DCS-ctrl: A Fast and Flexible Device-Control Mechanism for Device-Centric Server Architecture

Dongup Kwon¹, Jaehyung Ahn², Dongju Chae², Mohammadamin Ajdari², Jaewon Lee¹, Suheon Bae¹, Youngsok Kim¹, and Jangwoo Kim¹

¹Dept. of Electrical and Computer Engineering, Seoul National University
²Dept. of Computer Science and Engineering, POSTECH
Conventional Server Architecture

- Primarily rely on “CPU and memory”
  - CPU-centric computing & in-memory storage
  - **Slow** and **low-bandwidth** peripheral devices
Conventional Server Architecture

- **Primarily rely on “CPU and memory”**
  - CPU-centric computing & in-memory storage
  - *Slow* and *low-bandwidth* peripheral devices
Device-centric Server Architecture

- Exploit “fast & high-bandwidth devices”
  - Data processing **accelerators** (e.g., GPU, FPGA)
  - **Storage** (e.g., SSD), **network** (e.g., 100GbE), **PCIe** Gen3

![Diagram showing device-centric architecture](image)
Index

• Existing approaches
  • DCS-ctrl: HW-based device-control mechanism
  • Experimental results
• Conclusion
Existing Approaches

• **Software optimization**
  - Memory mgmt. optimization, user-level device interface
  - *Do not address multi-device tasks*

• **P2P communication**
  - Transfer data directly through PCI Express ➔ **D2D comm.**

• **Device integration**
  - Integrate heterogeneous devices ➔ **D2D comm.**
Limitations of Existing D2D Comm.

- **P2P communication**
  - Direct data transfers through PCI Express → D2D comm.
  - Slow and high-overhead control path
Limitations of Existing D2D Comm.

- **Integrated devices**
  - Integrating heterogeneous devices \(\rightarrow\) D2D comm.
  - Fast data & control transfers
  - Fixed and inflexible aggregate implementation
Limited Performance Potential

```
while (true) {
    rc_recv = recv(fd_sock, buffer, recv_size, 0);
    if (rc_recv <= 0) break;
    processing(&md_ctx, buffer, recv_size);
    rc_write = write(fd_file, buffer, recv_size);
    ...
}
```

- “Intermediate” processing between device ops
  - Prevent applications from using direct D2D comm.
  - Cause host-side resource contention (CPU and memory)
Design Goals

• **Performance & scalability**
  - **Faster** inter-device data & control communication
  - **More scalable** with CPU-efficient device operations

• **Flexibility**
  - Support any types of **off-the-shelf devices**

• **Applicability**
  - **Increase the opportunity** of applying D2D comm.
Index

• Existing approaches

• DCS-ctrl: HW-based device-control mechanism
  – Key ideas and benefits
  – Architecture

• Experimental results

• Conclusion
DCS-ctrl: Key Ideas & Benefits

- **DCS-ctrl: PCIe P2P + “HDC”**
  - **Hardware-based device-control** (HDC) mechanism
  - **HDC Engine**: “FPGA-based” device orchestrator + “near-device” processing unit
    - Performance & scalability ➔ HDC, device orchestrator
    - Flexibility ➔ FPGA-based, low-cost device controller
    - Applicability ➔ near-device processing unit
HDC Engine: Overview

**SW-controlled P2P**

Application

Dev A \rightarrow Dev B \rightarrow Dev C

Device driver A \rightarrow Device driver B \rightarrow Device driver C

Dev A \rightarrow Dev B \rightarrow Dev C

**DCS-ctrl (HW)**

Application

HDC Engine (FPGA)

Dev A \rightarrow Dev B \rightarrow Dev C

Device ctrl A \rightarrow Device ctrl B \rightarrow Device ctrl C

Dev A \rightarrow Dev B \rightarrow Dev C
DCS-ctrl: Key Ideas & Benefits

Optimized dev. control ⇒ Faster & scalable communication
Generic dev. interfaces ⇒ Higher flexibility
Near-device processing ⇒ Higher applicability

```c
void ssd_to_nic()
{
    get_from_ssd(&data);
    process_in_HDC(&data);
    write_to_nic(&data);
}
```
Key Idea #1: Device Orchestrator

- Perform multi-device tasks w/o CPU involvement
  - Offload a multi-device task to HDC Engine
  - Manage all device operations and their dependencies

Fast hardware-level device control

Scoreboard

<table>
<thead>
<tr>
<th>Dev</th>
<th>R/W</th>
<th>Src</th>
<th>Dst</th>
<th>Aux</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Read</td>
<td>Addr(DevA)</td>
<td>Addr(NDP-A)</td>
<td>-</td>
<td>Done</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>Addr(NDP-A)</td>
<td>Addr(NDP-B)</td>
<td>Hash</td>
<td>Issue</td>
</tr>
<tr>
<td>B</td>
<td>Write</td>
<td>Addr(NDP-B)</td>
<td>Addr(DevB)</td>
<td>-</td>
<td>Ready</td>
</tr>
</tbody>
</table>
Key Idea #2: Device Controller

- Provide interfaces between HDC Engine & devices
  - Include submission & completion queues
  - Build standard & vendor-specific device commands

Flexible & low-cost device control
**Key Idea #3: Near-device Processing**

- **Near-device processing units**
  - Execute intermediate processing between device ops
  - Scale-out storage app ➔ hash, encryption, compression

<table>
<thead>
<tr>
<th>Processing units</th>
<th>LUTs</th>
<th>Registers</th>
<th>Applications</th>
</tr>
</thead>
<tbody>
<tr>
<td>MD5</td>
<td>3.0%</td>
<td>0.69%</td>
<td>Swift</td>
</tr>
<tr>
<td>AES256</td>
<td>3.52%</td>
<td>0.99%</td>
<td>HDFS, Swift</td>
</tr>
<tr>
<td>GZIP</td>
<td>5.36%</td>
<td>2.09%</td>
<td>HDFS</td>
</tr>
</tbody>
</table>

Highly applicable to existing applications
Index

• Existing approaches

• DCS-ctrl: HW-based device-control mechanism
  - Key idea and benefits
    - Architecture

• Experimental results

• Conclusion
Baseline Architecture

- **Software-controlled P2P**
  - P2P comm. + *indirect device-control path*
DCS-ctrl: HW-based Device Control (1/3)

- **Offload device-control path to HDC Engine**
  - **Scoreboard**: schedule device operations in a multi-dev task
DCS-ctrl: Low-cost Integration (2/3)

- **Implement an FPGA-based device controller**
  - **Device controller:** directly control devices using P2P
DCS-ctrl: Near-device Processing (3/3)

- **Provide units for intermediate processing**
  - **NDP unit**: perform data processing on a data path

![Diagram of near-device processing](image)
DCS-ctrl Prototype

HDC Engine implemented on Xilinx Virtex-7 VC707

Supports off-the-shelf devices – Intel 750 SSDs, Broadcom 10Gbe NICs, NVIDIA GPUs
Index

• Existing approaches
• DCS-ctrl: HW-based device-control mechanism
• Experimental results
• Conclusion
Reducing Device Control Latency

- `encrypted_sendfile()`: SSD $\rightarrow$ hash $\rightarrow$ NIC
  - SW opt (+P2P): frequent boundary crossings, complex software
  - DCS-ctrl: less crossings, hardware-based device control

**without processing**

**with processing (AES256)**
Reducing CPU Utilization

- **Swift & HDFS workloads**
  - Offload device control & data transfers to hardware

**Swift**

- Kernel (GET)
- Kernel (PUT)
- GPU control
- Others

**HDFS**

- Kernel (Sender)
- Kernel (Receiver)
- GPU control
- Others

Normalized CPU utilization

<table>
<thead>
<tr>
<th>SW opt</th>
<th>SW opt + P2P</th>
<th>DCS-ctrl</th>
</tr>
</thead>
<tbody>
<tr>
<td>50%</td>
<td>25%</td>
<td>0%</td>
</tr>
</tbody>
</table>

Normalized CPU utilization

<table>
<thead>
<tr>
<th>Send SW opt</th>
<th>Send SW opt + P2P</th>
<th>Send DCS-ctrl</th>
</tr>
</thead>
<tbody>
<tr>
<td>52%</td>
<td>49%</td>
<td></td>
</tr>
</tbody>
</table>
Scalability: More Devices

- **Swift & HDFS workloads**
  - More CPU-efficient \(\rightarrow\) support more high-performance devices

![CPU utilization diagrams for Swift and HDFS](image)

**Swift**

- SW opt
- SW opt + P2P
- DCS-ctrl

**HDFS**

- SW opt
- SW opt + P2P
- DCS-ctrl
Conclusion

- **Fast & flexible device-control mechanism**
  - **Hardware-based device-control** (HDC) mechanism
  - **FPGA-based** standard device controllers
  - **Near-device data processing** (NDP) units

- **Real hardware prototype evaluation**
  - **72%** faster inter-device communication
  - **50%** lower CPU utilization for Swift & HDFS
Thank you!

We will release our IP & tools soon!