



# 登临 Hamming™ V2

## GDS 使用说明

DL-DG/SW-058A-02

2024-12-19

Copyright©苏州登临科技有限公司，2019 - 2025，版权所有。

未经苏州登临科技有限公司事先书面同意，不得以任何形式或方式复制或传播本文件的任何部分。

## 商标和许可



和其它苏州登临科技有限公司的其它登临科技的图标为苏州登临科技有限公司的商标。本手册中提及的所有其他商标均为其各自所有者的财产。

## 通知

所购买的产品、服务和特性由苏州登临科技有限公司与客户签订的合同规定。本文件中描述的所有或部分产品、服务和特性可能不在采购范围或使用范围内。除非合同中另有规定，本文件中的所有声明、信息和建议均按“原样”提供，无任何明示或暗示的保证或陈述。本手册中的信息如有更改，恕不另行通知。本文件在编制过程中已尽一切努力确保内容的准确性，本文件中的所有声明、信息和建议不构成任何明示或暗示的保证。

苏州登临科技有限公司

苏州工业园区扬富路11号南岸新地一期商务楼5号1101室，江苏，中国

<http://www.denglin.ai>

email : support@denglin.ai

## 更新历史

版本	更新描述
02	更新章节 <b>2 安装和验证</b> 。
01	第一次发布。

# 目录

[目录](#)

[1 简介](#)

[2 安装和验证](#)

[3 Cufire函数使用说明](#)

## 1 简介

GDS使GPU内存和存储之间的直接内存访问（DMA）传输路径成为可能，从而避免通过CPU的中转缓冲区。这条直接路径增加了系统带宽，减少了延迟和CPU的利用负载。

本文介绍了GDS在登临平台上的外部安装依赖包安装、驱动安装、功能验证的步骤和cufile的函数说明。本文适用于登临Goldwasser™ II 平台。

## 2 安装和验证

### 1. 硬件环境准备包括以下内容：

- x86 CPU
- NVME磁盘
- 登临 AI 加速卡

### 2. 硬件安装

- 执行以下命令检查机器的ACS是否关闭。若没有输出，表明机器没有ACS相关Feature。

```
sudo lspci -vvv | grep -i acsctl
```

如果出现如下打印，需要确保每一项都是“-”；若出现“+”，请在BIOS设置关闭ACS。

```
ACSctl: SrcValid- TransBlk- ReqRedir- CmpltRedir- UpstreamFwd- EgressCtrl-  
DirectTrans
```

- 如需关闭IOMMU，具体操作请查阅相关设备BIOS配置手册。
- 可以使用lspci命令得到设备PCI ID。如下图所示，GPU和NVME磁盘PCI ID分别是01:00.0和02:00.0。

```

(sdk) jenkins@amd7700x-opc-18-180:/LocalRun/jenkins/gds/sdk/tools$ lspci
00:00.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14d8
00:01.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14da
00:01.1 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 14db
00:01.2 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 14db
00:02.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14da
00:02.1 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 14db
00:03.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14da
00:04.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14da
00:08.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14da
00:08.1 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 14dd
00:08.3 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 14dd
00:14.0 SMBus: Advanced Micro Devices, Inc. [AMD] FCH SMBus Controller (rev 71)
00:14.3 ISA bridge: Advanced Micro Devices, Inc. [AMD] FCH LPC Bridge (rev 51)
00:18.0 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e0
00:18.1 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e1
00:18.2 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e2
00:18.3 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e3
00:18.4 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e4
00:18.5 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e5
00:18.6 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e6
00:18.7 Host bridge: Advanced Micro Devices, Inc. [AMD] Device 14e7
01:00.0 Co-processor: Device 1e27:0003
02:00.0 Non-Volatile memory controller: Micron/Crucial Technology Device 5407
03:00.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 43f4 (rev 01)
04:00.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 43f5 (rev 01)
04:01.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 43f5 (rev 01)
04:02.0 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 43f5 (rev 01)

```

需要确保NVME磁盘和GPU处在同一个PCIE 主桥下。如果不在同一个主桥下磁盘和GPU无法确保能直接通信，应关机重新调整GPU插卡位置。

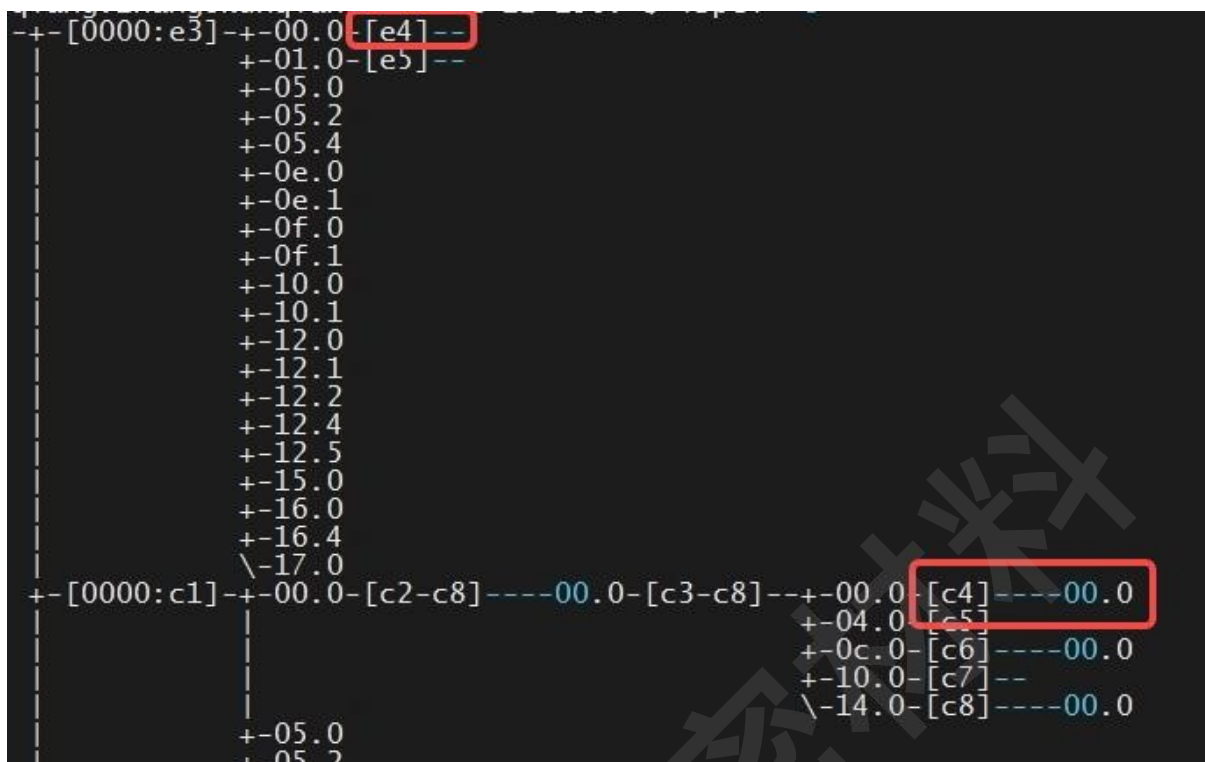
NVME磁盘和GPU在同一个主桥下，如下图所示。

```
(sdk) jenkins@amd7700x-opc-18-180:/LocalRun/jenkins/gds/sdk/tools$ lspci -v -s 01:00.0
01:00.0 Co-processor: Device 1e27:0003
Subsystem: Device 1e27:0608
Flags: bus master, fast devsel, latency 0, IRQ 121
Memory at fb000000 (32-bit, non-prefetchable) [size=16M]
Memory at fc000000 (32-bit, non-prefetchable) [size=4K]
Memory at f000000000 (64-bit, prefetchable) [size=32G]
Memory at f800000000 (64-bit, prefetchable) [size=256M]
Capabilities: <access denied>
Kernel driver in use: denglin

(sdk) jenkins@amd7700x-opc-18-180:/LocalRun/jenkins/gds/sdk/tools$ lspci -v -s 02:00.0
02:00.0 Non-Volatile memory controller: Micron/Crucial Technology Device 5407 (prog-if 02 [NVM Express])
Subsystem: Micron/Crucial Technology Device 0100
Flags: bus master, fast devsel, latency 0, IRQ 47, NUMA node 0
Memory at fcb00000 (64-bit, non-prefetchable) [size=16K]
Capabilities: <access denied>
Kernel driver in use: nvme
Kernel modules: nvme

(sdk) jenkins@amd7700x-opc-18-180:/LocalRun/jenkins/gds/sdk/tools$ lspci -t
-[0000:00]-+-00.0
              +-01.0
                +-01.1-[01]----00.0
                +-01.2-[02]----00.0
                +-02.0
                +-02.1-[03-0f]----00.0-[04-0f]--+00.0-[05]--
                |                                     +-01.0-[06]--
                |                                     +-02.0-[07]----00.0
                |                                     +-03.0-[08]----00.0
                |                                     +-04.0-[09]--
                |                                     +-05.0-[0a]--
                |                                     +-06.0-[0b]--
                |                                     +-07.0-[0c]--
                |                                     +-08.0-[0d]--
                |                                     +-0c.0-[0e]----00.0
                |                                     \-0d.0-[0f]----00.0
                +-03.0
                +-04.0
                +-08.0
                +-08.1-[10]--+00.0
                |               +-00.1
                |               +-00.2
                |               +-00.3
                |               +-00.4
                |               \-00.6
                +-08.3-[11]----00.0
                +-14.0
                +-14.3
```

PCI设备0xe4:00.0（在主桥0000:e3下）和0xc4:00.0（在主桥0000:c1下）不在同一PCI主桥下，如下图所示。



3. 由于GPU PCI只支持128B读写，需要对设备提前进行配置。

以下配置方法任选一种即可。

方法一：

1. 编辑 `/etc/default/grub` 文件，新增 `pci=pcie_bus_peer2peer` 配置，如下图所示：

```
GRUB_CMDLINE_LINUX="iommu=pt pci=pcie_bus_peer2peer"
```

2. 执行如下命令：

```
sudo update-grub
```

3. 重启系统。

方法二：（这种方法要求 GPU Firmware 版本在v1.4以上）。

1. 执行如下命令：

```
sudo dlsmi -mps 1
sudo dlsmi -mrrs 1
```

2. 重启系统。

3. 执行命令：

```
sudo setpci -s 00:01.1 61.b=09
```

00:01.1是与GPU直接相连的端口设备，如下图。

```
(sdk) qiang.zhang@amd7700x-opc-13-216:~$ sudo lspci -xxx -s 00:01.1
-[0000:00]--00.0
+00.2
+01.0
+01.1 [01]----00.0 GPU
+01.2-[02]----00.0
+02.0
```

```
(sdk) qiang.zhang@amd7700x-opc-13-216:~$ sudo lspci -xxx -s 00:01.1
00:01.1 PCI bridge: Advanced Micro Devices, Inc. [AMD] Device 14db
00: 22 10 db 14 07 04 10 00 00 00 04 06 10 00 81 00
10: 00 00 00 00 00 00 00 00 00 01 01 00 f1 01 00 20
20: 00 80 70 81 01 00 f1 ff 10 00 00 00 3f 00 00 00
30: 00 00 00 00 50 00 00 00 00 00 00 00 ff 00 12 00
40: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
50: 01 58 03 c8 00 00 00 00 10 a0 42 01 22 80 00 00
60: 10 09 00 00 05 71 73 00 00 00 05 b1 80 25 04 00
70: 00 00 40 01 18 00 01 00 00 00 00 00 ff 19 73 00
80: 06 04 00 00 3e 00 80 01 45 00 01 00 00 00 00 00
90: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
a0: 05 c0 81 00 00 00 e0 fe 00 00 00 00 00 00 00 00
b0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
c0: 0d c8 00 00 22 10 53 14 08 00 03 a8 00 00 00 00
d0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0: 00 00 00 00 fc 7f e0 9f 00 00 00 00 00 00 00 00
f0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

**注意：**

系统重启后会被重新设置为默认值，所以每次系统重启后需检查并重新设置。

**4. 执行命令检查是否设置成功：**

```
sudo lspci -vvv -s 00:01.1 | grep -v DevCap | grep MaxPayload
```

如下图：

```
amd7700x-opc-13-216:~$ sudo lspci -vvv -s 00:01.1 | grep -v DevCap | grep MaxPayload
MaxPayload 128 bytes, MaxReadReq 128 bytes
```

**4. 安装MLNX\_OFED软件包**

- 下载 [MLNX\\_OFED软件包](#) 并安装，验证版本是MLNX\_OFED\_LINUX-24.01-0.3.3.1。

```
./mlnxofedinstall --with-nvme --add-kernel-support --dkms --force
```

- 安装成功后重启系统。
- 执行如下命令检查NVME驱动是否正常。

```
cat /proc/kallsyms | grep nvme_v1_register_nvfs_dma_ops
```

如果没有出现下图相关信息，需要确认NVME磁盘是否存在。



```
qiang.zhang@amd7700x-opc-18-180:~$ cat /proc/kallsyms | grep nvme_v1_register_nvfs_dma_ops
0000000000000000 r __kstrtab_nvme_v1_register_nvfs_dma_ops [nvme]
0000000000000000 r __kstrtabns_nvme_v1_register_nvfs_dma_ops [nvme]
0000000000000000 r __ksymtab_nvme_v1_register_nvfs_dma_ops [nvme]
0000000000000000 T nvme_v1_register_nvfs_dma_ops [nvme]
```

- 如需确定读写文件所在磁盘是否使用NVME驱动，使用如下命令：

```
lspci -v -s 0000:xx:xx.x
```

- 其中 xx:xx.x 为磁盘PCI ID。如果看到 `kernel driver in use: nvme` 说明该磁盘使用的是NVME驱动。

5. 安装SDK。请参考 `sdk/documents` 目录下相关文档。

6. 执行如下命令，然后重启。

```
dlsmi -pbm 1
```

重启后，执行如下命令，其中 xx:xx.x 是GPU PCI ID。

```
lspci -v -s xx:xx.x
```

输出结果如下：

```
01:00.0 Co-processor: Device 1e27:0003
Subsystem: Device 1e27:0209
Flags: bus master, fast devsel, latency 0, IRQ 89
Memory at 80000000 (32-bit, non-prefetchable) [size=16M]
Memory at 81000000 (32-bit, non-prefetchable) [size=1M]
Memory at 1000000000 (64-bit, prefetchable) [size=64G]
Memory at 2000000000 (64-bit, prefetchable) [size=256M]
Capabilities: <access denied>
Kernel driver in use: denglin
```

说明：

根据具体卡型号，可能显示size为32G，64G，128G。如果显示其它，则表明未正确执行dlsmi命令。如果更新了firmware，需要重新设置。

7. 安装 gds-kernel driver ( gds-kernel driver需要与sdk在同一目录 )

- 进入 `sdk/kernel` 目录执行 `make` 编译GPU驱动。
- 进入 `gds-kernel/src/` 执行 `make` 命令编译，编译成功后生成 `denglin-fs.ko`。

可以使用如下命令，直接安装驱动。

```
sudo insmod denglin-fs.ko
```

8. 执行如下命令，初始化CUDA执行环境。

```
source sdk/env.sh
```

## 9. gds-test工具使用介绍

### ◦ gds-test命令参数

- --from, 从文件读数据到GPU Buffer, 后面跟文件名, 文件偏移和数据长度, 用逗号分隔, 中间不能有空格。
- --to, 把GPU Buffer中的内容写入文件, 后面跟文件名, 文件偏移和数据长度, 用逗号分隔, 中间不能有空格。
- --buffer, GPU Buffer, 后面配置buffer偏移, 可选, 不设置默认偏移是0。

from和to需要至少跟一个参数, 如果不跟from参数, 写入文件的内容是随机生成的可见字符。

### ◦ 测试实例

#先往GPU buffer写入65536字节随机可见ascii码数据, 然后通过GDS把GPU buffer中数据直接写入文件1.txt起始位置。

```
gds-test --to=/mount/gds-test/1.txt,0,65536
```

#先从1.txt 4096偏移处直接读取4096字节到GPU buffer的8192处, 然后再把GPU buffer中内容直接写入2.txt文件。

```
gds-test --from=/mount/gds-test/1.txt,4096,4096 --to=/mount/gds-test/2.txt,0,4096 --buffer=8192
```

### ◦ 测试异常结果分析

以下测试中, 由于文件dlcc只有0x2c5字节, 但请求读取长度length是0x1000字节, 所有实际只读取了0x2c5字节。后面测试在文件校验的时候, 校验了0x2c5字节, 数据校验无误。

写入文件789.log 0x1000字节, 其中文件前面0x2c5字节和dlcc文件相同, 后面剩余字节内容随机, 可能是0。实际使用中应确保被读文件有足够内容供读取。

```
(sdk) jenkins@amd7700x-opc-18-180:/LocalRun/jenkins/gds/sdk/tools$ ./gds-test --from=/nvme/jenkins/dlcc --to /nvme/jenkins/789.log --buffer=0
--from=/nvme/jenkins/dlcc,0x0,0x1000
--to=/nvme/jenkins/789.log,0x0,0x1000
--buffer=0x0
cufile: do_file_work(461): read fd=6 offset=0 length=0x1000 failed, errno=0 ioctl_ret=0x2c5
cufileReadAsync /nvme/jenkins/dlcc failed, ret=2c5
Check /nvme/jenkins/dlcc 0x2c5 bytes succeed
Check /nvme/jenkins/789.log 0x1000 bytes succeed
```

### 注意：

- 文件必须在NVME磁盘上（如果NVME磁盘未挂载，应使用 mount 命令先挂载磁盘）。
- 文件偏移, buffer偏移和数据长度必须4096对齐。
- 如果是读取文件, 需确保文件上有足够数据供读取, 否则执行结果不确定。

## 3 Cufire函数使用说明

1. `CufileError_t cuFileDriverOpen(void)`, 启动GDS相关驱动。初始化时必须调用。
2. `CufileError_t cuFileDriverClose(void)`, 关闭相关驱动。由于该函数会等待所有磁盘操作完成, 所以进程退出前应等待返回。

3. `CufileError_t cuFileStreamRegister(cudaStream_t stream, unsigned int flags)` , 注册CUDA Stream , 当前flag必须为0xf , 标志含义如下 :
  - bit0, buffer offset value is valid at submission time;
  - bit1, file offset value is valid at submission time;
  - bit2, size is valid at submission time;
  - bit3, all inputs are 4k aligned.
4. `CufileError_t cuFileStreamDeregister(cudaStream_t stream)` , 取消cuFileStreamRegister注册的Stream。
5. `CufileError_t cuFileHandleRegister(CUfileHandle_t *fh, CUfileDescr_t *descr)` , 注册文件描述符。当前仅支持 `CU_FILE_HANDLE_TYPE_OPAQUE_FD` 类型 , 成功后通过@fh返回注册的handler。文件必须使用 `O_DIRECT` 标志打开 , 例如 `open(file_name, O_DIRECT | O_RDWR | O_CREAT, 0644)` 。
6. `void cuFileHandleDeregister(CUfileHandle_t fh)` , 取消cuFileHandleRegister注册的handler。
7. `CufileError_t cuFileBufRegister(const void *devPtr_base, size_t length, int flags)` , 注册GPU memory。

参数解释如下 :

- `devPtr_base` 输入GPU memory首地址, 必须4096字节对齐。
  - `length` 输入memory 字节数, 必须4096字节对齐。
  - `flags` 必须是0。
8. `cuFileBufDeregister(const void* devPtr_base)` , 取消cuFileBufRegister注册的GPU memory。  
@devPtr\_base 输入GPU memory首地址。
  9. `CufileError_t cuFileReadAsync(CUfileHandle_t fh, void* bufPtr_base, size_t* size_p, offset* file_offset_p, off_t* bufPtr_offset_p, ssize_t* bytes_read_p, CUSTream stream)` ;  
`CufileError_t cuFileWriteAsync(CUfileHandle_t fh, void* bufPtr_base, size_t* size_p, offset* file_offset_p, off_t* bufPtr_offset_p, ssize_t* bytes_written_p, CUSTream stream)` ;

异步读写文件 ( 当前还是同步读写 , 读写操作结束才会返回 , 后续会变成真异步操作 , 需按异步操作调用 ) , 这两个函数返回失败说明没有操作成功且读写操作已结束 , 但返回 {`CU_FILE_SUCCESS`, `CUDA_SUCCESS`} 并不表示读写了相应字节数 , 调用者应检测 `bytes_read_p/bytes_written_p` 输出 , 确定读写是否结束及读写字节数 , 所有起始地址和偏移必须4096字节对齐且输入时有效。参数解释如下 :

- `fh` , 输入使用cuFileHandleRegister注册返回的handler。
- `bufPtr_base` , GPU memory起始地址。
- `size_p` , 存放读写字节数的地址。
- `file_offset_p` , 存放文件起始字节偏移地址。
- `bufPtr_offset_p` , 存放GPU memory偏移地址。
- `bytes_read_p/bytes_written_p` , 输出实际读写字节数。只有函数返回 {`CU_FILE_SUCCESS`, `CUDA_SUCCESS`} 后该输出才有效 , 读写操作结束后值才会被设置 , 调用者应检测该输出 , 确定读写结果。如果为正 , 则表示实际成功读写的字节数。如果为负 , 则表明读写失败。调用者可以先设置初始值0 , 如果该值被修改 , 说明读写操作已经结束。
- `stream` , 同步CUDA流。只有流前面所有操作完成后 , 才开始文件读写。只有读写完成后 , 才会执行后续CUDA操作。