Intel Mount Evans IPU Based NVMe/TCP Initiator Offload

Presented by
Ziye Yang, Staff Software Engineer, Intel
Yadong Li, Principal Engineer, Intel
Sudheer Mogilappagari, Software Engineer, Intel
Agenda

- Brief Introduction of IPU
- Intel Mount Evans IPU Introduction
- SPDK NVMe/TCP Initiator Design
- SPDK NVMe/TCP Initiator Performance Evaluation with Intel® Ethernet 800 Series with Application Device Queues (ADQ)
- Summary
Brief Introduction of IPU
Brief Introduction of IPU

- Infrastructure Processing Unit (IPU) is an evolution of SmartNIC, provides full infrastructure offload.
  - Highly intelligent infrastructure acceleration. Minimize latency and jitter and maximize revenue from CPU.
  - Separation of Infrastructure & Tenant, system-level security, control and isolation.
  - HW and SW programmable, built to customer needs.

- In this topic, we will briefly introduce Intel’s ASIC based IPU – Mount Evans, and then describe NVMe over TCP Initiator implementation as an example for IPU based storage offload.
Intel Mount Evans IPU Introduction
Mount Evans

Intel’s 200G IPU

Hyperscale Ready
- Co-designed with a top cloud provider
- Integrated learnings from multiple gen. of FPGA sNIC/IPU
- High performance under real world load
- Security and isolation from the ground up

Technology Innovation
- Highly Programmable Packet Processing Engine
- NVMe storage interface scaled up from Intel Optane Tech
- Next Generation Reliable Transport
- Advanced crypto and compression accel.

Software
- SW/HW/Accel co-design
- P4 Studio based on Barefoot
- Leverage and extend DPDK and SPDK
- Enable broad adoption of IPUs
Mount Evans

NVMe Initiator

- Multi-Host support (4 PFs for Hosts)
- SR-IOV support
- Data-at-rest crypto (AES-XTS)
- IOPS and BW limiting
- LCE for lookaside crypto, compression and CRC offloads, support chained ops
- Support local attached storage (4 PCIe root ports)
Scale out Storage Architecture

Software

- Customer’s Software
- Standard OS
- NVMe – SSD1

Hardware

- CPU
- PCIe

Control Plane

- Create New Virtual SSD
- Flexible Storage
  - NVMeoTCP
  - NVMeoRDMA
  - Ceph
  - BYOS
- Create Virtual NVMe Device
- Virt + NVMe
  - protocol layer
  - QoS
  - Load balance
  - HW crypto, compress, CRC

Storage Service

- Create New Virtual Volume - SSD1
- SSD1 – 250GB
- Virtualized Network
- Shared Storage
Mount Evans Based NVMe over TCP Initiator

- NVMe Protocol Initiator (NPI) HW auto fetches NVMe CMDs into SoC memory.
- SoC SW uses LCE/QAT DMA engine to fetch PRP lists and move data payloads.
- LCE/QAT used for CRC, crypto and compression offloads. Support chained ops.
- Fully integrated with SPDK NVMe-oF SW stack
  - NPI Transport for integration with nvmf layer.
  - NPI Transport uses NVMe Initiator Config Driver for device configurations and CMDs processing.
SPDK NVMe over TCP Initiator Introduction
SPDK NVMe over TCP Initiator Introduction

Following topics will be introduced in this section:

- SPDK quick overview with NVMe-oF related components.
- Detailed information of user space NVMe over TCP initiator.
- How Application Device Queues (ADQ) Technology is used in SPDK NVMe over TCP initiator.
SPDK Architecture (NVMe-oF view)

IPU storage initiator SW for NVMe-oF TCP can be integrated with SPDK at
- Nvmf layer
- bdev layer
- NVMe tcp driver

https://SPDK.IO
https://SPDK.IO/CN

Main Web Presence
SPDK NVMe over TCP Initiator

It is part of SPDK user space NVMe-oF solution for the initiator side based on user space NVMe device drivers, socks etc.

- **POSIX** (Stable, no dependency on kernel)
- **Uring** (Requires Linux kernel > 5.4.3, currently it is experimental)
Intel® Ethernet 800 Series with Application Device Queues (ADQ)

ADQ is an open technology designed to improve application specific queuing and steering.

ADQ works by:
• Filtering application traffic to a dedicated set of queues
• Application threads of execution are connected to specific queues within the ADQ queue set
• Bandwidth control of application egress (Tx) network traffic

**Without ADQ**
Application traffic intermixed with other traffic types

**With ADQ**
Application traffic to a dedicated set of queues

**ADQ benefits:**
- Increases application predictability
- Reduces application latency
- Improves application throughput
ADQ Enabling in SPDK Socket Layer

SPDK sock library support

- Common support for both server & client side.
  - Implement a function to get NAPI_ID for the socket file descriptor (FD).
    - Through `getsockopt` function on optname “SO_INCOMING_NAPI_ID”

- Support for server side
  - In order to address the efficiency usage issue on ADQ inside a SPDK application which is started with multiple CPU cores.
    - Implement a function to get optimal socket polling group for each socket FD.
    - Then the upper layer can create a scheduler to distribute socket FD with the same NAPI_IDS into the same socket polling group to better leverage the ADQ features.
ADQ Enabling in SPDK Socket Layer to Support Initiator Usage

- **Challenges:**
  - Socket is already created on a dedicated SPDK_thread on the initiator side, so cannot use the same methodology on the server side, i.e., seek an optimal socket polling group and schedule the socket into it.
    - We need leverage `spdk_thread_send_msg` among `spdk_threads` to do socket I/O and there is performance penalty.

- **Solutions:**
  - Leverage "NAPI_ID & SO_MARK" optnames on the socket. The purpose is to enforce all the sockets in each polling group with the same value for `SO_MARK optname`.
  - Then the NIC driver (with ADQ enabled) in Linux kernel queues packets of all sockets with same value of `SO_MARK optname` on same hardware queue.
Socket and Group Management: Add a socket into a group

Create a socket A:
1. Get a valid value from “NAPI_ID” optname and save as placement_id
2. Maintain a global list L with: <placement_id, group, ref> structure

To Add a socket A into a group (G):

- PLACEMENT_MARK based strategy enabled?
  - NO
    - Add the socket A into group(G)
  - YES
    - G’s placement_ID is valid?
      - NO
        - 1. Placement_id (G) init (Get an unused placement_id from List L)
        - 2. SO_MARK(A) = Placement_id(G)
        - 3. Update all any socket X in group G, with
          SO_MAKR(X) = Placement_id(G)
        - 4. Add the socket A into group(G)
      - YES
        - 1. SO_MARK(A) = Placement_id (G)
        - 2. Add the socket A into group(G)
SPDK NVMe/TCP Initiator Performance Evaluation on Intel® Ethernet 800 Series with Application Device Queues (ADQ)

* For SPDK NVMe/TCP Target performance, please refer to SDC’20 presentation – Optimizing User Space NVMe-oF TCP Transport Solution
SPDK NVMe/TCP with ADQ Test Topology

Intel® Ethernet Network Adapter E810-CQDA2 with ADQ

**Initiator**
Intel® Xeon® Scalable Platform
OS: Linux RHEL 8.2 k5.13.1

<table>
<thead>
<tr>
<th>Userspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application (FIO)</td>
</tr>
<tr>
<td>SPDK FIO plugin</td>
</tr>
<tr>
<td>SPDK NVMe/TCP Initiator</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
</tr>
<tr>
<td>Ethernet Driver</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Userspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP/IP</td>
</tr>
<tr>
<td>Ethernet Driver</td>
</tr>
</tbody>
</table>

**Target**
Intel® Xeon® Scalable Platform
OS: Linux RHEL 8.2 k5.13.1

<table>
<thead>
<tr>
<th>Userspace</th>
</tr>
</thead>
<tbody>
<tr>
<td>nvmf_tgt</td>
</tr>
<tr>
<td>SPDK NVMe/TCP Target</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel</th>
</tr>
</thead>
<tbody>
<tr>
<td>uio</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Intel® Ethernet Network Adapter E810-CQDA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
</tr>
<tr>
<td>P2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSD</th>
<th>SSD</th>
<th>SSD</th>
</tr>
</thead>
</table>

100 Gbps
SPDK NVMe/TCP Initiator Read Performance
Intel® Ethernet Network Adapter E810-CQDA2 with ADQ

Throughput & Average Latency Performance
(4KB Reads; 1500B MTU; QD=32; randomread)

SEE APPENDIX FOR TESTING CONFIGURATION. Results may vary. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. See configuration disclosure for details. For more information regarding performance and optimization choices in Intel software products., please visit https://software.intel.com/en-us/articles/optimization-notice.
SPDK NVMe/TCP Initiator Tail Latency
Intel® Ethernet Network Adapter E810-CQDA2 with ADQ

Tail Latency Comparison
(4KB Reads; 1500B MTU; QD=32; randomread)

Lower is better.

SEE APPENDIX FOR TESTING CONFIGURATION. Results may vary. For more complete information about performance and benchmark results, visit www.intel.com/benchmarks. See configuration disclosure for details. For more information regarding performance and optimization choices in Intel software products, please visit https://software.intel.com/en-us/articles/optimization-notice.
Summary
Summary

- IPU provides efficient infrastructure offload and improves data center efficiency. Designed to help minimize latency and jitter and maximize revenue from CPU.

- In this session, we shared
  - Intel Mount Evans IPU based NVMe/TCP initiator offload.
  - Integration of SPDK NVMe/TCP initiator into Mount Evans SW stack.
  - Tests with Intel® Ethernet 800 Series Network Adapter showed significant performance improvement with ADQ enabled for NVMe/TCP initiator.
    - Integration of Intel ADQ feature provides a low latency and high performance SPDK NVMe/TCP initiator solution.

- Finally, we welcome any SPDK optimizations and enhancements to make it best for IPU storage solutions.
Legal Information

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex. Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure. Your costs and results may vary. Intel technologies may require enabled hardware, software or service activation.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document. Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps. All product plans and roadmaps are subject to change without notice.

Statements in this document that refer to future plans or expectations are forward-looking statements. These statements are based on current expectations and involve many risks and uncertainties that could cause actual results to differ materially from those expressed or implied in such statements. For more information on the factors that could cause actual results to differ materially, see our most recent earnings release and SEC filings at www.intc.com.

© INTEL CORPORATION. Intel, the Intel logo, and other Intel marks are trademarks of Intel Corporation or its subsidiaries. Other names and brands may be claimed as the property of others.
Q & A
Appendix: Configuration
### SPDK NVMe/TCP with ADQ Testing Configuration

<table>
<thead>
<tr>
<th></th>
<th>Initiator</th>
<th>Target</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test by</strong></td>
<td>Intel</td>
<td>Intel</td>
</tr>
<tr>
<td><strong>Test date</strong></td>
<td>9/9/2021</td>
<td>9/09/2021</td>
</tr>
<tr>
<td><strong>Platform</strong></td>
<td>Sawtooth Pass (S2600STQ)</td>
<td>Sawtooth Pass (S2600STB)</td>
</tr>
<tr>
<td><strong># Nodes</strong></td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td><strong># Sockets</strong></td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Intel(R) Xeon(R) Platinum 8280L CPU @ 2.70GHz</td>
<td>Intel(R) Xeon(R) Platinum 8180 CPU @ 2.50GHz</td>
</tr>
<tr>
<td><strong>Cores/socket, Threads/socket</strong></td>
<td>28 cores per socket, 56 threads per socket</td>
<td>28 cores per socket, 56 threads per socket</td>
</tr>
<tr>
<td><strong>ucode</strong></td>
<td>0x5003102</td>
<td>0x2006b06</td>
</tr>
<tr>
<td><strong>HT</strong></td>
<td>Disabled</td>
<td>Disabled*</td>
</tr>
<tr>
<td><strong>Turbo</strong></td>
<td>Enabled</td>
<td>Enabled</td>
</tr>
<tr>
<td><strong>BIOS version</strong></td>
<td>SE5C620.86B.02.01.0008.031920191559</td>
<td>SE5C620.86B.02.01.0008.031920191559</td>
</tr>
<tr>
<td><strong>System DDR Mem Config: slots / cap / run-speed</strong></td>
<td>16 slots / 16GB / 2666 MT/s</td>
<td>12 slots / 16GB / 2666 MT/s</td>
</tr>
<tr>
<td><strong>System DCPMM Config: slots / cap / run-speed</strong></td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>Total Memory/Node (DDR+DCPMM)</strong></td>
<td>128GB DDR4-2666 DIMM</td>
<td>96GB DDR4-2666 DIMM</td>
</tr>
<tr>
<td><strong>Storage - boot</strong></td>
<td>100GB SATA SSD</td>
<td>100GB SATA SSD</td>
</tr>
<tr>
<td><strong>Storage - application drives</strong></td>
<td>N/A</td>
<td>6x Intel Corporation Optane SSD 900P Series, PCIe 3.0, x4</td>
</tr>
<tr>
<td><strong>NIC</strong></td>
<td>Intel® Ethernet Network Adapter E810-CQDA2</td>
<td>Intel® Ethernet Network Adapter E810-CQDA2</td>
</tr>
<tr>
<td><strong>PCH</strong></td>
<td>Intel Corporation C620 Series Chipset Family</td>
<td>Intel Corporation C620 Series Chipset Family</td>
</tr>
<tr>
<td><strong>Other HW (Accelerator)</strong></td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>Red Hat Enterprise Linux 8.2 (Ootpa)</td>
<td>Red Hat Enterprise Linux 8.2 (Ootpa)</td>
</tr>
<tr>
<td><strong>Kernel</strong></td>
<td>5.13.1</td>
<td>5.13.1</td>
</tr>
<tr>
<td><strong>Workload &amp; version</strong></td>
<td>Fio 3.25; SPDK 21.07</td>
<td>SPDK 21.01</td>
</tr>
<tr>
<td><strong>Compiler</strong></td>
<td>GCC 8.3.1 20191121 (Red Hat 8.3.1-5)</td>
<td>GCC 8.3.1 20191121 (Red Hat 8.3.1-5)</td>
</tr>
<tr>
<td><strong>NIC Driver</strong></td>
<td>ice-1.6.8; FW: 0x80008125</td>
<td>ice-1.6.8; FW: 0x80008256</td>
</tr>
</tbody>
</table>

*HT was turned off for benchmark purposes to not schedule FIO on threads of same physical core*
SPDK NVMe/TCP with ADQ: OS and Network Adapter Configuration

**Red Hat® Enterprise Linux 8.2 Configuration**

stopped: irqbalance, cpupower, firewalld, SELINUX disabled

tuned-adm profile throughput-performance

Kernel 5.13.1

```bash
sysctl -w net.core.somaxconn=4096
sysctl -w net.core.netdev_max_backlog=8192
sysctl -w net.ipv4.tcp_max_syn_backlog=16384
sysctl -w net.core.rmem_max=16777216
sysctl -w net.core.wmem_max=16777216
sysctl -w net.ipv4.tcp_mem="764688 1019584 16777216"
```

**Network Adapter Configuration**

**ADQ “On”**

```bash
ethtool -coalesce <interface> adaptive-rx off rx-usecs 0
ethtool -coalesce <interface> adaptive-tx off tx-usecs 500
ethtool --set-priv-flags <interface> channel-inline-flow-director on
ethtool --set-priv-flags <interface> channel-pkt-clean-bp-stop on
ethtool --set-priv-flags <interface> channel-pkt-clean-bp-stop-cfg on
ethtool --offload <interface> hw-tc-offload on
```

```bash
sysctl -w net.core.busy_read=1
sysctl -w net.core.busy_poll=1
```

```bash
<path-to-ice>/scripts/set_irq_affinity -X local <interface>
```

```bash
<path-to-ice>/scripts/set_xps_rxqs <interface>
```

```bash
tc qdisc add dev <interface> root mqprio num_tc 2 map 0 1 queues 2@0 $adq_qs@2 hw 1 mode channel
```

```bash
tc qdisc add dev <interface> ingress
```

```bash
tc filter add dev <interface> protocol ip parent ffff: prio 1 flower dst_ip $addr/32 ip_proto tcp dst_port $((port)) skip_sw hw_tc 1 [on Target system]
```

```bash
tc filter add dev <interface> protocol ip parent ffff: prio 1 flower dst_ip $addr/32 ip_proto tcp src_port $((port)) skip_sw hw_tc 1 [on Initiator system]
```

**ADQ “Off” (Baseline)**

```bash
ethtool -coalesce <interface> adaptive-rx off rx-usecs 50
ethtool -coalesce <interface> adaptive-tx off tx-usecs 50
```

```bash
sysctl -w net.core.busy_read=0
```

```bash
<path-to-ice>/scripts/set_irq_affinity -X local <interface>
```

```bash
ethtool -L <interface> rx $queues tx $queues
```
Please take a moment to rate this session.

Your feedback is important to us.