WrapDrive

A Common Accelerator Framework for User Space Applications

Kenneth Lee<liguozhu@hisilicon.com>, Hisilicon

LEADING
COLLABORATION
IN THE ARM
ECOSYSTEM

Why WrapDrive

- **We want to create a fast paths between user applications and hardware accelerators**
  - E.g. the crypto framework is good enough for kernel space usage, but not for user space applications
- **When we dive deeper, we find that the VFIO has solved most of the problem**
  - But it works only for VF. This is not good enough for accelerators that should service many processes
- **So we build the framework on top of VFIO-mdev and extend its functionality**
  - Providing unified resource management API for applications
  - Let the mdev make use of the iommu facility of its parent device
  - Enable multiple asid/pasid support on iommu drivers
- **WrapDrive is intended to be a general framework to expose some of the hardware facility to the user applications.**
- **It can be used:**
  - to provide crypto features to the user space
  - to expose some queues of NIC for ODP/DPDK/Netmap like applications
  - for other hardware such as AI engines or CCIX

Performance: af_alg vs wd on openssl speed test

- Hardware: Hisilicon D05
- OS: Ubuntu 16.04 with Kernel 4.11-r1
- Open SSL 1.0.2

- openssl crypto plugin: https://github.com/sarnold/af_alg

Block size(bytes)	16	64	256	1024	8192
AF_ALG	37.86k	140.05k	565.33k	2530.30k	19802.79k
WrapDrive	6327.14k	24477.50k	97456.55k	335090.47k	1797931.01k
CPU only	635934.24k	1248170.84k	1623808.68k	1732138.33k	1795028.31k

Overhead	Command	Shared Object	Symbol
23.67%	openssl	[kernel.kallsyms]	[k] _raw_spin_unlock_irqrestore
6.38%	openssl	[kernel.kallsyms]	[k] __bi_memset
4.96%	openssl	[hisi_hacv1]	[k] hisi_sec_queue_send
3.29%	openssl	[kernel.kallsyms]	[k] __arch_copy_from_user
2.83%	openssl	[kernel.kallsyms]	[k] __slab_alloc.isra.21
2.74%	openssl	[kernel.kallsyms]	[k] memcg_kmem_put_cache
2.66%	openssl	[kernel.kallsyms]	[k] kmem_cache_alloc
2.28%	openssl	[kernel.kallsyms]	[k] __local_bh_enable_ip
1.90%	openssl	[kernel.kallsyms]	[k] _raw_spin_unlock_irq

AF_ALG

Overhead	Command	Shared Object	Symbol
22.76%	openssl	libwdsec.so	[.] wd_reg_read
19.70%	openssl	[kernel.kallsyms]	[k] el0_svc_naked
6.76%	openssl	libwdsec.so	[.] hisi_sec_v1_batch_recv
5.66%	openssl	[kernel.kallsyms]	[k] _raw_spin_unlock_irqrestore
4.45%	openssl	libwdsec.so	[.] wd_enging_aes_128_cbc_cipher
2.93%	openssl	libwdsec.so	[.] hisi_sec_v1_batch_send
2.90%	openssl	libwdsec.so	[.] wd_msg_2_hisi_sec_v1_msg
1.79%	openssl	[kernel.kallsyms]	[k] sys_epoll_wait

WrapDrive

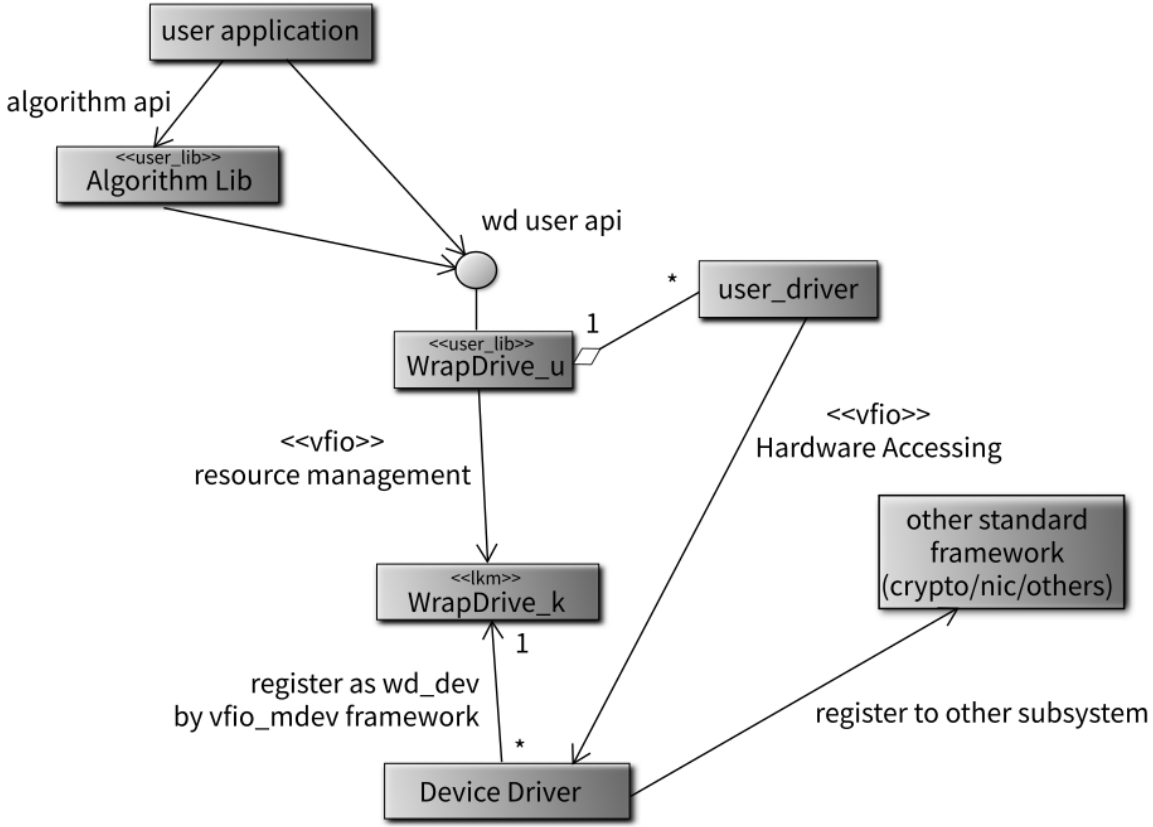
Alternative Solution

- Ibm-capi (drivers/misc/cxl): it is more focusing on FPGA setup for Power hardware
- KTLS (https://lwn.net/Articles/666509/): It is for network flow and it supports sending only
- Crypto user space if (Documentation/crypto/userspace-if.rst): It is not fast enough

Status

- It is in PoC stage. We encourage other accelerator developers come to join and help
- A demo with a dummy driver and a Hisilicon D05 driver is published on github: https://github.com/Kenneth-Lee/linux-kernel-wrapdrive/tree/wrapdrive-4.13-v1
- Documents: Documentation/wrapdrive/*
  - Framework: drivers/crypto/hisilicon/wd/*
  - Dummy Driver: drivers/crypto/hisilicon/dummy_drv/*
  - D05 Sec Driver: drivers/crypto/hisilicon/hacv1/*
  - A sample user library: samples/wrapdrive/*
  - Some test application: samples/wrapdrive/test/*
- A lot of work still needs to be done to reach the ideal solution

The Architecture



The WrapDrive_u interface

```
int wd_request_queue(struct wd_queue *q, struct wd_capa *capa);
void wd_release_queue(struct wd_queue *q);

int wd_send(struct wd_queue *q, void *req);
int wd_recv(struct wd_queue *q, void **req);
int wd_send_sync(struct wd_queue *q, void *req);
int wd_recv_sync(struct wd_queue *q, void **req);

int wd_mem_share(struct wd_queue *q, const void *addr, size_t size, int flags);
void wd_mem_unshare(struct wd_queue *q, const void *addr, size_t size);
```

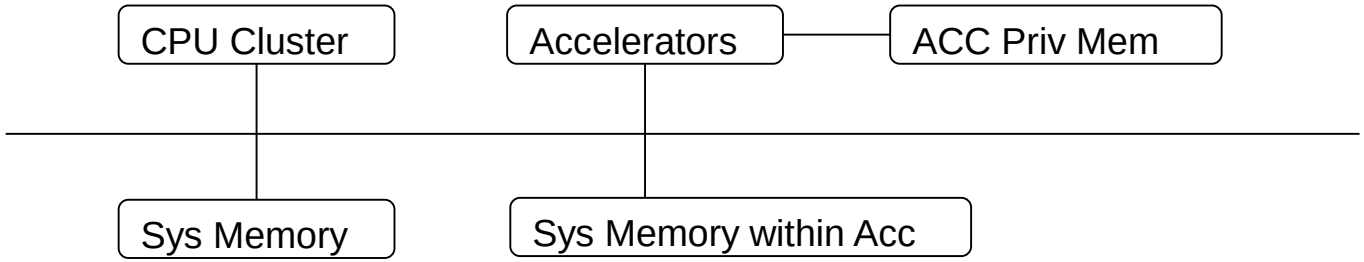
notes:

1. capa is used to identify a hardware capability and a wd_queue is used to identify a session to the capable hardware
2. A request to the hardware is done by send and recv from the queue. **It can be done by directly memory accessing without syscall.**
3. The recv reuses the same memory of the request queue.
4. send/recv can be strong or weak order according to the queue capability
5. DMA memory can be optionally shared to the hardware explicitly or by requesting a queue with *WD_CAPA_SHARE_ALL* flag.
6. An “Algorithm Library” can be created upon it for a particular algorithm. It can use hardware accelerators or the CPU to fulfill the request.

The VFIO-mdev interface

1. The accelerator driver is registered to VFIO-mdev with helper functions of WrapDrive_k (wd_register_dev)
2. All devices registered to WrapDrive belong to the class /sys/class/wd_dev
3. Its attributes are described in /sys/class/wd_dev/<dev_id>wrapdrive_dev
    1. numa_zone_id(rw): (it is writeable to leave the administer a chance to bind the device to another numa node)
    2. priority(rw)
4. Its capabilities are described in /sys/class/wd_dev/<dev_id>/mdev_supported_types/<dev_driv>-<algo_type>
    1. name(ro): name of the algorithm. it is the same as <algo_type>
    2. flags(ro): a bit mask to identify the feature supported in the algorithm
    3. available_instances(ro)
    4. device_api(ro): a string to identify the API version of the queue, it is used to identify the user driver
5. The queue is created by the mdev create interface. An mdev will be created for the user application to access the queue. It is created under /sys/class/wd_dev/<dev_id>/<mdev_id>. The params directory under it is used to described its parameters.

The Memory Model



The following memory usage models are considered:

1. The user application uses general memory and wd_share_memlist with hardware and send request upon it.
2. The user application uses memory in the same numa zone with the accelerator and wd_share_it with the hardware.
3. The user application requests a queue with the WD_CAPA_SHARE_ALL flag and adopts the same pattern as above without wd_share_it().
4. The user application requests a queue with the WD_CAPA_ALLOW_LEGACY flag. It dose not support wd_share_memlist. The memory can be used only with ioctl or read to the mdev. This model is not recommended.

1. The DMA mapping is done by VFIO and wrapped with wd_mem_share/wd_mem_unshared().
2. WD_CAPA_SHARE_ALL will be a new feature we are going to add to the VFIO and IOMMU framework. So we can share the whole application user space to the hardware.

The dummy driver

1. A demonstration driver doing memcpy from the kernel
2. It requires the noiommu mode of vfio, so requires cap_sys_rawio permission for the application.
3. No hardware is required, and it can be executed on any hardware platform. It is used for testing the WrapDrive resource management feature.
4. It can be tried on your PC within a virtual machine or on local hardware

The D05 HAC driver

1. D05 HAC driver is a platform drive which can be used to test the VFIO/IOMMU facility to prove the WrapDrive concept.
2. It registers with crypto and wrapdrive at the same time and can serve both kinds of application.
3. It is just a demonstration: it works only for one process.
    1. VFIO need to be upgraded to support multiple iommu_domain
    2. The vfio type1 driver and iommu should be upgraded to make use of the mdev parent device bus to create the iommu_domain (We have made a demo version of change in the shared public tree)
    3. The iommu driver need to be upgraded to support multiple domain
    4. The D05 SMMU does not support HAC hardware with a substream ID. We are waiting for the D06 to support multiple process
4. We will be happy to help you to enable your hardware to test the framework

Now let us talk about the problem

Please interrupt me if you have some feedback / suggestions

The SVM problem

1. With SVM, the DMA mapping can be done without pre fault the memory. Some update will be needed in VFIO framework (maybe some DMA parameters)
2. SVM is the most likely way to implement the WD_CAPA_SHARE_ALL feature of WrapDrive
3. But currently, the SVM feature of ARM is still in RFC status: http://www.spinics.net/lists/linux-pci/msg58650.html
4. We have checked it with some hardware and make some fixes: https://lkml.org/lkml/2017/8/31/211
5. PASID number is defined in PCIE but not in ACPI IORT

7.29.2. PASID Capability Register (Offset 04h)

Figure 7-140 details the allocation of register bits of the PASID Capability register; Table 7-118 provides the respective bit definitions.

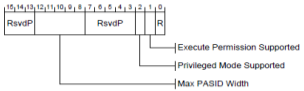


Figure 7-140: PASID Capability Register

Table 7-127: PASID Capability Register

Bit Location	Register Description	Attributes
0	**Reserved**	RsvdP
1	**Execute Permission Supported** – If Set, the Endpoint supports sending TLPs that have the Execute Requested bit Set. If Clear, the Endpoint will never Set the Execute Requested bit.	RO
2	**Privileged Mode Supported** – If Set, the Endpoint supports operating in Privileged and Non-Privileged modes, and supports sending requests that have the Privileged Mode Requested bit Set. If Clear, the Endpoint will never Set the Privileged Mode Requested bit.	RO
7:3	**Reserved**	RsvdP
12:8	**Max PASID Width** – Indicates the width of the PASID field supported by the Endpoint. The value n indicates support for PASID values 0 through 2^n-1 (inclusive). The value 0 indicates support for a single PASID (0). The value 20 indicates support for all PASID values (20 bits). This field must be between 0 and 20 (inclusive).	RO

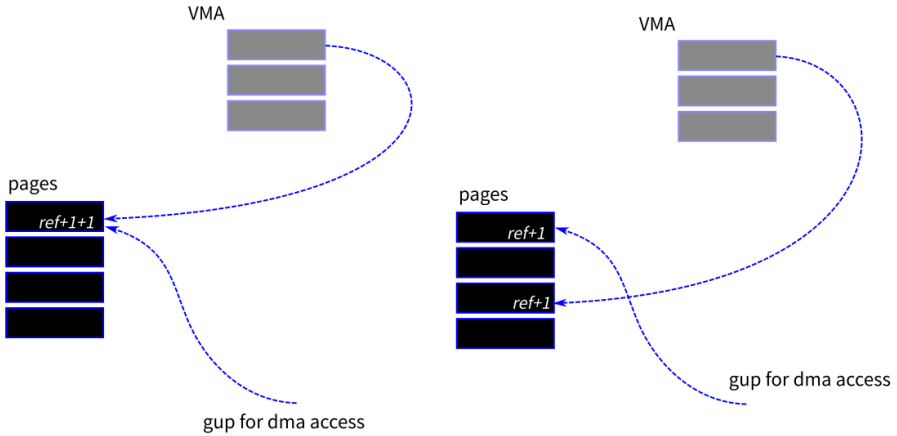
Table 13 Named component node format

Field	Byte Length	Byte Offset	Description
Type	1	0	For a named component, this field has a value of 1.
Length	2	1	The length of the node.
Revision	1	3	1.
Reserved	4	4	Must be zero.
Number of ID mappings	4	8	The number of ID mappings.
Reference to ID Array	4	12	Offset from the start of the IORT node to the start of its Array of ID mappings.
Named component specific data:			
Node flags	4	16	Bits 31:1 Reserved, must be zero Bit 0: Stall supported • This bit has a value of 1 if the system can tolerate transactions being stalled for this named device. • This bit has a value 0 if the system cannot tolerate transactions being stalled for this named device. WARNING: incorrectly setting this bit can lead to system deadlock and instability. If in doubt, set the bit to zero.

The GUP problem

1. Currently, the VFIO DMAMAP relies on gup. But according to the gup description, it is not going to work cross syscall.
2. This can be proved with MADVIDE(MADV_NOTNEED) after gup a user page.
3. This will remain as a risk until we can VM_PIN(https://lists.gt.net/linux/kernel/1931993) the vma.

```
/*
 *...
 * Must be called with mmap_sem held for read or write.
 *
 * get_user_pages walks a process's page tables and takes a reference to
 * each struct page that each user address corresponds to at a given
 * instant. That is, it takes the page that would be accessed if a user
 * thread accesses the given user virtual address at that instant.
 *
 * This does not guarantee that the page exists in the user mappings when
 * get_user_pages returns, and there may even be a completely different
 * page there in some cases (eg. if mmapped pagecache has been invalidated
 * and subsequently re faulted). However it does guarantee that the page
 * won't be freed completely. And mostly callers simply care that the page
 * contains data that was valid *at some point in time*. Typically, an IO
 * or similar operation cannot guarantee anything stronger anyway because
 * locks can't be held over the syscall boundary.
 *...
 */
```



Multiple asid support for the SMMU

1. Current SMMU driver does not support multiple domain at the same time
2. More detail: https://zhuanlan.zhihu.com/p/28853405
3. We are now working with this on Hi1620 (D06). But we will need ARM's help to upstream it.

mdev creating and releasing problem

1. The mdev can be created only with root permission. This is not acceptable for a user library. (maybe extend the mdev user interface with a misc device)
2. The mdev is created not binding to the process. Hence it will be remained if the process exits without remove it.
3. Currently, we clean them (those without user. In another word, those allocated by a non-exist pid) with a daemon. But it is not an elegant solution
4. So we think the best solution is:
    1. Take WrapDrive as part of mdev
    2. Expose the creation and release interface with a misc device
5. We will need Redhat's help on this.

Todo

1. Enable Multi-domain on SMMU
2. Enable parent iommu_domain for vfio-mdev
3. Enable SVM
4. Upstreaming: as part of hisi crypto driver (as the first step)? as part of crypto? or as part of vfio? (we prefer the third solution)
5. Upgrade Hi1616 HNS (NIC) to support WrapDrive
6. Add more drivers: Other SoC drivers, CCIX devices etc.

Open Discussion

Thank You

For further information: www.linaro.org

kLTS

setsockopt(socket, SOCK_ZEROCOPY);
send(socket, buffer, length, MSG_ZEROCOPY); //send only

recvmsg(socket, &message, MSG_ERRORQUEUE); //restore send buffer