GCMA: Guaranteed Contiguous Memory Allocator

SeongJae Park <sj38.park@gmail.com>

These slides were presented during

The Kernel Summit 2018

(https://events.linuxfoundation.org/events/linux-kernel-summit-2018/)

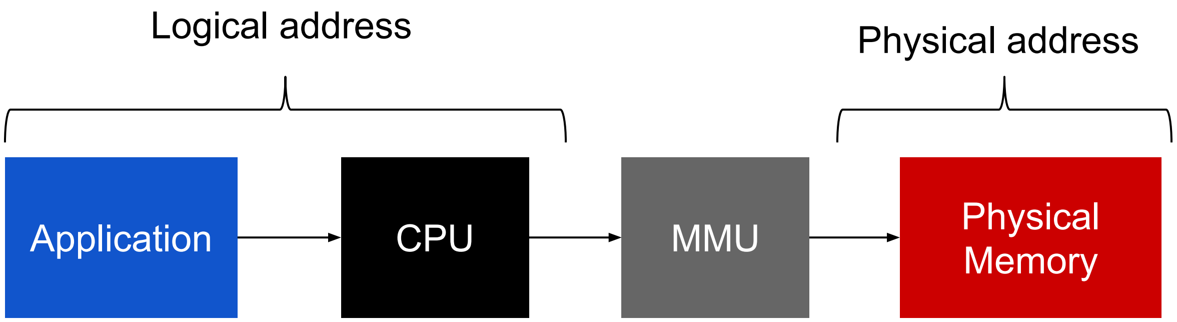
I, SeongJae Park

- SeongJae Park <sj38.park@gmail.com>
- PhD candidate at Seoul National University
- Interested in memory management and parallel programming for OS kernels

Contiguous Memory Requirements

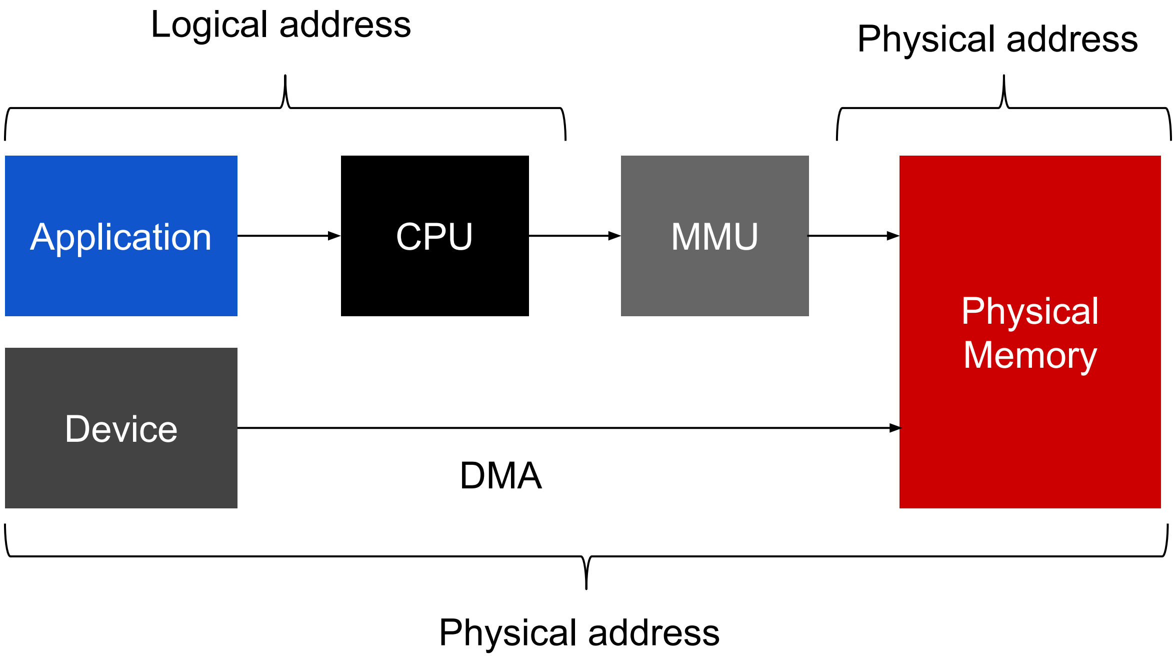
Who Wants Physically Contiguous Memory?

- Virtual Memory avoids need of physically contiguous memory
  - CPU uses virtual address; virtual regions can be mapped to any physical regions
  - MMU maps the virtual address to physical address
- Particularly, Linux uses demand paging and reclaim



Devices using Direct Memory Access (DMA)

- Lots of devices need large internal buffers (e.g., 12 Mega-Pixel Camera, 10 Gbps Network interface card, …)
- Because the MMU is behind the CPU, devices directly addressing the memory cannot use the virtual address



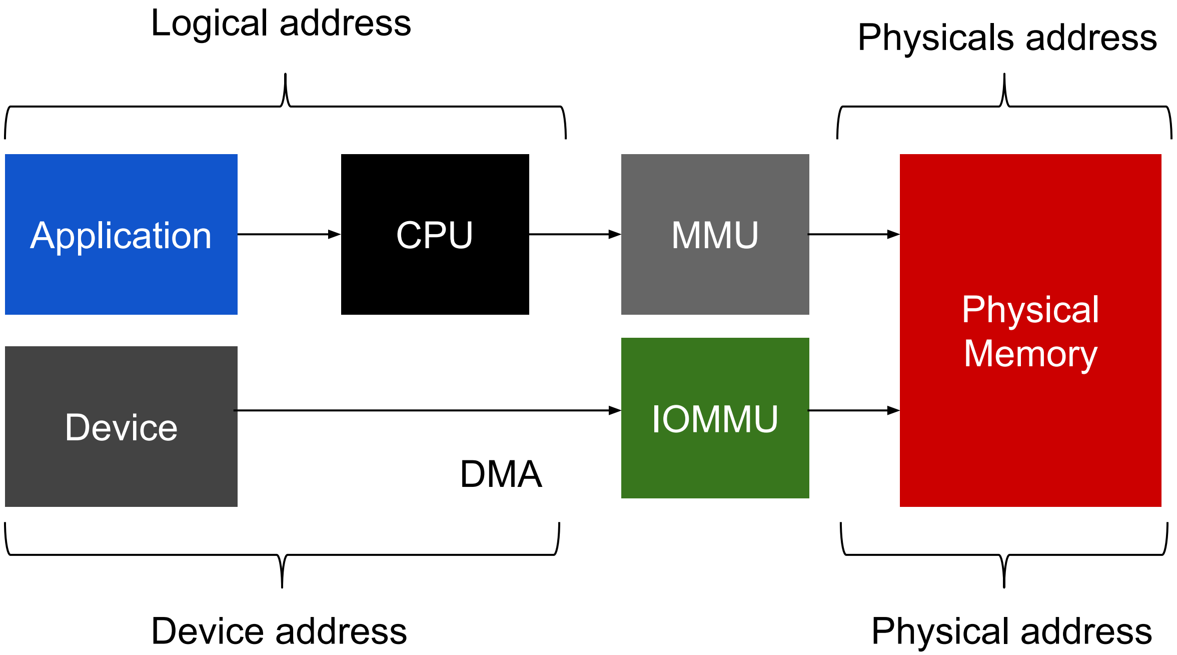
Systems using Huge Pages

- A system can have multiple sizes of pages, usually 4 KiB (regular) and 2 MiB
- Use of huge pages improve system performance by reducing TLB misses
- Importance of huge pages is growing
  - Modern workloads including big data, cloud, and machine learning are memory intensive
  - Systems equipping tera-bytes are not rare now
  - It’s especially important on virtual machine based environments
- A 2 MiB huge page is just 512 physically contiguous regular pages

Existing Solutions

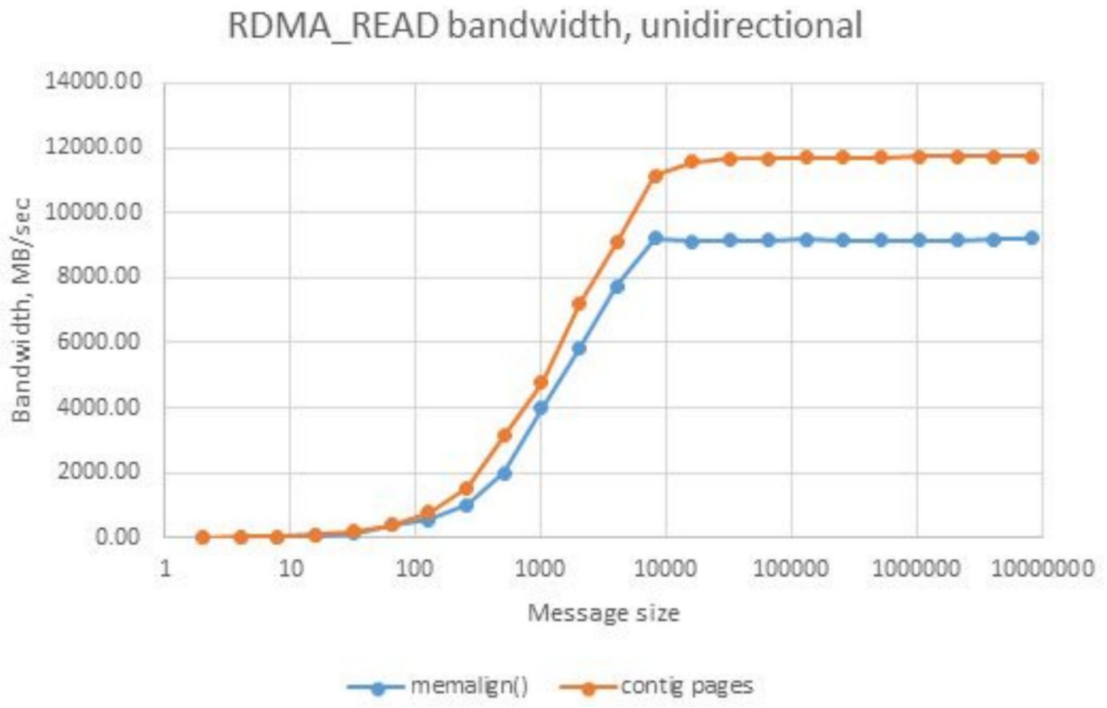
H/W Solutions: IOMMU, Scatter / Gather DMA

- Idea is simple; add MMU-like additional hardware for devices
- IOMMU and Scatter/Gather DMA gives the contiguous device memory illusion
- Additional H/W means increase of power consumption and price, which is unaffordable for low-end devices
- Useless for many cases including huge pages



Even H/W Solutions Impose Overhead

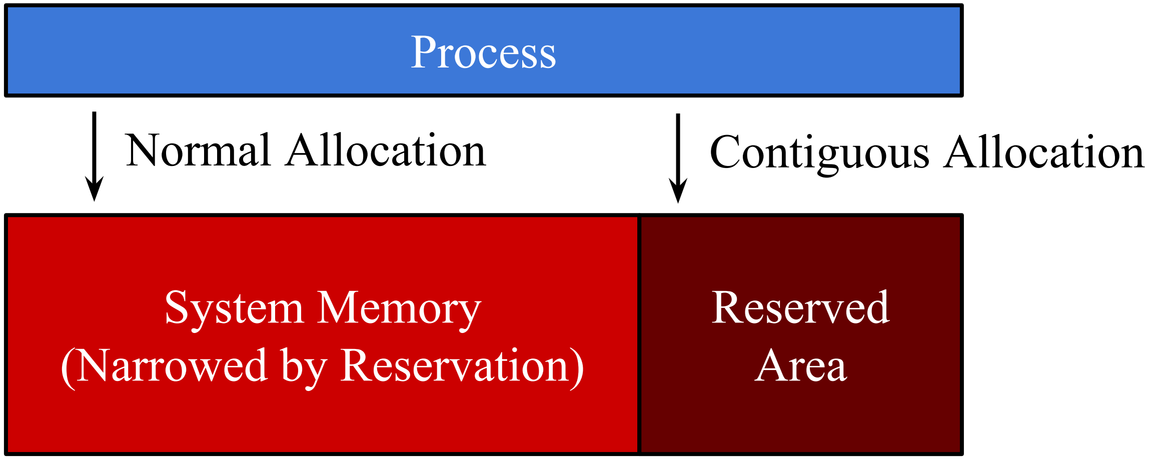
- Hardware based solutions are well known for low overhead
- Though it is low, the overhead is inevitable



Christoph Lameter et al., “*User space contiguous memory allocation for DMA*”,
Linux Plumbers Conference 2017

Reserved Area Technique

- Reserves sufficient amount of contiguous area at boot time and let only contiguous memory allocation to use the area
- Simple and effective for contiguous memory allocation
- Memory space utilization could be low if the reserved area is not fully used
- Widely adopted despite of the utilization problem

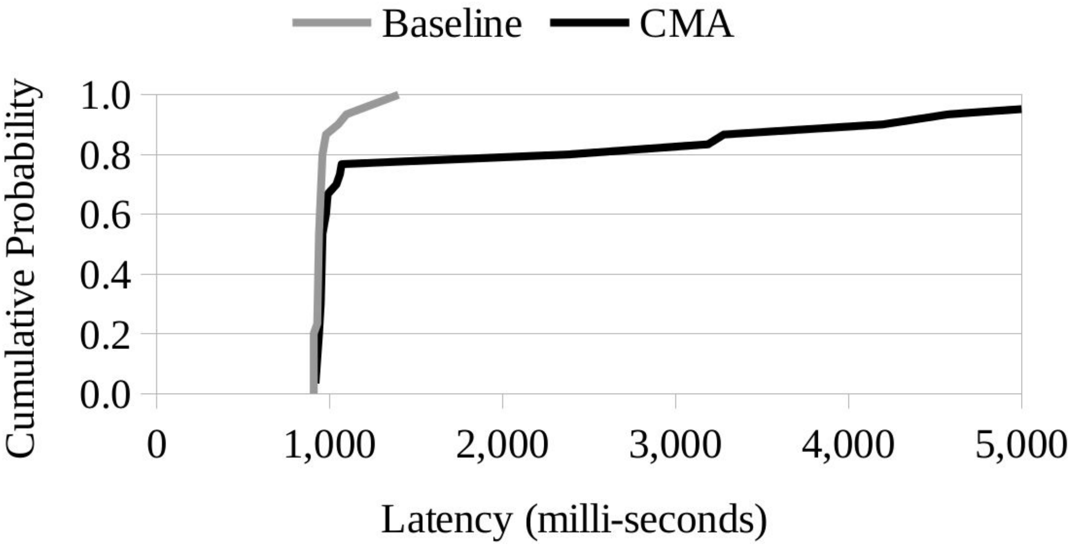


CMA: Contiguous Memory Allocator

- Software-based solution in the Linux kernel
- Generously extended version of the *Reserved area technique*
    - Mainly focus on memory utilization
    - Let movable pages to use the reserved area
    - If contig mem alloc requires pages being used for movable pages,
      move those pages out of the reserved area and use the vacant area for contig mem alloc
    - Solves the memory utilization problem well because general pages are movable
- In other words, CMA gives different priority to clients of the reserved area
    - Primary client: contiguous memory allocation
    - Secondary client: movable page allocation

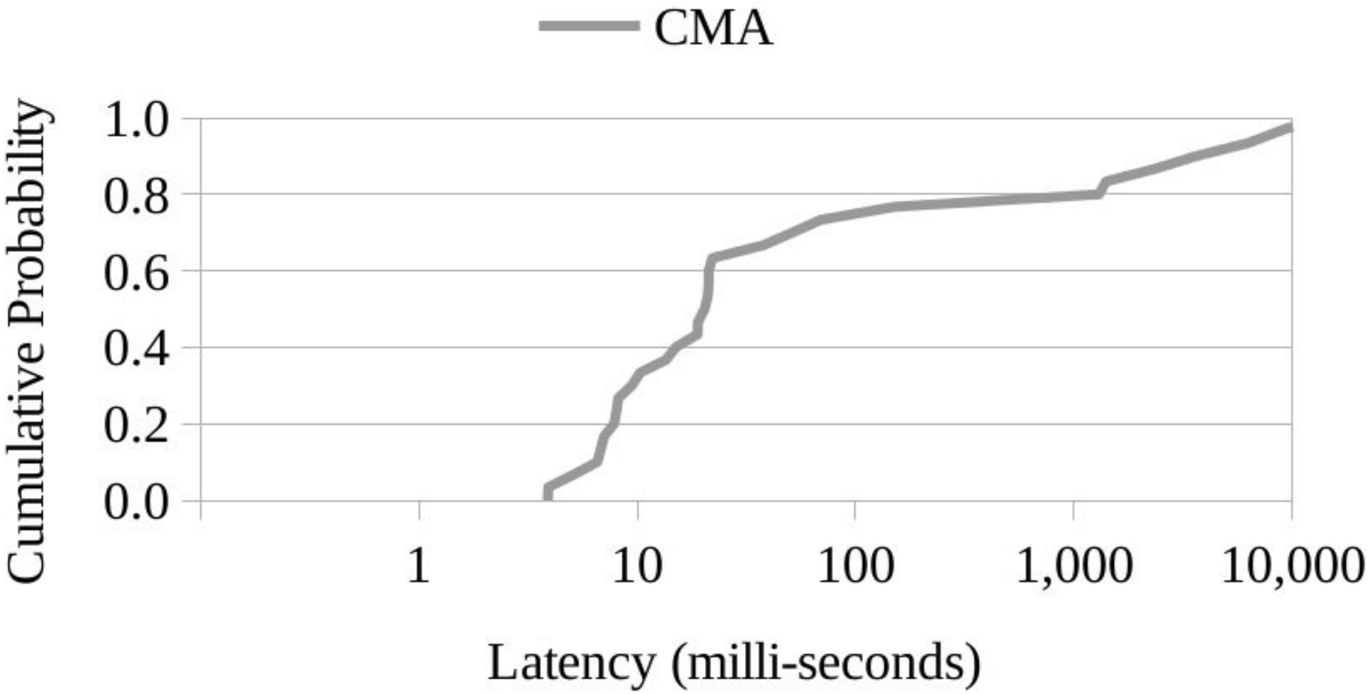
CMA in Real-land

- Measured latency for taking a photo using Camera app on Raspberry Pi 2 under memory stress (Blogbench) for 30 times
- 1.6 Seconds worst-latency with Reserved area technique,
- 9.8 seconds worst-latency with CMA; Unacceptable



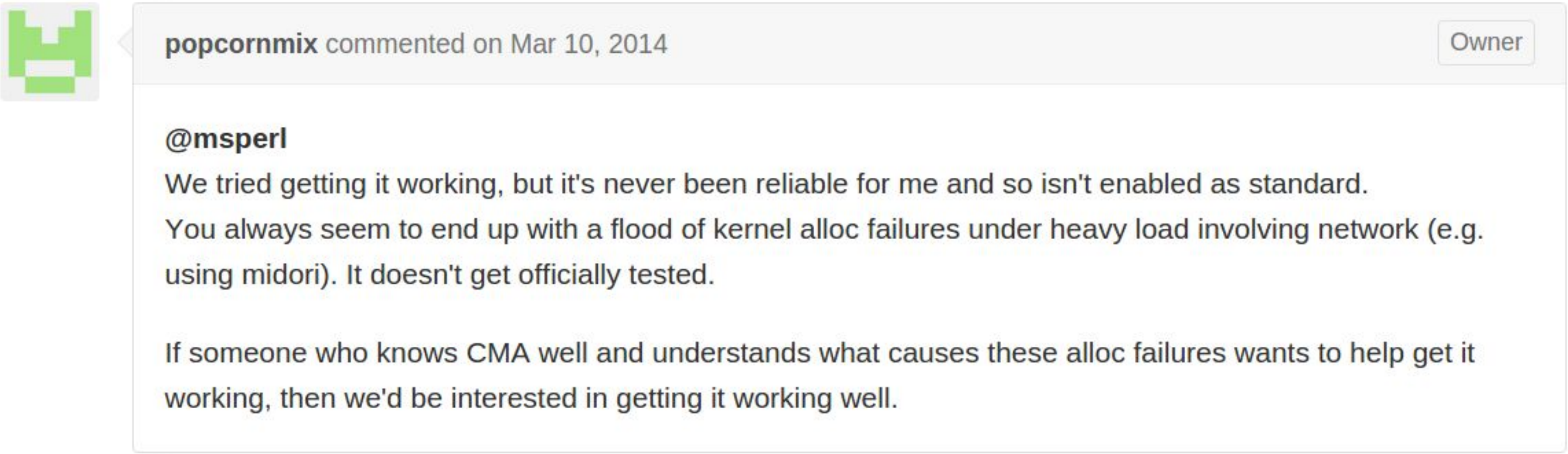
The Phantom Menace: Latency of CMA

- Measured the latency of CMA under the situation
- It’s clear that latency of Camera is bounded by CMA



CMA: Why So Slow?

- In short, secondary client of CMA is not so *nice* as expected
- Moving page out is an expensive task (copying, rmap control, LRU churn, …)
- If someone is holding the page, it could not be moved until the one releases the page (e.g., `get_user_page()`)
- Result: Unexpected long latency and even failure
- That’s why CMA is not adopted to many devices
- Raspberry Pi does not support CMA officially because of the problem



popcornmix commented on Mar 10, 2014

@msperl
We tried getting it working, but it's never been reliable for me and so isn't enabled as standard.
You always seem to end up with a flood of kernel alloc failures under heavy load involving network (e.g. using midori). It doesn't get officially tested.

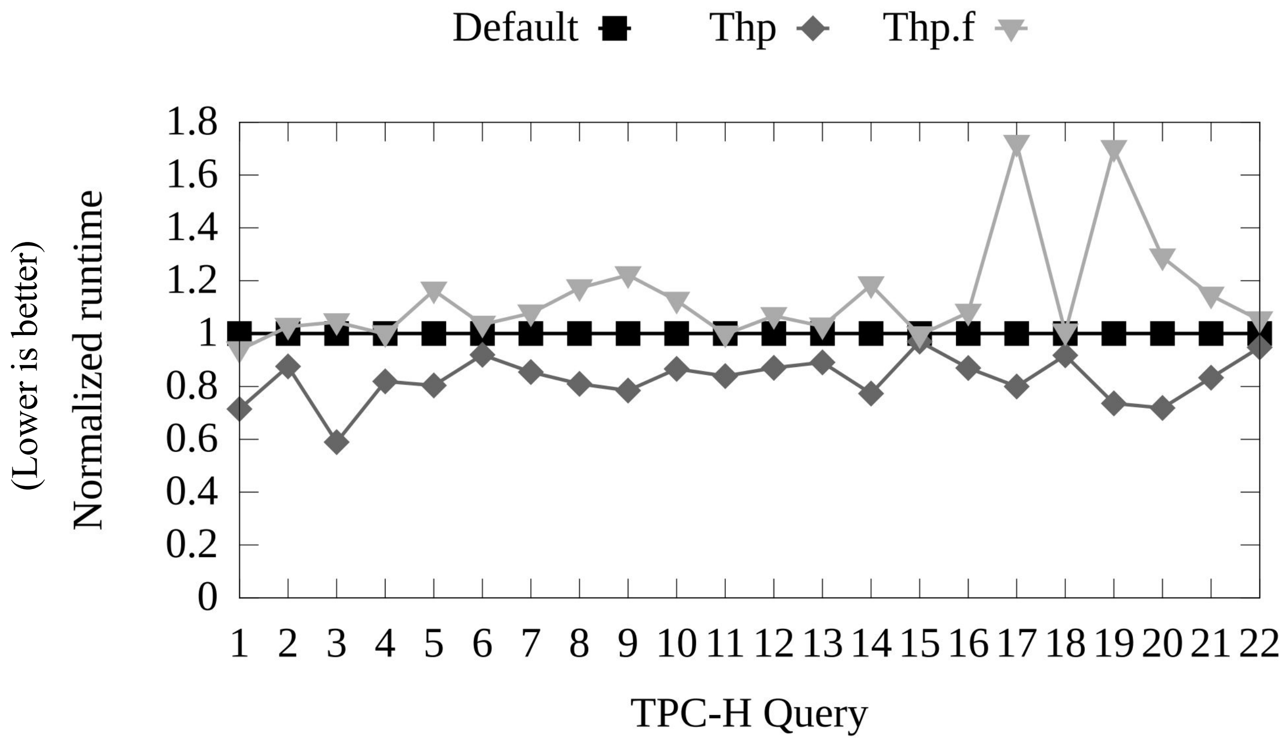
If someone who knows CMA well and understands what causes these alloc failures wants to help get it working, then we'd be interested in getting it working well.

Buddy Allocator

- Adaptively split and merge adjacent contiguous pages
- Highly optimized and heavily used in the Linux kernel
- Supports only multi-order contiguous pages and up to (`MAX_ORDER - 1`) contiguous pages
- Under fragmentation, it does time-consuming compaction and retry or just fails
  - Not a good news for contiguous memory requesting tasks, either
- THP uses buddy allocator to allocates huge pages
  - Quickly falls back to regular pages if it fails to allocate contiguous pages
  - As a result, it cannot be used on highly fragmented memory

THP Performance under High Fragmentation

- Default: THP disabled, Thp: THP always
- .f suffix: On the fragmented memory
- Under fragmentation, THP benefits disappear because of the fragmentation



Guaranteed Contiguous Memory Allocator

GCMA: Guaranteed Contiguous Memory Allocator

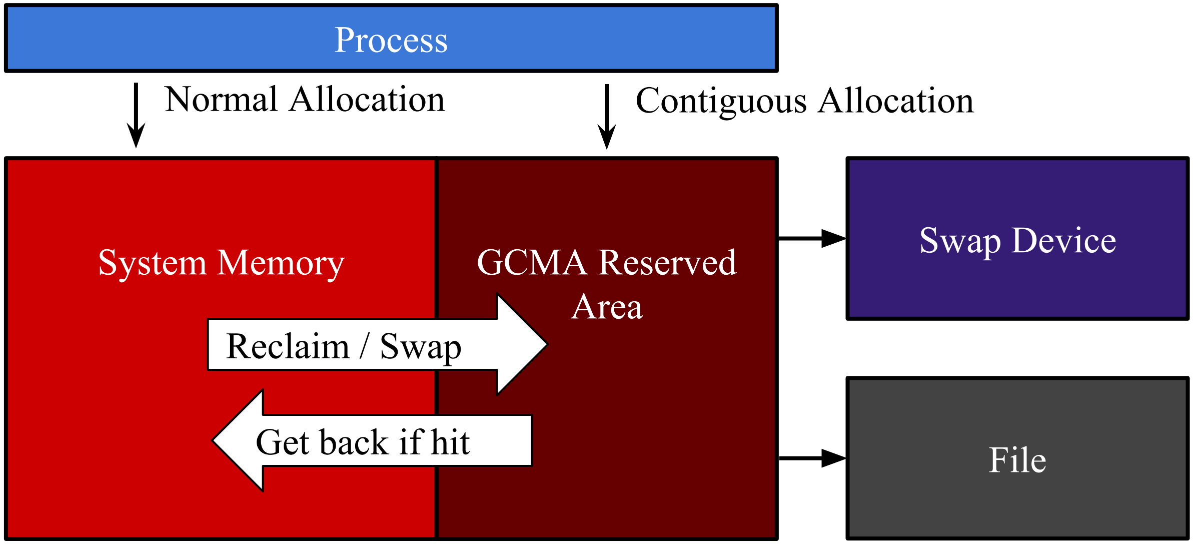
- GCMA is a variant of CMA that guarantees
  - Fast latency and success of allocation
  - While keeping memory utilization as well
- The idea is simple
  - Follow primary / secondary client idea of CMA to keep memory utilization
  - Select secondary client as only nice one unlike CMA
    - For fast latency, should be able to vacate the reserved area as soon as required without any task such as moving content
    - For success, should be out of kernel control
    - For memory utilization, most pages should be able to be secondary client
    - In short, **frontswap** and **cleancache**

Secondary Clients of GCMA

- Pages for a write-through mode Frontswap
    - Could be discarded immediately because the contents are written through to swap device
    - Kernel thinks it’s already swapped out
    - Most anonymous pages could be covered
    - We recommend users to use Zram as a swap device to minimize write-through overhead
- Pages for a Clean cache
    - Could be discarded because the contents in storage is up to date
    - Kernel thinks it’s already evicted out
    - Lots of file-backed pages could be the case
- Additional pros 1: Already expected to not be accessed again soon
    - Discarding the secondary client pages would not affect system performance much
- Additional pros 2: Occurs with only severe workloads
    - In peaceful case, no overhead at all

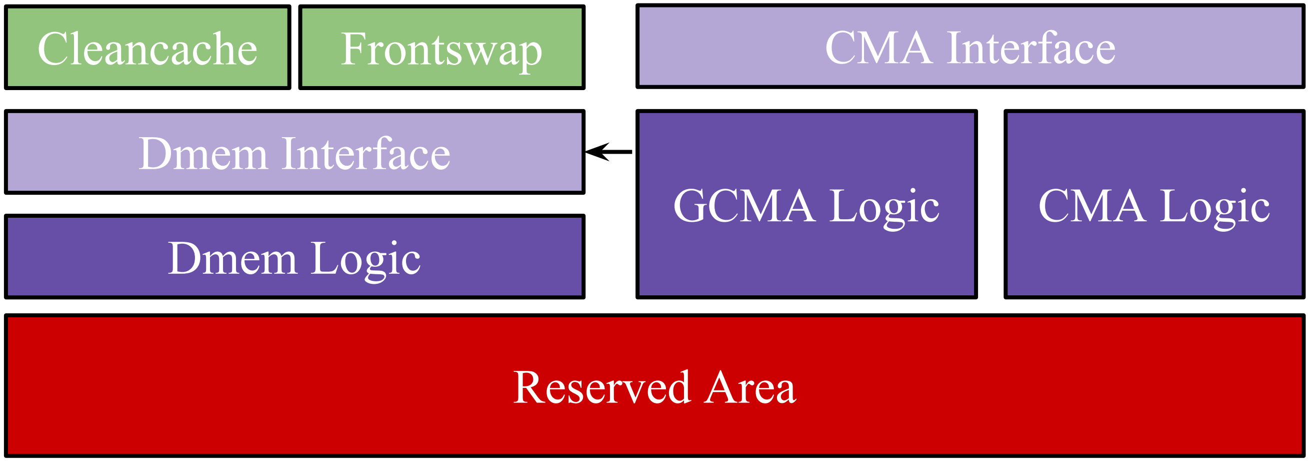
GCMA: Workflow

- Reserve memory area in boot time
- If a page is swapped out or evicted from the page cache, keep the content of the page in the reserved area
  - If the system requires the content of a page again, give it back from the reserved area
  - If a contiguous memory allocation requires area being used by those pages, discard those pages and use the area for contiguous memory allocation



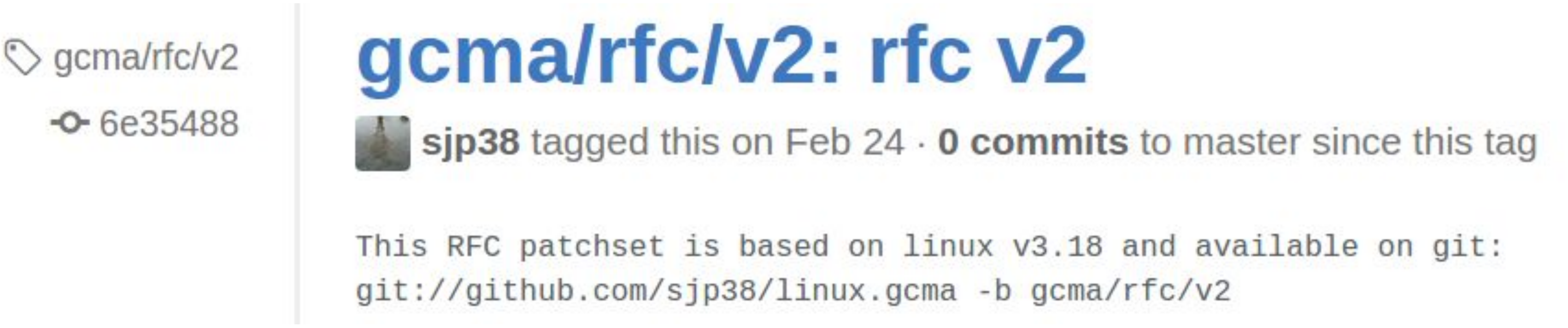
GCMA Architecture

- Reuse CMA interface
  - User can turn CMA to GCMA entire or use them selectively on single system
- DMEM: Discardable Memory
  - Abstraction for backend of frontswap and cleancache
  - Works as a last chance cache for secondary client pages using GCMA reserved area
  - Manages Index to pages using hash table of RB-tree based buckets
  - Evicts pages in LRU scheme



GCMA Implementation

- Implemented on Linux v3.18, v4.10 and v4.17
- About 1,500 lines of code
- Ported, evaluated on Raspberry Pi 2 and a high-end server
- Available under ***GPL v3*** at: https://github.com/sjp38/linux.gcma
- Submitted to LKML (https://lkml.org/lkml/2015/2/23/480)



gcma/rfc/v2

6e35488

gcma/rfc/v2: rfc v2

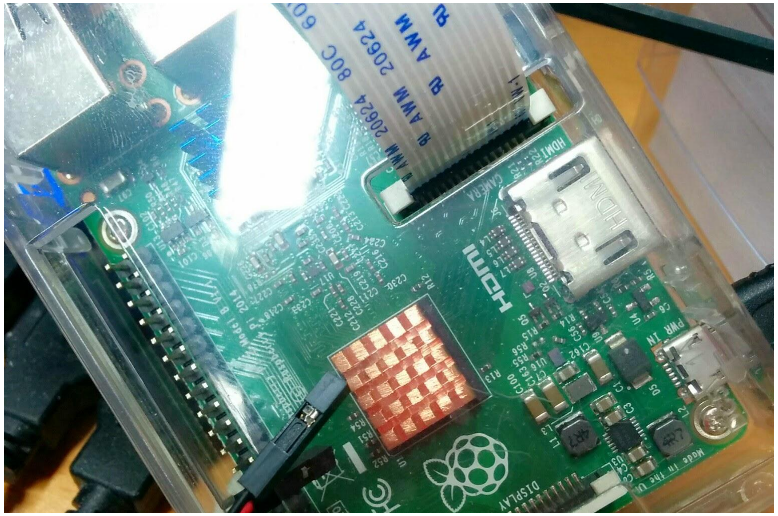
sjp38 tagged this on Feb 24 · **0 commits** to master since this tag

```
This RFC patchset is based on linux v3.18 and available on git:
git://github.com/sjp38/linux.gcma -b gcma/rfc/v2
```

Evaluations of GCMA for a Device

Experimental Setup

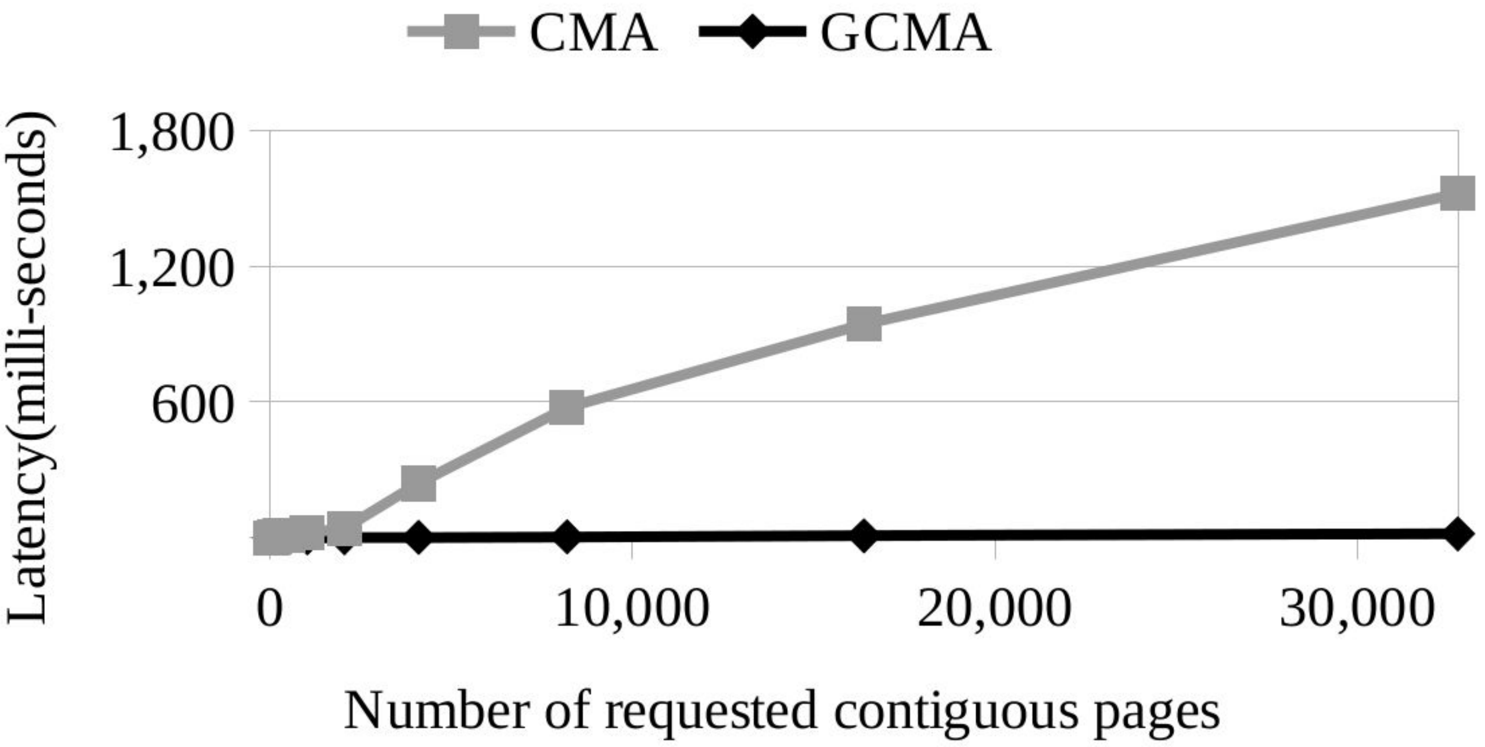
- Device setup
  - Raspberry Pi 2
  - ARM cortex-A7 900 MHz
  - 1 GiB LPDDR2 SDRAM
  - Class 10 SanDisk 16 GiB microSD card
- Configurations
  - Baseline: Linux rpi-v3.18.11 + 100 MiB swap + 256 MiB reserved area
  - CMA: Linux rpi-v3.18.11 + 100 MiB swap + 256 MiB CMA area
  - GCMA: Linux rpi-v3.18.11 + 100 MiB Zram swap + 256 MiB GCMA area



- Workloads
  - Contig Mem Alloc: Contiguous memory allocation using CMA or GCMA
  - Blogbench: Realistic file system benchmark
  - Camera shot: Repeatedly taking picture using Raspberry Pi 2 Camera module with 10 seconds interval between each shot

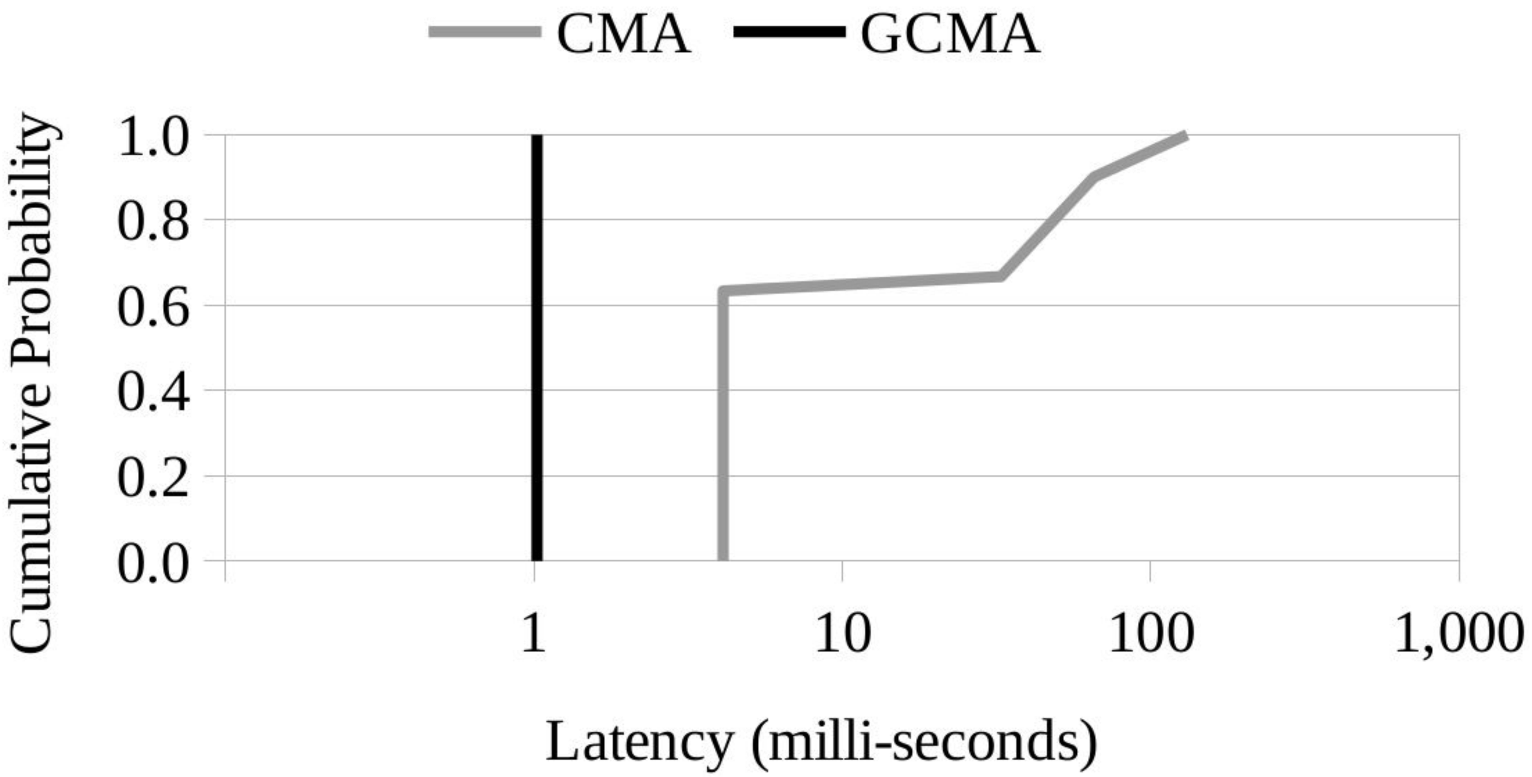
Contig Mem Alloc without Background Workload

- Average of 30 allocations
- GCMA shows 14.89x to 129.41x faster latency compared to CMA
- CMA even failed once for 32,768 contiguous pages allocation even though there is no background task



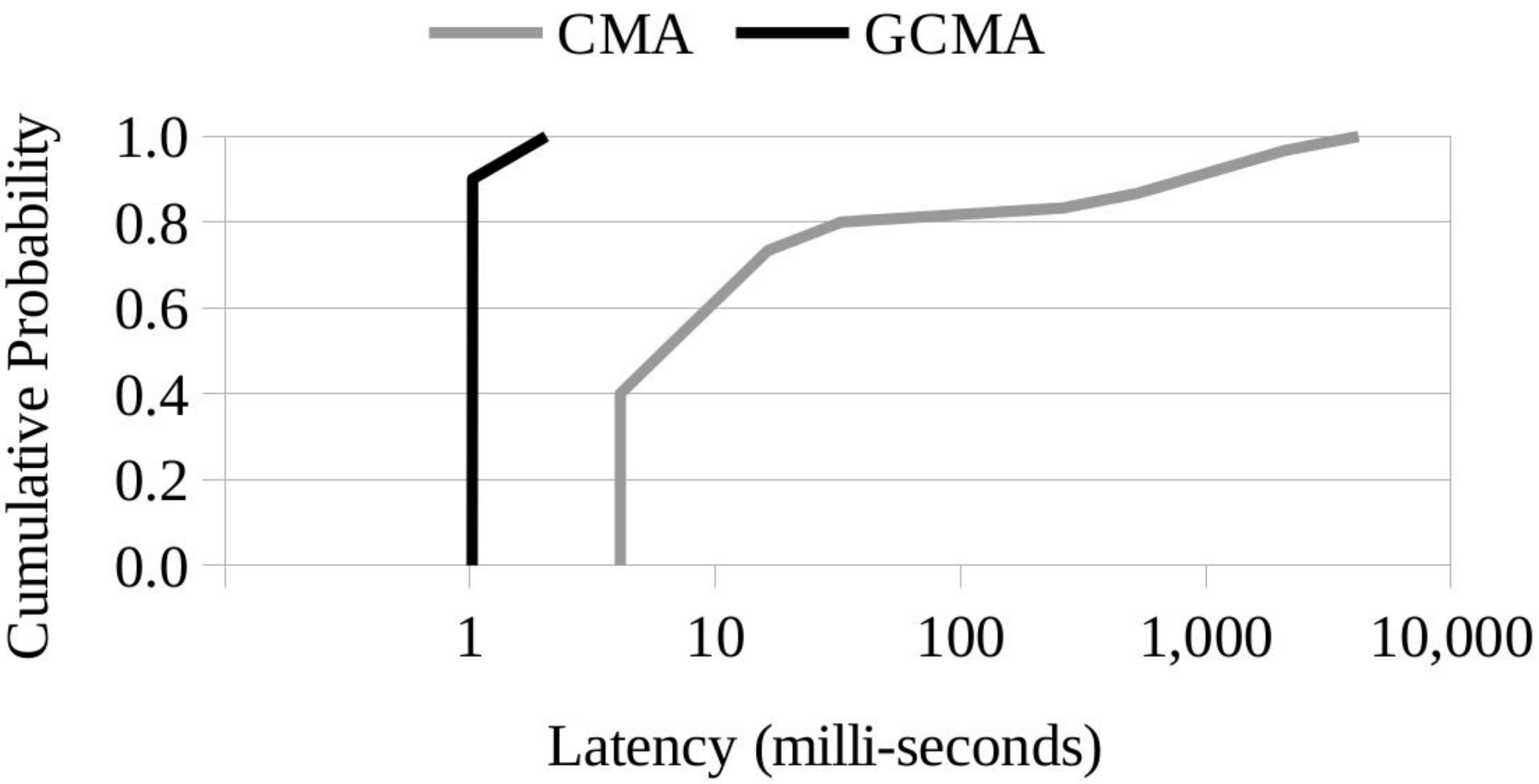
4MiB Contig Mem Alloc w/o Background Workload

- Latency of GCMA is more predictable compared to CMA



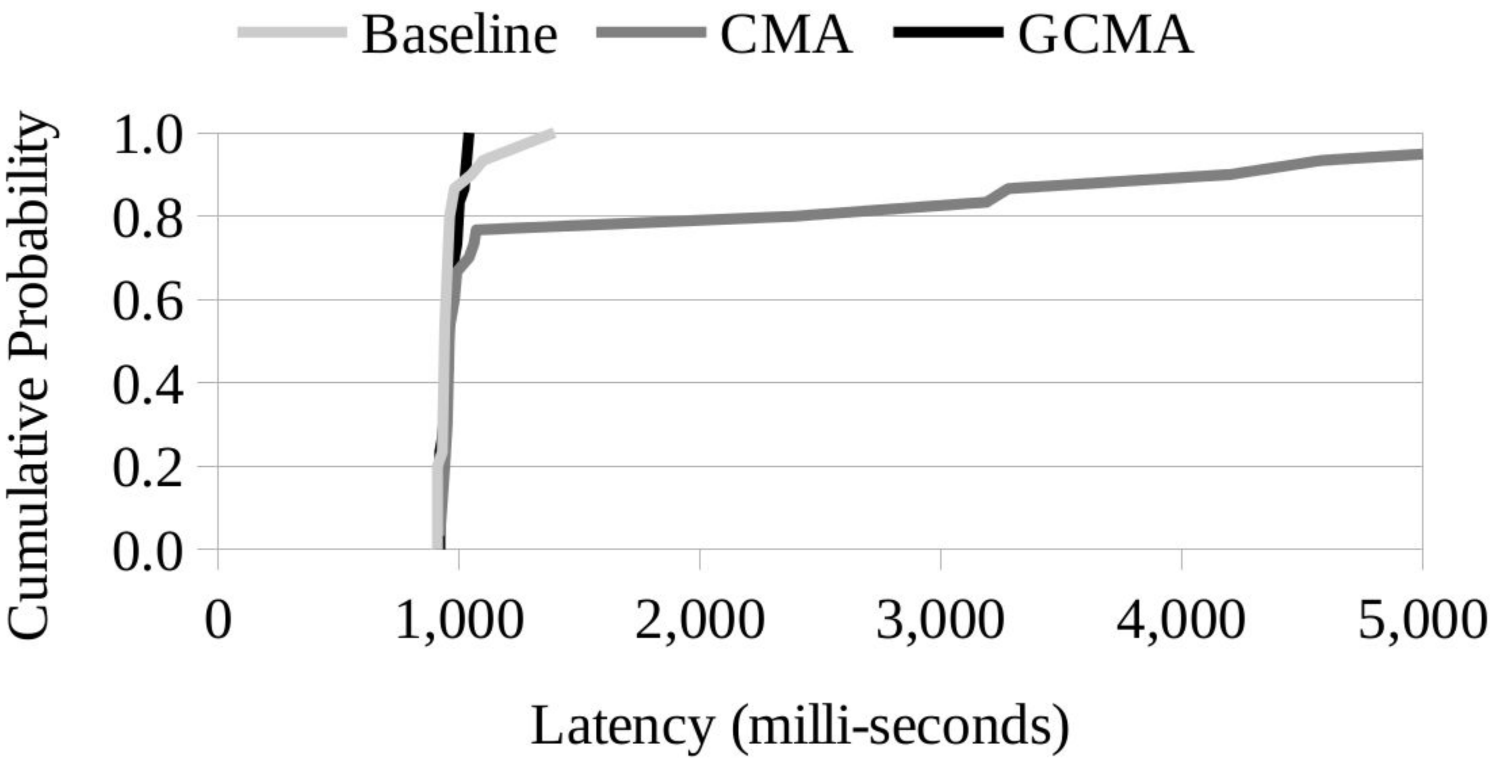
4MiB Contig Mem Allocation w/ Background Task

- Background workload: *Blogbench*
- CMA consumes 5 seconds (unacceptable!) for one allocation in bad case



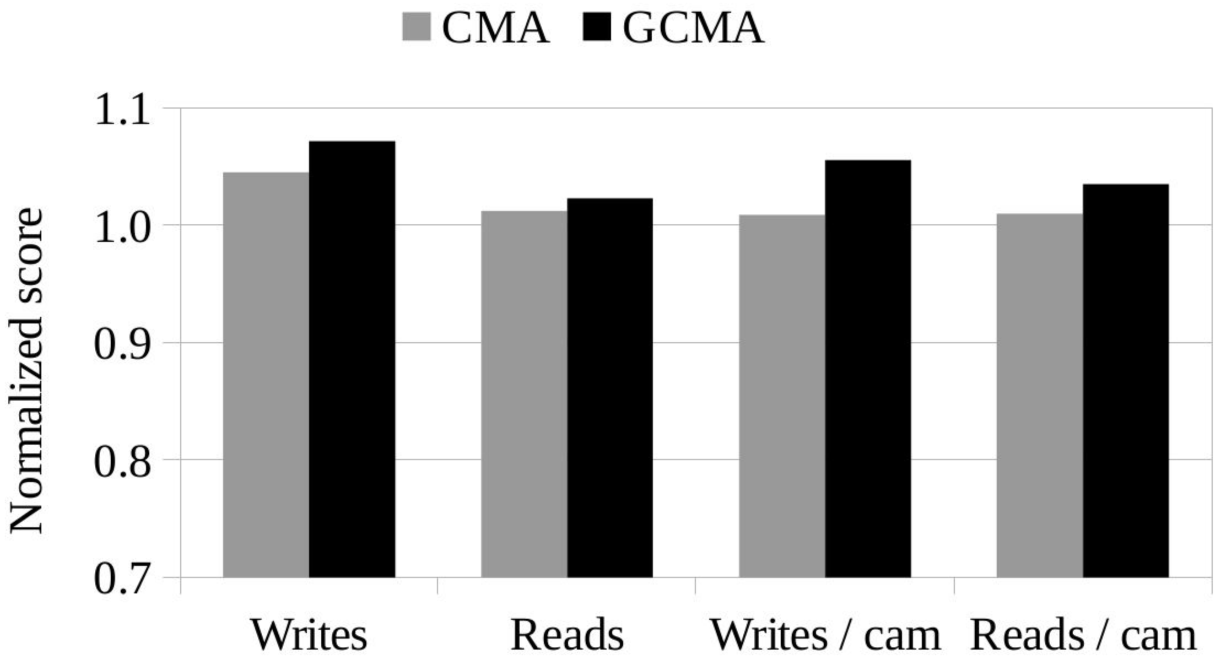
Camera Latency w/ Background Task

- Camera is a realistic, important application of Raspberry Pi 2
- Background task: *Blogbench*
- GCMA keeps latency as fast as Reserved area technique configuration



Blogbench on CMA / GCMA

- Scores are normalized to scores of Baseline configuration
- *'/ cam'* means camera workload as a background workload
- System using GCMA even slightly outperforms CMA version owing to its light overhead
- Allocation using CMA even degrades system performance because it should move out pages in reserved area



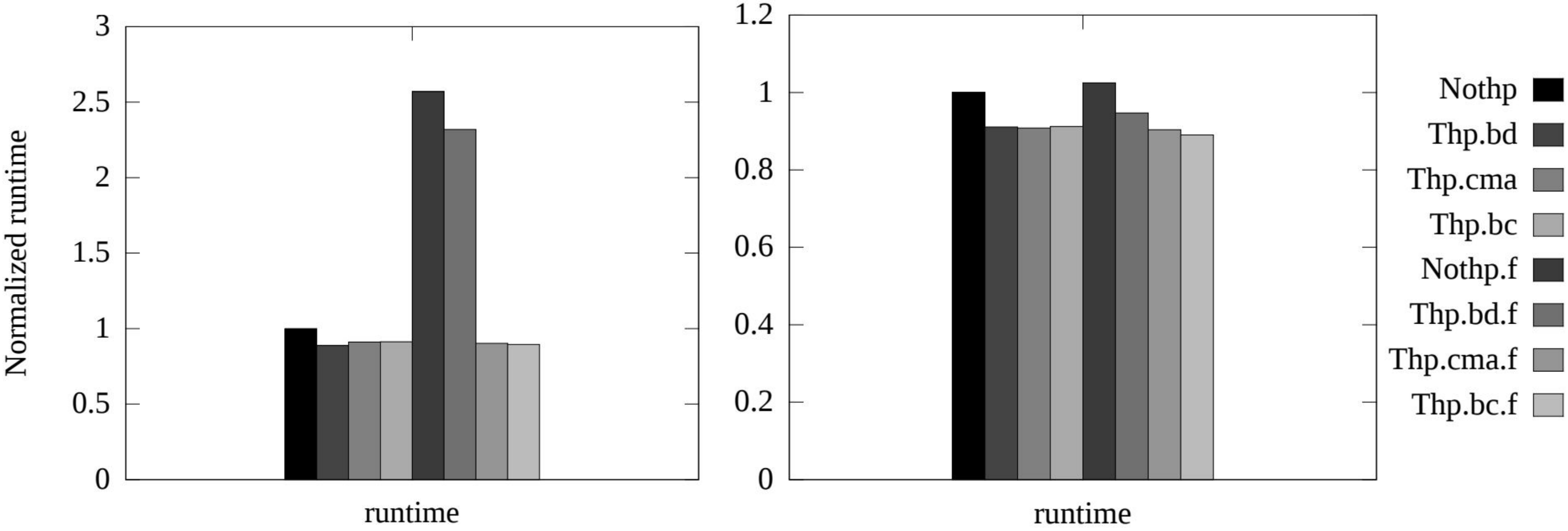
Evaluations of GCMA for THP

Experimental Setup

- Device setup
    - Intel Xeon E7-8870 v3
    - L2 TLB with 1024 entries
    - 600 GiB DDR4 memory
    - 500 GiB Intel SSD
    - Linux v4.10 based kernels
- Configurations
    - Nothp: THP disabled
    - Thp.bd: Buddy allocator based THP enabled
    - Thp.cma: GCMA based THP enabled
    - Thp.bc: Buddy allocator based, GCMA fallback using THP enabled
    - .f suffix: On fragmented system
- Workloads
    - 429.mcf and 471.omnetpp from SPEC CPU2006
    - TPC-H Strong test

Performance of SPEC CPU 2006

- Memory intensive two workloads are chosen
- Fragmentation decreases performance of regular pages, too
- GCMA for THP shows 2.56x higher performance compared to the original THP under fragmentation
- The impact is workload dependent, though

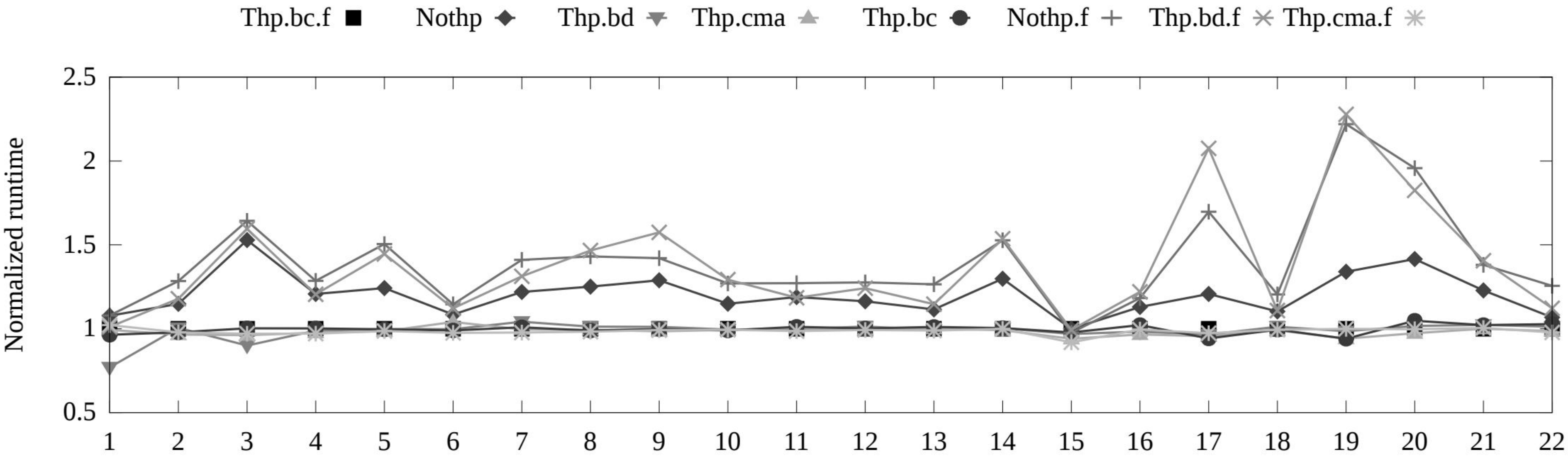


(a) 429.mcf

(b) 471.omnetpp

Performance of TPC-H

- Power test: Measure latency of each query for OLAP workloads
- Runtime is normalized by Thp.bc.f
- Queries 17 and 19 produces speedup more than 2x
- Four queries (3, 9, 14 and 20) show a speedup of 1.5x-2x



Plan for GCMA

Unifying Solutions Under CMA Interface

- There are many solutions and interfaces for contiguous memory allocations
- Too many interfaces can confuse new coming programmers; We don’t want to increase the confusion with GCMA
- GCMA is developed to coexist with CMA, rather than substituting it; It is already using CMA interface
- The interface could select the secondary clients for each CMA region
    - None (Reservation)
    - Migratables (CMA)
    - Discardables (GCMA)
- Currently, we are developing a patchset; It aims to be merged to the mainline
- The patchset will include updated evaluation result

Conclusion

- Contiguous memory allocation needs improvement
  - H/W solutions are expensive for low-end devices, imposes overhead, and cannot provide real contig memory
  - CMA is slow in many cases
  - Buddy allocator is restricted
- GCMA guarantees fast latency, success of allocation and reasonable memory utilization
  - It achieves the goals by utilizing nice clients, the pages for Frontswap and the Cleancache
  - Allocation latency of GCMA is as fast as Reserved area technique
  - GCMA can be used for THP allocation fallback
  - 130x and 10x shorter latency for contiguous memory allocation and taking a photo compared to CMA based system, repectively
  - More than 50% and 100% performance improvement for 7/24, 3/24 realistic workloads
- Planning to release the official version and updated evaluation results in near future

Thanks