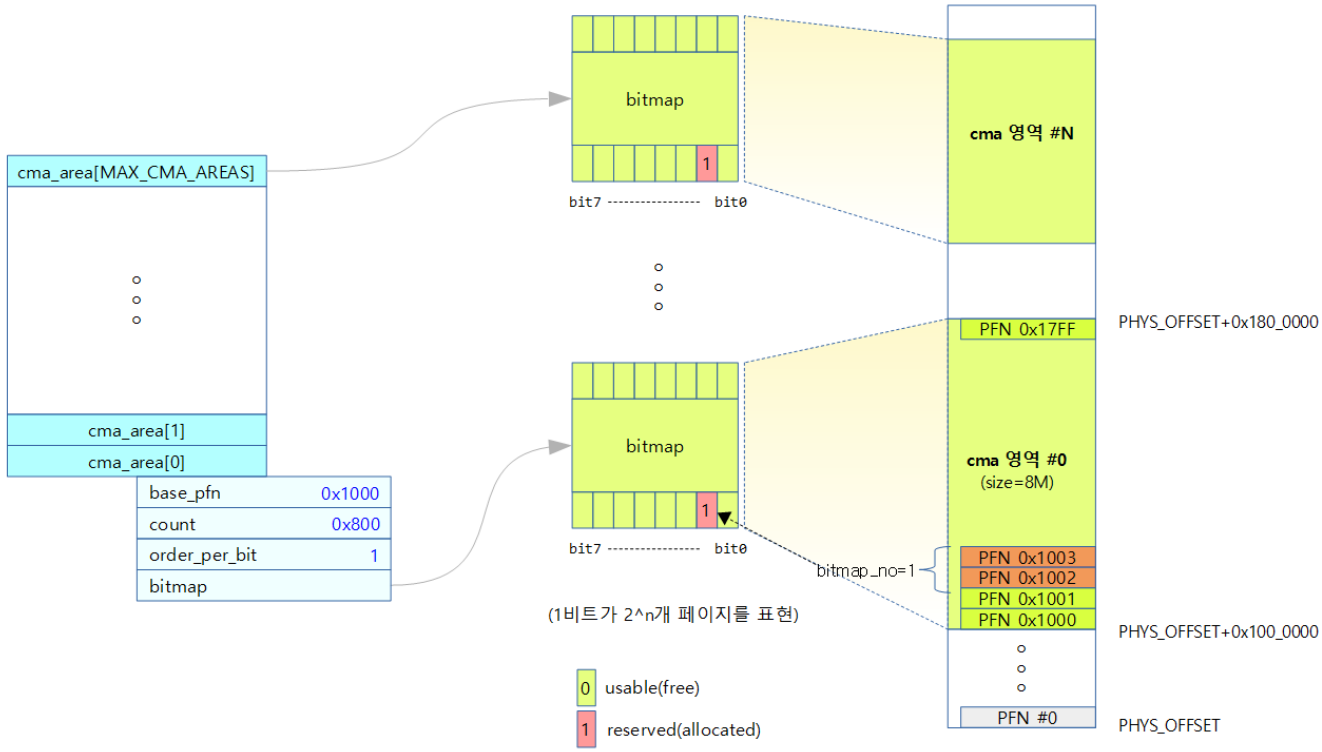
CMA(Contiguous Memory Allocator)

2016-03-09 (http://jake.dothome.co.kr/cma/) Moon Young-il (http://jake.dothome.co.kr/author/admin/) Linux Kernel (http://jake.dothome.co.kr/category/linux/)

The Buddy system is a system that manages the allocation of contiguous physical memory, and the system that is used to give priority to contiguous physical memory used by devices is CMA.
If you set a CMA area specifically for the device device, the Buddy system will normally manage to assign this area as a movable page. If the device requests contiguous physical memory when this area is full of movable pages, it will migrate some pages from the CMA region to a region other than the CMA area, and the CMA memory allocator will allocate as many pages of memory as the device needs.

- It is a continuous memory allocator created by Samsung in 2010 and developed with the participation of IBM and LG.
- CONFIG_CMA use kernel options.
- The CMA function is mainly used to manage the physically contiguous memory required by the DMA subsystem so that it can be allocated and used when the kernel is executed without having to reserve.
    - CONFIG_DMA_CMA use kernel options.
    - In other PowerPC architectures, it is also used for KVM.
    - Specify the CMA area with the "cma=" kernel parameter, which is limited to using only CMA and movable pages, and if the CMA area is insufficient, the movable pages will be migrated elsewhere.
- Unlike the buddy allocator, the assignment is not limited to the size of the order of two. (2M area allocation, etc.)
- The CMA allocator is fully integrated with the DMA subsystem, eliminating the need for device driver developers using DMA to use a separate CMA API.
    - Reference: Document/DMA-API.txt (http://www.kernel.org/doc/Documentation/DMA-API.txt) | kernel.org

The following figure shows that the cma is managed in an array of cma_area[] structures, and each cma area is bitmap-managed.



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/cma-1.png)

Cache-Coherent Issues

- If H/W does not provide cache coherent functionality, then S/W itself needs cache coherent functionality, which uses cacheless mapping via the dma_alloc_coherent() function when allocating pages.
  - If you use it via the dma_map_single() function, you can also create a buffer that can be accessed quickly without the need for a cache coherent.
    - The DMA Pool allocator also allows you to use the dma_pool_create() and dma_pool_alloc() functions.
- Unfortunately, there is an issue with the current CMA structure not being complete for cache mappings. The problem is that the two mappings conflict for the uncached memory pages that are allocated to the DMA region because the cache mapping is required for the cache to be used for the cache coherent function, which is already mapped to the lowmem area of the kernel.
  - There are two ways to solve this. One way is to assign a cma area to the highmem. (However, small memory or 2-bit systems do not have highmem...^^;) The second method is to remove the area that is allocated and used as a dma-buffer from the kernel's linear mapping.
    - Currently I am pushing the second method. With this approach, the kernel memory area (lowmem) is linearly mapped to a large extent using huge pages, and it is inevitable that part of this area will be partitioned and mapped, resulting in a decrease in TLB cache performance.
    - CMA and ARM (https://lwn.net/Articles/450286/) | LWN.net

Add a CMA admin area

The CMA management areas defined in this function will be initialized later when the CMA driver is loaded, and then allocation management will be available. If you don't use the CONFIG_MODULE kernel options, it will be initialized by the following routine:

- core_initcall(cma_init_reserved_areas);
  - All registered initcall functions are called from the do_initcalls() function of the kernel_init thread.

cma_declare_contiguous()

mm/cma.c

```
01 /**
02  * cma_declare_contiguous() - reserve custom contiguous area
03  * @base: Base address of the reserved area optional, use 0 for any
04  * @size: Size of the reserved area (in bytes),
05  * @limit: End address of the reserved memory (optional, 0 for any).
06  * @alignment: Alignment for the CMA area, should be power of 2 or zero
07  * @order_per_bit: Order of pages represented by one bit on bitmap.
08  * @fixed: hint about where to place the reserved area
09  * @res_cma: Pointer to store the created cma region.
10  *
11  * This function reserves memory from early allocator. It should be
12  * called by arch specific code once the early allocator (memblock or bo
    otmem)
13  * has been activated and all other subsystems have already allocated/re
    served
14  * memory. This function allows to create custom reserved areas.
15  *
16  * If @fixed is true, reserve contiguous area at exactly @base.  If fals
    e,
17  * reserve in range from @base to @limit.
18  */
19 int __init cma_declare_contiguous(phys_addr_t base,
20                         phys_addr_t size, phys_addr_t limit,
21                         phys_addr_t alignment, unsigned int order_per_bi
    t,
22                         bool fixed, struct cma **res_cma)
23 {
24         phys_addr_t memblock_end = memblock_end_of_DRAM();
25         phys_addr_t highmem_start;
26         int ret = 0;
27
28 #ifdef CONFIG_X86
29         /*
30          * high_memory isn't direct mapped memory so retrieving its phys
    ical
31          * address isn't appropriate.  But it would be useful to check t
    he
32          * physical address of the highmem boundary so it's justfiable t
    o get
33          * the physical address from it.  On x86 there is a validation c
    heck for
34          * this case, so the following workaround is needed to avoid it.
35          */
36         highmem_start = __pa_nodebug(high_memory);
37 #else
38         highmem_start = __pa(high_memory);
39 #endif
40         pr_debug("%s(size %pa, base %pa, limit %pa alignment %pa)\n",
41                 __func__, &size, &base, &limit, &alignment);
```

```
42
43          if (cma_area_count == ARRAY_SIZE(cma_areas)) {
44                  pr_err("Not enough slots for CMA reserved regions!\n");
45                  return -ENOSPC;
46          }
47
48          if (!size)
49                  return -EINVAL;
50
51          if (alignment && !is_power_of_2(alignment))
52                  return -EINVAL;
```

Set the memory area you want to manage with CMA. The memory area is limited to a maximum of MAX_CMA_AREAS.

- phys_addr_t memblock_end = memblock_end_of_DRAM();
  - Locate the end address of the last item registered in memory memblock by base + size.
- highmem_start = __pa(high_memory);
  - high_memory
    - Compare the physical memory size to the size of the maximum lowmem region
      - If it is exceeded, the high_memory is max at the end of the lowmem zone's virtual address
      - If not exceeded, the high_memory is the end virtual address of the lowmem zone
- if (cma_area_count == ARRAY_SIZE(cma_areas)) {
  - If the cma_areas array is full, it prints an error message and returns an error.
- if (alignment && !is_power_of_2(alignment))
  - If alignment is specified, if it is not of order 2, it simply returns an error.

```
01 /*
02  * Sanitise input arguments.
03  * Pages both ends in CMA area could be merged into adjacent unmovable
04  * migratetype page by page allocator's buddy algorithm. In the case,
05  * you couldn't get a contiguous memory, which is not what we want.
06  */
07 alignment = max(alignment,
08         (phys_addr_t)PAGE_SIZE << max(MAX_ORDER - 1, pageblock_order));
09 base = ALIGN(base, alignment);
10 size = ALIGN(size, alignment);
11 limit &= ~(alignment - 1);
12
13 if (!base)
14         fixed = false;
15
16 /* size should be aligned with order_per_bit */
17 if (!IS_ALIGNED(size >> PAGE_SHIFT, 1 << order_per_bit))
18         return -EINVAL;
19
20 /*
21  * If allocating at a fixed base the request region must not cross the
22  * low/high memory boundary.
23  */
24 if (fixed && base < highmem_start && base + size > highmem_start) {
25         ret = -EINVAL;
26         pr_err("Region at %pa defined on low/high memory boundary (%pa)
    \n",
27                 &base, &highmem_start);
28         goto err;
29 }
```

- alignment = max(alignment, (phys_addr_t)PAGE_SIZE << max(MAX_ORDER – 1, pageblock_order))
  - pageblock_order
    - 32bit ARM does not support CONFIG_HUGETLB_PAGE option.
  - MAXORDER(11=used by buddy)-1.
- Assign alignment and 4M (PAGE_SIZE << 10), whichever is greater.
- base = ALIGN(base, alignment);
  - base to alignment unit
- size = ALIGN(size, alignment);
  - Size to Alignment Units
- limit &= ~(alignment – 1);
  - Limit is rounded down by alignment unit
- if (!base)
  - If base is 0, change the value of the argument fixed to false.
  - After all, even if you call this function with fixed as true, it will only remain true if base is 0.
- if (!IS_ALIGNED(size >> PAGE_SHIFT, 1 << order_per_bit))
  - If the size is not aligned to 2^order_per_bit pages (4K), it returns an error.
- if (fixed && base < highmem_start && base + size > highmem_start) {
  - fixed is true, but if the area overlaps the highmem area, the
    - Prints an error message that the zone is defined on a low/high memory boundary and returns an error.

```
01          /*
02           * If the limit is unspecified or above the memblock end, its ef
    fective
03           * value will be the memblock end. Set it explicitly to simplify
    further
04           * checks.
05           */
06          if (limit == 0 || limit > memblock_end)
07                  limit = memblock_end;
08
09          /* Reserve memory */
10          if (fixed) {
11                  if (memblock_is_region_reserved(base, size) ||
12                      memblock_reserve(base, size) < 0) {
13                          ret = -EBUSY;
14                          goto err;
15                  }
16          } else {
17                  phys_addr_t addr = 0;
18
19                  /*
20                   * All pages in the reserved area must come from the sam
    e zone.
21                   * If the requested region crosses the low/high memory b
    oundary,
22                   * try allocating from high memory first and fall back t
    o low
23                   * memory in case of failure.
24                   */
25                  if (base < highmem_start && limit > highmem_start) {
26                          addr = memblock_alloc_range(size, alignment,
27                                                      highmem_start, limi
    t);
28                          limit = highmem_start;
29                  }
30
```

```
31                  if (!addr) {
32                          addr = memblock_alloc_range(size, alignment, base,
33                                                      limit);
34                          if (!addr) {
35                                  ret = -ENOMEM;
36                                  goto err;
37                          }
38                  }
39
40                  /*
41                   * kmemleak scans/reads tracked objects for pointers to other
42                   * objects but this address isn't mapped and accessible
43                   */
44                  kmemleak_ignore(phys_to_virt(addr));
45                  base = addr;
46          }
47
48          ret = cma_init_reserved_mem(base, size, order_per_bit, res_cma);
49          if (ret)
50                  goto err;
51
52          pr_info("Reserved %ld MiB at %pa\n", (unsigned long)size / SZ_1M,
53                  &base);
54          return 0;
55
56  err:
57          pr_err("Failed to reserve %ld MiB\n", (unsigned long)size / SZ_1M);
58          return ret;
59  }
```

- if (limit == 0 || limit > memblock_end)
  - If the limit is 0 or the limit is greater than memblock_end, set the limit to memblock_end.
  - Register the size of the memory in the CMA.
- if (fixed) {
  - If you want to add a static base address in lowmem when adding
- if (memblock_is_region_reserved(base, size) || memblock_reserve(base, size) < 0) {
  - Returns an error if the zone was registered with the reserve memblock, or if there is an error in registering the zone with the reserve memblock.

If fixed is false, then if the area to be added is the boundary between lowmem and highmem zones, it will first try to allocate to the highmem area, and if that fails, it will be assigned to the lowmem area.

- if (base < highmem_start && limit > highmem_start) {
  - If the area you want to add is the boundary between the highmem and lowmem zones.
- addr = memblock_alloc_range(size, alignment, highmem_start, limit);
  - Assign the request size between the range of highmem_start and limit.
- if (!addr) {
  - If the allocation to the highmem zone fails
- addr = memblock_alloc_range(size, alignment, base, limit);
  - It allocates a request size between base and limit and returns an error if it fails.
- kmemleak_ignore(phys_to_virt(addr));
  - This address is scanned and ignored so that it is not reported as leak.
- ret = cma_init_reserved_mem(base, size, order_per_bit, res_cma);

- Perform a sanity check on the reserved area once again and set up the CMA management items.
- CMA management items (CMA Area information) must be initialized at a later date.
  - When the CMA module is loaded, cma_init_reserved_area() is called, at which point each area is initialized.

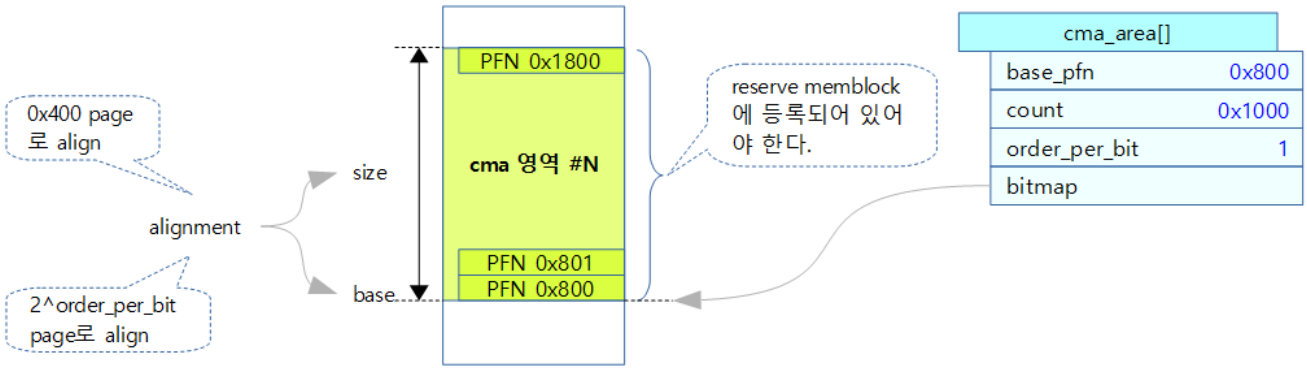
cma_init_reserved_mem()

mm/cma.c

```
01 /**
02  * cma_init_reserved_mem() - create custom contiguous area from reserved
   memory
03  * @base: Base address of the reserved area
04  * @size: Size of the reserved area (in bytes),
05  * @order_per_bit: Order of pages represented by one bit on bitmap.
06  * @res_cma: Pointer to store the created cma region.
07  *
08  * This function creates custom contiguous area from already reserved me
   mory.
09  */
10 int __init cma_init_reserved_mem(phys_addr_t base, phys_addr_t size,
11                                  int order_per_bit, struct cma **res_cm
   a)
12 {
13         struct cma *cma;
14         phys_addr_t alignment;
15
16         /* Sanity checks */
17         if (cma_area_count == ARRAY_SIZE(cma_areas)) {
18                 pr_err("Not enough slots for CMA reserved regions!\n");
19                 return -ENOSPC;
20         }
21
22         if (!size || !memblock_is_region_reserved(base, size))
23                 return -EINVAL;
24
25         /* ensure minimal alignment requied by mm core */
26         alignment = PAGE_SIZE << max(MAX_ORDER - 1, pageblock_order);
27
28         /* alignment should be aligned with order_per_bit */
29         if (!IS_ALIGNED(alignment >> PAGE_SHIFT, 1 << order_per_bit))
30                 return -EINVAL;
31
32         if (ALIGN(base, alignment) != base || ALIGN(size, alignment) !=
   size)
33                 return -EINVAL;
34
35         /*
36          * Each reserved area must be initialised later, when more kerne
   l
37          * subsystems (like slab allocator) are available.
38          */
39         cma = &cma_areas[cma_area_count];
40         cma->base_pfn = PFN_DOWN(base);
41         cma->count = size >> PAGE_SHIFT;
42         cma->order_per_bit = order_per_bit;
43         *res_cma = cma;
44         cma_area_count++;
45         totalcma_pages += (size / PAGE_SIZE);
46
47         return 0;
48 }
```

- if (cma_area_count == ARRAY_SIZE(cma_areas)) {
  - If there is no more space to allocate to cma_areas[], it prints and returns an error.
- if (!size || !memblock_is_region_reserved(base, size))
  - If size is 0 or if the request area is not in the reserve memblock, it returns an error.
- alignment = PAGE_SIZE << max(MAX_ORDER – 1, pageblock_order);
  - pageblock_order
    - 32bit ARM does not support CONFIG_HUGETLB_PAGE option.
    - MAXORDER(11=used by buddy)-1.
  - 4M = PAGE_SIZE(4K) << (MAXORDER(11) – 1)
- if (!IS_ALIGNED(alignment >> PAGE_SHIFT, 1 << order_per_bit))
  - Returns an error if alignment is not aligned by order_per_bit units.
- if (ALIGN(base, alignment) != base || ALIGN(size, alignment) != size)
  - If the start address and size of the request are not aligned by the alignment unit, an error is returned.
- Add a request to the cma_areas[], increasing the cma_area_count by 1 and increasing the number of pages added to the totalcma_pages.

The figure below shows that when created through the cma_init_reserved_mem() function, the size and base of the area must be aligned to 4M units and 2^order_per_bit pages.



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/cma-4.png)

Assign and release CMAs

cma_alloc()

mm/cma.c

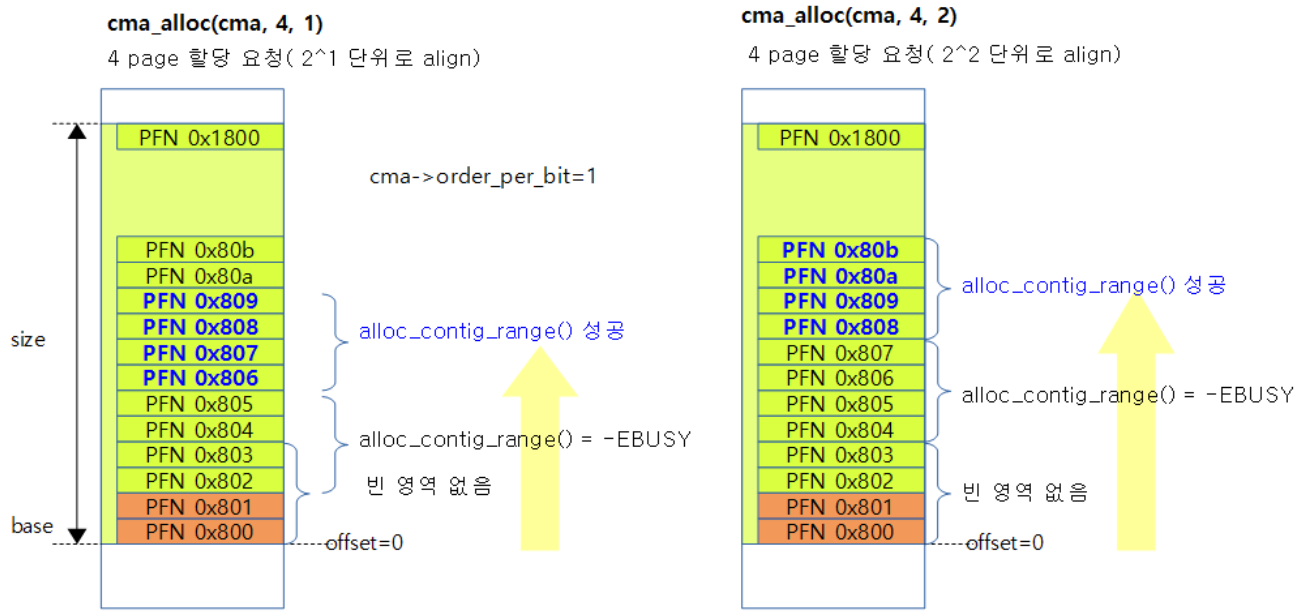
```
01 /**
02  * cma_alloc() - allocate pages from contiguous area
03  * @cma:   Contiguous memory region for which the allocation is performe
    d.
04  * @count: Requested number of pages.
05  * @align: Requested alignment of pages (in PAGE_SIZE order).
06  *
07  * This function allocates part of contiguous memory on specific
08  * contiguous memory area.
09  */
10 struct page *cma_alloc(struct cma *cma, int count, unsigned int align)
11 {
12         unsigned long mask, offset, pfn, start = 0;
13         unsigned long bitmap_maxno, bitmap_no, bitmap_count;
```

```
14          struct page *page = NULL;
15          int ret;
16
17          if (!cma || !cma->count)
18                  return NULL;
19
20          pr_debug("%s(cma %p, count %d, align %d)\n", __func__, (void *)c
    ma,
21                   count, align);
22
23          if (!count)
24                  return NULL;
25
26          mask = cma_bitmap_aligned_mask(cma, align);
27          offset = cma_bitmap_aligned_offset(cma, align);
28          bitmap_maxno = cma_bitmap_maxno(cma);
29          bitmap_count = cma_bitmap_pages_to_bits(cma, count);
30
31          for (;;) {
32                  mutex_lock(&cma->lock);
33                  bitmap_no = bitmap_find_next_zero_area_off(cma->bitmap,
34                                  bitmap_maxno, start, bitmap_count, mask,
35                                  offset);
36                  if (bitmap_no >= bitmap_maxno) {
37                          mutex_unlock(&cma->lock);
38                          break;
39                  }
40                  bitmap_set(cma->bitmap, bitmap_no, bitmap_count);
41                  /*
42                   * It's safe to drop the lock here. We've marked this re
    gion for
43                   * our exclusive use. If the migration fails we will tak
    e the
44                   * lock again and unmark it.
45                   */
46                  mutex_unlock(&cma->lock);
47
48                  pfn = cma->base_pfn + (bitmap_no << cma->order_per_bit);
49                  mutex_lock(&cma_mutex);
50                  ret = alloc_contig_range(pfn, pfn + count, MIGRATE_CMA);
51                  mutex_unlock(&cma_mutex);
52                  if (ret == 0) {
53                          page = pfn_to_page(pfn);
54                          break;
55                  }
56
57                  cma_clear_bitmap(cma, pfn, count);
58                  if (ret != -EBUSY)
59                          break;
60
61                  pr_debug("%s(): memory range at %p is busy, retrying\n",
62                           __func__, pfn_to_page(pfn));
63                  /* try again with a bit different memory target */
64                  start = bitmap_no + mask + 1;
65          }
66
67          pr_debug("%s(): returned %p\n", __func__, page);
68          return page;
69  }
```

- if (!cma || !cma->count)
  - Returns null if the CMA is null or there are no bits available for the CMA
- if (!count)
  - Returns null if there is no request count(bit)
- mask = cma_bitmap_aligned_mask(cma, align);

        - If align is greater than CMA->order_per_bit, the number of bits equal to the difference is made into mask bits 1 and returned.
            - 예) align=4(64K), cma->order_per_bit=2(16K)
                - 3 (Mask made with two bits as 1)
- offset = cma_bitmap_aligned_offset(cma, align);
    - Determine the size that cannot be used when aligning the starting address with 2 ^ align page as offset.
- bitmap_maxno = cma_bitmap_maxno(cma);
    - Find out the maximum number of bits that can be used in the bitmap.
- bitmap_count = cma_bitmap_pages_to_bits(cma, count);
    - Number of bitmap bits required for the request count (page)
- bitmap_no = bitmap_find_next_zero_area_off(cma->bitmap, bitmap_maxno, start, bitmap_count, mask, offset);
    - Find a blank spot in the beatmap and find the bitmap_no.
        - CMA->bitmap is searched and the size of the bits is masked, and if 0 is found, it is judged to be an empty space and the bitmap_no is known. (offset does not assign area)
- if (bitmap_no >= bitmap_maxno) {
    - If there is no space to add, return null.
- bitmap_set(cma->bitmap, bitmap_no, bitmap_count);
    - Set the number of bits bitmap_count bitmap_no to 1.
- pfn = cma->base_pfn + (bitmap_no << cma->order_per_bit);
    - PFN for the location you assign
    - e.g. base_pfn=0x1000, bitmap_no=1, order_per_bit=1
        - 0x1002 = 0x1000 + (1 << 1)
- ret = alloc_contig_range(pfn, pfn + count, MIGRATE_CMA);
    - It allocates all the pages it needs as MIGRATE_CMA and returns a pointer to the page structure if it succeeds. All existing movable pages in this area are migrated through the compaction process.
- cma_clear_bitmap(cma, pfn, count);
    - If the allocation fails, clear the bitmap again.
    - If the failure is not -EBUSY, it returns null.
- start = bitmap_no + mask + 1;
    - Try to allocate again at the next bitmap location.

In the figure below, the align unit is different to show how the page is assigned when the cma_alloc() function is called.



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/cma-5.png)

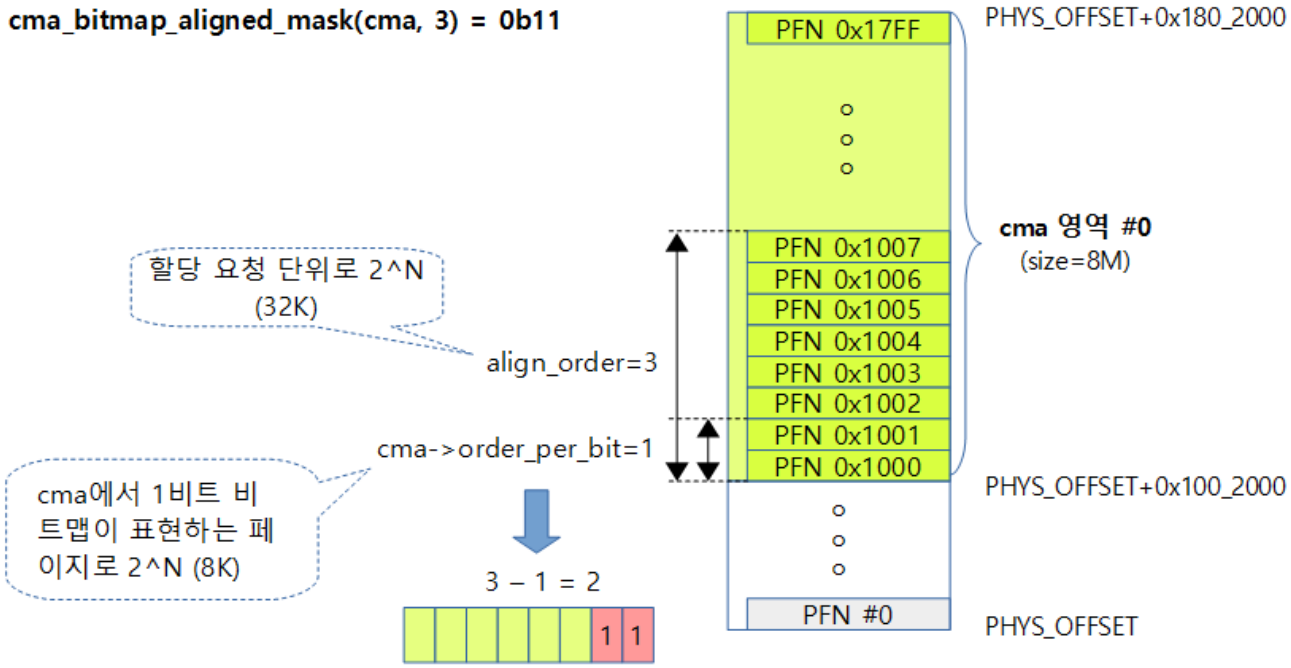
cma_bitmap_aligned_mask()

mm/cma.c

```
1 static unsigned long cma_bitmap_aligned_mask(struct cma *cma, int align_
  order)
2 {
3         if (align_order <= cma->order_per_bit)
4                 return 0;
5         return (1UL << (align_order - cma->order_per_bit)) - 1;
6 }
```

- Return by making the number of bits equal to the difference between align_order and CMA->order_per_bit equal to 1
  - If the page unit managed by the bit is greater than or equal to the requested align_order in that CMA entry, it returns 0.
  - e.g. align_order = 4 (64K increments), cma->order_per_bit = 2 (16K increments)
    - 3 (Set 2 beats to 1 to make a mask.)

The figure below shows that when you want to allocate a request unit of 2^3 pages, the mask value is equal to the bit difference for the existing CMA area that is managed in 2 page increments.



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/cma-3.png)

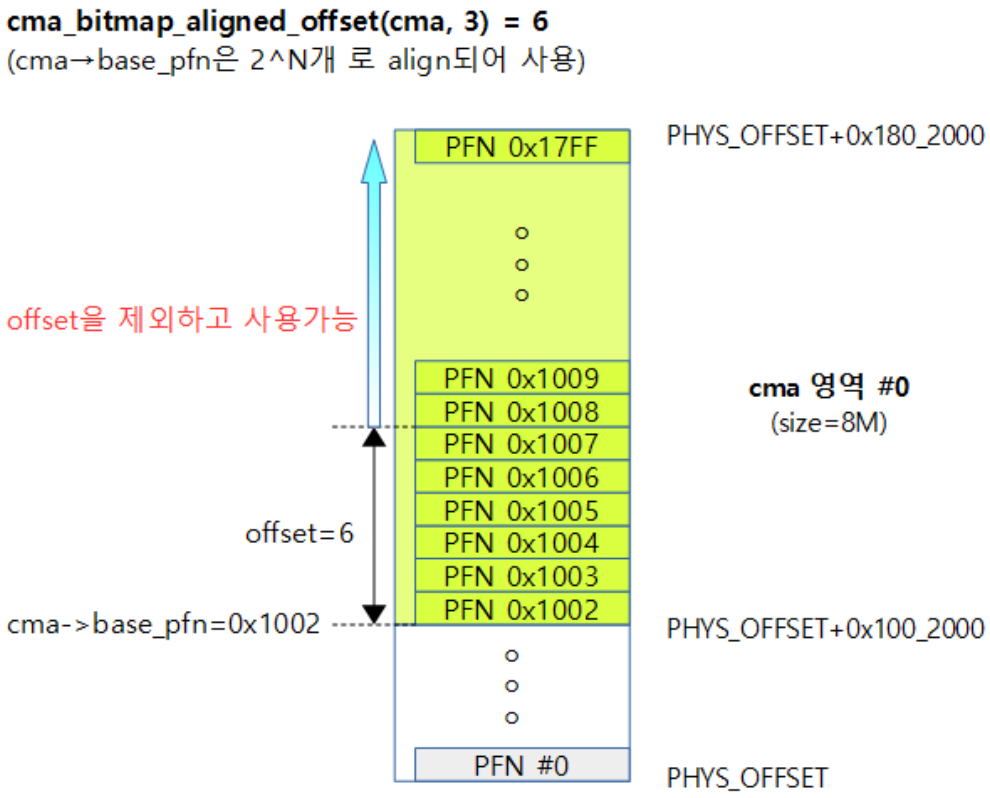
cma_bitmap_aligned_offset()

mm/cma.c

```
01 /*
02  * Find a PFN aligned to the specified order and return an offset repres
   ented in
03  * order_per_bits.
04  */
05 static unsigned long cma_bitmap_aligned_offset(struct cma *cma, int alig
   n_order)
06 {
07         if (align_order <= cma->order_per_bit)
08                 return 0;
09
10         return (ALIGN(cma->base_pfn, (1UL << align_order))
11                 - cma->base_pfn) >> cma->order_per_bit;
12 }
```

- If the align_order of the CMA->base_pfn is not in the requested align_order, it returns the offset required for the search.
  - If the page unit managed by the bit is greater than or equal to the requested align_order in that CMA entry, it returns 0.
  - Shift the starting pfn to the left by align_order, subtract the starting pfn, and shift the right by CMA->order_per_bit
    - e.g. 0x0001_2344, cma->order_per_bit = 2 (16K increments), align_order = 4 (64K increments)
      - 0xC(0x0001_2350 – 0x0001_2344) >> 2 = 3

The figure below shows that if the cma->base_pfn is not aligned by the number of 2^3 pages, the offset is determined by the difference.



(http://jake.dothome.co.kr/wp-content/uploads/2016/03/cma-2.png)

cma_bitmap_maxno()

mm/cma.c

```
1 static unsigned long cma_bitmap_maxno(struct cma *cma)
2 {
3         return cma->count >> cma->order_per_bit;
4 }
```

- The maximum number of bits used by a bitmap
  - 예) cma->count=800, cma->order_per_bit=2
    - 200 bits

cma_bitmap_pages_to_bits()

mm/cma.c

```
1 static unsigned long cma_bitmap_pages_to_bits(struct cma *cma,
2                                                unsigned long pages)
3 {
4         return ALIGN(pages, 1UL << cma->order_per_bit) >> cma->order_per
_bit;
5 }
```

- The number of bits to use in the bitmap required for the request page.
  - Returns the number of bits required when aligned to the minimum unit size used by the bitmap.
  - e.g. pages=5 (20K requests), cma->order_per_bit=2 (16K increments)
    - 2(16K x 2) = 8 >> 2

cma_release()

mm/cma.c

```
01 /**
02  * cma_release() - release allocated pages
03  * @cma:   Contiguous memory region for which the allocation is performe
   d.
04  * @pages: Allocated pages.
05  * @count: Number of allocated pages.
06  *
07  * This function releases memory allocated by alloc_cma().
08  * It returns false when provided pages do not belong to contiguous area
   and
09  * true otherwise.
10  */
11 bool cma_release(struct cma *cma, struct page *pages, int count)
12 {
13         unsigned long pfn;
14
15         if (!cma || !pages)
16                 return false;
17
18         pr_debug("%s(page %p)\n", __func__, (void *)pages);
19
20         pfn = page_to_pfn(pages);
21
22         if (pfn < cma->base_pfn || pfn >= cma->base_pfn + cma->count)
23                 return false;
24
25         VM_BUG_ON(pfn + count > cma->base_pfn + cma->count);
26
27         free_contig_range(pfn, count);
28         cma_clear_bitmap(cma, pfn, count);
29
30         return true;
31 }
```

- if (!cma || !pages)
  - Returns false if CMA is null or if the pages pointer is null.
- Returns false if the start physical address of the request page is outside the CMA area.
- free_contig_range(pfn, count)
  - Count (number of pages) from the PFN address in the CMA area.
- cma_clear_bitmap(cma, pfn, count);
  - Clear the page in that area from the bitmap.

free_contig_range()

mm/page_alloc.c

```
01 void free_contig_range(unsigned long pfn, unsigned nr_pages)
02 {
03         unsigned int count = 0;
04
05         for (; nr_pages--; pfn++) {
06                 struct page *page = pfn_to_page(pfn);
07
08                 count += page_count(page) != 1;
09                 __free_page(page);
```

```
10          }
11          WARN(count != 0, "%d pages are still in use!\n", count);
12  }
```

- From pfn, it loops around the requested number of pages and releases the pages.

cma_clear_bitmap()

mm/cma.c

```
01  static void cma_clear_bitmap(struct cma *cma, unsigned long pfn, int cou
    nt)
02  {
03          unsigned long bitmap_no, bitmap_count;
04
05          bitmap_no = (pfn - cma->base_pfn) >> cma->order_per_bit;
06          bitmap_count = cma_bitmap_pages_to_bits(cma, count);
07
08          mutex_lock(&cma->lock);
09          bitmap_clear(cma->bitmap, bitmap_no, bitmap_count);
10          mutex_unlock(&cma->lock);
11  }
```

- bitmap_no = (pfn – cma->base_pfn) >> cma->order_per_bit;
  - Get the bitmap number.
- bitmap_count = cma_bitmap_pages_to_bits(cma, count)
  - count to find the number of bits needed.
- bitmap_clear(cma->bitmap, bitmap_no, bitmap_count);
  - Clear the number of bitmap_count bits from the bitmap_no position in the bitmap.

Initialize CMA Zones

cma_init_reserved_areas()

mm/cma.c

```
01  static int __init cma_init_reserved_areas(void)
02  {
03          int i;
04
05          for (i = 0; i < cma_area_count; i++) {
06                  int ret = cma_activate_area(&cma_areas[i]);
07
08                  if (ret)
09                          return ret;
10          }
11
12          return 0;
13  }
14  core_initcall(cma_init_reserved_areas);
```

- core_initcall(cma_init_reserved_areas);
  - CONFIG_MODULE Depending on the kernel options, the location where it is called varies.
    - If you don't have CONFIG_MODULE settings
      - All registered initcall functions are called from the do_initcalls() function of the kernel_init thread.

- If you have CONFIG_MODULE settings
  - It's the same as the cma_init_reserved_areas module_init declaration inside the drivers. Therefore, when the CMA driver module is loaded, the cma_init_reserved_areas() function is called.

cma_activate_area()

mm/cma.c

```
01 static int __init cma_activate_area(struct cma *cma)
02 {
03         int bitmap_size = BITS_TO_LONGS(cma_bitmap_maxno(cma)) * sizeof
   (long);
04         unsigned long base_pfn = cma->base_pfn, pfn = base_pfn;
05         unsigned i = cma->count >> pageblock_order;
06         struct zone *zone;
07
08         cma->bitmap = kzalloc(bitmap_size, GFP_KERNEL);
09
10         if (!cma->bitmap)
11                 return -ENOMEM;
12
13         WARN_ON_ONCE(!pfn_valid(pfn));
14         zone = page_zone(pfn_to_page(pfn));
15
16         do {
17                 unsigned j;
18
19                 base_pfn = pfn;
20                 for (j = pageblock_nr_pages; j; --j, pfn++) {
21                         WARN_ON_ONCE(!pfn_valid(pfn));
22                         /*
23                          * alloc_contig_range requires the pfn range
24                          * specified to be in the same zone. Make this
25                          * simple by forcing the entire CMA resv range
26                          * to be in the same zone.
27                          */
28                         if (page_zone(pfn_to_page(pfn)) != zone)
29                                 goto err;
30                 }
31                 init_cma_reserved_pageblock(pfn_to_page(base_pfn));
32         } while (--i);
33
34         mutex_init(&cma->lock);
35         return 0;
36
37 err:
38         kfree(cma->bitmap);
39         cma->count = 0;
40         return -EINVAL;
41 }
```

Modules

The macro code below was moved to include/linux/init.h -> include/linux/module.h in kernel v2015.5-rc4 in May 2.

core_initcall()

include/linux/init.h

```
#define core_initcall(fn)               module_init(fn)
```

- If a CONFIG_MODULE is declared, call module_init().

module_init()

include/linux/init.h

```
/* Each module must use one module_init(). */
#define module_init(initfn)                                     \
        static inline initcall_t __inittest(void)               \
        { return initfn; }                                      \
        int init_module(void) __attribute__((alias(#initfn)));
```

- initcall_t
  - It is a function pointer without arguments, as follows:
  - typedef int (*initcall_t)(void);
- Call the argument initfn function and return the result as an int type.
- int init_module(void) __attribute__((alias(#initfn)));
  - Use the alias compiler attribute to call the argument initfn when using the init_module() function name as alias.

Structs and Key Variables

struct cma

mm/cma.c

```
struct cma {
        unsigned long   base_pfn;
        unsigned long   count;
        unsigned long   *bitmap;
        unsigned int order_per_bit; /* Order of pages represented by one bit */
        struct mutex    lock;
};
```

- base_pfn
  - The PFN number of the starting physical address
- count
  - Number of Manage Pages
- bitmap
  - Bitmap data
- order_per_bit
  - The order page represented by 1 bit of the bitmap (2^N pages, 0=1 pages, 1=2 pages, 2=4 pages, ...)
- lock
  - Used for synchronization when manipulating bitmaps

Global Variables

mm/cma.c

```
1 static struct cma cma_areas[MAX_CMA_AREAS];
2 static unsigned cma_area_count;
3 static DEFINE_MUTEX(cma_mutex);
```

- cma_areas[]
  - If the CONFIG_CMA_AREAS kernel option is set (1 + CONFIG_CMA_AREAS), otherwise it is 0.
- cma_area_count
  - The number of items used by the cma_areas[] array.
- cma_mutex
  - Used for synchronization when assigning CMA zones

mm/page_alloc.c

```
1 unsigned long totalcma_pages __read_mostly;
```

- totalcma_pages
  - Number of pages used in CMA

consultation

- CMA: generalize CMA reserved area management functionality (https://github.com/torvalds/linux/commit/a254129e8686bff7a340b58f35241b04927e81c0)
- CMA(Contiguous Memory Allocator) for DMA (http://jake.dothome.co.kr/cma-dma) | 문c
- Contiguous Memory Allocator (https://lwn.net/Articles/477343/) | LWN.net
- A reworked contiguous memory allocator (http://lwn.net/Articles/447405/) | LWN.net
- CMA and ARM (http://lwn.net/Articles/450286/) | LWN.net
- Contiguous memory allocation for drivers (http://lwn.net/Articles/396702/) | LWN.net
- CMA documentation file (http://lwn.net/Articles/396707/) | LWN.net
- A deep dive into CMA (https://lwn.net/Articles/486301/) | LWN.net
- Linux Kernel CMA | Mark Veltzer – Download odf file (https://www.google.co.kr/url?sa=t&rct=j&q=&esrc=s&source=web&cd=37&ved=0ahUKEwjA-pK9iqnMAhXLI5QKHVY4DYo4HhAWCEYwBg&url=http%3A%2F%2Fkerneltlv.com%2Fwp-content%2Fuploads%2F2016%2F01%2Flinux_kernel_cma.odp&usg=AFQjCNFnAWGV92Q3Q0K300EG7F0RT6I9Wg&sig2=d6btRh6saKoO12Ihpte5qg)
- Document/DMA-API-HOWTO.txt (http://www.kernel.org/doc/Documentation/DMA-API-HOWTO.txt) | kernel.org
- Document/DMA-API.txt (http://www.kernel.org/doc/Documentation/DMA-API.txt) | kernel.org
- Guaranteed Contiguous Memory Allocator | Sungjae Park – Download PDF (https://www.oss.kr/editor/file/c4cb5e93/download/bc1c4329-82c9-447e-9e93-2548518f196a)

6 thoughts to “CMA(Contiguous Memory Allocator)”

TUNDRA
2020-03-06 17:07 (http://jake.dothome.co.kr/cma/#comment-235462)

Hello Author,
Could you please explain more detail about two things:
– what is the purpose of mask (cma_bitmap_aligned_mask). ?
– Why do we need align when allocate?

Thank you in advance!

RESPONSE (/CMA/?REPLYTOCOM=235462#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2020-03-06 21:41 (http://jake.dothome.co.kr/cma/#comment-235495)

The CMA allocator allocates memory for the device to use for DMA purposes, which usually requires alignment due to DMA HW start address constraints.

case 1) Example of 8M buffer allocation aligned in 4M units according to user’s intention
Memory range allocated: 0x10C0_0000 to 0x1140_0000

case 2) Example of 8M buffer allocation that the device cannot use because of misalignment
Memory range allocated: 0x1022_2000 to 0x10A2_2000

Thank you.

RESPONSE (/CMA/?REPLYTOCOM=235495#RESPOND)

TUNDRA
2020-03-07 10:08 (http://jake.dothome.co.kr/cma/#comment-235591)

thank you so much

RESPONSE (/CMA/?REPLYTOCOM=235591#RESPOND)

ANDO KI (HTTP://WWW.FUTURE-DS.COM)
2020-03-19 16:54 (http://jake.dothome.co.kr/cma/#comment-237442)

It is a very helpful post and thank you.
Could you kindly give answer to questions?

1. Is there any way (API) to figure out the offset and size of CMA, which is displayed at the dmesg.
– In kernel module
– In user program
2. Is it possible to make non-cacheable for the CMA area?
– I want to fill the CAM by PCIe DMA and want to disable while DMA fills the CMA memory.
3. Is it possible to allocate a huge one such as 20GByte or ore (of course, there is sufficient main memory, e.g., 36GByte on board)
4. Is it possible to refer the CMA from user program using mmap() on /dev/mem.

RESPONSE (/CMA/?REPLYTOCOM=237442#RESPOND)

MOON YOUNG-IL (HTTP://JAKE.DOTHOME.CO.KR)
2020-03-20 00:03 (http://jake.dothome.co.kr/cma/#comment-237489)

Hi ANDO KI,

I answered your question as follows:

1.Is there any way (API) to figure out the offset and size of CMA, which is displayed at the dmesg.
– In kernel module
– In user program

-> There are two ways. Let's check each method.
1) use "cma=xxM" command line kernel parameter
Example) "cma=16M"

* If you use dmsg, you can see the following log.
cma: Reserved 16 MiB at 0x000000007f000000
Kernel command line: cma = 16M root = / dev / vda rw rootwait

2) use device tree
Example)

```
reserved-memory {
        # address-cells = <2>;
        # size-cells = <2>;
        ranges;

        linux, cma {
                compatible = "shared-dma-pool";
                reusable;
                size = <0 0x02000000>;
                linux, cma-default;
        };
};
```

* If you use dmsg, you can see the following log.
Reserved memory: created CMA memory pool at 0x000000007e000000, size 32 MiB
OF: reserved mem: initialized node linux, cma, compatible id shared-dma-pool

* There is also a way to check the device tree after booting the Linux kernel.
$ cd /sys/firmware/devicetree/base/reserved-memory/linux,cma
$ ls
compatible linux, cma-default name reusable size
$ xxd size
00000000: 0000 0000 0200 0000 ……..

2. Is it possible to make non-cacheable for the CMA area?
– I want to fill the CAM by PCIe DMA and want to disable while DMA fills the CMA memory.

-> yes. You can use dma_alloc_coherent() or dma_alloc_wc() api.

3.Is it possible to allocate a huge one such as 20GByte or ore (of course, there is sufficient main memory, e.g., 36GByte on board)

-> case ARM64)
When CONFIG_ZONE_DMA32 is declared for compatibility with 32-bit pci dma devices, the maximum dma area is limited to 4G.
When using a pcie device that supports 64-bit, the cma area can be largely captured without 4G limitation.

4. Is it possible to refer the CMA from user program using mmap () on / dev / mem.

-> I have never used it, but it will be possible.
Look for the DMA_ATTR_NO_KERNEL_MAPPING attribute in dma_mmap_attrs().

More details on DMA can also be found in my blog.
DMA -1- (Basic)
DMA -2- (DMA Coherent Memory)
DMA -3- (DMA Pool)
DMA -4- (DMA Mapping)
DMA -5- (IOMMU)

응답 (/CMA/?REPLYTOCOM=237489#RESPOND)

문영일 (HTTP://JAKE.DOTHOME.CO.KR)
2020-03-21 14:16 (http://jake.dothome.co.kr/cma/#comment-237747)

One more things to know about DMA performace

1) Using SRAM buffer – Bestest performance
If the system uses a dedicated SRAM buffer as fast as the cache for DMA, no cache mapping is required, so no cache coherent is needed.

2) HW DMA cache coherent – Best performance
If the system supports HW DMA cache coherent (eg CCI-500 and DMA devices using it), using cached (L1, L2, ...) system memory as a DMA buffer is no problem.

3) Use dma mapping API – High performance
If you need to use DMA at high speed, you should use dma_alloc_contiguous () instead of dma_alloc_cohernet () or dma_alloc_wc () to use cache. However, dma mapping (sync) API should be used in this case.

4) Use Write combine mapping – Slow performance
When the system memory is changed to the memory mapping for write combine using the dma_alloc_coherent () or dma_alloc_wc () command, the cache cohent problem does not occur because the cache is not used and only the buffer is used. Instead, the cache is not used, so performance is sacrificed.

응답 (/CMA/?REPLYTOCOM=237747#RESPOND)

댓글 남기기

이메일은 공개되지 않습니다. 필수 입력창은 * 로 표시되어 있습니다

댓글

이름 *

이메일 *

웹사이트

댓글 작성

‹ 페이지 테이블 API – ARM32 (http://jake.dothome.co.kr/pt-api/)

CMA(Contiguous Memory Allocator) for DMA > (http://jake.dothome.co.kr/cma-dma/)

문c 블로그 (2015 ~ 2023)