DMA 相关概念以及 arm 实现

2018-12-19 2023-07-05 Linux 844
6.2k 6 mins.

Linux 内核中 DMA 及 Cache 分析，涉及以下函数

- dma_alloc_coherent
- dma_map_single
- dma_alloc_writecombine
- pgprot_noncached
- remap_pfn_range

- Linux Kernel: 4.9.22
- Arch: arm

arm

```
arch/arm/mm/dma-mapping.c
include/linux/dma-mapping.h
```

几个关键变量和函数

- atomic_pool_init 和 DEFAULT_DMA_COHERENT_POOL_SIZE
- dma zone、 dma pool、 setup_dma_zone 和 CONFIG_ZONE_DMA
- coherent_dma_mask 和 dma_zone_size

DMA ZONE

存在 DMA ZONE 的原因是某些硬件的 DMA 引擎 不能访问到所有的内存区域，因此，加上一个 DMA ZONE，当使用 GFP_DMA 方式申请内存时，获得的内存限制在 DMA ZONE 的范围内，这些特定的硬件需要使用 GFP_DMA 方式获得可以做 DMA 的内存;
如果系统中所有的设备都可选址所有的内存，那么 DMA ZONE 覆盖所有内存。
DMA ZONE 的大小，以及 DMA ZONE 要不要存在，都取决于你实际的硬件是什么。
由于设计及硬件的使用模式， DMA ZONE 可以不存在

> 由于现如今绝大多少的 SoC 都很牛逼，似乎 DMA 都没有什么缺陷了，根本就不太可能给我们机会指定 DMA ZONE 大小装逼了，那个这个 ZONE 就不太需要存在了。反正任何 DMA 在任何地方申请的内存，这个 DMA 都可以存取到。

DMA ZONE 的内存只能做 DMA 吗?

DMA ZONE 的内存做什么都可以。 DMA ZONE 的作用是让有缺陷的 DMA 对应的外设驱动申请 DMA buffer 的时候从这个区域申请而已，但是它不是专有的。其他所有人的内存（包括应用程序和内核）也可以来自这个区域。

dma_mask 与 coherent_dma_mask 的定义

include/linux/device.h

```
struct device {
    ...
    u64     *dma_mask;  /* dma mask (if dma'able device) */
    u64     coherent_dma_mask;/* Like dma_mask, but for
                        alloc_coherent mappings as
                        not all hardware supports
                        64 bit addresses for consistent
                        allocations such descriptors. */
    unsigned long   dma_pfn_offset;

    struct device_dma_parameters *dma_parms;

    struct list_head    dma_pools;  /* dma pools (if dma'ble) */

    struct dma_coherent_mem *dma_mem; /* internal for coherent mem
    ...
};
```

dma_mask 与 coherent_dma_mask 这两个参数表示它能寻址的物理地址的范围，内核通过这两个参数分配合适的物理内存给 device 。 dma_mask 是 设备 DMA 能访问的内存范围， coherent_dma_mask 则作用于申请 一致性 DMA 缓冲区 。因为不是所有的硬件都能够支持 64bit 的地址宽度。如果 addr_phy 是一个物理地址，且 (u64)addr_phy <= *dev->dma_mask ，那么该 device 就可以寻址该物理地址。如果 device 只能寻址 32 位地址，那么 mask 应为 0xffffffff 。依此类推。

例如内核代码 arch/arm/mm/dma-mapping.c

```
static void *__dma_alloc(struct device *dev, size_t size, dma_addr_t *handle,
             gfp_t gfp, pgprot_t prot, bool is_coherent,
             unsigned long attrs, const void *caller)
{
    u64 mask = get_coherent_dma_mask(dev);
    struct page *page = NULL;
    void *addr;
    bool allowblock, cma;
    struct arm_dma_buffer *buf;
    struct arm_dma_alloc_args args = {
        .dev = dev,
        .size = PAGE_ALIGN(size),
        .gfp = gfp,
        .prot = prot,
        .caller = caller,
        .want_vaddr = ((attrs & DMA_ATTR_NO_KERNEL_MAPPING) == 0),
        .coherent_flag = is_coherent ? COHERENT : NORMAL,
```

```
    };

#ifdef CONFIG_DMA_API_DEBUG
    u64 limit = (mask + 1) & ~mask;
    if (limit && size >= limit) {
        dev_warn(dev, "coherent allocation too big (requested %#x mask %#llx)\n",
            size, mask);
        return NULL;
    }
#endif
...
}
```

limit 就是通过 mask 计算得到的设备最大寻址范围

dma_alloc_coherent 分配的内存一定在 DMA ZONE 内吗?

dma_alloc_coherent() 申请的内存来自于哪里，不是因为它的名字前面带了个 dma_ 就来自 DMA ZONE 的，本质上取决于对应的 DMA 硬件是谁。应该说绝对多数情况下都不在 DMA ZONE 内，代码如下

dma_alloc_coherent -> dma_alloc_attrs

```
static inline void *dma_alloc_attrs(struct device *dev, size_t size,
                       dma_addr_t *dma_handle, gfp_t flag,
                       unsigned long attrs)
{
    struct dma_map_ops *ops = get_dma_ops(dev);
    void *cpu_addr;

    BUG_ON(!ops);

    if (dma_alloc_from_coherent(dev, size, dma_handle, &cpu_addr))
        return cpu_addr;

    if (!arch_dma_alloc_attrs(&dev, &flag))
        return NULL;
    if (!ops->alloc)
        return NULL;

    cpu_addr = ops->alloc(dev, size, dma_handle, flag, attrs);
    debug_dma_alloc_coherent(dev, size, *dma_handle, cpu_addr);
    return cpu_addr;
}
```

在 dma_alloc_attrs 首先通过 dma_alloc_from_coherent 从 device 自己的 dma memory 中申请，如果没有再通过 ops->alloc 申请， arm 如下

```
static struct dma_map_ops *arm_get_dma_map_ops(bool coherent)
{
    return coherent ? &arm_coherent_dma_ops : &arm_dma_ops;
}

struct dma_map_ops arm_coherent_dma_ops = {
    .alloc          = arm_coherent_dma_alloc,
    .free           = arm_coherent_dma_free,
    .mmap           = arm_coherent_dma_mmap,
    .get_sgtable        = arm_dma_get_sgtable,
    .map_page       = arm_coherent_dma_map_page,
    .map_sg         = arm_dma_map_sg,
};
EXPORT_SYMBOL(arm_coherent_dma_ops);

static void *arm_coherent_dma_alloc(struct device *dev, size_t size,
    dma_addr_t *handle, gfp_t gfp, unsigned long attrs)
{
    return __dma_alloc(dev, size, handle, gfp, PAGE_KERNEL, true,
                attrs, __builtin_return_address(0));
}

static void *__dma_alloc(struct device *dev, size_t size, dma_addr_t *handle,
             gfp_t gfp, pgprot_t prot, bool is_coherent,
             unsigned long attrs, const void *caller)
{
    u64 mask = get_coherent_dma_mask(dev);
    struct page *page = NULL;
    void *addr;
    bool allowblock, cma;
    struct arm_dma_buffer *buf;
    struct arm_dma_alloc_args args = {
        .dev = dev,
        .size = PAGE_ALIGN(size),
        .gfp = gfp,
        .prot = prot,
        .caller = caller,
        .want_vaddr = ((attrs & DMA_ATTR_NO_KERNEL_MAPPING) == 0),
        .coherent_flag = is_coherent ? COHERENT : NORMAL,
    };

#ifdef CONFIG_DMA_API_DEBUG
    u64 limit = (mask + 1) & ~mask;
    if (limit && size >= limit) {
        dev_warn(dev, "coherent allocation too big (requested %#x mask %#llx)\n",
            size, mask);
        return NULL;
    }
#endif

    if (!mask)
```

```
        return NULL;

    buf = kzalloc(sizeof(*buf),
              gfp & ~(__GFP_DMA | __GFP_DMA32 | __GFP_HIGHMEM));
    if (!buf)
        return NULL;

    if (mask < 0xffffffffULL)
        gfp |= GFP_DMA;

    /*
     * Following is a work-around (a.k.a. hack) to prevent pages
     * with __GFP_COMP being passed to split_page() which cannot
     * handle them.  The real problem is that this flag probably
     * should be 0 on ARM as it is not supported on this
     * platform; see CONFIG_HUGETLBFS.
     */
    gfp &= ~(__GFP_COMP);
    args.gfp = gfp;

    *handle = DMA_ERROR_CODE;
    allowblock = gfpflags_allow_blocking(gfp);
    cma = allowblock ? dev_get_cma_area(dev) : false;

    if (cma)
        buf->allocator = &cma_allocator;
    else if (nommu() || is_coherent)
        buf->allocator = &simple_allocator;
    else if (allowblock)
        buf->allocator = &remap_allocator;
    else
        buf->allocator = &pool_allocator;

    addr = buf->allocator->alloc(&args, &page);

    if (page) {
        unsigned long flags;

        *handle = pfn_to_dma(dev, page_to_pfn(page));
        buf->virt = args.want_vaddr ? addr : page;

        spin_lock_irqsave(&arm_dma_bufs_lock, flags);
        list_add(&buf->list, &arm_dma_bufs);
        spin_unlock_irqrestore(&arm_dma_bufs_lock, flags);
    } else {
        kfree(buf);
    }

    return args.want_vaddr ? addr : page;
}
```

&pool_allocator 从 DMA POOL 中分配，使用函数 atomic_pool_init 创建

代码段

```
    if (mask < 0xffffffffULL)
        gfp |= GFP_DMA;
```

GFP_DMA 标记被设置，以指挥内核从 DMA ZONE 申请内存。但是 mask 覆盖了整个 4GB，调用 dma_alloc_coherent() 获得的内存就不需要一定是来自 DMA ZONE

dma_alloc_coherent() 申请的内存是非 cache 的吗?

缺省情况下， dma_alloc_coherent() 申请的内存缺省是进行 uncache 配置的。但是现代 SOC 有可能会将内核的通用实现 overwrite 掉，变成 dma_alloc_coherent() 申请的内存也是可以带 cache 的。

```
static struct dma_map_ops *arm_get_dma_map_ops(bool coherent)
{
    return coherent ? &arm_coherent_dma_ops : &arm_dma_ops;
}

struct dma_map_ops arm_coherent_dma_ops = {
    .alloc          = arm_coherent_dma_alloc,
    .free           = arm_coherent_dma_free,
    .mmap           = arm_coherent_dma_mmap,
    .get_sgtable        = arm_dma_get_sgtable,
    .map_page       = arm_coherent_dma_map_page,
    .map_sg         = arm_dma_map_sg,
};
EXPORT_SYMBOL(arm_coherent_dma_ops);
```

Ref

1. kernel 如何保证 cache 数据一致性
2. 关于 DMA ZONE 和 dma_alloc_coherent 若干误解的澄清
3. DMA 及 cache 一致性的学习心得
4. DMA 导致的 CACHE 一致性问题解决方案
5. Linux 内存管理 —— DMA 和一致性缓存
6. cache 一致性问题
7. 简单粗暴有效的 mmap 与 remap_pfn_range
8. 认真分析 mmap：是什么 为什么 怎么用
9. mmap 函数：原理与使用（含代码）
10. Linux 内存映射函数 mmap（）函数详解
11. 宋宝华： 关于 DMA ZONE 和 dma alloc coherent 若干误解的彻底澄清
12. Loongson3A 的 DMA 传输