Getting Started with NVIDIA GPUDirect Storage

Release r1.12

NVIDIA Corporation

Mar 11, 2025

Contents

Chapter 1. Getting Started with NVIDIA GPUDirect Storage

Getting started information for NVIDIA® Magnum IO GPUDirect® Storage.

Chapter 2. Introduction

NVIDIA® GPUDirect® Storage (GDS) enables a direct data path for direct memory access (DMA) transfers between GPU memory and storage, which avoids a bounce buffer through the CPU. This direct path increases system bandwidth and decreases the latency and utilization load on the CPU.

The purpose of this guide is to help the user evaluate and test GDS functionality and performance by using sample applications. These applications can be run after you set up and install GDS and before you run the custom applications that have been modified to take advantage of GDS.

Refer to the following guides for more information about GDS:

- GPUDirect Storage Design Guide
- GPUDirect Storage Overview Guide
- cuFile API Reference Guide
- GPUDirect Storage Release Notes
- GPUDirect Storage Best Practices Guide
- GPUDirect Storage Installation and Troubleshooting Guide
- GPUDirect Storage Benchmarking and Configuration Guide
- GPUDirect Storage O_DIRECT Requirements Guide

To learn more about GDS, refer to the following posts:

- GPUDirect Storage: A Direct Path Between Storage and GPU Memory.
- The Magnum IO series.

Chapter 3. GDS Installation

GDS can be enabled in two ways: using the GDS NVIDIA kernel driver (`nvidia-fs.ko`) and updating the NVMe driver patches, or optionally as of 12.8 (when Linux Kernel and GPU version/driver requirements are met) with P2PDMA mode that does not require patching the NVMe driver nor installing the nvidia-fs driver. Please refer to the GPUDirect Storage Installation and Troubleshooting Guide for mroe information.

Chapter 4. Demonstrating GDS with GDSIO

GDSIO is a benchmarking tool designed to leverage NVIDIA's GPUDirect Storage (GDS) technology. Similar to the popular FIO utility, GDSIO allows users to configure various parameters such as block sizes, threads, read/write patterns, and interfaces. However, GDSIO is specifically focused on measuring the performance of GPU memory to storage transfers.

With GDSIO, users can evaluate throughput and latency metrics, enabling meaningful comparisons between different data transfer modes. Below, we provide examples to illustrate how GDSIO can be used to get started.

4.1. Examples

Example 1: Basic GPU-to-Storage Write

The simplest GDSIO command performs a GPU-to-storage write operation. The following command writes 8 KB (-s 8K) of data from GPU 0 (-d 0) to the file /mnt/gds/test. The data is written in two 4 KB blocks (-i 4K), and the -I 1 flag specifies a single iteration.

```
/usr/local/cuda/gds/tools/gdsio -x 0 -d 0 -s 8K -i 4K -f /mnt/gds/test -I 1

IoType: WRITE XferType: GPUD Threads: 1 DataSetSize: 8/8(KiB) IOSize: 4(KiB)
↪Throughput: 0.001081 GiB/sec, Avg_Latency: 14.000000 usecs ops: 2 total_time 0.
↪007057 secs
```

This operation measures GPU-to-storage transfer performance. The reported throughput, latency, and total execution time provide insights into the efficiency of direct data transfer.

Example 2: Comparing GDS Mode to CPU-Mediated Transfers

To compare GPU-Direct transfers (GPUD) with traditional GPU □ CPU □ Storage transfers, simply switch to the -x 2 mode:

```
/usr/local/cuda/gds/tools/gdsio -x 2 -d 0 -s 8K -i 4K -f /mnt/gds/a -I 1

IoType: WRITE XferType: CPU_GPU Threads: 1 DataSetSize: 8/8(KiB) IOSize: 4(KiB)
↪Throughput: 0.001127 GiB/sec, Avg_Latency: 40.000000 usecs ops: 2 total_time 0.
↪006769 secs
```

This comparison highlights the efficiency of GDS by contrasting it with the added latency and slightly reduced throughput of CPU-mediated transfers.

Example 3: Batch Mode for GPU Writes

Batch mode (-x 6) groups multiple small write operations into a single submission to improve efficiency. In this example, we write 8 KB (-s 8K) in batches of 10 entries (-w 10), each 4 KB in size (-i 4K). This configuration results in two batches, with a total of 20 I/O operations (2 batches × 10 I/Os per batch).

The next operation using the batch mode of GDS:

- Batch mode (-x 6) will gather 10 entries (-w 10), each of size 4k (-i 4K) into a single group and submit a write. There are a total of two batches to generate the entire desired write size (-s 8K). The aggregate number of I/Os is 20 (2 batches * 10 I/Os per batch)

  ```
  /usr/local/cuda/gds/tools/gdsio -x 6 -d 0 -s 8K -i 4K -D /mnt/gds/dir -I 1 -w 10

  IoType: WRITE XferType: GPU_BATCH Threads: 1 IoDepth: 10 DataSetSize: 80/80(KiB)
  ↪IOSize: 4(KiB) Throughput: 0.010523 GiB/sec, Avg_Latency: 3476.650000 usecs
  ↪ops: 20 total_time 0.007250 secs
  ```

  This operation demonstrates the performance benefits of batching, showing improved throughput for grouped I/O operations. The directory contents are as follows to show 10 files, each of size 8K.

  ```
  ls -l /mnt/gds/dir

  total 80
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.0
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.1
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.2
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.3
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.4
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.5
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.6
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.7
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.8
  -rw-r--r-- 1 root root 8192 Dec  5 17:12 gdsio.9
  ```

Additional information on GDSIO can be found at GDSIO Benchmark.

Chapter 5. Linking with GDSIO

The following is a simple example program that performs a GPU-to-storage write using the GDSIO library:

```
#include <iostream>
#include <fcntl.h>
#include <unistd.h>
#include <cuda_runtime.h>
#include <cufile.h>
int main() {

    CUfileHandle_t cfHandle;
    CUfileDescr_t cfDescr = {};
    const char *filename = "/mnt/gds/testfile";
    int fd = open(filename, O_CREAT | O_RDWR, 0664);
    if (fd < 0) {
        perror("File open failed");
        return 1;
    }

    // Set up GDS descriptor
    cfDescr.handle.fd = fd;
    cfDescr.type = CU_FILE_HANDLE_TYPE_OPAQUE_FD;
    CUfileError_t status = cuFileHandleRegister(&cfHandle, &cfDescr);
    if (status.err != CU_FILE_SUCCESS) {
        std::cerr << "cuFileHandleRegister failed: " << status.err << std::endl;
        close(fd);
        return 1;
    }

    // Alloc GPU memory and fill GPU memory with data
    void *devPtr;
    size_t bufferSize = 8192;
    cudaMalloc(&devPtr, bufferSize);
    cudaMemset(devPtr, 0xAB, bufferSize);

    // Perform the write
    ssize_t writtenBytes = cuFileWrite(cfHandle, devPtr, bufferSize, 0, 0);
    if (writtenBytes < 0) {
        perror("cuFileWrite failed");
    } else {
        std::cout << "Wrote " << writtenBytes << " bytes to the file." << std::endl;
    }
getBoolParameter()
    // Clean up
```

(continues on next page)

(continued from previous page)

```
    cuFileHandleDeregister(cfHandle);
    close(fd);
    cudaFree(devPtr);
    return 0;
}
```

Compile the sample above with the following command:

```
g++ -o gds_example gds_example.cc -I/usr/local/cuda/include -L/usr/local/cuda/lib64 -
↪lcuda -lcufile -lcudart
```

More information regarding the API can be found in the cuFile API Reference Guide.

Chapter 6. Troubleshooting GDS issues

The GPUDirect Storage Installation and Troubleshooting Guide has separate sections for basic and advanced troubleshooting.

The Cheat Sheet for Diagnosing Problems covers commonly seen problems and how to solve them.

Chapter 7. Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation ("NVIDIA") makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer ("Terms of Sale"). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

NVIDIA products are not designed, authorized, or warranted to be suitable for use in medical, military, aircraft, space, or life support equipment, nor in applications where failure or malfunction of the NVIDIA product can reasonably be expected to result in personal injury, death, or property or environmental damage. NVIDIA accepts no liability for inclusion and/or use of NVIDIA products in such equipment or applications and therefore such inclusion and/or use is at customer's own risk.

NVIDIA makes no representation or warranty that products based on this document will be suitable for any specified use. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer's sole responsibility to evaluate and determine the applicability of any information contained in this document, ensure the product is suitable and fit for the application planned by customer, and perform the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer's product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this document. NVIDIA accepts no liability related to any default, damage, costs, or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this document or (ii) customer product designs.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Chapter 8. OpenCL

OpenCL is a trademark of Apple Inc. used under license to the Khronos Group Inc.

Chapter 9. Trademarks

NVIDIA, the NVIDIA logo, CUDA, DGX, DGX-1, DGX-2, DGX-A100, Tesla, and Quadro are trademarks and/or registered trademarks of NVIDIA Corporation in the United States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright