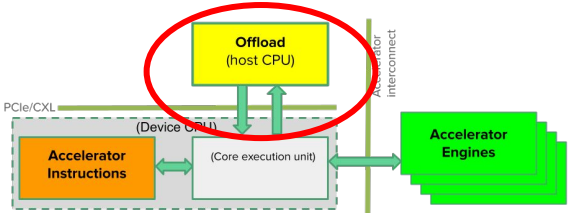
Jesse Brandeburg
Netdev0x17

Manifestations of DSA in networking domain

- Offloads
- Acceleration instructions
- Accelerator engines



Offloads



- Run processing path with accelerations in hardware device
- Use cases
  - NIC offloads, Transforms in the data path, full application offload
  - Checksum, TSO, GRO, TC Flower, TLS, TCP offload, RDMA over TCP
- API: Essentially a “tail call” from host CPU or device
  - CPU calls device: TX descriptor contains arguments requesting accelerations
  - Device calls CPU: RX descriptor contains results of accelerations
- Advantages: Relatively easy programming model, kernel takes care of security and resource isolation
- Disadvantages: “All or nothing”-- no granularity to access sub-functions

Network offload in Linux has been a disappointment!

- Few truly ubiquitous offloads
- Disconnects between SW and offload implementation cause problems

  How does SW know hardware is doing what it wants or is even correct?
- Kernel interfaces are a mess. eg.:

  NETIF_F_TSO, NETIF_F_GSO_ROBUST, NETIF_F_TSO_ECN, NETIF_F_TSO_MANGLEID, NETIF_F_TSO6, NETIF_F_FSO, NETIF_F_GSO_GRE, NETIF_F_GSO_GRE_CSUM, NETIF_F_GSO_IPXIP4, NETIF_F_GSO_IPXIP6, NETIF_F_GSO_UDP_TUNNEL, NETIF_F_GSO_UDP_TUNNEL_CSUM, NETIF_F_GSO_PARTIAL, NETIF_F_GSO_TUNNEL_REMCSUM, NETIF_F_GSO_SCTP_BIT, NETIF_F_GSO_ESP, NETIF_F_GSO_UDP, NETIF_F_GSO_UDP_L4, NETIF_F_GSO_FRAGLIST X features, vlan_features, hw_enc_features, mpls_features
- Buggy, especially protocol specific checksum offloads (see RX/TX csum offload)
- Hard to specify normative requirements for offloads

  E.g from OCP NIC Core Features Specification on Receive Segment Coalescing:

  It takes the software Generic Receive Offload (GRO) in Linux v6.3 as ground truth.

The “fundamental” Offload Requirement

*The functionality of a hardware offload must be **exactly** the same as that in the CPU software being offloaded*

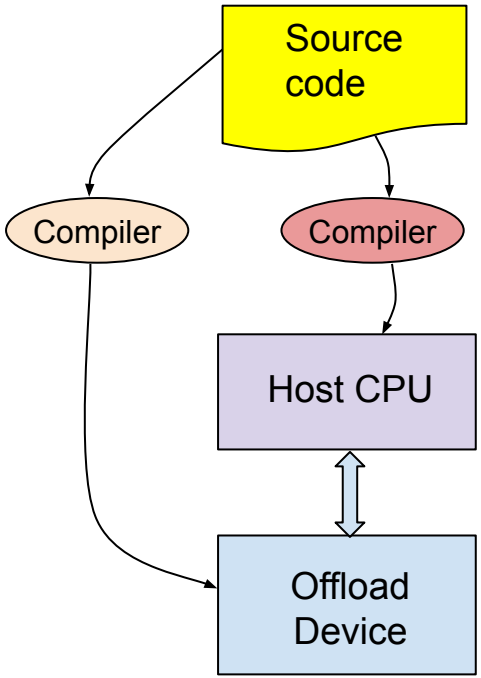
Design principles

- Adhere to fundamental offload requirement
- Core stack *offloads to driver*
  - Simple interface. E.g. just NETIF_F_HWCSUM, NETIF_F_GSO
- Driver decides what can be offloaded to hardware (on TX at least)
  - Driver decides on per packet basis what can be offloaded to HW
  - Use helper functions if offload not okay (e.g. skb_checksum_help)
- When to parse?
  - TX: Offload should not require parsing (except loopback to RX)
  - RX: Device needs to parse (except for RX checksum)
- Programmable devices are an enabler
  - Consistent methods and APIs for programmable devices
  - Mechanism to know that hardware offloads the exact kernel functionality
  - Programmable parser needed for RX

Fixing the five basic offloads (establishes path to fix others)

- TX/RX checksum
  - Eliminate NETIF_F_IPCSUM, NETIF_F_IPV6CSUM, just use NETIF_F_HWCSUM
  - Call skb_checksum_help if device can handle a packet
  - Eliminate CHECKSUM_UNNECESSARY, just use CHECKSUM_COMPLETE
  - Helper function for legacy devices to do csum-unnecessary to csum-complete conversion
- RSS (and aRFS)
  - Need programmable parser
  - Flow dissector to eBPF is enabler
- GSO (TSO)/GRO (LRO)
  - Really want GSO and GRO in eBPF
  - Eliminate as many NETIF_F_GSO_* flags a possible, just use NETIF_F_GSO*
  - Need helper function if cannot offload to the device

Running the **same** code in the CPU and target!



Requirements

- Programmable devices, compilers
- Host/device interfaces (offload processing)
- Deal with resource limits in device
- Method for host CPU to query device to see what programs are supported. Proposal:
  1. Take hash of source (or IR)
  2. Save hash in compiled images
  3. Load images in CPU and device
  4. Compare hashes at runtime. If hash of CPU image matches one reported by HW then offload is a go!

Enabling HW offload (orig. Jakub Kicinski)

1. User writes their parser in whatever ~~DSL~~ language they want
2. User compiles the parser in user space (front end->IR (CPR)->backend)
   2.1. Compiler embeds a representation of the graph in the blob
   2.2. Compile to executable for running in kernel (e.g. to XDP/eBPF)
   2.3. Take SHA1 of source code, attach hash to all executable files
3. User puts the HW blob in /lib/firmware
4. devlink dev $dev reload action parser-fetch $filename
5. devlink loads the file, parses it to extract the representation from 2.1, and passes the blob to the driver
   5.1. driver/fw reinitializes the HW parser
   5.2. user can inspect the graph by dumping the common representation from 2.1 (via something like devlink dpipe, perhaps)
6. The parser tables are annotated with Linux offload targets (routes, classic ntuple, nftables, flower etc.) with some tables being left as "raw"* (* better name would be great)
7. ethtool ntuple is extended to support insertion of arbitrary rules into the "raw" tables
8. The other tables can only be inserted into using the subsystem they are annotated for
9. To validate functional equivalency in offload compare hash (compare hash of kernel program to device's hash)
   9.1. Kernel queries driver for list of offloaded programs by hash
   9.2. Driver queries device for loaded programs

Status: Deprecating protocol specific checksum offload

- Background https://netdevconf.info/1.1/keynote-hardware-checksumming-less-more-david-s-miller.html
- Prerequisites patch sets
  - drivers: Fix drivers doing TX csum offload with EH (ipv6_skip_exthdr_no_rthdr)
  - crc-offload: Split RX CRC offload from csum offload
  - Flow_dissector: Parse into UDP encapsulations
- Convert drivers to NETIF_F_HWCSUM
  - Helper function: skb_csum_hwoffload_legacy_check
  - Fairly minor change to most drivers
- Eliminating CHECKSUM_UNNECESSARY
  - Helper function: skb_csum_rx_legacy_convert_unnecessary
  - ~2 LOC change for most drivers
  - Some uses of CHECKSUM_UNNECESSARY should be CHECKSUM_IGNORE
- Testing: A good use case netdev CI testing!

Thanks!