Alluxio - CAPI Flash Integration for Big Data Frameworks

Kavana N Bhat & Shajith Chandran
Power Systems, IBM
Big Data Ecosystem Today
Alluxio (formerly Tachyon)

- Memory Speed Virtual Distributed Storage
- Enables Virtualized Data Across Multiple Types of Storage
Why use Alluxio?

Memory-speed data sharing across jobs and frameworks

Data survive in memory after computation crashes

Off-heap storage, no GC
Storage Tier Hierarchy

- Faster
- Higher Capacity

MEM

SSD

HDD
Automatic Data Migration

- Data can be evicted to lower layers if it is “cooling down”
- Data can be promoted to upper layers if it is “warming up”
CAPI: Coherent Accelerator Processor Interface
CAPI vs. I/O Device Driver: Data Prep

Typical I/O Model Flow: Total ~13µs for data prep

Flow with a Coherent Model: Total 0.36µs
Alluxio – CAPI Integration Choices

- Modify Alluxio to use the new user-level CAPI Flash block APIs
  - Requires changes to Alluxio
  - Alluxio-specific Implementation
  - Non-reusable

- Create a User-Space Filesystem for CAPI Flash
  - Generic Implementation
  - Provides standard interfaces for file operations - No changes to Alluxio
CAPI Userspace Filesystem (USFS)

- The userspace filesystem is primarily contained in a library(libcflsh_usfs), that runs on top of the CAPI flash block library API.

- Provides analogs of all major filesystem APIs (open, close, read, write, aio_read, aio_write etc).

- Provides wrapper layer that allows applications to work without modifications with use of the library preload feature.
  - Wrapper intercepts filesystem calls from libc intended for libcflsh_usfs and routes them internally.

Library Preload for existing Application POCs

```
Application
Libc
Kernel FS (ext4fs)
Kernel Storage Drivers

Traditional Disk with Legacy FS (ext4fs etc)

Same Unmodified Application

Application
Libcflsh_usfs Libc

Application
Libcflsh_black

CAPI Flash Disk USFS
```
Alluxio with CAPI USFS

User Space

Kernel Space

PHYP/Hardware

CAPI Flash Adapter

Alluxio with Traditional IO

VFS

Filesystem

Disk Driver

CAPI Flash Block Library

User space FS

Alluxio with CAPI

CAPI Flash Adapter

© 2017 IBM Corporation
Alluxio Performance with CAPI

Alluxio with Traditional FS on Flash
Alluxio Performance with CAPI

Alluxio with CAPI USFS on Flash
References

- Alluxio Project: www.alluxio.org
- IBM CAPI: http://ibm.biz/powercapi
- IBM CAPI User Space Block Library: https://github.com/open-power/capiflash
- What is CAPI?
  - For Partners: http://www.ibm.com/services/weblectures/dlv/partnerworld/ltu48850
- SuperVessel Website: https://ptopenlab.com(cloudlabconsole/#/
- Contact: kavana.bhat@in.ibm.com, shajithchandran@in.ibm.com
Questions?
Thank You
Prior to CAPI, an application called a device driver to utilize an FPGA Accelerator. The device driver performed a memory mapping operation.

3 versions of the data (not coherent).
1000s of instructions in the device driver.
With CAPI, the FPGA shares memory with the cores

Memory Subsystem

1 coherent version of the data. No device driver call/instructions.
Typical I/O Model Flow: Total ~13µs for data prep

Flow with a Coherent Model: Total 0.36µs
CAPI Works

Acceleration Portion: Data or Compute Intensive, Storage or External I/O

Sharing the same memory space
Accelerator is a peer to POWER8 Core

Application Portion: Data Set-up, Control

POWER8 Processor
CAPI technology connections

- Proprietary hardware to enable coherent acceleration
- Operating system enablement
  - Ubuntu LE
  - Libcxl function calls
- Customer application and accelerator
  - Application sets up data and calls the accelerator functional unit (AFU)
  - AFU reads and writes coherent data across the PCIe and communicates with the application
    - PSL cache holds coherent data for quick AFU access
CAPI solution flow

1. Connect to accelerator
   - OS
   - IBM Supplied PSL
   - AFU
   - App

2. Set Work Element Descriptor (WED) at AddrX – may contain addresses of other data structures
   - AFU reservation for work
   - Understands WED content - and any other addressed data structures

3. Start accelerator
   - AFU continues to work using this interface
   - AFU fetches AddrX (the WED) starts operation

4. If required, App can read or write AFU registers
   - AFU continues to work using this interface

5. App knows AFU is finished (Mechanism is user defined)
   - App can start again from top or free AFU

6. AFU finishes (Mechanism is user defined)
   - De-assert RUNNING
   - Assert DONE
Infrastructure Consolidation

Before: NoSQL in memory (x86)

- Large Distributed (Scale out)
- Large Memory per node
- Networking Bandwidth Needs
- Load Balancing

24U

24:1 physical server consolidation = 6x less rack space

24:1 infrastructure consolidation

Upto 3x cost savings

6x less rack space 2U server + 2U FlashSystem vs. typical deployment

IBM Data Engine for NoSQL

After: NoSQL POWER8 + CAPI Flash

- 10Gb Uplink
- POWER8 Server
- Flash Array w/ up to 40TB

WWW

Backup Nodes

500GB Cache Node

Load Balancer

WWW

Infrastructure Requirements

- Large Distributed (Scale out)
- Large Memory per node
- Networking Bandwidth Needs
- Load Balancing

Acceptable latency

Memory

CAPI

Conventional PCIe I/O

network

network

network

© 2017 IBM Corporation