# GPUDirect Storage API使用

注意：

cuFile API是线程安全的。

初始化GDS库后，不应使用fork系统调用。fork系统调用后API的行为在子进程中没有定义。API的设计不与fork调用工作。

应在有效的CUDA上下文环境中调用带GPU缓存的API。

## 1 cuFileDriverOpen

在启用任何其他GDS API之前，各进程仅激活一次该API。应用程序应调用该API来避免在第一次IO调用时的驱动程序的延迟。

## 2. cuFileHandleRegister

该API将文件描述词转换为cuFileHandle，检查在挂载点上的命名文件的能力，在该平台上通过GDS得到支持。

注意：各文件描述词应有一个句柄。

多线程共享相同句柄。查看示例代码展示使用多个线程使用相同句柄的信息。

注意：在兼容模式，无需O\_DIRECT模式可打开另外的fd。该模式还可以处理unaligned的读和写，甚至当POSIX无法处理时。

## 3. cuFileBufRegister, cuFileRead, and cuFileWrite

GPU内存应暴露给第三方设备来启动被这些设备使用的DMA。页面跨越这些在GPU虚拟地址空间中的缓冲，需要映射到BAR空间，该映射有一个overhead。

注意：实现该映射的进程称之为注册(registration)。

使用cuFileBufRegister显式执行缓冲注册是可选的。如果没有注册用户缓冲，使用cuFile实施的一个中间的预注册的GPU缓冲，从该缓冲复制到用户缓冲。下表列出是否注册有益的参考。

注意：IO Pattern 1是不优化的baseline情况，不在此表中。

| **Use Case** | **Description** | **Recommendation** |
| --- | --- | --- |
| A 4KB-aligned GPU buffer is reused as an intermediate buffer to read or write data by using optimal IO sizes for storage systems in multiples of 4KB. | The GPU buffer is used as an intermediate buffer to stream the contents or to populate a different data structure in GPU memory.  You can implement this use case for IO libraries with DSG. | Register this reusable intermediate buffer to avoid the additional internal staging of data by using GPU bounce buffers in the cuFile library.  See [IO Pattern 2](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#io-pattern-2) for the recommended usage. |
| Filling a large GPU buffer for one use. | The GPU buffer is the final location of the data. Since the buffer will not be reused, the registration cost will not be amortized. A usage example is reading large preformatted checkpoint binary data.  Registering a large buffer can have a latency impact when the buffer is registered. | This can also cause BAR memory exhaustion because running multiple threads or applications will compete for BAR memory.  Read or write the data without buffer registration.  See [IO Pattern 3](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#io-pattern-3) for the recommended usage. |
| Partitioning a GPU buffer to be accessed across multiple threads. | The main thread allocates a large chunk of memory and creates multiple threads. Each thread registers a portion of the memory chunk independently and uses that as in [IO Pattern 2](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#io-pattern-2).  You can also register the entire memory in the parent thread and use this registered buffer with the size and devPtr\_offset parameters set appropriately with the buffer offsets for each thread. A cudaContext must be established in each thread before registering the GPU buffers. | Allocate, register, and deregister the buffers in each thread independently for simple IO workflows.  For cases where the GPU memory is preallocated, each thread can set the appropriate context and register the buffers independently.  See IO Pattern 6 for the recommended usage.  After you install the GDS package, see cufile\_sample\_016.cc and cufile\_sample\_017.cc under /usr/local/CUDA-X.y/samples/ for more details. |
| GPU offsets, file offsets, and IO request sizes are unaligned. | The IO reads or writes are mostly unaligned. An intermediate aligned buffer might be needed to handle alignment issues with GPU offsets, file offsets, and IO sizes. | **Do not** register the buffer.  See [IO Pattern 4](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#io-pattern-4) and [IO Pattern 5](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#io-pattern-5). |
| Working on a GPU with a small BAR space as compared to the available GPU memory. | In some GPU SKUs, the BAR memory is smaller than the total device memory. | To avoid failures because of BAR memory exhaustion, do not register the buffer.  See [IO Pattern 3](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#io-pattern-3). |

通过O\_DIRECT方式打开文件，同时在IO时要求块大小对齐。

[aligned IO的性能要比unaligned IO的性能好很多](http://www.mysqlperformanceblog.com/2011/06/09/aligning-io-on-a-hard-disk-raid-the-theory/)。

## 4. cuFileHandleDeregister

要求：在调用该API之前，应用程序必须确保已经完成IO在句柄上，不在被使用。文件描述词应该打开状态。总是使用该API，在结束进程之前重申资源。

## 5、[cuFileBufDeregister](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#cufile-bug-deregister)

要求：

## 6、[cuFileDriverClose](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html#cufile-driver-close)

要求：

# cuFile APIs

cuFile API是对CUDA新的扩展，因此需要关注将来API的变化。例如，一些领域还在审查，包括cuFileDriver属性和cuFileBuf管理。

## 1 使用

1.1动态交互

Some of the cuFile APIs are optional, and if they are not called proactively, their actions will occur reactively:

if cuFile{Open, HandleRegister, BufRegister} are called on a driver, file, or buffer, respectively that has been opened or registered by a previous cuFile\* API call, results in an error. Calling cuFile{BufDeregister, HandleDeregister, DriverClose} on a buffer, file, or driver, respectively that has never been opened or registered by a previous cuFile\* API call results in an error. For these errors, the output parameters of the APIs are left in an undefined state, but there are no other side effects.

* cuFileDriverOpen 显式地初始化设备驱动

Its use is optional, and if it is not used, driver initialization happens implicitly at the first use of the cuFile{HandleRegister, Read, Write, BufRegister} APIs.

* cuFileHandleRegister

turns an OS-specific file descriptor into a cuFileHandle and performs checking on the GDS supportability based on the mount point and the way that the file was opened.

**Note:** Calling this API is a hard requirement.

* cuFileBufRegister explicitly registers a memory buffer.

If this API is not called, a memory buffer is registered the first time the buffer is used, for example, in cuFile{Read, Write}.

* cuFile{BufDeregister, HandleDeregister} explicitly frees a buffer and file resources.

If this API is not called, the buffer and resources are implicitly freed when the driver is closed.

* cuFileDriverClose explicitly frees driver resources.

If this API is not called, the driver resources are implicitly freed when the process is terminated.

If cuFile{Open, HandleRegister, BufRegister} is called on a driver, file, or buffer, respectively that has been opened or registered by a previous cuFile\* API call, results in an error. Calling cuFile{BufDeregister, HandleDeregister, DriverClose} on a buffer, file, or driver, respectively that has never been opened or registered by a previous cuFile\* API call also results in an error. For these errors, the output parameters of the APIs are left in an undefined state and there are no other side effects.

1.2驱动，文件和缓冲器管理

下面是管理驱动，文件和缓冲器的整体流程：

1. Call cuFileDriverOpen() to initialize the state of the critical performance path.
2. Allocate the GPU memory with cudaMalloc.
3. To register the buffer, call cuFileBufRegister to initialize the buffer state of the critical performance path.
4. Complete the following IO workflow:
   1. For Linux, open a file with POSIX open.
   2. Call cuFileHandleRegister to wrap an existing file descriptor in an OS-agnostic cuFileHandle. This step evaluates the suitability of the file state and the file mount for GDS and initializes the file state of the critical performance path.
   3. Call cuFileRead/cuFileWrite on an existing cuFile handle and existing buffer.
      * If the cuFileBufRegister has not been previously called, the first time that cuFileRead/cuFileWrite is accessed, the GDS library performs a validation check on the GPU buffer and an IO is issued.
      * Not using cuFileBufRegister might not be performant for small IO sizes.
      * Refer to the [GPUDirect Best Practices Guide](https://docs.nvidia.com/gpudirect-storage/best-practices-guide/index.html) for more information.
   4. Unless an error condition is returned, the IO is performed successfully.
5. Call cuFileBufDeregister to free the buffer-specific cuFile state.
6. Call cuFileHandleDeregister to free the file-specific cuFile state.
7. Call cuFileDriverClose to free up the cuFile state.

## 2 cuFile API定义

2.1数据类型

首先，cuFile API使用数据类型，然后typedef使用数据类型，最后枚举也使用数据类型。

2.2 cuFile驱动API

用来初始化、终止、查询和调整设置cuFile系统的API：

/\* Initialize the cuFile infrastructure \*/

CUfileError\_t cuFileDriverOpen();

/\* Finalize the cuFile system \*/

CUfileError\_t cuFileDriverClose();

/\* Query capabilities based on current versions, installed functionality \*/

CUfileError\_t cuFileGetDriverProperties(CUfileDrvProps\_t \*props);

/\*API to set whether the Read/Write APIs use polling to do IO operations \*/

CUfileError\_t cuFileDriverSetPollMode(bool poll, size\_t poll\_threshold\_size);

/\*API to set max IO size(KB) used by the library to talk to nvidia-fs driver \*/

CUfileError\_t cuFileDriverSetMaxDirectIOSize(size\_t max\_direct\_io\_size);

/\* API to set maximum GPU memory reserved per device by the library for internal buffering \*/

CUfileError\_t cuFileDriverSetMaxCacheSize(size\_t max\_cache\_size);

/\* Sets maximum buffer space that is pinned in KB for use by cuFileBufRegister \*/

CUfileError\_t cuFileDriverSetMaxPinnedMemSize(size\_t max\_pinned\_memory\_size);

2.3 cuFile IO APIs

The core of the cuFile IO APIs are the read and write functions.

ssize\_t cuFileRead(CUFileHandle\_t fh, void \*devPtr\_base, size\_t size, off\_t file\_offset, off\_t devPtr\_offset);

This API reads the data from a specified file handle at a specified offset and size bytes into the GPU memory by using GDS functionality. The API works correctly for unaligned offsets and any data size, although the performance might not match the performance of aligned reads.This is a synchronous call and blocks until the IO is complete.

Note: For the devPtr\_offset, if data will be read starting exactly from the devPtr\_base that is registered with cuFileBufRegister, devPtr\_offset should be set to 0. To read starting from an offset in the registered buffer range, the relative offset should be specified in the devPtr\_offset, and the devPtr\_base must remain set to the base address that was used in the cuFileBufRegister call.

ssize\_t cuFileWrite(CUFileHandle\_t fh, const void \*devPtr\_base, size\_t size, off\_t file\_offset, off\_t devPtr\_offset);

This API writes the data into a specified file handle at a specified offset and size bytes from the GPU memory by using GDS functionality. The API works correctly for unaligned offset and data sizes, although the performance is not on-par with aligned writes.This is a synchronous call and will block until the IO is complete.

If the file is opened with an O\_SYNC flag, the metadata will be written to the disk before the call is complete.

The buffer on the device has both a base (devPtr\_base) and offset (devPtr\_offset). This offset is distinct from the offset in the file.

2.4 cuFile File Handle APIs

The cuFileHandleRegister API makes a file descriptor or handle that is known to the cuFile subsystem by using an OS-agnostic interface. The API returns an opaque handle that is owned by the cuFile subsystem.

To conserve memory, the cuFileHandleDeregister API is used to release cuFile-related memory objects. Using only the POSIX close will not clean up resources that were used by cuFile. Additionally, the clean up of cufile objects that are associated with the files that were operated on in the cuFile context will occur at cuFileDriverClose.

CUfileError\_t cuFileHandleRegister(CUFileHandle\_t \*fh, CUFileDescr\_t \*descr);

void cuFileHandleDeregister(CUFileHandle\_t fh);

2.5 cuFile buffer APIs

The cuFileBufRegister API incurs a significant performance cost, so registration costs should be amortized where possible. Developers must ensure that buffers are registered up front and off the critical path.

The cuFileBufRegister API is optional. If this is not used, instead of pinning the user’s memory, cuFile-managed and internally pinned buffers are used.

The cuFileBufDeregister API is used to optimally clean up cuFile-related memory objects, but CUDA currently has no analog to cuFileBufDeregister. The cleaning up of objects that are associated with the buffers that were operated on in the cuFile context occur at cuFileDriverClose. If explicit APIs are used, the incurred errors are reported immediately, but if the operations of these explicit APIs are performed implicitly, error reporting and handling is less clear.

CUfileError\_t cuFileBufRegister(const void \*devPtr\_base, size\_t size, int flags);

CUfileError\_t cuFileBufDeregister(const void \*devPtr\_base);

## 3 cuFile API Functional Specification

3.1 cuFile 驱动 API Functional Specification

3.2 cuFile IO API Functional Specification

3.3 cuFile内存管理Functional Specification

3.4 cuFile流API Functional Specification

The stream APIs are similar to Read and Write, but they take a stream parameter to support asynchronous operations and execute in the CUDA stream order.

1、[cuFileReadAsync](https://docs.nvidia.com/gpudirect-storage/api-reference-guide/index.html" \l "cufilereadasync)

CUfileError\_t cudaFileReadAsync(CUFileHandle\_t fh, void \*devPtr\_base,

size\_t \*size, off\_t file\_offset,

off\_t devPtr\_offset,

int \*bytes\_read, cudaStream\_t stream);

CUfileError\_t cuFileReadAsync(CUFileHandle\_t fh, void \*devPtr\_base,

size\_t \*size, off\_t file\_offset,

off\_t devPtr\_offset,

int \*bytes\_read, CUstream stream);

2、[cuFileWriteAsync](https://docs.nvidia.com/gpudirect-storage/api-reference-guide/index.html#cufilewriteasync)

CUfileError\_t cudaFileWriteAsync(CUFileHandle\_t fh, void \*devPtr\_base,

size\_t \*size, off\_t file\_offset,

off\_t devPtr\_offset,

int \*bytes\_written, cudaStream\_t stream);

CUfileError\_t cuFileWriteAsync(CUFileHandle\_t fh, void \*devPtr\_base,

size\_t \*size, off\_t file\_offset,

off\_t devPtr\_offset,

int \*bytes\_written, CUstream\_t stream);

## 4 cuFile Batch API Functional Specification

## 5 示例代码

见cuFile\_test.cu