Coherent Accelerator (CXL) Flash

Introduction

The IBM Power architecture provides support for CAPI (Coherent Accelerator Power Interface), which is available to certain PCIe slots on Power 8 systems. CAPI can be thought of as a special tunneling protocol through PCIe that allow PCIe adapters to look like special purpose co-processors which can read or write an application's memory and generate page faults. As a result, the host interface to an adapter running in CAPI mode does not require the data buffers to be mapped to the device's memory (IOMMU bypass) nor does it require memory to be pinned.

On Linux, Coherent Accelerator (CXL) kernel services present CAPI devices as a PCI device by implementing a virtual PCI host bridge. This abstraction simplifies the infrastructure and programming model, allowing for drivers to look similar to other native PCI device drivers.

CXL provides a mechanism by which user space applications can directly talk to a device (network or storage) bypassing the typical kernel/device driver stack. The CXL Flash Adapter Driver enables a user space application direct access to Flash storage.

The CXL Flash Adapter Driver is a kernel module that sits in the SCSI stack as a low level device driver (below the SCSI disk and protocol drivers) for the IBM CXL Flash Adapter. This driver is responsible for the initialization of the adapter, setting up the special path for user space access, and performing error recovery. It communicates directly the Flash Accelerator Functional Unit (AFU) as described in Documentation/powerpc/cxl.rst.

The cxlflash driver supports two, mutually exclusive, modes of operation at the device (LUN) level:

- Any flash device (LUN) can be configured to be accessed as a regular disk device (i.e.: /dev/sdc). This is the default mode.
- Any flash device (LUN) can be configured to be accessed from user space with a special block library. This mode further specifies the means of accessing the device and provides for either raw access to the entire LUN (referred to as direct or physical LUN access) or access to a kernel/AFU-mediated partition of the LUN (referred to as virtual LUN access). The segmentation of a disk device into virtual LUNs is assisted by special translation services provided by the Flash AFU.

Overview

The Coherent Accelerator Interface Architecture (CAIA) introduces a concept of a master context. A master typically has special privileges granted to it by the kernel or hypervisor allowing it to perform AFU wide management and control. The master may or may not be involved directly in each user I/O, but at the minimum is involved in the initial setup before the user application is allowed to send requests directly to the AFU.

The CXL Flash Adapter Driver establishes a master context with the AFU. It uses memory mapped I/O (MMIO) for this control and setup. The Adapter Problem Space Memory Map looks like this:

```
+-------------------------------+
|    512 * 64 KB User MMIO      |
|        (per context)          |
|       User Accessible         |
+-------------------------------+
|    512 * 128 B per context    |
|    Provisioning and Control   |
|   Trusted Process accessible  |
+-------------------------------+
|         64 KB Global          |
|   Trusted Process accessible  |
+-------------------------------+
```

This driver configures itself into the SCSI software stack as an adapter driver. The driver is the only entity that is considered a Trusted Process to program the Provisioning and Control and Global areas in the MMIO Space shown above. The master context driver discovers all LUNs attached to the CXL Flash adapter and instantiates scsi block devices (/dev/sdb, /dev/sdc etc.) for each unique LUN seen from each path.

Once these scsi block devices are instantiated, an application written to a specification provided by the block library may get access to the Flash from user space (without requiring a system call).

This master context driver also provides a series of ioctls for this block library to enable this user space access. The driver supports two modes for accessing the block device.

The first mode is called a virtual mode. In this mode a single scsi block device (/dev/sdb) may be carved up into any number of distinct virtual LUNs. The virtual LUNs may be resized as long as the sum of the sizes of all the virtual LUNs, along with the meta-data associated with it does not exceed the physical capacity.

The second mode is called the physical mode. In this mode a single block device (/dev/sdb) may be opened directly by the block library and the entire space for the LUN is available to the application.

Only the physical mode provides persistence of the data. i.e. The data written to the block device will survive application exit and restart and also reboot. The virtual LUNs do not persist (i.e. do not survive after the application terminates or the system reboots).

Block library API

Applications intending to get access to the CXL Flash from user space should use the block library, as it abstracts the details of interfacing directly with the cxlflash driver that are necessary for performing administrative actions (i.e.: setup, tear down, resize). The block library can be thought of as a 'user' of services, implemented as IOCTLs, that are provided by the cxlflash driver specifically for devices (LUNs) operating in user space access mode. While it is not a requirement that applications understand the interface between the block library and the cxlflash driver, a high-level overview of each supported service (IOCTL) is provided below.

The block library can be found on GitHub: http://github.com/open-power/capiflash

CXL Flash Driver LUN IOCTLs

Users, such as the block library, that wish to interface with a flash device (LUN) via user space access need to use the services provided by the cxlflash driver. As these services are implemented as ioctls, a file descriptor handle must first be obtained in order to establish the communication channel between a user and the kernel. This file descriptor is obtained by opening the device special file associated with the scsi disk device (/dev/sdb) that was created during LUN discovery. As per the location of the cxlflash driver within the SCSI protocol stack, this open is actually not seen by the cxlflash driver. Upon successful open, the user receives a file descriptor (herein referred to as fd1) that should be used for issuing the subsequent ioctls listed below.

The structure definitions for these IOCTLs are available in: uapi/scsi/cxlflash_ioctl.h

DK_CXLFLASH_ATTACH

This ioctl obtains, initializes, and starts a context using the CXL kernel services. These services specify a context id (u16) by which to uniquely identify the context and its allocated resources. The services additionally provide a second file descriptor (herein referred to as fd2) that is used by the block library to initiate memory mapped I/O (via mmap()) to the CXL flash device and poll for completion events. This file descriptor is intentionally installed by this driver and not the CXL kernel services to allow for intermediary notification and access in the event of a non-user-initiated close(), such as a killed process. This design point is described in further detail in the description for the DK_CXLFLASH_DETACH ioctl.

There are a few important aspects regarding the "tokens" (context id and fd2) that are provided back to the user:

- These tokens are only valid for the process under which they were created. The child of a forked process cannot continue to use the context id or file descriptor created by its parent (see DK_CXLFLASH_VLUN_CLONE for further details).
- These tokens are only valid for the lifetime of the context and the process under which they were created. Once either is destroyed, the tokens are to be considered stale and subsequent usage will result in errors.
- A valid adapter file descriptor (fd2 >= 0) is only returned on the initial attach for a context. Subsequent attaches to an existing context (DK_CXLFLASH_ATTACH_REUSE_CONTEXT flag present) do not provide the adapter file descriptor as it was previously made known to the application.
- When a context is no longer needed, the user shall detach from the context via the DK_CXLFLASH_DETACH ioctl. When this ioctl returns with a valid adapter file descriptor and the return flag DK_CXLFLASH_APP_CLOSE_ADAP_FD is present, the application _must_ close the adapter file descriptor following a successful detach.
- When this ioctl returns with a valid fd2 and the return flag DK_CXLFLASH_APP_CLOSE_ADAP_FD is present, the application _must_ close fd2 in the following circumstances:
    - Following a successful detach of the last user of the context
    - Following a successful recovery on the context's original fd2
    - In the child process of a fork(), following a clone ioctl, on the fd2 associated with the source context
- At any time, a close on fd2 will invalidate the tokens. Applications should exercise caution to only close fd2 when appropriate (outlined in the previous bullet) to avoid premature loss of I/O.

DK_CXLFLASH_USER_DIRECT

This ioctl is responsible for transitioning the LUN to direct (physical) mode access and configuring the AFU for direct access from user space on a per-context basis. Additionally, the block size and last logical block address (LBA) are returned to the user.

As mentioned previously, when operating in user space access mode, LUNs may be accessed in whole or in part. Only one mode is allowed at a time and if one mode is active (outstanding references exist), requests to use the LUN in a different mode are denied.

The AFU is configured for direct access from user space by adding an entry to the AFU's resource handle table. The index of the entry is treated as a resource handle that is returned to the user. The user is then able to use the handle to reference the LUN during I/O.

DK_CXLFLASH_USER_VIRTUAL

This ioctl is responsible for transitioning the LUN to virtual mode of access and configuring the AFU for virtual access from user space on a per-context basis. Additionally, the block size and last logical block address (LBA) are returned to the user.

As mentioned previously, when operating in user space access mode, LUNs may be accessed in whole or in part. Only one mode is allowed at a time and if one mode is active (outstanding references exist), requests to use the LUN in a different mode are denied.

The AFU is configured for virtual access from user space by adding an entry to the AFU's resource handle table. The index of the entry is treated as a resource handle that is returned to the user. The user is then able to use the handle to reference the LUN during I/O.

By default, the virtual LUN is created with a size of 0. The user would need to use the DK_CXLFLASH_VLUN_RESIZE ioctl to adjust the grow the virtual LUN to a desired size. To avoid having to perform this resize for the initial creation of the virtual LUN, the user has the option of specifying a size as part of the DK_CXLFLASH_USER_VIRTUAL ioctl, such that when success is returned to the user, the resource handle that is provided is already referencing provisioned storage. This is reflected by the last LBA being a non-zero value.

When a LUN is accessible from more than one port, this ioctl will return with the DK_CXLFLASH_ALL_PORTS_ACTIVE return flag set. This provides the user with a hint that I/O can be retried in the event of an I/O error as the LUN can be reached over multiple paths.

DK_CXLFLASH_VLUN_RESIZE

This ioctl is responsible for resizing a previously created virtual LUN and will fail if invoked upon a LUN that is not in virtual mode. Upon success, an updated last LBA is returned to the user indicating the new size of the virtual LUN associated with the resource handle.

The partitioning of virtual LUNs is jointly mediated by the cxlflash driver and the AFU. An allocation table is kept for each LUN that is operating in the virtual mode and used to program a LUN translation table that the AFU references when provided with a resource handle.

This ioctl can return -EAGAIN if an AFU sync operation takes too long. In addition to returning a failure to user, cxlflash will also schedule an asynchronous AFU reset. Should the user choose to retry the operation, it is expected to succeed. If this ioctl fails with -EAGAIN, the user can either retry the operation or treat it as a failure.

DK_CXLFLASH_RELEASE

This ioctl is responsible for releasing a previously obtained reference to either a physical or virtual LUN. This can be thought of as the inverse of the DK_CXLFLASH_USER_DIRECT or DK_CXLFLASH_USER_VIRTUAL ioctls. Upon success, the resource handle is no longer valid and the entry in the resource handle table is made available to be used again.

As part of the release process for virtual LUNs, the virtual LUN is first resized to 0 to clear out and free the translation tables associated with the virtual LUN reference.

DK_CXLFLASH_DETACH

This ioctl is responsible for unregistering a context with the cxlflash driver and release outstanding resources that were not explicitly released via the DK_CXLFLASH_RELEASE ioctl. Upon success, all "tokens" which had been provided to the user from the DK_CXLFLASH_ATTACH onward are no longer valid.

When the DK_CXLFLASH_APP_CLOSE_ADAP_FD flag was returned on a successful attach, the application _must_ close the fd2 associated with the context following the detach of the final user of the context.

DK_CXLFLASH_VLUN_CLONE

This ioctl is responsible for cloning a previously created context to a more recently created context. It exists solely to support maintaining user space access to storage after a process forks. Upon success, the child process (which invoked the ioctl) will have access to the same LUNs via the same resource handle(s) as the parent, but under a different context.

Context sharing across processes is not supported with CXL and therefore each fork must be met with establishing a new context for the child process. This ioctl simplifies the state management and playback required by a user in such a scenario. When a process forks, child process can clone the parents context by first creating a context (via DK_CXLFLASH_ATTACH) and then using this ioctl to perform the clone from the parent to the child.

The clone itself is fairly simple. The resource handle and lun translation tables are copied from the parent context to the child's and then synced with the AFU.

When the DK_CXLFLASH_APP_CLOSE_ADAP_FD flag was returned on a successful attach, the application _must_ close the fd2 associated with the source context (still resident/accessible in the parent process) following the clone. This is to avoid a stale entry in the file descriptor table of the child process.

This ioctl can return -EAGAIN if an AFU sync operation takes too long. In addition to returning a failure to user, cxlflash will also schedule an asynchronous AFU reset. Should the user choose to retry the operation, it is expected to succeed. If this ioctl fails with -EAGAIN, the user can either retry the operation or treat it as a failure.

DK_CXLFLASH_VERIFY

This ioctl is used to detect various changes such as the capacity of the disk changing, the number of LUNs visible changing, etc. In cases where the changes affect the application (such as a LUN resize), the cxlflash driver will report the changed state to the application.

The user calls in when they want to validate that a LUN hasn't been changed in response to a check condition. As the user is operating out of band from the kernel, they will see these types of events without the kernel's knowledge. When encountered, the user's architected behavior is to call in to this ioctl, indicating what they want to verify and passing along any appropriate information. For now, only verifying a LUN change (ie: size different) with sense data is supported.

DK_CXLFLASH_RECOVER_AFU

This ioctl is used to drive recovery (if such an action is warranted) of a specified user context. Any state associated with the user context is re-established upon successful recovery.

User contexts are put into an error condition when the device needs to be reset or is terminating. Users are notified of this error condition by seeing all 0xF's on an MMIO read. Upon encountering this, the architected behavior for a user is to call into this ioctl to recover their context. A user may also call into this ioctl at any time to check if the device is operating normally. If a failure is returned from this ioctl, the user is expected to gracefully clean up their context via release/detach ioctls. Until they do, the context they hold is not relinquished. The user may also optionally exit the process at which time the context/resources they held will be freed as part of the release fop.

When the DK_CXLFLASH_APP_CLOSE_ADAP_FD flag was returned on a successful attach, the application _must_ unmap and close the fd2 associated with the original context following this ioctl returning success and indicating that the context was recovered (DK_CXLFLASH_RECOVER_AFU_CONTEXT_RESET).

DK_CXLFLASH_MANAGE_LUN

This ioctl is used to switch a LUN from a mode where it is available for file-system access (legacy), to a mode where it is set aside for exclusive user space access (superpipe). In case a LUN is visible across multiple ports and adapters, this ioctl is used to uniquely identify each LUN by its World Wide Node Name (WWNN).

CXL Flash Driver Host IOCTLs

Each host adapter instance that is supported by the cxlflash driver has a special character device associated with it to enable a set of host management function. These character devices are hosted in a class dedicated for cxlflash and can be accessed via */dev/cxlflash/**.

Applications can be written to perform various functions using the host ioctl APIs below.

The structure definitions for these IOCTLs are available in: uapi/scsi/cxlflash_ioctl.h

HT_CXLFLASH_LUN_PROVISION

This ioctl is used to create and delete persistent LUNs on cxlflash devices that lack an external LUN management interface. It is only valid when used with AFUs that support the LUN provision capability.

When sufficient space is available, LUNs can be created by specifying the target port to host the LUN and a desired size in 4K blocks. Upon success, the LUN ID and WWID of the created LUN will be returned and the SCSI bus can be scanned to detect the change in LUN topology. Note that partial allocations are not supported. Should a creation fail due

to a space issue, the target port can be queried for its current LUN geometry.

To remove a LUN, the device must first be disassociated from the Linux SCSI subsystem. The LUN deletion can then be initiated by specifying a target port and LUN ID. Upon success, the LUN geometry associated with the port will be updated to reflect new number of provisioned LUNs and available capacity.

To query the LUN geometry of a port, the target port is specified and upon success, the following information is presented:

- Maximum number of provisioned LUNs allowed for the port
- Current number of provisioned LUNs for the port
- Maximum total capacity of provisioned LUNs for the port (4K blocks)
- Current total capacity of provisioned LUNs for the port (4K blocks)

With this information, the number of available LUNs and capacity can be can be calculated.

HT_CXLFLASH_AFU_DEBUG

This ioctl is used to debug AFUs by supporting a command pass-through interface. It is only valid when used with AFUs that support the AFU debug capability.

With exception of buffer management, AFU debug commands are opaque to cxlflash and treated as pass-through. For debug commands that do require data transfer, the user supplies an adequately sized data buffer and must specify the data transfer direction with respect to the host. There is a maximum transfer size of 256K imposed. Note that partial read completions are not supported - when errors are experienced with a host read data transfer, the data buffer is not copied back to the user.