Computational Storage APIs

Oscar P Pinto, Principal Engineer
Samsung Semiconductor Inc.
Agenda

- Overview
- Introducing CS APIs
- API Usage by Example
- Example in Code
- Advanced Topics
- Summary
Computational Storage

Why an API Library?
Why Computational Storage?

- Data is being created at an exponential rate
- Storage has also grown to account for this growth

- NVMe SSDs provide better performance than ever before
  - But their bandwidth not fully utilized by Host
- General purpose CPUs not able to fully tap this bandwidth
  - Scaling limited by PCIe lanes
- SSDs have more internal bandwidth than utilized

- Computational Storage & Offloads tap into this
  - Process data near storage
  - Add compute to storage
CS API Library Overview

- One Set of APIs across all CSx types
  - CSP, CSD, CSA
  - Common set of APIs for different CS devices
- One interface to different device and connectivity choices
  - Hardware ASIC, CPU, FPGA, etc
  - NVMe/NVMe-oF, PCIe, custom, etc
- Configurations may be local/remote attached
- Hides vendor specific implementation details below library
- Abstracts device specific details
- APIs to be OS agnostic

Computational Storage Processor (CSP)
Computational Storage Drive (CSD)
Computational Storage Array (CSA)
About API Library

- Uniform interface for multiple configurations
  - APIs provided in common library
- Each CSx managed through its own device stack
  - Library may interface with additional plugins based on implementation requirements
  - Plugins help connect CSx to abstracted CS interfaces
- Extensible Interface
- CS APIs abstract
  - Discovery
  - Device Access
  - Device Memory (mapped/unmapped)
  - Near Storage Access
  - Copy Device Memory
  - Download CSFs
  - Execute CSFs
  - Device Management
API Requirements

- One interface across CS devices
  - CSDs, CSPs, CSAs
- Discovery
- Access
- Configure
- Device Memory Allocation
- Data Movement
  - Input: Host memory, Storage, Device memory
  - Output: Device memory, Storage, Host memory
- Execute
- Device Management
- Queued I/O Requests
- Transparent Local/Remote usages
- Security
How to use Computational Storage
Usage by example
Applying Computational Storage

1. Input data does not get transferred to Host DRAM

2. Application

3. CS API Library
Computational Storage API Details
Discovery & Access APIs

Discovery
- Discover CSx devices
  - By device path, file/directory path or all
  - Discover CSFs by requirement

Access
- Access CSx once discovered for CS usage
- Get access to a specific CSF for execution

CS_STATUS csGetCSxFromPath(char *Path, unsigned int *Length, char *DevName);

CS_STATUS csQueryFunctionList(char *Path, unsigned int *Length, char *Buffer);

CS_STATUS csOpenCSx(char *DevName, void *DevContext, CS_DEV_HANDLE *DevHandle);

CS_STATUS csCloseCSx(CS_DEV_HANDLE DevHandle);

CS_STATUS csGetFunction(CS_DEV_HANDLE DevHandle, char *Name, void *Context, CS_FUNCTION_ID *FunctionId);
Device Memory

CS_STATUS csAllocMem(CS_DEV_HANDLE DevHandle, int Bytes, unsigned int MemFlags, CS_MEM_HANDLE *MemHandle, CS_MEM_PTR *VAddressPtr);
CS_STATUS csFreeMem(CS_MEM_HANDLE MemHandle);

- Allocate / Deallocate Device Memory
- Manage Device Memory
  - Memory scheme
    - Memory mapped (PCIe BAR)
    - Opaque
  - Memory organization
    - Host managed
    - Device managed
- Mapping of memory to application space depends on the device
- Transparent to fabric usages
- Returns memory handle
  - Virtual address pointer when applicable
### Storage I/O

- Initiate direct internal transfers between storage (SSD) and allocated Device Memory
  - Seamlessly manages mapped/unmapped device memory
  - P2P transfers through file system if device supports memory mapped P2P BAR
- Single interface to support block & file; extensible
- Transparent to fabric usages

- Follows common completion modes
  - Synchronous
  - Asynchronous callback
  - Asynchronous event

```c
typedef struct {
    enum CS_STORAGE_REQ_MODE Mode;
    CS_DEV_HANDLE DevHandle;
    union {
        csBlockIo BlockIo;
        csFileIo FileIo;
    } u;
} csStorageRequest;

CS_STATUS csQueueStorageRequest(csStorageRequest *Req, void *Context, csQueueCallbackFn CallbackFn, CS_EVT_HANDLE EventHandle, u32 *CompValue);
```

Some modes not available in all configurations.
Compute

- Initiate execution of a CSF with its input and output parameters
- API extensible for parameters
- Transparent to fabric usages

- Follows common completion modes
  - Synchronous
  - Asynchronous callback
  - Asynchronous event

```c
typedef struct {
    CS_DEV_HANDLE DevHandle;
    CS_FUNCTION_ID FunctionId;
    int NumArgs;
    CsComputeArg Args[1];
} csComputeRequest;

CS_STATUS csQueueComputeRequest(csComputeRequest *Req, void *Context, csQueueCallbackFn CallbackFn,
                                  CS_EVT_HANDLE EventHandle, u32 *CompValue);
```
Copy Device Memory

- Transfer data between Host memory and allocated Device Memory
- Single interface for transfer operations
  - Transparent to fabric usages
- Follows common completion modes

- Common completion modes
  - Synchronous
  - Asynchronous callback
  - Asynchronous event

Some modes not available in all configurations
Coding the Example

Applying APIs to example
APIs Required for Example

1. Discover CSx & CSF
2. Allocate Device Memory
3. Queue Storage Request
4. Queue Compute Request
5. Queue Copy Memory Request
# include <cs.h>

```c
int cs_decode(char *file_path, int fd, void *decode_buf)
{
    // discover my CS device (CSx) and CSF
    length = sizeof(csxBuffer);
    status = csGetCSxFromPath(file_path, &length, &csxBuffer);
    status = csOpenCSx(csxBuffer, &MyDevContext, &devHandle);
    status = csGetFunction(devHandle, myFunction, NULL, &functId);
    // allocate device memory for input and output buffers
    status = csAllocMem(devHandle, CHUNK_SIZE, 0, &inMemHandle, NULL);
    status = csAllocMem(devHandle, CHUNK_SIZE, 0, &outMemHandle, NULL);
    // allocate storage request & read chunk size data from file handle fd
    storReq = calloc(1, sizeof(CsStorageRequest));
    if (!storReq) { ERROR_OUT("memory alloc error\n"); }
    storReq->Mode = CS_STORAGE_FILE_IO;
    storReq->DevHandle = devHandle;
    storReq->u.CsFileIo.Type = CS_STORAGE_LOAD_TYPE;
    storReq->u.CsFileIo.FileHandle = fd;
    storReq->u.CsFileIo.Offset = 0;
    storReq->u.CsFileIo.Bytes = CHUNK_SIZE;
    storReq->u.CsFileIo.DevMem.MemHandle = inMemHandle;
    storReq->u.CsFileIo.DevMem.ByteOffset = 0;
    status = csQueueStorageRequest(storReq, storReq, NULL, NULL, NULL);
    // allocate compute request for 3 args & issue compute request
    compReq = calloc(1, sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
    if (!compReq) { ERROR_OUT("memory alloc error\n"); }
    compReq->DevHandle = devHandle;
    compReq->FunctionId = functId;
    compReq->NumArgs = 3;
    argPtr = &compReq->Args[0];
    csHelperSetComputeArg(argPtr[0], CS_AFDM_TYPE, inMemHandle, 0);
    csHelperSetComputeArg(argPtr[1], CS_32BIT_VALUE_TYPE, CHUNK_SIZE);
    csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, outMemHandle, 0);
    status = csQueueComputeRequest(compReq, NULL, NULL, NULL, NULL);
    // allocate copy request & copy results to host buffer
    copyReq = calloc(1, sizeof(CsCopyMemRequest));
    if (!copyReq) { ERROR_OUT("memory alloc error\n"); }
    copyReq->Type = CS_COPY_FROM_DEVICE;
    copyReq->HostVAddress = decode_buf;
    copyReq->DevMem.MemHandle = outMemHandle;
    copyReq->DevMem.ByteOffset = 0;
    copyReq->Bytes = CHUNK_SIZE;
    status = csQueueCopyMemRequest(copyReq, NULL, NULL, NULL, NULL);
    return 0;
}
```

*API return status values are not shown to check for success and errors to ease readability
Other APIs

What else can the APIs do?
Other Interfaces

- `csQueueBatchRequest()`
- `csDownload()`
- `csRegisterPlugin()`
- `csQueryDeviceProperties()`
- `csQueryDeviceStatistics()`
- `csConfig()`
- `csSetDeviceCapability()`
Call for Action

- **Other sessions on Computational Storage**
  - Samsung Keynote – Yang Seok Ki
  - Moving forward with an Architecture & API – Bill Martin
  - Computational Storage Update from SNIA WG – Scott Shadley & Jason Molgaard
  - NVMe Computational Storage Update – Kim Malone & Stephen Bates

- **Join the standardization efforts**
  - SNIA, NVMe

- **Help build the ecosystem**
Thank You
Please take a moment to rate this session.

Your feedback is important to us.