Continuous Memory Allocator
Allocating big chunks of physically contiguous memory

Michał Nazarewicz
mina86@mina86.com

Google

November 6, 2012
Outline

1. Introduction
   - Why physically contiguous memory is needed
   - Solutions to the problem

2. Usage & Integration
   - Using CMA from device drivers
   - Integration with the architecture
   - Private & not so private CMA regions

3. Implementation
   - Page allocator
   - CMA implementation
   - CMA problems and future work
Outline

1. Introduction
   - Why physically contiguous memory is needed
   - Solutions to the problem

2. Usage & Integration
   - Using CMA from device drivers
   - Integration with the architecture
   - Private & not so private CMA regions

3. Implementation
   - Page allocator
   - CMA implementation
   - CMA problems and future work
Modern CPUs have MMU.

- Virtual $\rightarrow$ physical address.

Virtually contiguous $\not\iff$ physically contiguous.

So why bother?
Modern CPUs have MMU.
- Virtual $\rightarrow$ physical address.
- Virtually contiguous $\not\implies$ physically contiguous.

MMU stands behind CPU.
- There are other chips in the system.
- Some require large buffers.
  - 5-megapixel camera anyone?
- On embedded, there’s plenty of those.
The mighty DMA

- DMA can do vectored I/O.
- Gathering buffer from scattered parts.
  - Hence also another name: DMA scatter-gather.
- Contiguous for the device $\Rightarrow$ physically contiguous.
- So why bother?
The mighty DMA

- DMA can do vectored I/O.
- Gathering buffer from scattered parts.
  - Hence also another name: DMA scatter-gather.
- Contiguous for the device $\Leftrightarrow$ physically contiguous.
- DMA may lack vectored I/O support.
- DMA can do linear access only.
The mighty I/O MMU

- What about an I/O MMU?
  - Device → physical address.
- Same deal as with CPU’s MMU.
- So why bother?
The mighty I/O MMU

- What about an I/O MMU?
  - Device → physical address.
- Same deal as with CPU’s MMU.

- I/O MMU is not so common.
- I/O MMU takes time.
- I/O MMU takes power.
Reserve and assign at boot time

- Reserve memory during system boot time.
  - `mem` parameter.
  - Memblock / bootmem.

- Assign buffers to each device that might need it.
- While device is not being used, memory is wasted.
Reserve and allocate on demand

- Reserve memory during system boot time.
- Provide API for allocating from that reserved pool.
- Less memory is reserved.
- But it’s still wasted.

- bigphysarea
- Physical Memory Manager
Reserve but give back

- Reserve memory during system boot time.
- Give it back
  - but set it up so only movable pages can be allocated.
- Provide API for allocating from that reserved pool.
- Migrate pages on allocation.

- Contiguous Memory Allocator
Outline

1 Introduction
   - Why physically contiguous memory is needed
   - Solutions to the problem

2 Usage & Integration
   - Using CMA from device drivers
   - Integration with the architecture
   - Private & not so private CMA regions

3 Implementation
   - Page allocator
   - CMA implementation
   - CMA problems and future work
Using CMA from device drivers

- CMA is integrated with the DMA API.
- If device driver uses the DMA API, nothing needs to be changed.
- In fact, device driver should always use the DMA API and never call CMA directly.
void *my_dev_alloc_buffer(
    unsigned long size_in_bytes, dma_addr_t *dma_addrp)
{
    void *virt_addr = dma_alloc_coherent(
        my_dev,
        size_in_bytes,
        dma_addrp,
        GFP_KERNEL);
    if (!virt_addr)
        dev_err(my_dev, "Allocation failed.");
    return virt_addr;
}
Releasing memory from device driver

Freeing

```c
void *my_dev_free_buffer(
    unsigned long size, void *virt, dma_addr_t dma)
{
    dma_free_coherent(my_dev, size, virt, dma);
}
```
Documentation

- Documentation/DMA-API-HOWTO.txt
- Documentation/DMA-API.txt
- Linux Device Drivers, 3rd edition, chapter 15.
  - http://lwn.net/Kernel/LDD3/
Integration with the architecture

- CMA needs to be integrated with the architecture.
- Memory needs to be reserved.
- There are early fixups to be done. Or not.
- The DMA API needs to be made aware of CMA.
- And Kconfig needs to be instructed to allow CMA.
Memory reservation

- Memblock must be ready, page allocator must not.
- On ARM, `arm_memblock_init()` is a good place.
- All one needs to do, is call `dma_contiguous_reserve()`.

```c
void __init dma_contiguous_reserve(
    phys_addr_t limit);
```

`limit` Upper limit of the region (or zero for no limit).
Reserving memory on ARM

```c
if (mdesc->reserve)
    mdesc->reserve();

+/*
+ * reserve memory for DMA contiguous allocations,
+ * must come from DMA area inside low memory
+ */
+dma_contiguous_reserve(min(arm_dma_limit, arm_lowmem_limit));
+
arm_memblock_steal_permitted = false;
memblock_allow_resize();
memblock_dump_all();
```
Early fixups

- On ARM
  - cache is not coherent, and
  - having two mappings with different cache-ability gives undefined behaviour.

- Kernel linear mapping uses huge pages.
- So on ARM an “early fixup” is needed.
  - This fixup alters the linear mapping so CMA regions use 4 KiB pages.
  - The fixup is defined in `dma_contiguous_early_fixup()` function
    - which architecture needs to provide
    - with declaration in a `asm/dma−contiguous.h` header file.
Early fixups, cont.

No need for early fixups

```c
#ifndef ASM_DMA_CONTIGUOUS_H
#define ASM_DMA_CONTIGUOUS_H

#ifdef __KERNEL__

#include <linux/types.h>
#include <asm-generic/dma-contiguous.h>

static inline void
dma_contiguous_early_fixup(phys_addr_t base, unsigned long size)
{
    /* nop, no need for early fixups */
}

#endif /* __KERNEL__ */
#endif
```

The DMA API needs to be modified to use CMA.
CMA most likely won’t be the only one.
Allocating CMA memory

Allocate

```c
struct page *dma_alloc_from_contiguous(
    struct device *dev,
    int count,
    unsigned int align);
```

dev  Device the allocation is performed on behalf of.

count  *Number of pages* to allocate. Not number of bytes nor order.

align  Order which to align to. Limited by Kconfig option.

Returns page that is the first page of count allocated pages.

It's not a compound page.
Releasing CMA memory

```
bool dma_release_from_contiguous(
    struct device *dev,
    struct page *pages,
    int count);
```

- **dev** Device the allocation was performed on behalf of.
- **pages** The first of allocated pages. As returned on allocation.
- **count** Number of allocated pages.

Returns `true` if memory was freed (ie. was managed by CMA) or `false` otherwise.
Let it compile!

- There’s one thing that needs to be done in Kconfig.
- Architecture needs to select `HAVE_DMA_CONTIGUOUS`.
- Without it, CMA won’t show up under “Generic Driver Options”.
- Architecture may also select CMA to force CMA in.
Default CMA region

- Memory reserved for CMA is called CMA region or CMA context.
- There’s one default context devices use.
- So why does `dma_alloc_from_contiguous()` take device as an argument?
- There may also be per-device or private contexts.
What is a private region for?

- Separate a device into its own pool.
  - May help with fragmentation.
  - For instance big vs small allocations.
  - Several devices may be grouped together.
- Use different contexts for different purposes within the same device.
  - Simulating dual channel memory.
  - Big and small allocations in the same device.
Declaring private regions

```c
int dma_declare_contiguous(
  struct device *dev,
  unsigned long size,
  phys_addr_t base,
  phys_addr_t limit);
```

- **dev** Device that will use this region.
- **size** *Size in bytes* to allocate. Not pagas nor order.
- **base** Base address of the region (or zero to use anywhere).
- **limit** Upper limit of the region (or zero for no limit).

Returns zero on success, negative error code on failure.
Region shared by several devices

- The API allows to assign a region to a single device.
- What if more than one device is to use the same region.
- It can be easily done via “copying” the context pointer.
Region shared by several devices, cont

Copying CMA context pointer between two devices

```c
static int __init foo_set_up_cma_areas(void) {
    struct cma *cma;
    cma = dev_get_cma_area(device1);
    dev_set_cma_area(device2, cma);
    return 0;
}
postcore_initcall(foo_set_up_cma_areas);
```
CMA uses a one-to-many mapping from device structure to CMA region.

As such, one device can only use one CMA context. . .

. . . unless it uses more than one device structure.

That’s exactly what S5PV110’s MFC does.
Outline

1 Introduction
   • Why physically contiguous memory is needed
   • Solutions to the problem

2 Usage & Integration
   • Using CMA from device drivers
   • Integration with the architecture
   • Private & not so private CMA regions

3 Implementation
   • Page allocator
   • CMA implementation
   • CMA problems and future work
Linux kernel memory allocators

- memblock
- kmalloc()
- kmem_cache
- vmalloc()
- mempool
- page allocator
- the DMA API

- memblock gives memory to page allocator
- kmalloc() uses page allocator
- kmem_cache uses page allocator
- mempool may use page allocator
- the DMA API may use page allocator
Linux kernel memory allocators

- memblock
  - gives memory to
  - page allocator

- page allocator
  - uses
  - kmalloc()
  - vmalloc()

- kmem_cache
  - uses
  - mempool

- the DMA API
  - may use
Buddy allocator

- Page allocator uses buddy allocation algorithm.
  - Hence different names: buddy system or buddy allocator.
- Allocations are done in terms of orders.
- User can request order from 0 to 10.
- If best matching page is too large, it’s recursively split in half (into two buddies).
- When releasing, page is merged with its buddy (if free).
Pages and page blocks, cont

- 32 × 0-order pages
- 16 × 1-order pages
- 8 × 2-order pages
- 4 × 3-order pages
- 2 × 4-order pages
- 1 × 5-order pages
- 32 × 5-order pages
- 1 × 10-order pages

pageblock
max-order page
Migrate types

- On allocation, user requests an unmovable, a reclaimable or a movable page.
  - For our purposes, we treat reclaimable as unmovable.
- To try keep pages of the same type together, each free page and each page block has a migrate type assigned.
  - But allocator will use fallback types.
  - And migrate type of a free page and page blocks can change.
- When released, page takes migrate type of pageblock it belongs to.
Interaction of CMA with Linux allocators

memblock

page allocator

cma

the DMA API

Usage & Integration
CMA migrate type

- CMA needs guarantees that large number of contiguous pages can be migrated.
  - 100% guarantee is of course never possible.

- CMA introduced a new migrate type.
  - MIGRATE_CMA

- This migrate type has the following properties:
  - CMA pageblocks never change migrate type.\(^1\)
  - Only movable pages can be allocated from CMA pageblocks.

\(^1\) Other than while CMA is allocating memory from them.
Preparing CMA region

- At the boot time, some of the memory is reserved.
- When page allocator initialises, that memory is released with CMA’s migrate type.
- This way, it can be used for movable pages.
  - Unless the memory is allocated to a device driver.
- Each CMA region has a bitmap of “CMA free” pages.
  - “CMA free” page is one that is not allocated for device driver.
  - It may still be allocated as a movable page.
Allocation

Start → Choose range of pages to allocate

- Found? (Yes) → Isolate pageblocks containing range
  - Succeeded? (Yes) → Try to migrate
  - Succeeded? (No) → Undo isolate

- Found? (No) → Choose another range of pages
  - Allocation failed → Stop
  - Allocation succeeded → Undo isolate

Mark pages in range as non-free
Migration

- Pages allocated as movable are set up so that they can be migrated.
  - Such pages are only references indirectly.
  - Examples are anonymous process pages and disk cache.

- Roughly speaking, migration consists of:
  1. allocating a new page,
  2. copying contents of the old page to the new page,
  3. updating all places where old page was referred, and
  4. freeing the old page.

- In some cases, content of movable page can simply be discarded.
Problems

- `get_user_pages()` makes migration impossible.
- ext4 does not support migration of journal pages.
- Some filesystems are not good on migration.
Future work

- Only swap.
- Transcendent memory.
- POSIX_FADV_VOLATILE.
Thank you!

Michał Nazarewicz
mina86@mina86.com
http://mina86.com/cma/