Deflate your Data with DPDK

Fiona Trahe & Lee Daly
Intel
Agenda

- Overview of DPDK compressdev
- Deep dive into some API concepts
- Poll-mode drivers
  - Intel QuickAssist PMD
  - Intel ISA-L PMD
Agenda

- Overview of DPDK compressdev
- Deep dive into some API concepts
- Poll-mode drivers
  - Intel QuickAssist PMD
  - Intel ISA-L PMD
DPDK and SPDK overview

- DPDK https://www.dpdk.org/ is the Data Plane Development Kit that consists of libraries to accelerate packet processing workloads running on a wide variety of CPU architectures.

- SPDK http://spdk.io/ provides a set of tools and libraries for writing high performance, scalable, user-mode storage applications. It relies on DPDK's proven base functionality to implement its memory management.
# DPDK libraries

<table>
<thead>
<tr>
<th>Core libraries</th>
<th>Packet classification</th>
<th>Accelerated SW libraries</th>
<th>Stats</th>
<th>QoS</th>
<th>Packet Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core functions such as memory management, software rings, timers etc.</td>
<td>Software libraries for hash/exact match, LPM, ACL etc.</td>
<td>Common functions such as IP fragmentation, reassembly, reordering etc.</td>
<td>Libraries for collecting and reporting statistics.</td>
<td>Libraries for QoS scheduling and metering/policing</td>
<td>Libraries for creating complex pipelines in software.</td>
</tr>
<tr>
<td>ETHDEV</td>
<td>CRYPTODEV</td>
<td>EVENTDEV</td>
<td>SECURITY</td>
<td>COMPRESSDEV</td>
<td>BBDEV</td>
</tr>
<tr>
<td>PMDs for physical and virtual Ethernet devices</td>
<td>PMDs for HW and SW crypto accelerators</td>
<td>Event-driven PMDs</td>
<td>Hardware acceleration APIs for security protocols</td>
<td>PMDs for HW and SW compression accelerators</td>
<td>PMDs for HW and SW wireless accelerators</td>
</tr>
</tbody>
</table>
# dpdk/compressdev Key Features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asynchronous burst API</td>
<td>Support HW &amp; SW acceleration to allow compression for data greater than 64K.</td>
</tr>
<tr>
<td>Chained Mbufs</td>
<td>To allow compression for data greater than 64K.</td>
</tr>
<tr>
<td>Compression Algorithms</td>
<td>Deflate LZS</td>
</tr>
<tr>
<td>Compression Levels</td>
<td>-1: PMD Default 1: Fastest 9: Best Ratio</td>
</tr>
<tr>
<td>Checksum</td>
<td>#1 CRC32 #2 Adler32 #3 Combined Adler32_CR C32</td>
</tr>
<tr>
<td>Hash Generation</td>
<td>#1 SHA1 #2 SHA256</td>
</tr>
</tbody>
</table>
Compression API Flow

Application Start
- Allocate src/dst memory
  - Op Pool Creation
  - Device Config
  - Queue Pair setup
  - Device Start
  - Create Private Xform

Main Loop
- Op allocate
  - Build the Ops
  - Enqueue Burst
  - Dequeue Burst
  - Free ops

- Free Private Xform
- Stop Device
- Close Device

Application End
So you want to run a compression app?

### Download and setup DPDK
1. `git clone http://dpdk.org/git/dpdk`
2. `./usertools/dpdk-setup.sh`
   - Option=[22/21]
   - num hugepages= [64], [64]

### Compile DPDK
3. `meson [build directory]`
4. `cd [build directory]`
5. `ninja`

### Run Compressdev Unit Test
6. `./[build directory]/test/test/dpdk-test --vdev=compress_isal`
7. `RTE>> compressdev_autotest`
Agenda

- Overview of DPDK compressdev
- Deep dive into some API concepts
- Poll-mode drivers
  - Intel QuickAssist PMD
  - Intel ISA-L PMD
compressdev – operation

Operations contain both input and output parameters
Compression terminology - stateless

Raw data → Compression operation → Compressed data → Decompression operation → Decompressed data

1. Raw data
2. Compressed data
3. Decompressed data

chunk1 → 1 → chunk1
chunk2 → 2 → chunk2
chunk3 → 3 → chunk3
Compression terminology - stateful

Related data-set

chunk1  chunk2  chunk3

1  2  3

chunk1  chunk2  chunk3

Depends on

1 Depends on 2 Depends on 3

2018 Storage Developer Conference. © Intel. All Rights Reserved.
Compression terminology recap

- **Stateless compression**
  - Data in each operation is treated independently.
  - Operations can be carried out in parallel.
  - Each chunk of compressed data can be independently decompressed.
  - Order is not important.
  - May give higher throughput.

- **Stateful compression**
  - Data in later operations may depend on data in earlier operations.
  - History and state is saved at end of each operation to be used in processing subsequent operations.
  - Can only do one operation at a time.
  - Order is important.
  - Is useful if all the data is not available at the same time, e.g. related chunks are being received from a network.
  - May give better compression ratio.
compressdev – private_xform

A private_xform is used for STATELESS operations

Application

rte_comp_xform {
compress or decompress,
algorithm, huffman type,
level, window_size,
checksum,
hash_algo }

create_private_xform()

PMD

private_xform pool

Private data
derived
from xform
compressdev – stream

A stream is used for STATEFUL operations

rte_comp_xform {
  compress or decompress,
  algorithm, huffman type,
  level, window_size,
  checksum,
  hash_algo }

create_stream()

Private data derived from xform
+ state and history
meta-data
Where do SGLs fit in?

Don’t confuse stateful with scatter-gather lists (SGLs), also called chained mbufs.

- Stateless/Stateful refers to data relationship across operations.
- SGLs are chained data buffers within an operation.
- SGLs can be used in stateless or stateful ops
Compression – stateless with SGLs

- A chunk passed in or out of an operation may be comprised of one or more buffers (segments) chained together.
- Segments can be any size.
- There is no correlation between the number of segments passed in for compression and the number of segments it will decompress to.
Capabilities

Compression service capability flags

Device capability flags

RTE_COMPDEV_FF_HW_ACCELERATED
RTE_COMPDEV_FF_CPU_SSE
RTE_COMPDEV_FF_CPU_AVX512
etc

```
struct rte_compressdev_capabilities {
    enum rte_comp_algorithm algo;
    /* Compression algorithm */
    uint64_t comp_feature_flags;
    /**< Bitmask of flags for compression service features */
    struct rte_param_log2_range window_size;
    /**< Window size range in base two log byte values */
};
```
Agenda

- Overview of DPDK compressdev
- Deep dive into some API concepts
- Poll-mode drivers
  - Intel QuickAssist PMD
  - Intel ISA-L PMD
compressdev poll mode drivers

Software Drivers

- Intel ISA-L PMD
  - Employs the compression engine from Intel’s ISA-L library, optimized for Intel Architecture.
- Zlib PMD
  - Software driver uses Zlib’s compression library.

Hardware Drivers

- Intel QAT PMD
  - Utilizes Intel’s QuickAssist family of hardware accelerators.
- Cavium Octeontx PMD
  - Uses HW offload device, found in Cavium’s Octeontx SoC family.
Agenda

- Overview of DPDK compressdev
- Deep dive into some API concepts
- Poll-mode drivers
  - Intel QuickAssist PMD
  - Intel ISA-L PMD
ISA-L PMD Features

- “Deflate” compression algorithm
- Adler32 & CRC32 Checksum available
- Stateless functionality
- Levels
  - #1 Fastest
  - #2 Higher ratio
  - #3 Best Ratio (AVX512/2 Only)
- Allows for Shareable Private Xfrom
- Single & Chained Mbuf Support
- Virtual Device [ --vdev= ]
- Fixed & Dynamic Huffman
How to use the ISA-L PMD

1. Install ISA-L Library
2. Compile the ISA-L with DPDK
3. Initialize driver as virtual device in app

Make

Meson

App Command line arg
--vdev=compress_isal

Function in application
rte_dev_init(compress_isal)
Agenda

- Overview of DPDK compressdev
- Deep dive into some API concepts
- Poll-mode drivers
  - Intel QuickAssist PMD
  - Intel ISA-L PMD
Intel QuickAssist (QAT) poll mode driver

- Offloads compression operations to QAT hardware engines
- Many engines so can process ops in parallel
- MMIOs are expensive so offload cost can be minimised by sending larger bursts
QAT capabilities

- Hardware offload
- SGLs
- Stateless compression
- Deflate algorithm with Huffman fixed and dynamic encoding
- Shareable private_xform
- Multi-packet checksum
- Adler, CRC32 and combined checksum generation
QAT PMD depends on QAT kernel driver

- QAT kernel driver initialises PF device and exposes Virtual Functions
- VFs are enabled to DPDK QAT PMDs by vfio-pci or igb_uio
- A QAT VF can support both crypto and compression PMDs simultaneously
Questions?

More information @

or contact us @
- fiona.trahe@intel.com
- lee.daly@intel.com