CMA (Contiguous Memory Allocator) in the Linux kernel is a framework designed to allocate large contiguous blocks of physical memory, primarily for devices that require physically contiguous memory, such as multimedia devices (e.g., GPUs, camera drivers, etc.). It was introduced to address the challenges of allocating contiguous memory dynamically without causing system memory fragmentation.

* Devices like video encoders, GPUs, cameras, and network adapters often require physically contiguous memory for direct DMA (Direct Memory Access) operations.
* **Traditional Approach:** Earlier, dma\_alloc\_coherent() was used to allocate physically contiguous memory, but it had limitations like increased memory fragmentation, high memory reservation, and wastage of memory.
* **CMA Solution:** CMA allows reserving a memory region at boot time (using a reserved memory region) and dynamically allocates large contiguous blocks to devices without severe fragmentation.

**How CMA Works Internally (Full Kernel Call Flow)?**

CMA works in conjunction with the **buddy allocator** and **page allocator**. During boot, CMA reserves a large block of contiguous memory based on kernel parameters or device tree settings. This block is **not exclusively reserved**; the kernel can still use it for movable page allocations. Once a device requests memory, CMA tries to reclaim the block and allocate it.

**Flow at Boot Time:**

* drivers/base/dma-contiguous.c → cma\_declare\_contiguous()
* This function sets up the CMA memory region.

int cma\_declare\_contiguous( phys\_addr\_t base, phys\_addr\_t size, phys\_addr\_t limit, bool fixed, struct cma \*\*res\_cma)

* 1. **Memory Allocation (During Runtime)**

When a driver requests memory (via dma\_alloc\_coherent() or dma\_alloc\_attrs()), the kernel attempts to allocate memory from the CMA region. If the memory is occupied, CMA triggers **page migration** (if pages are movable). If pages are unmovable (like pinned kernel pages), allocation fails.

**Flow During Allocation:**

* kernel/dma/contiguous.c → dma\_alloc\_from\_contiguous()
* Internally uses cma\_alloc().

void \*dma\_alloc\_from\_contiguous( struct device \*dev, size\_t size, dma\_addr\_t \*dma\_handle)

**Key Functions Involved:**

* **cma\_alloc()** – Tries to allocate contiguous pages.
* **alloc\_contig\_range()** – Allocates contiguous pages and migrates movable pages.
* **migrate\_page()** – Migrates user-space pages to free up CMA memory.

**3.3 Page Migration (If CMA Region Is Occupied)**

If the CMA region has movable user-space pages (like file cache, anonymous memory), CMA will migrate them. Uses **page migration API** like:

int migrate\_page(struct page \*oldpage, struct page \*newpage, enum migrate\_mode mode);

If the pages are unmovable (kernel allocations, pinned memory), the allocation fails.

* 1. **Releasing CMA Memory**

When the driver releases the memory (dma\_free\_coherent()), the pages are returned to the CMA region and marked as movable. This avoids long-term memory fragmentation.

**Release Flow:**

* dma\_free\_from\_contiguous() → cma\_release() → free\_contig\_range()

**4. CMA Configuration and Device Tree Bindings**

You can configure CMA in multiple ways:

**4.1 Kernel Command Line Parameters**

cma=512M

Reserves **512MB** of contiguous memory for CMA.

**4.2 Device Tree Configuration**

In device tree, you can define reserved memory like:

reserved-memory {

cma\_region: cma@0 {

compatible = "shared-dma-pool";

reg = <0x0 0x80000000 0x0 0x20000000>;

reusable;

};

};

The device driver can request this memory via:

dma\_declare\_coherent\_memory();

**5. CMA APIs and Usage**

Core CMA APIs used by drivers:

|  |  |
| --- | --- |
| **API** | **Description** |
| cma\_alloc() | Allocates contiguous memory from CMA. |
| cma\_release() | Releases CMA memory. |
| alloc\_contig\_range() | Allocates a contiguous physical range. |
| dma\_alloc\_coherent() | Allocates DMA-coherent memory (uses CMA internally). |

struct page \*page = cma\_alloc(my\_cma, size, GFP\_KERNEL);

void \*vaddr = page\_address(page);

dma\_addr\_t dma\_addr = virt\_to\_phys(vaddr);

**6. CMA Interaction with Buddy Allocator and Memory Zones**

CMA relies on the **buddy allocator** for memory management. It uses **ZONE\_MOVABLE** to place reclaimable pages (like user-space page cache). During high memory pressure, the kernel can migrate these pages to free up CMA.

**Flow Summary:**

* If a driver requests memory → CMA tries to allocate.
* If memory is occupied → Page migration happens.
* If unmovable pages exist → Allocation fails.

**7. CMA Limitations and Issues**

Although CMA is powerful, it has certain limitations:

1. **Unmovable Pages Block Allocation:**
   * If kernel pages are pinned (like slab cache, pinned user pages), CMA allocation fails.
2. **High Memory Fragmentation:**
   * CMA tries to migrate pages, but if they are unmovable, fragmentation increases.
3. **Fixed Memory Allocation:**
   * The memory is fixed at boot time. Changing it requires a reboot.
4. **No Dynamic Region Growth:**
   * CMA cannot dynamically expand beyond its predefined region.

**8. Best Practices for Using CMA in Linux Drivers**

1. Always use standard APIs (dma\_alloc\_coherent()) instead of cma\_alloc() for portability.
2. Avoid unmovable kernel memory allocations to maximize CMA efficiency.
3. Pre-calculate the worst-case contiguous memory requirements and reserve accordingly.
4. Use /proc/meminfo or /sys/kernel/debug/cma to monitor CMA usage:

cat /proc/meminfo | grep Cma

1. Avoid long-term pinned pages (e.g., get\_user\_pages()) inside the CMA region.

|  |  |
| --- | --- |
| **Aspect** | **CMA Behavior** |
| **Purpose** | Provides large physically contiguous memory to devices (like GPUs, cameras, etc.) |
| **Allocation** | Uses cma\_alloc() or dma\_alloc\_coherent() |
| **Memory Management** | Uses page migration, buddy allocator, and movable zones |
| **Failure Case** | Allocation fails if unmovable pages block the region |
| **Configuration** | Via kernel cmdline or device tree |

* In **real-time multimedia devices** (like camera capture, video playback, etc.), CMA is critical to ensure zero-latency memory access.
* Major drivers (like V4L2, GPU, and multimedia drivers) extensively use CMA to optimize performance.