optimal performance everywhere

Rallying with a Formula 1

Thomas Monjalon – 6WIND
DPDK Summit Userspace – Dublin – 2016
DPDK is about performance with various architectures/devices/environments thanks to optimizations/offloads and simplicity.

call to participation to fill the gaps

more details?
1/ high throughput
   ▶ main priority

2/ low latency?
   ▶ may be studied

3/ no guarantee on low jitter
   ▶ real-time system?
Run on several Architectures

- DPDK is not Intel®
  - (not anymore Intel® DPDK)
  - however the largest contributor

- Regression must be checked on every supported machines
  - best effort from contributors
<table>
<thead>
<tr>
<th></th>
<th>ARMv7</th>
<th>ARMv8</th>
<th>Power8</th>
<th>x86-32</th>
<th>x86-64</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib/acl</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lib/distributor</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lib/hash</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lib/ip_frag</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lib/lpm</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lib/reorder</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>lib/sched</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/bnx2x</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/bnxt</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/cxgbe</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/e1000</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/enax</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/enic</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/fm10k</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/i40e</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/ixgbe</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/mlx4</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/mlx5</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/nfp</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/qede</td>
<td>-</td>
<td>-</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/szedata2</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/thunderx</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/vhost</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>net/virtio</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Architecture-specific Implementation

- reusable and generic code in EAL
  - `librte_eal/common/include/generic/`
  - `librte_eal/common/include/arch/`
  - `librte_eal/common/arch/`

- library or driver specific code in separate files
  - `librte_acl/acl_run_altivec.c`
  - `librte_acl/acl_run_avx2.c`
  - `librte_acl/acl_run_neon.c`
  - `librte_acl/acl_run_scalar.c`
  - `librte_acl/acl_run_sse.c`

- build-time CPU features supported by the compiler
  - `#ifdef RTE_MACHINE_CPUFLAG_*`

- run-time CPU detection
  - `rte_cpu_get_flag_enabled(RTE_CPUFLAG_*)`

- best available optimization in only one build/package (e.g. SSE3/SSE4/AVX2/AVX512)
Function Multi-Versioning

- manual/legacy method (currently used in DPDK)
  - specific compilation of whole file
  - function pointer defined at run-time

- flatten function attribute
  - inline calls in the function
  - allow more code to be optimized by compiler

- target function attribute
  - build function with specific flags

- target_clones function attribute
  - build function clones with specific flags
  - select the best one at run-time through ifunc resolver
  - no manual tuning
Vector (SIMD) Optimizations

- ISA-specific intrinsic functions
- generic GCC vector type
  - __attribute__((vector_size(n)))
  - limited to simple operations

- Maintenance of vectorized code
  - Who is responsible and/or expert? lib maintainer? arch maintainer?
  - How to coordinate a change affecting several drivers on several architectures?
  - Risk of deviating features/behaviour in driver paths
Many techniques
- hot/cold attributes
- inlining
- cache alignment
- bulk
- prefetch
- ...

How generic is the performance gain (or loss)?
Long list of supported Devices

- multi-bus
- PCI
- SoC
- virtual

- generic interfaces
  - net (ethdev)
  - crypto (cryptodev)
Unlock the full power of the Devices

- offloads in NIC
  - load balancing (flow steering)
    - new Rx filtering API
  - segmentation offloads (LRO, TSO)
    - new software implementation for virtio
  - checksum offloads
    - new flags
- common support
  - software emulation to fill the gaps
- early access to hardware features
  - unstable API
  - specific features in common interface or picked in drivers?
Unlock the full power of the Machine

- custom mempool handlers
  - not used yet?

- event driven model
  - NPU

- other usages in software design?
Flexible Packaging

- split in multiple libraries
  - or combined in one linker script
- static .a
  - more efficient
- dynamic .so
  - distributions choice
- drivers as plugins
- standard make install (since v2.2)

- integrated in some distributions
Choices of Linux kernel Bypass

- userspace-friendly kernel modules
  - vmxnet3-usermap
  - mlx/verbs thanks to RDMA
    - no root access required
    - less code in PMD
  - i40e: 34.8 kSLOC
  - mlx4: 4.2 kSLOC

- UIO kernel modules
  - igb_uio (out-of-tree)
  - uio_pci_generic (no MSI/MSI-X, i.e. no VF device)

- VFIO kernel module
  - vfio-pci (IOMMU, performance loss?)
  - vfio-pci noiommu mode (since v4.5)
Multiple Environments

- not only Linux
- FreeBSD
- not only kernel bypass in a full blown OS?
  - OsV unikernel?

- no hugepage
  - works with virtual devices
  - requires more work for DMA
Usability

- more/better default values
  - command line (-m, -n, -c, etc)
  - thresholds
- avoid build-time configuration
- run-time configuration
  - by application
    - argc/argv must be replaced by a simpler API
  - by user
    - command line
    - file
    - specific to the application
- a long road
- ease compilation in existing projects
  - must generate a pkg-config file
- pluggable logs
  - should be fixed now
- avoid forcing application design
  - thread management?
- no exit()
  - kill rte_panic() in libraries
More debug tools
- pdump
- valgrind

Language Bindings
- C native
- C++ supported as best effort
- Other generic languages? Go? Rust?
- Specific languages? P4? eBPF/XDP?
Who is driving this super car?

- **Vendors**
  - show capabilities of their devices and processors

- **R&D labs**
  - userspace development accelerate time to market

- **Manufacturers**
  - highest performance
Where is it Used?

- **Equipments**
  - Telecom, High End Switches, Large Volume Servers, Security

- **Technology**
  - Legacy, SDN, NFV

- **OS**
  - RHEL, Fedora, Ubuntu, Clear Linux, Mirantis OpenStack

- **Stacks**
  - 6WIND, OVS, BESS, VPP, ODP, OpenFastPath, Seastar, ANS, mTCP, Butterfly, Packet-journey

- **Traffic Generators**
  - pktgen-dpdk, Moongen, TRex, WARP17
started for x86 with Intel drivers only

2012: DPDK users working without cooperation
   - private DPDK forks

2013: 6WIND launched dpdk.org initiative

other similar projects were started
   - Cisco VPP (closed source before this year)
   - Italian projects from Pisa University (Netmap, PF_RING, PFQ)
   - 2016: Linux kernel start building XDP solution

2016: major hardware vendors involved in DPDK
   - IBM Power and ARM architectures
   - drivers for almost all fast NICs

2017: network processors (NPU)?

New Governance?
Questions?

Thomas Monjalon
thomas.monjalon@6wind.com
tmonjalo / freenode #DPDK