

An approach towards improvement of contiguous memory allocation linux kernel: a review

Anmol Suryavanshi, Sanjeevkumar Sharma

Department of Computer Science and Engineering, Faculty of Computer Science and Engineering, Oriental University, Madhya Pradesh, India

Article Info

Article history:

Received Jul 10, 2021

Revised Jan 4, 2022

Accepted Jan 12, 2022

Keywords:

Contiguous memory allocator

Input-output memory

management units

Reservation techniques

Scatter/gather direct memory

access

ABSTRACT

The demand of contiguous memory allocation has been expanded in day-to-day life in all the devices. It is achieved in existing systems by using various reservation techniques. There are various other methods to achieve the goal of contiguous memory allocation in linux kernel such as, input output memory management units (IOMMU's), scatter/gather direct memory access (DMA) and reserved static memory at boot time. But these solutions have its own drawbacks such as, IOMMU requires hardware. However, the configuration of additional hardware's increases the cost. The power consumption of the system and the reserved static memory in the system goes waste when not in used for specific purpose. It is very difficult to access contiguous memory in low-end devices that are unable to provide real contiguous memory. There is one existing method called contiguous memory allocator (CMA), which provides dynamic contiguous memory. It overcomes most of the problems but CMA itself has some drawbacks, which do not provide the guarantee of failure in future of contiguous memory. The motivation behind this study is to review existing contiguous memory allocation (CMA) method by identifying and removing its drawbacks.

This is an open access article under the [CC BY-SA](#) license.



Corresponding Author:

Anmol Suryavanshi

Department of Computer Science and Engineering, Faculty of Computer Science and Engineering

Oriental University

Indore, Madhya Pradesh, India

Email: eranmol89@gmail.com, anmol.suryavanshi@svkm.ac.in

1. INTRODUCTION

Nowadays, the requirement for physical contiguous memory allocation is extremely demanded. Particularly in low-end 32-bit devices as the currently available solutions are not sufficient. The frequently used method is reservation technique. Even though it serves memory allocation in good manner, yet it can seriously downgrade the memory usage or wastage of reserved memory. Although, there are some hardware solutions such as scatter/gather direct memory access (DMA) and input output memory management unit (IOMMU) for resolving this issue. However, the cost of this additional hardware is much more for the low-end 32-bit devices. contiguous memory allocator (CMA) is a linux software product that targets to resolve memory allocation as well as efficient memory utilization issues. There are different devices on embedded frameworks, that have no scatter/gather DMA or IOMMU facility and contiguous memory blocks for allocation [1], [2]. They incorporate devices like cameras and hardware video decoders-encoders. However, such devices regularly requires large memory that makes systems inadffigequate. Some embedded devices force extra necessities on the buffers. For example, they can just operate on buffers allocated in specific memory bank (if one or more memory bank available in the system) or buffers allocated to a specific memory limit. The enormous growth has been observed in the development of embedded devices recently

(particularly in the V4L region) and there were various type of drivers exists which incorporate their own memory allocation code. A large portion of them use bootmem-based strategies [3], [4].

CMA structure is an dynamic reserved memory that binds contiguous memory allocation systems. It gives a straightforward application-programming interface (API) to device drivers together, while remaining as adjustable and modular. The contiguous allocation makes this possible without harming genuine CMA. The CMA has emphasis on memory proficiency. It reserves large memory region during the boot time which utilises for contiguous allocation [5], [6]. If the contiguous memory do not fully utilizes the reserved memory then it is available for 2nd-class clients like other processes which do not require contiguous memory, else that memory will go waste. The pages allocated to the 2nd-class clients were required for contiguous memory allocation. It disposes the pages when required for CMA process and utilizes them for contiguous memory allocation [7]–[9]. The detailed block diagram structure of CMA is shown in the Figure 1 [10], [11].

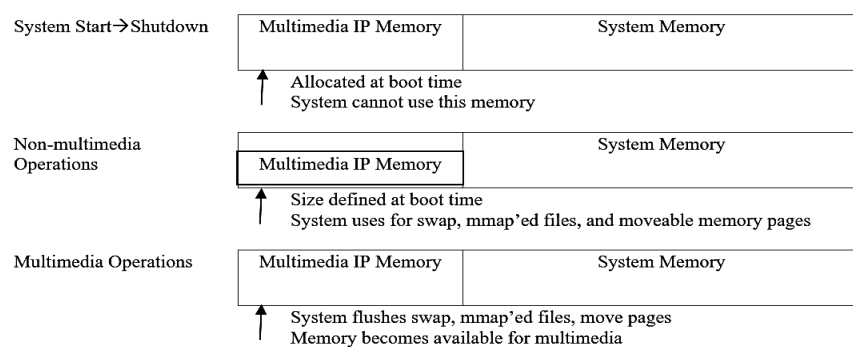


Figure 1. Block diagram of CMA

In CMA, required memory can be reserved. It will be reserved at boot time by kernel commandline or by device tree. In linux, these pages are marked as movable, unmovable, or locked etc. from the reserved CMA memory. The movable pages are `_2nd-class client_` that can be used by non-CMA process. The movable pages can be moved or disposed- off if the contiguous allocation requires for them [3], [5]. As mentioned in Figure 1, the CMA works with specific CMA memory, which will be useful for multimedia and non-multimedia operations or any purpose. The CMA reserves the memory at boot time. Despite the fact that it is utilized to take care of the memory assignment, issue dynamically. CMA benefits through DMA planning structure. Toward the start, the idea of CMA region was global memory that is available for all-purpose. At the reserved time, CMA has its own starting address and end address where all the pages are distributed either in movable or unmovable pages. Everyone can use the memory inside this CMA start and end address. Other processes can use only movable pages.

To allocate CMA memory using DMA, an API can be used to provide contiguous memory attributes. When CMA memory is requested through DMA API, it will allocate from CMA memory region. The movable pages can be moved with mapping of physical memory to virtual memory [12], [13]. Therefore, it can be used by requested process. In Linux, all the processes work on virtual memory. The existing reservation-based technique and CMA based technique may be compared as shown in Figure 1. The CMA size is defined at boot time. The system used the memory for swap, mmpaed files and moveable memory pages when CMA does not use this memory area process a shown in Figure 1. When this memory area required by CMA process, it can be reclaim the movable pages and get the memory area which is required for CMA process [14]–[16].

The CMA is required for I/O devices that can just work with contiguous physical memory. On systems with an I/O IOMMU, this would not be an issue on the grounds that IOMMU can map to non-contiguous memory locations to contiguous locations of physical memory [17], [18]. Additionally, a few gadgets can do scatter/gather DMA Preferably. All I/O gadgets ought to be intended to one or the other work behind an IOMMU or ought to be fit for scatter/gather DMA [19]–[21]. Sadly, this isn't the situation and there are gadgets that require physically contiguous buffers. There are two different ways for a device driver to dispense a contiguous buffer [6], [22]–[24]. The device driver can allocate a chunk of physical memory at boot-time. This is reliable because most of the physical memory would be available at boot-time. However, if the I/O device is not used, then the allocated physical memory is just wasted [25]–[28]. A chunk of physical memory can be allocated on demand, but it may be difficult to find a contiguous free range of the required

size. The advantage, though, is that memory is only allocated when needed [6], [29]. CMA solves this exact problem by providing the advantages of both approaches with none of their downsides. The basic idea is to make it possible to migrate allocated physical pages to create enough space for a contiguous buffer. More information on how CMA works can be found here [22], [30]–[32].

The Figure 2 explains the workflow of existing CMA in linux kernel and the way it works to migrate and reclaim the pages: i) When application requested memory from reserved CMA area, reservation of CMA memory can be done through mentioning size of memory to be reserved at compile time; ii) Request of reservation comes to CMA than it starts choosing range of pages to allocate from cma reserved area, if CMA driver found range requested from application to allocate memory. Then it isolate the pageblock from that range so that no one can use the area at same time., if range of pages is not found allocation is failed for requested memory; iii) Once CMA driver isolate the pageblock. It then start looking for non free pages and try to migrate the non free pages or pages used by another non cma process; iv) If CMA succeed to migrate the pages from selected pageblock it will undo the isolation of pageblock and go for finding another range of pages until requested memory can be allocated; and v) if from isolation block there no pages were used by other processes then mark pages in that range as free and undo the isolation of pageblocks will succeed the allocation of requested memory.

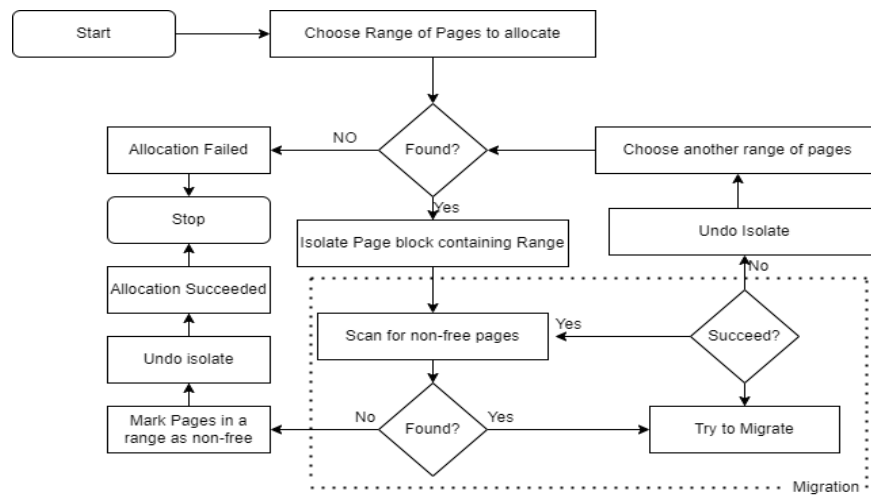


Figure 2. Workflow of CMA

2. LITERATURE REVIEW

Park *et al.* [1] introduced a guaranteed contiguous memory allocator (GCMA) that assures low latency, perfect allocation & efficient memory area. The GCMA uses memory reservation technique which increases efficient memory utilization by sharing memory region with immediately discardable data [1]. Park *et al.* [1]. have introduced alternative approach for CMA and called it as Guaranteed CMA (GCMA). But it degrades the system performance and uses transcendent memory as it does not utilize memory effectively. The authors had studied over existing solutions i.e., the MMU like hardware scatter/gather DMA or IOMMU. In their work, authors stated that hardware solutions are too much costly for low-end devices. As a result, it imposes function overhead and it can't achieve real contiguous memory. The research also states that the buddy allocators are restricted ones [1], [29].

Corbet *et al.* [5] Have introduced five level page tables which are originated from four level page tables. It does not support on all platforms and increase only linear addresses size [29]. The study states that the results are having emphasis on translation look-a-side buffer (TLB) and the huge pages which require fewer lookups can easily work. The data stored when a virtual memory address maps itself to the physical address in a page table. The virtual addresses are indexed via indexed pages and the page frame numbers are yielded for linked physical page. The array used for this purpose mostly unusual and wasteful [3]. The 32-bit systems are mostly don't use all the available virtual address [7].

Corbet *et al.* [6] have introduced efficiency in huge page management system and coordinated while improving decency and execution, the Ingens decreases tail-latency and bloat. But it has a drawback that huge pages will reserve the memory for special purpose only [31]. The first class resources are being maintained contiguity, tracks utilization and access memory frequency for huge pages. Ingens improves huge page allocation by eliminating plague in the current systems [10].

Stultz [18] described the Android ION subsystem which is considered for allotting and sharing buffers between Hardware gadgets & user spaces to empower zero copy memory sharing amongst the different devices. The approach allows centralize allocation of different "types" of memories or "heaps" [1]. The ION plays a vital role through central buffer allocator & manager who handles cache maintenance for DMA [1]. But it allocates and shares data between devices using existing CMA for user space allocations [14]. Jeong *et al.* [18] have introduced rental memory allocation by reusing device reserved memory [31]. proposed approach states that the rental memory returning time cannot be less than simple rental memory management. This prototype shows minimum return time in synthetic's and android's based workload [31]. Also it gives analytical improvement in the performance as compared with existing simple management policy [31]. The research gap in the system is the device memory needs to reserve, so that it can be reused/rented for other purpose [16].

Basu *et al.* [21] have introduced efficiency in virtual memory for big server with segmentation approach. The authors propose an approach to eliminate the inefficiency of the memory caused due to memory reservation. When device is in an idle state, the reserved memory space of the device can be used for different purposes. This approach needs minimal reallocating associated memory space. It only reduces the TLB misses [17].

Magenheimer [21] has introduced compression-based technique in Linux which compresses and decompress pages into RAM and does not support page cache pages. The kernel compression should have a byte sequences in memory, compresses it and afterwards keeps the compressed version in the RAM until data needed in future again. We can't read or write any one byte in compressed form. When we need data from compressed one then we can decompress the data to find individual byte to directly access it again [18].

Nazarewicz [22] has introduced CMA in Linux systems. This approach reserves memory by using mem parameter and mem block/bootmem and booting time of system. It assigns buffer space to device when needed. The memory might be wasted in idle state. The approach provides an API for allocating reserved pool. This method migrates pages for allocation [19].

Zeng [24] introduced ION memory manager for android. This approach suggests the specific hardware needs or to combat fragmentation, the ION managing more than one memory pools which are set aside a boot time. Some of the hardware blocks have special memory requirements like GPU, display controller and camera. IONS heaps are the representation of ION memory pool. Android devices are configured with various sets of ION heaps as per the device's memory requirement. But the underlying Linux kernel implementation is same [20].

Jeong [18] initiated memory reservation based on device. The authors have stated the on-demand reservations method i.e., eCache. The eCache method takes very less time for user latency due to reservation time. It also maximizes the efficiency system memory [4]. It also reduces read I/O operation and increase the launch performance of the application. With this approach, memory efficiency of the system maximizes due to nature of eviction-based memory allocation. But the memory reservation technique was Static memory allocation [21].

Ferdman *et al.* [23] have made study on data centers and workloads require efficient power and space. In the research, the authors have analyzed micro architectural behavior of the huge type of scale out workload for the performance counter. This approach identifies and analyze the important sources of power inefficiencies in a instruction fetch, core micro architectural memory system organization [3]. This analysis gives efficiently working scale-out workloads which required for optimizing an instruction fetching path for multi megabyte instructions working sets & reducing the core aggressiveness's and end level cache capacity to clear up an memory area [3]. But in this study, there is no solution for physical contiguous memory allocation failure issue in workloads [3], [7].

Magenheimer [21] has defined the motivation behind transcendent memory (tmem) is gives the kernel with the capacity to use memory. This research shows two existing tmem frontends, frontswap and cleancache. These two primary kernel memory types are impactful over memory allocati on pressure. These two methods are complementary to each other. The cleancache works on cleanly mapped pages which are reclaimed by the kernel. The frontswap works on anonymous pages which are to be swapped out by kernel. When cleancache_get executes successfully then disk read gets avoided. When frontswap_put executes successfully, a swap device read-write operation get avoided. The tmem is faster as compared to disk paging/swapping. But this method can't specify, now and again can't follow, and can't directly address [18].

Barr *et al.* [26] had designed a mechanism for speculative address translation. The author shows that the specific devices can raise latency through nested paging but the nested paging gives some performance penalty. While using huge pages, reduces the TLB misses frequency. But this method specifically applies because the hypervisor needs to control over guests physical address space permission. Speculative address translation allows the results of huge pages. But it requires the Special device named as SpecTLB [25]. Robert Love has introduced the techniques of Memory Zones, different page types and physical to virtual

translation technique. In this paper, the authors discussed various mechanisms of obtaining memory such as, Page allocators and the Slab allocators. To obtain the memory is not easy in the kernel because it depends on the allocation process which relies on certain conditions [26].

Corbet *et al.* [6] have introduced special class of memory, which is not used by kernel directly. The author has introduced “Cleancache” and “Frontswaps” methods, which has properties like transcendent memory. The CleanCache method provides a place where it can store pages and removes them if the space requires by another process, if not then it keeps it around. The Frontswaps works as an emergency safety valve when guest requires too much memory to work with. However, it is transcendent memory type and memory cannot be used by kernel directly [3].

3. PROBLEM IDENTIFICATION

Physical contiguous memory can be significant for different processing conditions such as low-end embedded systems and high-end devices [3], [13], [20], [21], [23], [25]–[27]. In any case, existing allocators all have various impediments. For instance, scatter/gather DMA and IOMMU like hardware-based arrangements increments the cost and power utilization, buddy system is useless in case of fragmentation. The reservation approaches decrease efficient memory usage and CMA is excessively slowest [2]. Along with the above-mentioned drawbacks of CMA, following are the other drawbacks in existing CMA in linux kernel.

3.1. Allocation failure

CMA may fail to allocate contiguous memory due to mentioned facts: i) Large size allocation: It fails due to fragmentation in virtual memory; ii) Direct pinning: The movable pages may be pinned by any kernel thread for a while. The migration may get fail in case of movable pages that are already pinned by someone else while migrating him or her [1]. As a result, the pages needs to be unpinned as soon as possible, otherwise the contiguous allocation may get in virtual memory [32]; and iii) Indirect pin: If one movable page is dependant on an object, the object may increase reference counts of a movable pages for getting assurance the is it safe to use the page [1]. The page that has not been free to be used by contiguous allocation when a movable page needs get migrate for a contiguous memory allocation [22]. As a result, contiguous allocation may get fails.

3.2. High cost

Contiguous memory allocation of CMA can have high cost due to Function overhead issue. When allocation is succeeded in physical memory it will be mapped with virtual contiguous memory and return to the CPU to use [9], [16]–[19]. This entire process causes function overhead and function overhead increases the high cost of the allocation process.

4. PROPOSED METHOD

The proposed approach resolves the drawbacks exists in the CMA this will guarantee the CMA allocation without failure. CMA requires physical contiguous memory as well as virtual contiguous memory because all the processes work in Linux on virtual memory addresses. We proposed that not all processes require virtually contiguous memory all the time. We observed a reason for the failure in existing CMA is mapping. This physical to virtual mapping also increases latency and allocation failure in existing CMA method. Therefore, we proposed the removal of mapping at the time of allocation of CMA memory and remap when it is required by process to work on virtual memory.

We just need to remap specific amount of size from whole allocation when process required working on virtual memory as shown in Figure 3. It will be initiated through the design and development of proposed wrapper device driver for CMA allocation and redirect all the CMA call through that driver. The driver itself will be implemented in such a way that it will allocate/de-allocate and maintain memory from CMA reserved area and internally this driver will use the original API with our provided attribute. Our provided attribute is do not map physically contiguous allocated memory with virtual contiguous memory, but we will maintain allocated physical address so that form that range whenever virtual memory required. We can have base address and size of memory, which need to be remapped virtually. Therefore, we propose to provide remap API in our driver to map it into virtual memory at the time of CMA allocation and when virtual memory is required from user/application. It will provide the interface to remap that allocated physical area to virtual area.

The proposed method provides the implementation of CMA allocation and deallocation. It remaps the specific physical address and its size to mapped into virtual address. It resolves the allocation failure, which is drawback of existing CMA, and the time taken to find out the available virtual address space, which

indirectly increase the latency of allocation as shown in Figure 3. The proposed approach provides very good solution to all the existing CMA drawbacks and resolving all the failure and latency issues. The performance of proposed approach has been evaluated on one of the 32-bit ARM board, which is beaglebone black.

Reserve CMA memory at boot time will be a same as existing CMA, but while allocating CMA, the memory will utilize the above-mentioned approach, i.e. the proposed new driver method and the driver will allocate CMA memory using `DMA_KERNEL_NO_MAPPING` attribute. It will not be mapped physical memory to virtual memory at the time of CMA allocation, but when application or the user of CMA required virtual memory, driver will provide the API to remap the physical memory to virtual at run time. Therefore, latency and allocation failure issues may be resolved in existing CMA through implementing 32-bit android device for this approach. The proposed driver can be developed with the help of this Beagle-bone Board.

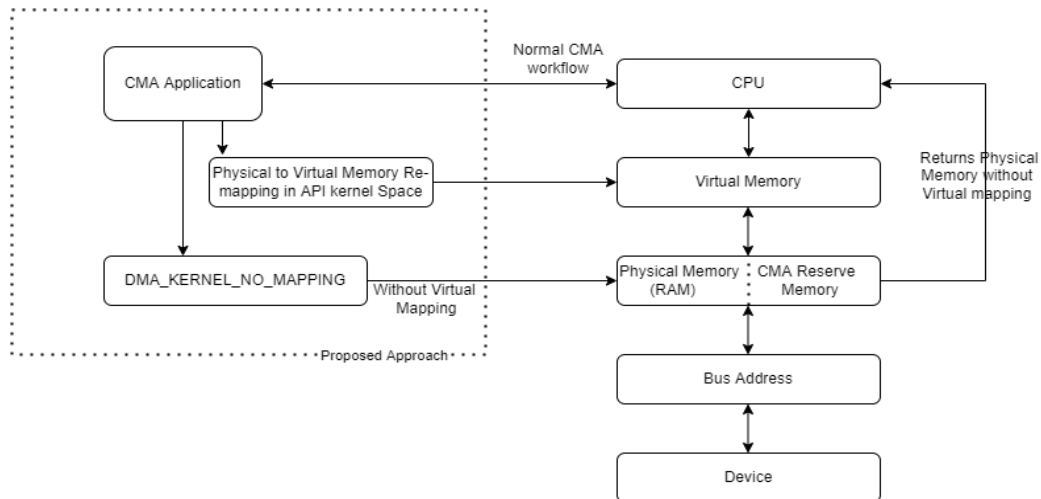


Figure 3. Proposed approach

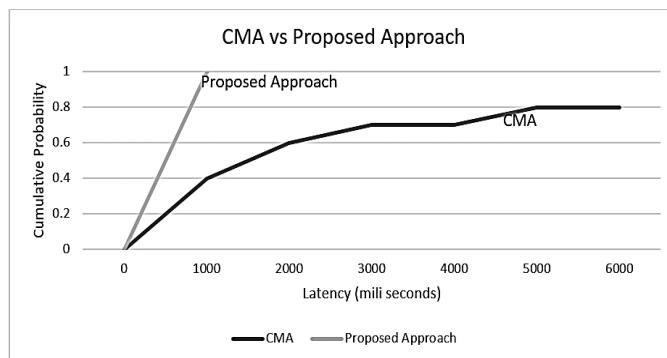
5. EXPECTED OUTCOMES

The expected outcomes are mentioned as: i) To reduce the latency and allocation failure issue in existing CMA by creating one driver in Linux kernel, which will be used to allocate and deallocate CMA memory; ii) When virtual memory (dereference pointer) is required, then only it will mapped with virtual memory (ultimately it will improve latency), otherwise the physical memory will be directly used by device without mapped to virtual memory; iii) By using `DMA_KERNEL_NO_MAPPING` attribute in DMA API, I will improve allocation failure issue; reduce workload of system to mapped and unmapped memory and latency issue in CMA; and iv) This solution will be useful in many applications like GPU/VPU for 4K Set Top box or Android TV, which requires large contiguous memory for performance and features with limited RAM.

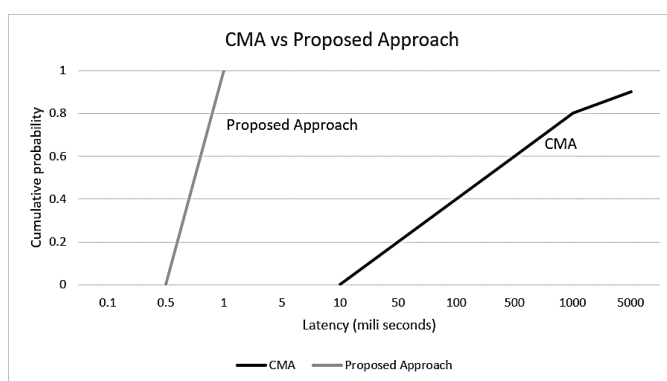
5.1. Proposed approach experiment on beagle bone black board for YouTube 4K video allocation and latency

This paper is evaluating the latency probability of CMA and proposed approach using the 32-bit Beagle bone device with YouTube 4K video. When a YouTube 4K video played by 32-bit Beagle bone device generally allocates 128 CMA pages 600 times with kernel threads and keeping the allocated memory as more as it can to utilize it again for forthcoming buffer stream. Hence, only the first buffer stream of video played requires CMA allocation. For conducting this study, one may configure the system which releases a memory allocation for a 4K video very quickly against subsequent buffer stream. Without subsequent tasks, every configuration demonstrates a 4K video latency of about 16 s. There is no difference between the latencies of the videos played using the CMA and the proposed approach. Because the latency is dominated by the YouTube 4K video application, not by contiguous memory allocation.

The expected evaluations are shown in Figure 4. In Figure 4(a), the CMA illustrates an extremely slower YouTube 4K video latency while the proposed approach has a much better latency. At its lowest, the CMA needs approx. 100 s to play a 4K video stream while a proposed approach requires 16 s. As mentioned in Figure 4(b), the latency of the proposed approach lies between 4 ms and 10 s, while that of the existing CMA lies between 900 ms and 550 ms in digital camera. This study suggests that contiguous memory allocation using the CMA gives very high and unpredictable latency in a real time application.



(a)



(b)

Figure 4. Latency comparisons between existing contiguous memory allocation and proposed approach for (a) YouTube 4K video and (b) digital camera

6. CONCLUSION

The proposed approach will remove all the drawbacks of existing CMA such as, large size memory allocation failure, direct pinning issue, indirect pinning issue in linux kernel. This approach improves the latency of the CMA system in linux kernel. This approach will remove extra expenses of the hardware's like scatter/gather DMA and IOMMU required to allocate contiguous memory in 32-bit devices. It will also overcome drawback of Buddy System has in fragmentation. Hence improved CMA will be supposed to give better results in those existing systems. This study will be help to resolve contiguous memory related issues in low end 32-bit devices in the future.





REFERENCES

- [1] S. J. Park, M. Kim, and H. Y. Yeom, "GCMA: Guaranteed Contiguous Memory Allocator," *IEEE Transactions on Computers*, vol. 68, no. 3, pp. 390–401, Mar. 2019, doi: 10.1109/TC.2018.2869169.
- [2] V. Kalpana, "A Novel Approach on Memory Management Systems," *International Journal for Research in Applied Science and Engineering Technology*, vol. 9, no. 3, pp. 509–512, Mar. 2021, doi: 10.22214/ijraset.2021.33259.
- [3] M. A. Ahmed, A. Aljumah, and M. G. Ahmad, "Design and Implementation of a Direct Memory Access Controller for Embedded Applications," *International Journal of Technology*, vol. 10, no. 2, p. 309, Apr. 2019, doi: 10.14716/ijtech.v10i2.795.
- [4] A. Changela, "VLSI Implementation of Direct Memory Access (DMA) Controller," *JETIR*, vol. 5, no. 5, pp. 18–23, 2018, [Online]. Available: www.jetir.org.
- [5] J. Corbet, "Five-level page tables," <https://lwn.net/Articles/717293/>. p. 1, 2017, [Online]. Available: <https://lwn.net/Articles/717293/>.
- [6] J. Corbet, "Cleanscache and Frontswap," <http://lwn.net/Articles/386090/>. p. 1, 2010, [Online]. Available: <http://lwn.net/Articles/386090/>.
- [7] J. Liu and X. Rival, "An array content static analysis based on non-contiguous partitions," *Computer Languages, Systems and Structures*, vol. 47, pp. 104–129, Jan. 2017, doi: 10.1016/j.cl.2016.01.005.
- [8] J. D. Leidel, "Bit contiguous memory allocation for processing in memory," in *Proceedings of MCHPC 2017: Workshop on Memory Centric Programming for HPC - Held in conjunction with SC 2017: The International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov. 2017, vol. 2017-January, pp. 11–19, doi: 10.1145/3145617.3145618.
- [9] M. Abadi et al., "TensorFlow: A system for large-scale machine learning," *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*, vol. 10, no. July, pp. 265–283, 2016, doi: 10.5555/3026877.3026899.





- [10] R. Williams, "Linux device drivers," in *Real-Time Systems Development*, Elsevier, 2006, pp. 410–434.
- [11] I. McDonald, *Linux kernel development*. Addison-Wesley, 2007.
- [12] R. Tucci, E. G. Khalaf, and R. P. Tucci, "Semi-Contiguous Memory Allocation for Efficient Sequential-Access. Semi-Contiguous Memory Allocation for Efficient Sequential-Access," in *Proceedings of the 2006 International Conference on Computer Design & Conference on Computing in Nanotechnology, CDES 2006*, 2006, pp. 135–140.
- [13] A. Faraz, "A Review of Memory Allocation and Management in Computer Systems," *Computer Science & Engineering: An International Journal*, vol. 6, no. 4, pp. 01–19, Aug. 2016, doi: 10.5121/cseij.2016.6401.
- [14] M. Kim, J. Koo, H. Lee, and J. R. Geraci, "Memory management scheme to improve utilization efficiency and provide fast contiguous allocation without a statically reserved area," *ACM Transactions on Design Automation of Electronic Systems*, vol. 21, no. 1, pp. 1–23, Dec. 2015, doi: 10.1145/2770871.
- [15] L. G. Kabari and T. S. Gogo, "Efficiency of Memory Allocation Algorithms Using Mathematical Model," *International Journal of Emerging Engineering Research and Technology*, vol. 3, no. 9, p. 55, 2015.
- [16] N. V. Patil A and P. S. Irabashetti, "Dynamic Memory Allocation: Role in Memory Management Government Polytechnic, Ahmednagar Maharashtra Government," *Research Article International Journal of Current Engineering and Technology*, vol. 4, no. 2, 2014, [Online]. Available: <http://inpressco.com/category/ijcet>.
- [17] J. Stultz, "Integrating the ION memory allocator," <http://lwn.net/Articles/565469/>. Sep. 2013, Accessed: Jan. 11, 2022. [Online]. Available: <http://lwn.net/Articles/565469/>.
- [18] J. Jeong, H. Kim, J. Hwang, J. Lee, and S. Maeng, "Rigorous rental memory management for embedded systems," *Transactions on Embedded Computing Systems*, vol. 12, no. SUPPL1, pp. 1–21, Mar. 2013, doi: 10.1145/2435227.2435239.
- [19] J. Jeong and J. Lee, "Rigorous Rental Memory Management for Embedded Systems KAIST CS / TR-2011-349 Department of Computer Science Rigorous Rental Memory Management for Embedded Systems," 2011.
- [20] A. Basu, J. Gandhi, J. Chang, M. D. Hill, and M. M. Swift, "Efficient virtual memory for big memory servers," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 237–248, Jun. 2013, doi: 10.1145/2508148.2485943.
- [21] D. Magenheimer, "In - kernel memory compression," <http://lwn.net/Articles/545244/>. 2013, [Online]. Available: <http://lwn.net/Articles/545244/>.
- [22] I. Usage and I. Implementation, "Continuous Memory Allocator," <https://events.static.linuxfound.org/>. p. 46, 2012.
- [23] M. Ferdman *et al.*, "Clearing the clouds: A study of emerging scale-out workloads on modern hardware," in *International Conference on Architectural Support for Programming Languages and Operating Systems - ASPLOS*, 2012, pp. 37–47, doi: 10.1145/2150976.2150982.
- [24] C. Stills, D. Memory, and A. Pools, "The Android ION Memory Manager," <http://lwn.net/Articles/48055/>, no. December 2011. Feb. 2013.
- [25] J. Jeong, H. Kim, J. Hwang, J. Lee, and S. Maeng, "DaaC: Device-reserved memory as an eviction-based file cache," *CASES'12 - Proceedings of the 2012 ACM International Conference on Compilers, Architectures and Synthesis for Embedded Systems, Co-located with ESWEEK*, pp. 191–200, 2012, doi: 10.1145/2380403.2380439.
- [26] T. W. Barr, A. L. Cox, and S. Rixner, "SpecTLB: A mechanism for speculative address translation," *Proceedings - International Symposium on Computer Architecture*, pp. 307–317, 2011, doi: 10.1145/2000064.2000101.
- [27] A. Jaleel, "Memory characterization of workloads using instrumentation-driven simulation," in *Web Copy*: <http://www.glue.umd.edu/ajaleel/workload>, 2010, pp. 1–12, [Online]. Available: <http://www.jaleels.org/ajaleel/workload/SPECanalysis.pdf>.
- [28] P. Hornyack, L. Ceze, S. Gribble, D. Ports, and H. Levy, "A Study of Virtual Memory Usage and Implications for Large Memory," 2013.
- [29] L. N. Bairavasundaram, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau, "Dependability analysis of virtual memory systems," in *Proceedings of the International Conference on Dependable Systems and Networks*, 2006, vol. 2006, pp. 355–364, doi: 10.1109/DSN.2006.26.
- [30] T. Nakajima and H. Tezuka, "Virtual memory management for interactive continuous media applications," in *International Conference on Multimedia Computing and Systems -Proceedings*, 1997, pp. 415–423, doi: 10.1109/mmcs.1997.609752.
- [31] D. Raghuvanshi, "Memory Management in Operating System," *International Journal of Trend in Scientific Research and Development*, vol. Volume-2, no. Issue-5, pp. 2346–2347, Aug. 2018, doi: 10.31142/ijtsrd18342.
- [32] M. Freedman, "The compression cache: virtual memory compression for handheld computers," 2000, [Online]. Available: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.8782>.

BIOGRAPHIES OF AUTHORS



Anmol Suryavanshi     as born in India 1st July 1989. He received my bachelor's degree in Information Technology Engineering from KBC North Maharashtra University (NMUJ), Jalgaon, India in 2011. Later he received his master's degree from Rajiv Gandhi Technological University, (RGPV), Bhopal, India in 2014. He is currently pursuing Ph.D. from Oriental University, Indore (OU), India since February 2020. He can be contacted at email: eranmol89@gmail.com.



Dr. Sanjeev Kumar Sharma     is working as Professor (CSE) and Dean Student Welfare in Oriental Institute of Science and Technology, Bhopal and Adjunct faculty in the Oriental University, Indore. Apart from teaching, he is also associated with the activities of Indian Army through National Cadet Corps (NCC) to motivate and encourage the students to join the armed forces. He completed his Ph. D in computer science and engineering from Devi Ahilya University. He has more than 40 research papers in various national, International Journals and Conferences. He is having 20 years of teaching experience and 12 years of experience in Research. He is also the member of various societies such as, AMIE, CSI, ACM. He can be contacted at email: sanjeevsharma@oriental.ac.in.