宋宝华：关于linux内存管理中DMA ZONE和dma_alloc_coherent若干误解的澄清

原创 宋宝华 于 2018-01-22 21:23:49 发布 13655 收藏 57 版权

分类专栏： Linux Kernel开发 文章标签： Linux DMA ZONE dma_alloc_coherent iommu

Linux Kernel开发 专栏收录该内容

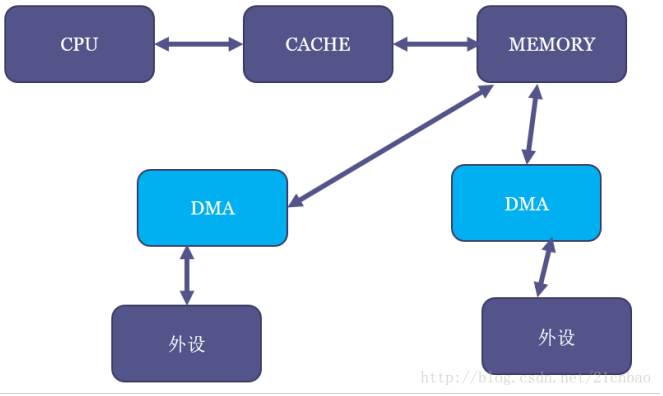
本文已首先在Linuxer公众号（ID： LinuxDev）发表，先转回我的blog也发表。转载请注明出处。

1.DMA ZONE的大小是16MB?

这个答案在32位X86计算机的条件下是成立的，但是在其他的绝大多数情况下都不成立。

首先我们要理解DMA ZONE产生的历史原因是什么。DMA可以直接在内存和外设之间进行数据搬移，对于内存的存取来讲，它和CPU一样，是一个访问master，可以直接访问内存。

DMA ZONE产生的本质原因是：不一定所有的DMA都可以访问到所有的内存，这本质上是硬件的设计限制。



在32位X86计算机的条件下,ISA实际只可以访问16MB以下的内存。那么ISA上面假设有个网卡，要DMA，超过16MB以上的内存，它根本就访问不到。所以Linux 内核干脆简单一点，把16MB砍一刀，这一刀以下的内存单独管理。如果ISA的驱动要申请DMA buffer，你带一个GFP_DMA标记来表明你想从这个区域申请，我保证申请的内存你是可以访问的。

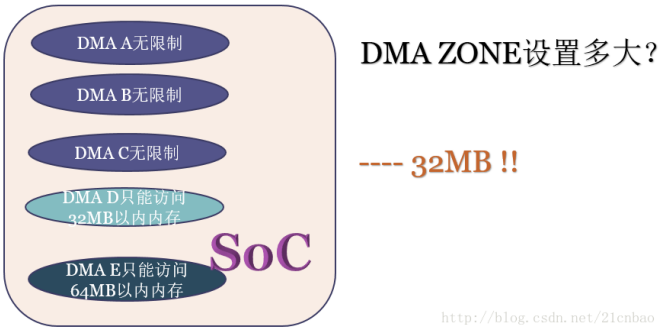
DMA ZONE的大小，以及DMA ZONE要不要存在，都取决于你实际的硬件是什么。比如我在CSR工作的时候，CSR的primaII芯片，尽管除SD MMC控制器以外的所有的DMA都可以访问整个4GB内存，但MMC控制器的DMA只能访问256MB，我们就把primaII对应Linux的DMA ZONE设为了256MB，详见内核：arch/arm/mach-prima2/common.c

```
#ifdef CONFIG_ARCH_PRIMA2

static const char *const prima2_dt_match[] __initconst = {

    "sirf,prima2",

    NULL

};


DT_MACHINE_START(PRIMA2_DT, "Generic PRIMA2 (Flattened Device Tree)")

    /* Maintainer: Barry Song <baohua.song@csr.com> */

    .l2c_aux_val    = 0,

    .l2c_aux_mask   = ~0,

    .dma_zone_size  = SZ_256M,

    .init_late      = sirfsoc_init_late,

    .dt_compat      = prima2_dt_match,

MACHINE_END

#endif
```

不过CSR这个公司由于早前已经被Q记收购，已经不再存在，一起幻灭的，还有当年挂在汽车前窗上的导航仪。这不禁让我想起我们当年在ADI arch/blackfin里面写的代码，也渐渐快几乎没有人用了一样。

一代人的芳华已逝,面目全非,重逢虽然谈笑如故,可不难看出岁月给每个人带来的改变。原谅我不愿让你们看到我们老去的样子,就让代码,留住我们芬芳的年华吧.....

下面我们架空历史，假设有一个如下的芯片，里面有5个DMA，A、B、C都可以访问所有内存，D只能访问32MB，而E只能访问64MB，你觉得Linux的设计者会把DMA ZONE设置为多大？当然是32MB，因为如果设置为64MB，D从DMA ZONE申请的内存就可能位于32MB-64MB之间，申请了它也访问不了。



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。

2.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。

3.dma_alloc_coherent()申请的内存来自DMA ZONE?

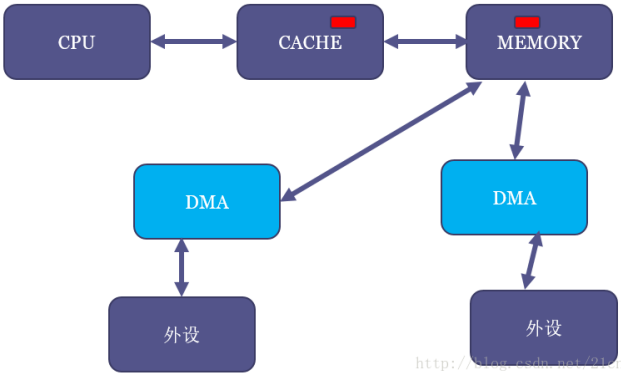
dma_alloc_coherent()申请的内存来自于哪里，不是因为它的名字前面带了个dma_就来自DMA ZONE的，本质上取决于对应的DMA硬件是谁。看代码：

```
static void *__dma_alloc(struct device *dev, size_t size, dma_addr_t *handle,
gfp_t gfp, pgprot_t prot, bool is_coherent, const void *caller)
{
        u64 mask = get_coherent_dma_mask(dev);
        …
        if (mask < 0xffffffffULL)
                gfp |= GFP_DMA;
        …
}
```

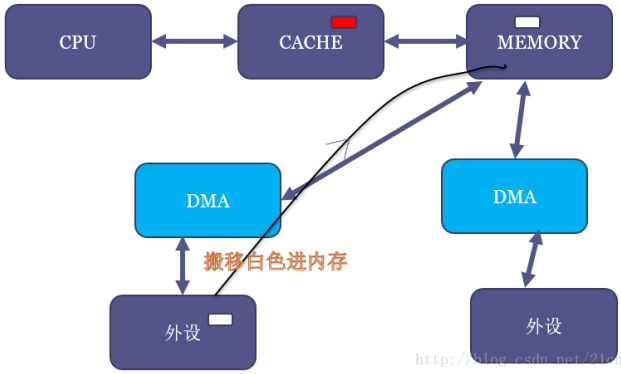
对于primaII而言，绝大多少的外设的dma_coherent_mask都设置为0Xffffffff（4GB内存全覆盖），但是SD那个则设置为256MB-1对应的数字。这样当primaII的SD驱动调用dma_alloc_coherent()的时候，GFP_DMA标记被设置，以指挥内核从DMA ZONE申请内存。但是，其他的外设，mask覆盖了整个4GB，调用dma_alloc_coherent()获得的内存就不需要一定是来自DMA ZONE。

4.dma_alloc_coherent()申请的内存是非cache的吗?

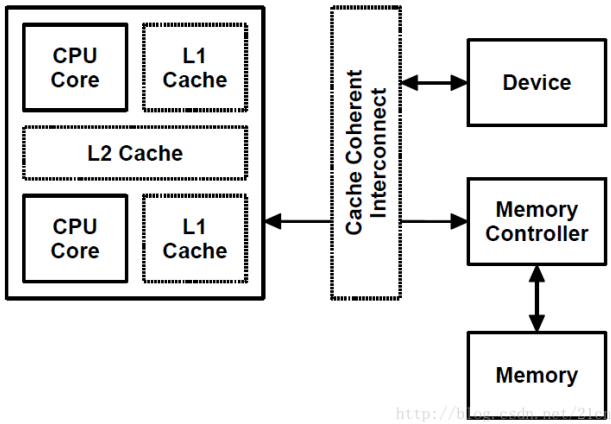
要解答这个问题，首先要理解什么叫cache coherent。还是继续看这个DMA的图，我们假设MEM里面有一块红色的区域，并且CPU读过它，于是红色区域也进CACHE：



但是，假设现在DMA把外设的一个白色搬移到了内存原本红色的位置：



这个时候，内存虽然白了，CPU读到的却还是红色，因为CACHE命中了，这就出现了cache的不coherent。
当然，如果是CPU写数据到内存，它也只是先写进cache（不一定进了内存），这个时候如果做一个内存到外设的DMA操作，外设可能就得到错误的内存里面的老数据。
所以cache coherent的最简单方法，自然是让CPU访问DMA buffer的时候也不带cache。事实上，缺省情况下，dma_alloc_coherent()申请的内存缺省是进行uncache配置的。
但是，由于现代SoC特别强，这样有一些SoC里面可以用硬件做CPU和外设的cache coherence，如图中的cache coherent interconnect：



这些SoC的厂商就可以把内核的通用实现overwrite掉，变成dma_alloc_coherent()申请的内存也是可以带cache的。这部分还是让大牛Arnd Bergmann童鞋来解释：

来自：https://www.spinics.net/lists/arm-kernel/msg322447.html

Arnd Bergmann：

dma_alloc_coherent() is a wrapper around a device-specific allocator,

based on the dma_map_ops implementation. The default allocator

from arm_dma_ops gives you uncached, buffered memory. It is expected

that the driver uses a barrier (which is implied by readl/writel

but not __raw_readl/__raw_writel or readl_relaxed/writel_relaxed)

to ensure the write buffers are flushed.

If the machine sets arm_coherent_dma_ops rather than arm_dma_ops,

the memory will be cacheable, as it's assumed that the hardware

is set up for cache-coherent DMAs.

当我grep内核源代码的时候，我发现部分SoC确实是这样实现的：

baohua@baohua-VirtualBox:~/develop/linux/arch/arm$ git grep arm_coherent_dma_ops

include/asm/dma-mapping.h:extern struct dma_map_ops arm_coherent_dma_ops;

mach-highbank/highbank.c: set_dma_ops(dev, &arm_coherent_dma_ops);

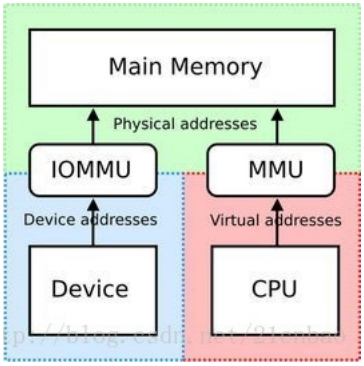
mach-mvebu/coherency.c: set_dma_ops(dev, &arm_coherent_dma_ops);

5.dma_alloc_coherent()申请的内存一定是物理连续的吗?

绝大多数的SoC目前都支持和使用CMA技术，并且多数情况下，DMA coherent APIs以CMA区域为申请的后端，这个时候，dma alloc coherent本质上用__alloc_from_contiguous()从CMA区域获取内存，申请出来的内存显然是物理连续的。这一点，在设备树dts里面就可以轻松配置,要么配置一个自己特定的cma区域，要么从“linux,cma-default”指定的缺省的CMA池子里面取内存：

```
reserved-memory {
		#address-cells = <1>;
		#size-cells = <1>;
		ranges;

		/* global autoconfigured region for contiguous allocations */
		linux,cma {
			compatible = "shared-dma-pool";
			reusable;
			size = <0x4000000>;
			alignment = <0x2000>;
			linux,cma-default;
		};

		display_reserved: framebuffer@78000000 {
			reg = <0x78000000 0x800000>;
		};

		multimedia_reserved: multimedia@77000000 {
			compatible = "acme,multimedia-memory";
			reg = <0x77000000 0x4000000>;
		};
	};
```

但是，如果IOMMU存在（ARM里面叫SMMU）的话，DMA完全可以访问非连续的内存，并且把物理上不连续的内存，用IOMMU进行重新映射为I/O virtual address (IOVA)：



所以dma_alloc_coherent()这个API只是一个前端的界面，它的内存究竟从哪里来，究竟要不要连续，带不带cache，都完全是因人而异的。

最后总结一句，千万不要被教科书和各种网上的资料懵逼了双眼，你一定要真正自己探索和搞清楚事情的本源。

今天看了《芳华》这部电影，感慨良多，遂作此文。

更多精华文章请扫描下方二维码关注Linux阅码场