LG Electronics
CTO Software Platform Laboratory
Minchan Kim <minchan.kim@lge.com>
Contents

- Background
- Problems
- GCMA Basic
- GCMA Pros and Cons
Background

• Need for physical contiguous memory for a long time
  – Reserve during boot
  – Array vs. malloc problem

• What is CMA (Contiguous memory allocator)?
  – Allow reserved memory to be able to be migrated out

• How it works
  – Reserves an amount of memory during boot
  – Allows it to be allocated for only movable pages
  – Migrates and/or discards them out if some drivers ask
CMA Problems - I

• No guarantee of successful allocation
  – Any one in kernel can pin any one of movable pages for a long time – get_user_pages
  – Object dependency – get_page

• High latency of the allocation
  – Migration overhead (rmap overhead, memory copy and LRU churn)
  – Reclaim overhead, sometime OOM
  – Stall caused by dirty page write
  – Swap out/throw workingset pages out
CMA Problems - II

• Code complexity
  – Add a ton of logics in core MM
  – Fragile hooks in core MM made lots of bugs
  – FS/MM/Driver dependency
Guaranteed CMA Basic

• Goals
  – Request of allocation should be always successful
  – The latency of allocation should be minimized

• How it works
  – Reserves an amount of memory during boot
  – Allows it to be allocated for the [least recently used] out-of-kernel pages
  – Discards and/or migrates them out if some driver ask
GCMA Pros

• Guarantee of successful allocation
  – GCMA pages are out of kernel's control

• Low latency
  – Least recently used pages by cleancache/frontswap
  – How to handle [ephemeral|persistent] pages
    • discard, write back, put back
  – No migration/reclaim overhead

• Complexity
  – Never touch core MM, all of logics are self-contained
  – Flexible for adding policy
GCMA Cons

- Low cache hit ratio imposes memcpy + reclaim cost
- Moves page reclaim overhead from CMA allocation time to runtime
GCMA
Implementation Details and Performance Evaluation

SeongJae Park <sj38.park@gmail.com>
Nice To Meet You

SeongJae Park <sj38.park@gmail.com>
dcslab@snu.ac.kr

Global Open Frontier Lab
**TL;DR**

- **GCMA is**
  - Abbrev of Guaranteed Contiguous Memory Allocator
  - Guarantees
    - Allocation success
    - Minimized latency
GCMA: Implementation
GCMA Basic Idea

1. Reserves an amount of memory during boot

2. Allow the memory to *contig allocation* and *LRU / easily discardable pages*

3. Drain *LRU / easily discardable pages* for *contig allocation* if necessary
GCMA vs CMA: Similarity

1. Reserves an amount of memory during boot

2. Allow the memory to *contig allocation* and *LRU / easily discardable pages*

3. Drain *LRU / easily discardable pages* for *contig allocation* if necessary
GCMA vs CMA: Similarity

1. Reserves an amount of memory during boot

2. Allow the memory to contig allocation and LRU / easily discardable pages

3. Drain LRU / easily discardable pages for contig allocation if necessary

So, GCMA simply use CMA code rather than re-inventing the wheel
GCMA Architecture

GCMA is integrated in CMA

CMA Interface

CMA Internal

Page Migration

GCMA Interface

GCMA Internal

LRU / Easily Discardable Pages Manager
GCMA Inside CMA

GCMA uses CMA reserved area

```c
struct cma {
    unsigned long   base_pfn;
    unsigned long   count;
    unsigned long   *bitmap;
    struct gcma     *gcma;
};
```

GCMA allocates free contig area which CMA found

```c
pfn = cma->base_pfn + (bitmap_no << cma->order_per_bit);
mutex_lock(&cma_mutex);

if (cma->gcma)
    ret = gcma_alloc_contig(cma->gcma, pfn, count);
else
    ret = alloc_contig_range(pfn, pfn + count, MIGRATE_CMA);
```
GCMA Basic Idea: Done

1. Reserves an amount of memory during boot

2. Allow the memory to \textit{contig allocation} and \textit{LRU / easily discardable pages}

3. Drain \textit{LRU / easily discardable pages} for \textit{contig allocation} if necessary
GCMA Basic Idea: TODOs

1. Reserves an amount of memory during boot

2. Allow the memory to contig allocation and LRU / easily discardable pages

3. Drain LRU / easily discardable pages for contig allocation If necessary
Background: Frontswap

Hook layer between Memory & Swap-Device

A page can be stored inside Frontswap before going to Swap-Device if possible

Fast, Narrow

Memory

Frontswap Front-end

Backend

Swap Device

Slow, Large
Page Inside Frontswap

- Least Recently Used

- Could be free
  - after write-back
  - as soon as want if it works as write-through (The data is already in swap-device)
  - after put-back (It would come back to frontswap again soon)
Page Inside Frontswap

- Least Recently Used

- Could be free
  - after write-back
  - as soon as want if it works as write-through
    (The data is already in swap-device)

1. Reserves an amount of memory during boot

2. Allow the memory to `contig allocation` and `LRU / easily discardable pages`

3. Drain `LRU / easily discardable pages` for `contig allocation` if necessary
Page Inside Frontswap

- Least Recently Used
  - Could be free
    - after write-back
    - as soon as want if it works as write-through
      (The data is already in swap-device)

1. Reserves an amount of memory during boot

2. Allow the memory to contig allocation and LRU / easily discardable pages

3. Drain LRU / easily discardable pages for contig allocation If necessary
Frontswap Backend Using GCMA

- Allow write-through Frontswap-backend page could be allocated from GCMA
  - Enhance memory efficiency

- If the region with the Frontswap-backend page should be allocated for CMA guest
  - Free the page and use it
  - The page could be free safely because the Frontswap-backend is working in write-through mode
Weakness of The Implementation

- It could degrade system performance
  - Write-through could slow system

- Wear-leveling needs for Flash swap-device
  - Write-through could crash flash swap-device

- Recommend Zram as swap-device
  - Could be helpful
GCMA Data Structures

- CMA bitmap
  - manages contiguously allocated regions
GCMA Data Structures

- CMA bitmap
  - manages contiguously allocated regions
- Frontswap tree
  - red-black tree for Frontswap Entries

[Diagram of a tree structure and a reserved area]
GCMA Data Structures

- **CMA bitmap**
  - manages contiguously allocated regions
- **Frontswap tree**
  - red-black tree For Frontswap Entries
- **GCMA bitmap**
  - CMA bitmap & Frontswap Pages

![Diagram of GCMA bitmap and Frontswap tree]

```c
#include <gcma.h>

// Use gcma->bitmap
```

GCMA Data Structures

- **CMA bitmap**
  - manages contiguously allocated regions
- **Frontswap tree**
  - red-black tree For Frontswap Entries
- **GCMA bitmap**
  - CMA bitmap & Frontswap Pages
GCMA Scenario

Let’s allocate contig area a
Let's allocate contig area a

Ah, there are two frontswap pages in the area. Hey, get off those two pages!
GCMA Scenario

- Let's allocate contig area a
- contig bitmap
- gcma->bitmap

Ah, there are two frontswap pages in the area. Hey, get off those two pages!

Got it! BAM!
Let’s allocate contig area a

gcma->bitmap

Reserved Area

area a

Done!
GCMA Scenario

Let’s allocate contig area a

contig bitmap

gcma->bitmap

Ok, it's clean now! Go ahead!

area a

Reserved Area

Done!
GCMA Scenario

Thanks!

contig bitmap

area a

Reserved Area

Reads:

Ok, it's clean now!
Go ahead!

Done!
GCMA: Evaluation
Evaluation Environment: Machine

- **CuBox-i4 Pro**
  - 4 Cores, ARMv7 1GHz
  - 800 MiB DDR3 RAM (shrinked for stress)
  - Class 10 SanDisk 16 GiB microSD Card
Evaluation Environment: Variants

Baseline: Linux v3.17, 128 MiB Swap

CMA: Baseline + 256 MiB CMA

GCMA: Baseline + 256 MiB GCMA

GCMA.ZRAM: GCMA + 128 MiB Zram swap
Evaluation Workload

● Background Workload
  ○ Linux v3.12.6 defconfig build
  ○ 4 jobs per core (make -j 16)

● Allocation (Foreground) Workload
  ○ Request 1 - 32K contiguous pages (4 KiB - 128 MiB)
  ○ Repeat 32 Requests, Get Average
Average Latency

CMA shows much higher allocation latency than GCMA & GCMA.ZRAM
Latency Distribution: 4 MiB alloc

- GCMA shows stabler distribution than CMA
- CMA takes 4-8 seconds in worst case
Latency Distribution: 4 MiB alloc

- GCMA shows stabler distribution than CMA
- CMA takes 4-8 seconds in worst case
Latency Distribution

Legends: Number of requested contig pages

CMA
- 1
- 512
- 1024
- 2048
- 4096
- 8192
- 16384
- 32000

GCMA
- 1
- 512
- 1024
- 2048
- 4096
- 8192
- 16384
- 32000

Latency (micro seconds)
Latency Distribution

Legends: Number of requested contig pages

CMA
Sparse: Latency is unpredictable

GCMA
Dense: Latency is predictable

Latency (micro seconds)
Latency Distribution

Legends: Number of requested contig pages

CMA
- 1
- 512
- 1024
- 2048
- 4096
- 8192
- 16384
- 32000

GCMA

Latency (micro seconds)
- 4-8 secs
- 32-64 msecs
Background Workload Performance

```
time make -j 16
```

Frontswap pages of gcma:
- 10,949 stored, 10,829 reclaimed

Frontswap pages of gcma.zram:
- 11,859 stored, 11,739 reclaimed
Background Workload Performance

time make -j 16

frontswap pages of gcma: 10,449 stored, 10,829 reclaimed
frontswap pages of gcma.zram: 11,859 stored, 11,739 reclaimed

zram helps system performance
Future Work

Cleancache backend

Configurable discard policy
How to Use GCMA

Turn on CONFIG_GCM

- cma_declare_contiguous()
+ gcma_declare_contiguous()

That’s all modification clients need
Profiling Tool

- See how the GCMA working using
  
  `$DEBUGFS_DIR/gcma/
  (stored_pages|
  loaded_pages|
  evicted_pages|
  reclaimed_pages)`

```
gcma# ls
evicted_pages  loaded_pages  reclaimed_pages  stored_pages
```
Disclaimer

- GCMA is not magic, it has trade-off

- GCMA is not CMA alternative or enemy
  - Each allocator has own merits and demerits
  - GCMA and CMA could coexist
Summary: GCMA...

Guarantees allocation success and fast latency

Have trade-off for system performance
    Recommends Zram swap

Could be used with very little modification on client code
Code Release

Just unmatured RFC yet, though

https://github.com/sjp38/linux.gcma.git

Also available from LKML

Appreciates every comments and review :D
Thank You

Any question?