SNIA COMPUTE + MEMORY + STORAGE SUMMIT

Architectures, Solutions, and Community
VIRTUAL EVENT, APRIL 11-12, 2023

SNIA CS APIs

A Programming Guide

Presented by Oscar P Pinto

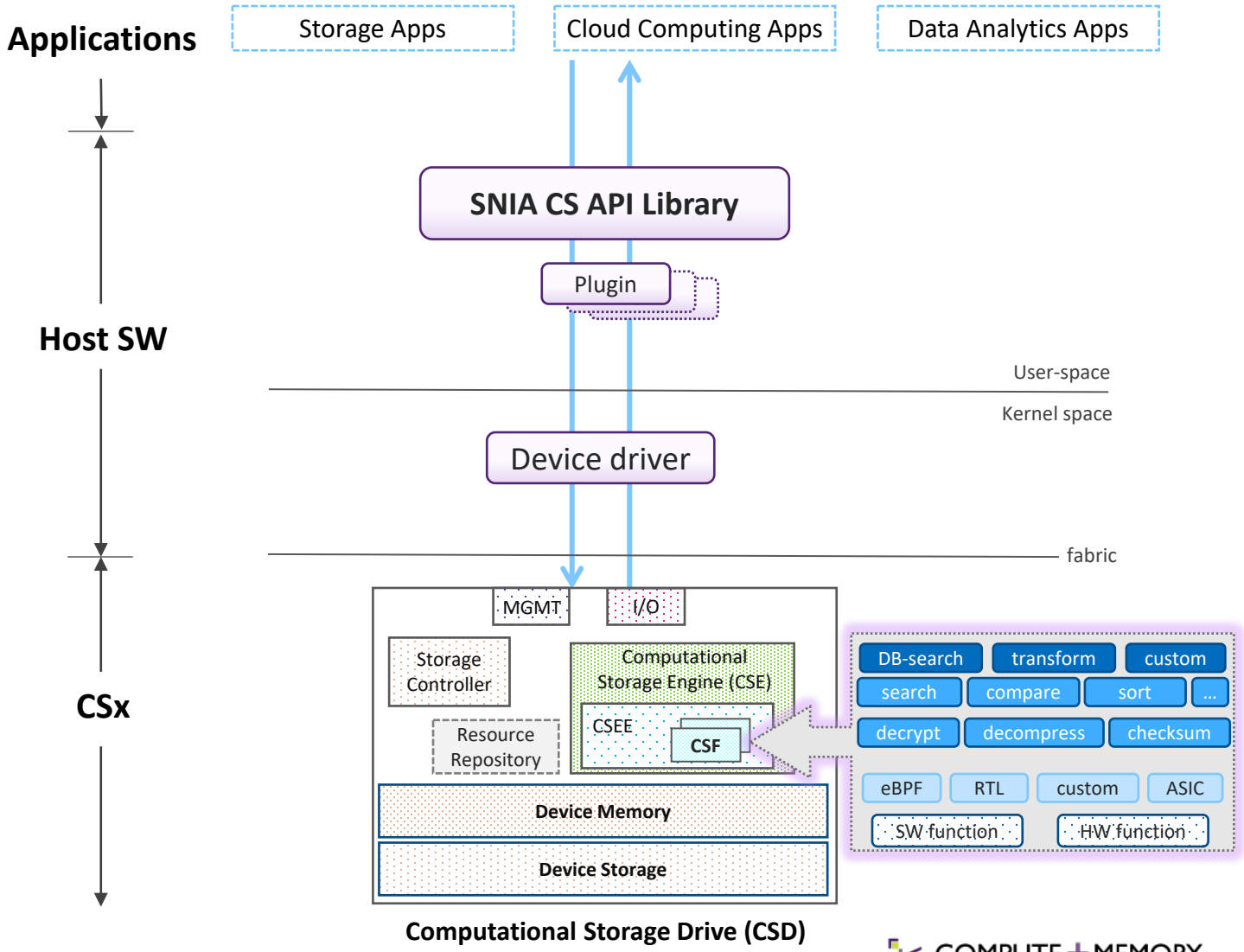
Principal Engineer, Samsung Semiconductor Inc

Agenda

- SNIA CS APIs
- Programming Usage
- How to Apply
  - Using an Example
  - Step by Step
- Progress Update
- Summary

SNIA Computational Storage APIs

1. One set for all CSx types
   - CSP, CSD, CSA
2. Hides Device Details
   - Hardware, Connectivity (local/remote)
   - Vendor specific Implementations
3. Abstracts Device Interface
   - Discovery
   - Access
   - Device Memory (mapped/unmapped)
   - Near Storage Access
   - Copy Device Memory
   - Download CSFs
   - Execute CSFs
   - Device Management
4. Extensible Interface
   - Plugins connect CSx to abstracted APIs
5. OS Agnostic



Programming Usage

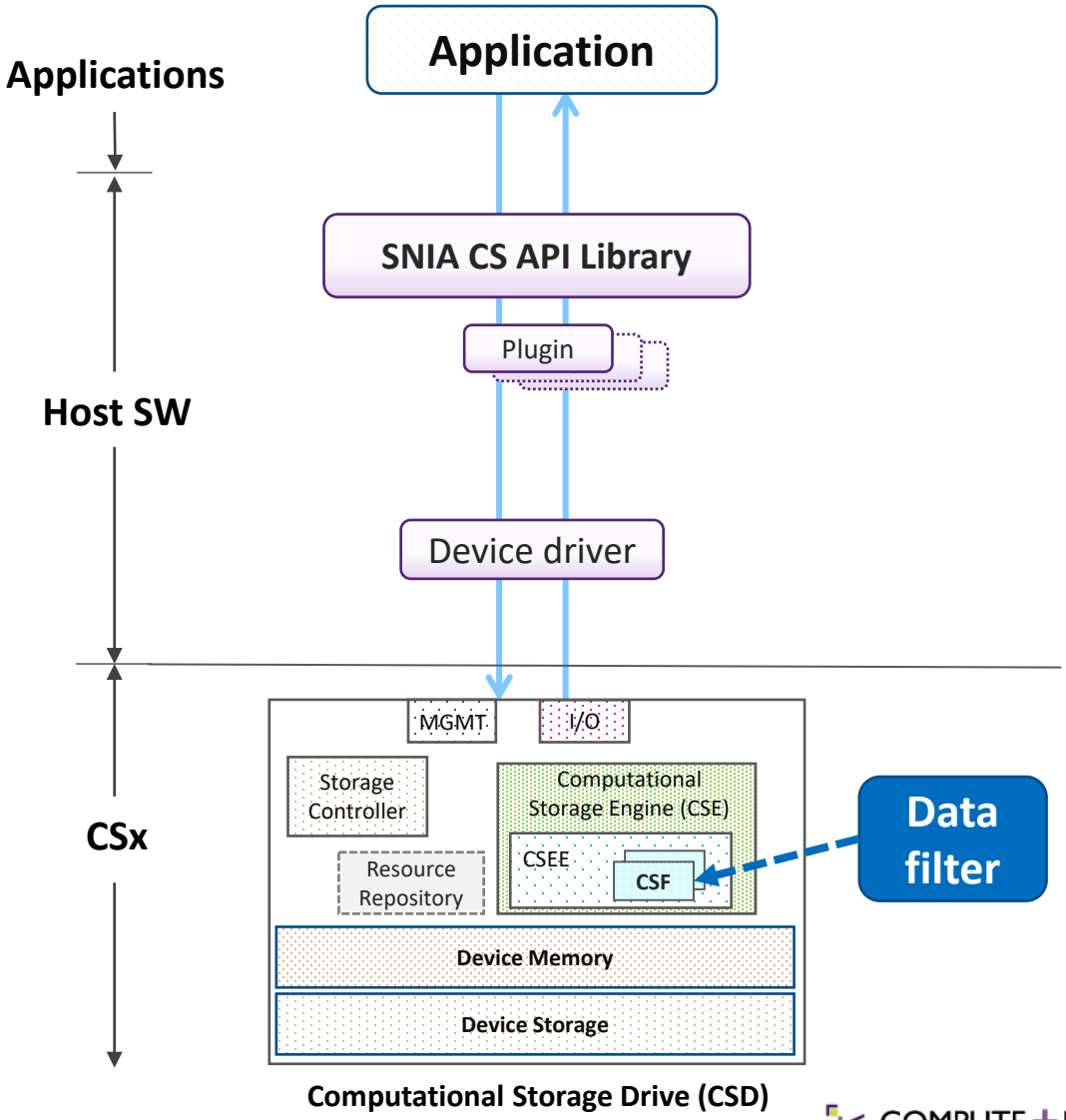
- 2 Programming Modes
  - Privileged Mode Operations (Administrator)
    - Configure Individual CS Resources
    - Download CSFs
    - Manage Device
  - Non-privileged Mode Operations (Normal)
    - Discover CSx, CSFs
    - Allocate Device Memory
    - Execute CSFs
    - Transfer Storage Data to/from Device Memory
    - Copy Device Memory to/from Host Memory

Using CS APIs

Non-privileged Users

Example

- Execute Data Filter CSF
  - Allocate Device Memory (FDM)
  - Load Data from Storage
  - Run Data Filter CSF on loaded Data
  - Copy Results to Host Memory



CSF – Computational Storage Function
FDM – Function Device Memory

Using CS APIs

- Simplified API set
  - Just Setup & Run Computational Storage
- Minimal Steps
  - 3 steps to Setup
  - 3 steps to perform Computational Storage I/O
- Apply Example

Prepare for Computational Storage (Setup)

Discover CSx

1

- Discover by name
- Access device

⇨

Discover CSF

2

- Discover the function(s) you want to execute

⇨

Allocate FDM

3

- Allocate Device Memory

⇨

Ready

CSx – Computational Storage Device

CS APIs: Discover CSx

step **1** **Discover CSx**

- Discover by name/storage entity
- Access device

Find CSx near storage

```
// Find my CSx near storage
status = csGetCSxFromPath(“my_file_path", &length, &csxBuffer);
if (status != CS_SUCCESS)
    ERROR_OUT("No CSx device found!\n");
// open device, init function and prealloc buffers
status = csOpenCSx(csxBuffer, &MyDevContext, &devHandle);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not access CSx\n");
```

Access CSx

CS APIs: Discover CSF

step **2**

Discover CSF

- Discover your CSF
- Pick the best one if > 1 available

Find CSF by name

Pick most performant in list

```
// Get access to the CSF to run
status = csGetCSFId(devHandle, “filter”, &infoLength, &count, &csfInfo);
if (status != CS_SUCCESS)
    ERROR_OUT("CSX does not contain any decrypt CSFs \n");
// pick highest performant CSF
CSFIdInfo *p = csfInfo;
CSFIdInfo *myCSF = NULL;
for (i=0; i< count; i++, p++) {
    if ((myCSF == NULL) ||
        ((myCSF != NULL) && (p->RelativePerformance > myCSF->RelativePerformance))) {
        myCSF = p;
    }
}
printf(“CSFId: %d, RelativePerformance: %d\n”, myCSF->CSFId, myCSF->RelativePerformance);
```


CS APIs: Allocate FDM

step **3**

Allocate FDM

- Pick from list if > 1 available
- Allocate FDM as necessary

```
// Pick most performant FDM from CSFIdInfo
FDMAccess *p = myCSF->FDMList;
FDMAccess *myFDM = NULL;
for (i = 0; i < myCSF->NumFDMs; i++, p++) {
    if ((myFDM == NULL) ||
        ((myFDM != NULL) && (p->RelativePerformance > myFDM->RelativePerformance))) {
        myFDM = p;
    }
}
// allocate FDM for CSF usage
CsMemFlags f;
f.s->FDMId = myFDM->FDMId;
f.s->Flags = 0;   // may also be CS_FDM_CLEAR
status = csAllocMem(devHandle, CHUNK_SIZE, &f, &afdmHandle1, NULL);
if (status != CS_SUCCESS)
    ERROR_OUT("AFDM alloc error\n");
```

Pick most performant FDM in list if > 1

Allocate desired size & as required

Perform Computational Storage I/O (Run)

Load Storage Data

- Load Data near Compute

Execute CSF

- Run Compute Operation

Copy Results

- Copy from FDM into Host Memory

Done

CS APIs: Load Storage Data

step 4

Load Storage Data

- Load data from storage into FDM
- Data does not leave the device

```
// Populate storage request with data from file
csStorageRequest storReq = malloc(sizeof(CsStorageRequest));
if (!storReq) { ERROR_OUT(“not enough memory!\n"); }

storReq->Mode = CS_STORAGE_FILE_IO;
storReq->DevHandle = devHandle;
storReq->u.CsFileIo.Type = CS_STORAGE_LOAD_TYPE;
storReq->u.CsFileIo.FileHandle = fd;        // file fd to access for data
storReq->u.CsFileIo.Offset = 0;             // data offset within file
storReq->u.CsFileIo.Bytes = CHUNK_SIZE;
storReq->u.CsFileIo.DevMem.MemHandle = afdmHandle1;
storReq->u.CsFileIo.DevMem.ByteOffset = 0;
status = csQueueStorageRequest(storReq, storReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not load storage data\n");
```

Event for completion

Callback for completion

Synchronous: NULL, NULL
Asynchronous: Callback / Event

CS APIs: Execute CSF

step 5

Execute CSF

- Run compute on loaded data

```
// Populate compute request on data loaded from file
CsComputeRequest compReq = malloc(sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }

compReq->CSFId = myCSF->CSFId;              // filter function id
compReq->NumArgs = 3;                       // takes 3 arguments

CsComputeArg argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, afdmHandle1, 0);        // input buffer
csHelperSetComputeArg(&argPtr[1], CS_32BIT_VALUE_TYPE, CHUNK_SIZE);     // length of input
csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, afdmHandle2, 0);        // output buffer
status = csQueueComputeRequest(compReq, compReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT("Error in CSF execution\n");
```

Event for completion

Callback for completion

Synchronous: NULL, NULL
Asynchronous: Callback / Event

CS APIs: Copy Results

step 6

Copy FDM contents to Host

- Copy Data from Device Memory to Host

```
// Populate copy request for results data
CsCopyMemRequest copyReq = malloc(sizeof(CsCopyMemRequest));
if (!copyReq) { ERROR_OUT("memory alloc error\n"); }

copyReq->Type = CS_COPY_FROM_DEVICE;
copyReq->u.HostVAddress = results_buf;
copyReq->DevMem.MemHandle = afdmHandle2;
copyReq->DevMem.ByteOffset = 0;
copyReq->Bytes = CHUNK_SIZE;
status = csQueueCopyMemRequest(copyReq, copyReq, NULL, NULL, NULL, NULL);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not copy data from FDM\n");
```

Event for completion

Callback for completion

Synchronous: NULL, NULL
Asynchronous: Callback / Event

Usage Summary

6 Steps

1
```
csGetCSxFromPath(“my_file_path", &length, &csxBuffer);
csOpenCSx(csxBuffer, &MyDevContext, &devHandle);
```

2
```
csGetCSFId(devHandle, “filter”, &infoLength, &count, &csfInfo);
```

3
```
csAllocMem(devHandle, CHUNK_SIZE, &f, &afdmHandle1, NULL);
```

4
```
csQueueStorageRequest(storReq, storReq, NULL, NULL, NULL, &compVal);
```

5
```
csQueueComputeRequest(compReq, compReq, NULL, NULL, NULL, &compVal);
```

6
```
csQueueCopyMemRequest(copyReq, copyReq, NULL, NULL, NULL, NULL);
```


SNIA COMPUTE + MEMORY + STORAGE SUMMIT

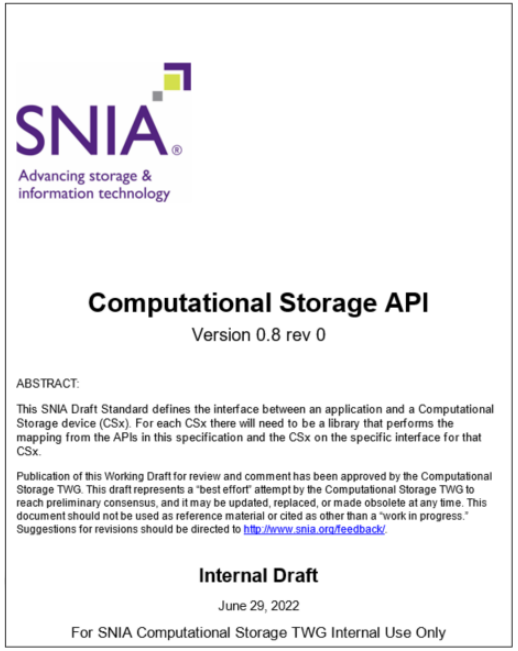
Architectures, Solutions, and Community
VIRTUAL EVENT, APRIL 11-12, 2023

Progress

Specification Progress

- Extend Storage Request to include LBA ranges
- Device Memory
  - Device Memory Pools and Compute Proximity
  - Initialization Options
- Cancelling I/O, Abort & Reset
- Compute Functions
  - Handling, Global Identifiers
- Mapping to NVMe

- SNIA CS API v0.8 Specification
  - Public review available


SNIA
Advancing storage & information technology

Computational Storage API

Version 0.8 rev 0

ABSTRACT:

This SNIA Draft Standard defines the interface between an application and a Computational Storage device (CSx). For each CSx there will need to be a library that performs the mapping from the APIs in this specification and the CSx on the specific interface for that CSx.

Publication of this Working Draft for review and comment has been approved by the Computational Storage TWG. This draft represents a "best effort" attempt by the Computational Storage TWG to reach preliminary consensus, and it may be updated, replaced, or made obsolete at any time. This document should not be used as reference material or cited as other than a "work in progress." Suggestions for revisions should be directed to http://www.snia.org/feedback/.

Internal Draft

June 29, 2022

For SNIA Computational Storage TWG Internal Use Only

Summary

Summary

- Simplified Programming Interface for Computational Storage
- APIs map to different device solutions
- Simple to follow and scalable
- Minimal steps to adopt

Additional Info

- Please attend other Computational Storage sessions
  - Standardizing Computational Storage *by Bill Martin & Jason Molgaard*
  - NVMe® Computational Storage *by Kim Malone & Bill Martin*
- Join the standardization efforts
  - SNIA, NVMe
- Help build the ecosystem

SNIA COMPUTE + MEMORY + STORAGE SUMMIT

Architectures, Solutions, and Community
VIRTUAL EVENT, APRIL 11-12, 2023

Please take a moment to rate this session.

Your feedback is important to us.

Backup

The Complete Code

```
// Find my CSx near storage
status = csGetCSxFromPath(“my_file_path", &length, &csxBuffer);
if (status != CS_SUCCESS)
    ERROR_OUT("No CSx device found!\n");
// open device, init function and prealloc buffers
status = csOpenCSx(csxBuffer, &MyDevContext, &devHandle);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not access CSx\n");

// Get access to the CSF to run
status = csGetCSFId(devHandle, “filter”, &infoLength, &count, &csfInfo);
if (status != CS_SUCCESS)
    ERROR_OUT("CSX does not contain any decrypt CSFs \n");
// pick highest performant CSF
CSFIdInfo *p = csfInfo;
CSFIdInfo *myCSF = NULL;
for (i=0; i< count; i++, p++) {
    if ((myCSF == NULL) ||
        ((myCSF != NULL) && (p->RelativePerformance > myCSF->RelativePerformance))) {
        myCSF = p;
    }
}

// Pick most performant FDM from CSFIdInfo
FDMAccess *p = myCSF->FDMList;
FDMAccess *myFDM = NULL;
for (i = 0; i < myCSF->NumFDMs; i++, p++) {
    if ((myFDM == NULL) ||
        ((myFDM != NULL) && (p->RelativePerformance > myFDM->RelativePerformance))) {
        myFDM = p;
    }
}
// allocate FDM for CSF usage
CsMemFlags f;
f.s->FDMId = myFDM->FDMId;
f.s->Flags = 0;   // may also be CS_FDM_CLEAR
status = csAllocMem(devHandle, CHUNK_SIZE, &f, &afdmHandle1, NULL);
if (status != CS_SUCCESS)
    ERROR_OUT("AFDM alloc error\n");
```

1 2 3

```
// Populate storage request with data from file
csStorageRequest storReq = malloc(sizeof(CsStorageRequest));
if (!storReq) { ERROR_OUT(“not enough memory!\n"); }
storReq->Mode = CS_STORAGE_FILE_IO;
storReq->DevHandle = devHandle;
storReq->u.CsFileIo.Type = CS_STORAGE_LOAD_TYPE;
storReq->u.CsFileIo.FileHandle = fd;                    // file fd to access for data
storReq->u.CsFileIo.Offset = 0;                         // data offset within file
storReq->u.CsFileIo.Bytes = CHUNK_SIZE;
storReq->u.CsFileIo.DevMem.MemHandle = afdmHandle1;
storReq->u.CsFileIo.DevMem.ByteOffset = 0;
status = csQueueStorageRequest(storReq, storReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT("Could not load storage data\n");

// Populate compute request on data loaded from file
CsComputeRequest compReq = malloc(sizeof(CsComputeRequest) + (sizeof(CsComputeArg) * 3));
if (!compReq) { ERROR_OUT("memory alloc error\n"); }
compReq->CSFId = myCSF->CSFId;                          // filter function id
compReq->NumArgs = 3;                                   // takes 3 arguments

CsComputeArg argPtr = &compReq->Args[0];
csHelperSetComputeArg(&argPtr[0], CS_AFDM_TYPE, afdmHandle1, 0);        // input buffer
csHelperSetComputeArg(&argPtr[1], CS_32BIT_VALUE_TYPE, CHUNK_SIZE);     // length of input
csHelperSetComputeArg(&argPtr[2], CS_AFDM_TYPE, afdmHandle2, 0);        // output buffer
status = csQueueComputeRequest(compReq, compReq, NULL, NULL, NULL, &compVal);
if (status != CS_SUCCESS)
    ERROR_OUT(“Error in CSF execution\n");

// Populate copy request for results data
CsCopyMemRequest copyReq = malloc(sizeof(CsCopyMemRequest));
if (!copyReq) { ERROR_OUT("memory alloc error\n"); }
copyReq->Type = CS_COPY_FROM_DEVICE;
copyReq->u.HostVAddress = results_buf;
copyReq->DevMem.MemHandle = afdmHandle2;
copyReq->DevMem.ByteOffset = 0;
copyReq->Bytes = CHUNK_SIZE;
status = csQueueCopyMemRequest(copyReq, copyReq, NULL, NULL, NULL, NULL);
if (status != CS_SUCCESS)
    ERROR_OUT(“Could not copy from FDM!\n);
```

4 5 6