DeLiBA: An Open-Source Hardware/Software Framework for the Development of Linux Block I/O Accelerators

Babar Khan, Carsten Heinz, Andreas Koch
Embedded Systems and Applications Group, TU Darmstadt, Germany

32nd International Conference on Field Programmable Logic and Application (FPL) 2022
hosted by Queens University Belfast, United Kingdom
29th August - 2nd September 2022
What is Multi-Queue (MQ) Linux Block I/O Layer

- Part of **Linux** operating system.
- Responsible for handling block devices like **Hard Disks, SSDs**.
- Interface between **Applications** and **Storage**
- **Multi-Queue (MQ)** = For **Multi-Core Systems**
Bottleneck & Complexity in Block I/O Layer

- **18K - 20K** instructions in single **4KB** request.

- Approx. **60%** and **90%** of total execution time in kernel on x86 and on ARM resp for **4KB** request.

- Around **64K** lines of codes excluding drivers.
Motivation: Ceph

What is **Ceph**: 

- Ceph is a software-defined distributed storage protocol.
- Ceph **Multi-Queue (MQ) Block Device Driver** is part of **Linux** operating system.
- Our first use case: **Ceph I/O Accelerator**
Research Problem

Revisiting MQ Linux Block I/O layer gives 2 research problems:

- **First**: MQ Block I/O Layer still has a *performance bottleneck*.

- **Second**: MQ Block I/O Layer codebase is *notoriously complex*.
DeLiBA Framework

DeLiBA addresses **both** research problems:

- Enables use of programming tools *at userspace* to tackle *complexity*

- Enables use of **FPGA accelerators** to tackle *performance bottleneck*
DeLiBA Architecture

*Stage 1:* Client Application launches read/write request

*Stage 2:* Network Block Device

*Stage 3:* Block Layer Libraries

*Stage 4:* Hardware/Software Interface (FPGA)

*Stage 5:* Network Interface Card (NIC)

*Stage 6:* Block I/O placement on remote server

Data Center
FPGA
Alveo U280

Block Devices

Linux Network stack

Network Interface Card (NIC)

Network library

HW-SW

task based interface

Block Layer Libraries

socket NBD request

socket NBD reply

/dev/nbdx

Network Block Device (NBD)

Cache & Schedule

I/O Pool

Userspace

Client read () write ()

User-level NBD Server

kernel
**Userspace: Network Block Device (NBD)**

Network Block Device moves Block I/O operations in userspace with 2 context switches.

### Stage 1: Client Application
- Client Application launches read/write request

### Stage 2: Network Block Device
- Client Application reads/write to /dev/nbdx
- Network Block Device (NBD) receives socket NBD requests and replies
- NBD Server handles userspace requests

**Diagram Details:**
- **Network Block Device (NBD)**
- **Client read () write ()**
- **User-level NBD Server**
- **Cache & Schedule**
- **I/O Pool**
- **Block Layer Libraries**
- **Network library**
- **Network Interface Card (NIC)**
- **Data Center FPGA Alveo U280**
- **Linux Network stack**
- **Network Interface**
- **HW-SW**
- **Task based interface**

**Notes:**
- Userspace
- Block Devices
- Kernel
Ceph Protocol Specific Libraries

Stage 1: Client Application launch read/write request
Stage 2: Network Block Device
Stage 3: Block Layer Libraries

Network InterFace Card (NIC)
Data Center FPGA
Alveo U280

Linux Network stack
Network library
HW-SW
Block Layer Libraries
I/O Pool
Cache & Schedule
User-level NBD Server
User read () write ()
Client

/dev/nbdx
Network Block Device (NBD)
socket NBD reply
Hardware/Software Interface for FPGA

DeLiBA relies on FPGA middleware Task-Parallel System Composer (TaPaSCo)
Overall Caveats i.e. Context Switches

**Stage 1:** Client Application launch read/write request

**Stage 2:** Network Block Device

**Stage 3:** Block Layer Libraries

**Stage 4:** Hardware/Software Interface (FPGA)

**Stage 5:** Network Interface Card (NIC)

**Stage 6:** Block I/O placement on remote server

Data Center FPGA

Alveo U280

Network library

Linux Network stack

Network Interface Card (NIC)

Block Devices

Userspace

Client read () write ()

User-level NBD Server

Cache & Schedule

I/O Pool

Block Layer Libraries

HW-SW

task based interface

Network Block Device (NBD)

/kernel
/dev/nbd/ /b

socket NBD request

socket NBD reply

Network Block Device (NBD)
I/O Accelerator

Ceph I/O Accelerator is based on C++ Vitis High-Level Synthesis (HLS) for target clock frequency **300 MHz** (pre-synthesis):

- **Interface Synthesis:**
  AXI memory mapped automated by TaPaSCo.

- **Algorithmic Synthesis:**
  HLS based transformations i.e. Loop and Memory optimizations.
### Hardware Results and Speedups

<table>
<thead>
<tr>
<th>kernel</th>
<th>Software Execution Time</th>
<th>Hardware kernel Execution</th>
<th>Total Execution with Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Straw Bucket (pure HLS code)</td>
<td>85 µs</td>
<td>0.675 µs</td>
<td>70 µs</td>
</tr>
<tr>
<td>Straw Bucket (using Vitis ln x function IP)</td>
<td>85 µs</td>
<td>0.885 µs</td>
<td>70 µs</td>
</tr>
<tr>
<td>List Bucket</td>
<td>65 µs</td>
<td>0.280 µs</td>
<td>72 µs</td>
</tr>
<tr>
<td>Uniform Bucket</td>
<td>20 µs</td>
<td>2.240 µs</td>
<td>25 µs</td>
</tr>
<tr>
<td>Tree Bucket</td>
<td>45 µs</td>
<td>0.810 µs</td>
<td>45 µs</td>
</tr>
</tbody>
</table>

- **Per kernel speedup:** 120x
- **Overall Speedup:** 1.2x (huge potential for further acceleration)
Evaluation on Hardware

Following Hardware setup:

- AMD EPYC Rome 7302P 16-core CPU with 128GB of memory, attached by 10 Gb/s Ethernet to the Ceph server.

- Xilinx Alveo U280 FPGA card attached to the client host by PCIe Gen3 x8 and uses a system clock of 200 MHz
### Evaluation Hardware – Throughput (128KB)

<table>
<thead>
<tr>
<th></th>
<th>Throughput MB/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sequential READ</strong></td>
<td>548</td>
</tr>
<tr>
<td><strong>Sequential WRITE</strong></td>
<td>280</td>
</tr>
<tr>
<td><strong>Random READ</strong></td>
<td>480</td>
</tr>
<tr>
<td><strong>Random WRITE</strong></td>
<td>200</td>
</tr>
<tr>
<td><strong>Software</strong></td>
<td>548</td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td>558</td>
</tr>
<tr>
<td></td>
<td>280</td>
</tr>
<tr>
<td></td>
<td>480</td>
</tr>
<tr>
<td></td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>558</td>
</tr>
<tr>
<td></td>
<td>320</td>
</tr>
<tr>
<td></td>
<td>550</td>
</tr>
<tr>
<td></td>
<td>240</td>
</tr>
</tbody>
</table>
Evaluation Hardware – IOPS (4KB)

<table>
<thead>
<tr>
<th></th>
<th>Sequential READ (KIOPS)</th>
<th>Sequential WRITE (KIOPS)</th>
<th>Random READ (KIOPS)</th>
<th>Random WRITE (KIOPS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software</td>
<td>9.7</td>
<td>5.4</td>
<td>5.5</td>
<td>4.8</td>
</tr>
<tr>
<td>Hardware</td>
<td>12</td>
<td>10</td>
<td>13</td>
<td>7.8</td>
</tr>
</tbody>
</table>
Conclusion

- Performance Gain (Speedups) Throughput:
  - **1.2x** & **1.9x** for Rand writes (128KB) & Seq Writes (4KB) resp.

- Performance Gain (Speedups) IOPS:
  - **2.36x** for 4KB Rand reads (4KB)

- Through DeLiBA initial goal of easy *programmability* achieved
Future Work – DeLiBA SmartNIC

Stage 1: Client Application launch requests
Stage 2: Network Block Device
Stage 3: Block Layer Libraries + Network Library APIs
Stage 4: Hardware/Software Interface (FPGA)
Stage 6: Block I/O placement on remote server
Future Work – DeLiBA MQ Driver
DeLiBA is open-source

- DeLiBA is available at our ESA github:
  
  https://github.com/esa-tu-darmstadt/deliba

QR code for our DeLiBA repo
References


THANKS FOR YOUR ATTENTION!

....... looking forward to interesting discussions in the **FPGA Design** panel of FPL 2022