FAT-Pointer based range addresses

Akilan Selvacoumar

Mathematics and Computer Sciences
Heriot Watt University

Year 2 progression report of:
Doctor of Philosophy

June 2024

Abstract

The increasing disparity between application workloads and the capacity of Translation Lookaside Buffers (TLB) has prompted researchers to explore innovative solutions to mitigate this gap. One such approach involves leveraging physically contiguous memory to optimize TLB utilization. Concurrently, advancements in hardware-level system security, exemplified by the Capability Hardware Enhanced RISC Instructions (CHERI) architecture, offer additional opportunities for improving memory management and security.

CHERI introduces capability-based addressing, a novel approach that enhances system security by associating capabilities with memory pointers. These capabilities restrict access to memory regions, thereby fortifying the system against various security threats. Importantly, the mechanisms implemented in CHERI for enforcing memory protection can also serve as accelerators for standard user-space memory allocators. By leveraging capability-based addressing, memory allocators can efficiently manage memory resources while ensuring robust security measures are in place.

Table of contents

Chapter 1

Introduction

In the dynamic landscape of computing, the pursuit of optimal performance is a constant endeavor, especially as applications evolve to handle increasingly complex workloads. One critical aspect influencing performance is memory management, where efficient utilization of resources is paramount. Translation Lookaside Buffers (TLBs) play a pivotal role in this regard, expediting memory access by storing recently accessed memory translations. However, as applications grow in size and complexity, the capacity of TLBs often struggles to keep pace, leading to performance bottlenecks. To address this challenge, researchers have turned to innovative solutions, one of which involves harnessing the benefits of huge pages. Huge pages, also known as large pages, allow for the allocation of memory in significantly larger chunks compared to traditional small pages. By reducing the number of TLB entries needed to access a given amount of memory, huge pages offer a potential avenue for optimizing TLB utilization and thereby enhancing overall system performance.

Simultaneously, advancements in hardware-level security, such as the Capability Hardware Enhanced RISC Instructions (CHERI) architecture, present additional opportunities for performance enhancement. CHERI's capability-based addressing approach not only strengthens system security by tightly controlling memory access but also provides avenues for accelerating memory management operations.

In this context, the integration of huge pages into memory management strategies alongside capability-based addressing in architectures like CHERI offers a compelling synergy. By optimizing TLB utilization through the utilization of huge pages and leveraging the security features of capability-based addressing, significant performance improvements can be realized. This approach not only enhances system security but also accelerates memory access.

1.0.1 TLB based approaches

Efficient memory management, particularly in the context of Translation Lookaside Buffer (TLB) optimization, has been a focal point of research and development within computer architecture. Various techniques have been proposed to mitigate TLB-related bottlenecks and improve overall system performance.

Huge Pages:

This is used to map a very large region of memory to a single entry. This small/large region of memory is physically contiguous. Most implementations of huge pages [Panwar et al.] are size aligned, For example for the x86 architecture the huge pages size are 4KB, 2MB and 1GB pages.

Segment:

A segment[Basu et al.] can be viewed as mapping between contiguous virtual memory and contiguous physical memory. The property of a segment allows it to be larger than a page. Direct Segment allows the user to set a single segment for an application. Two registers are added to mark the start and end of the segment. Any virtual address within this region can be translated by adding the fixed offset between the virtual and physical address.

Range Memory Mapping (RMM):

RMM[Karakostas et al.] introduces the concept of adding an additional range table. For large allocations RMM eagerly allocates contiguous physical pages. The following allocations creates large memory ranges that are both virtually and physically contiguous. RMM builds on the concept of Direct segment by adding offset to translate a virtual address to physical address. RMM compares address with range boundaries to decide which range it belongs to. RMM queries the range table ofter an L1 TLB miss.

FlexPointer:

FlexPointer[Chen et al.] is based on the RMM[Karakostas et al.] paper. FlexPointer does eagerly allocate pages which are physically contiguous and stores the ID to translate a virtual address to physical address on the remaining unused bits on the 64 bit virtual address. The paper contribution mentions shifting the TLB lookup to an earlier stage to improve latency of accessing the TLB entries. FlexPointer immediate queries the range TLB for translations rather than the RMM paper which waits for the L1 TLB miss.

1.0.2 CHERI

A capability functions as a token that provides the holder with the authority to execute specific actions. By carefully managing who possesses these capabilities, it is possible to enforce security measures, such as ensuring that a particular section of software can only read from a designated range of memory addresses.

The concept of cap abilities has a rich history in computer science, tracing back to early systems designed to enhance security and manage resources effectively. For instance, foundational work discussed in sources like [3] offers a comprehensive narrative on the evolution of capability architectures. This historical perspective can be further enriched by incorporating insights from more recent studies, such as those found in [Miller].

CHERI (Capability Hardware Enhanced RISC Instructions)[Woodruff et al.] represents a modern embodiment of this long-standing idea. It introduces more granular control over permissions, allowing for finer distinctions in what actions can be performed and by whom. Moreover, CHERI is designed to integrate seamlessly with contemporary processor instruction sets, ensuring that these advanced security features can be implemented on modern hardware platforms. This adaptation not only revitalizes the capability concept but also significantly enhances its applicability and effectiveness in current computing environments.

Chapter 2

Fat Pointer Based Range Addresses

FAT-Pointers, combined with the capabilities of the CHERI (Capability Hardware Enhanced RISC Instructions) architecture, introduce robust memory safety and security features by incorporating additional metadata with memory pointers. This enhanced architecture utilizes concepts such as FlexPointer, Range Memory Mapping (RMM) to manage memory effectively.

Range addresses play a pivotal role within this framework, defining memory regions bounded by a starting address (Upper) and an ending address (Lower). These range addresses are encoded within FAT-pointers, allowing for precise control over memory regions.

The functionality of ranges encompasses several key aspects:

- **Creation of Physically Contiguous Memory Ranges**: By defining memory regions that are physically contiguous, systems can achieve optimal memory access patterns, enhancing performance and efficiency.

- **Encoding Ranges as Bounds to the Pointer**: Integrating range bounds directly into FAT-pointers enables the architecture to enforce memory access restrictions at the pointer level thus allowing tracking of memory ranges on a pointer level.

- **Instrumenting Block-Based Allocators with Physically Contiguous Memory**: The integration of range-based memory concepts into memory allocation systems, such as block-based allocators, facilitates the efficient management and utilization of physically contiguous memory blocks, mitigating issues related to memory fragmentation.

Figure 2.1 illustrates the methodology employed to leverage the CHERI 128-bit FAT-pointer scheme for facilitating block-based memory management on physically contiguous memory, which is depicted on the right side of the figure. This technique contrasts with the conventional mmap approach, the details of which are elaborated in section ("To be added later").

In figure 2.1, the green-highlighted section marks the unused space between the 48th and 64th bits within the FAT-pointer. This area of unused bits presents an opportunity to store additional metadata, potentially enhancing the capabilities of the memory management system. The potential applications of this extra space are discussed in the future work section ("To be added later"), where we explore how this additional metadata storage could be used to further optimize memory allocation.

2.0.1 Range creation and huge pages

In this implementation, memory ranges are established using bounds encoded within the FAT-pointer, adhering to the CHERI 128-bit bounds compression scheme[Woodruff et al.]. The memory chunk defined by the upper and lower bounds is always physically contiguous. Initially, a huge page of arbitrary size is allocated. Within this huge page, custom-sized memory segments are allocated using a custom-designed mmap function, which overrides the existing block-based mmap function. Once the memory is physically allocated through this custom mmap function, bounds are set to track the memory block, eliminating the need for traditional TLB usage for this purpose. Traditional TLB usage involves maintaining numerous TLB entries, often supplemented by an L2 TLB and other hierarchical structures, to translate virtual addresses to physical addresses. This approach requires multiple entries to handle various memory segments, leading to increased overhead and complexity in address translation. Conversely, the current approach streamlines this process by using a single TLB entry to translate multiple addresses within a contiguous memory range. This reduces the number of required TLB entries, simplifying the translation process and improving efficiency. By consolidating address translations into a single TLB entry, this method minimizes the overhead associated with managing numerous TLB entries and leverages the bounds encoded within the FAT-pointer for efficient memory tracking and access. This approach allows for precise and efficient memory management within the allocated huge page.

Figure 2.2 illustrates a straightforward use-case in which the dark pink line represents a single, large contiguous memory area, or huge page. Within this huge page, the orange and blue lines indicate two separate memory allocations equivalent to invoking malloc twice to allocate memory in distinct regions. This scenario simulates a block-based memory allocator operating within the confines of the huge page. The allocations leverage the bounds encoded in the FAT-pointer, ensuring tracking and efficient management of the allocated memory regions. By using the FAT-pointer bounds, this method maintains the integrity and contiguity of the allocated blocks within the huge page.

2.0.2 Software Stack

The software stack is based on CHERIBSD[4], selected because ARM officially supports Morello's performance counters[1] on this operating system. As illustrated in the figure 2.3, the setup includes a C program that is linked to the prototype memory allocator or to various memory allocators being benchmarked, as described in section ("Evaluation section"). This linkage can occur in two ways: either as a shared object file during compile time for larger allocators, or as a header file for smaller allocators, ensuring flexibility and efficiency in memory management.

The custom mmap function, tailored to ensure physically contiguous memory allocation, is a key component of this system. This function is linked to the contigmem driver, which has been modified from the DPDK[Bi and Wang] library to meet the specific needs of this implementation. The contigmem driver is essential for managing large contiguous memory blocks and is loaded during the system boot process. It reserves a huge page of arbitrary size, with the size parameter set based on the requirements of the conducted experiments.

This integration ensures that the memory allocation process is optimized for performance, leveraging the contiguity of memory blocks and the capabilities provided by the CHERI architecture and the Morello platform. By using the contigmem driver and the custom mmap function, the system achieves efficient memory allocation and tracking, crucial for the high-performance needs of the application.

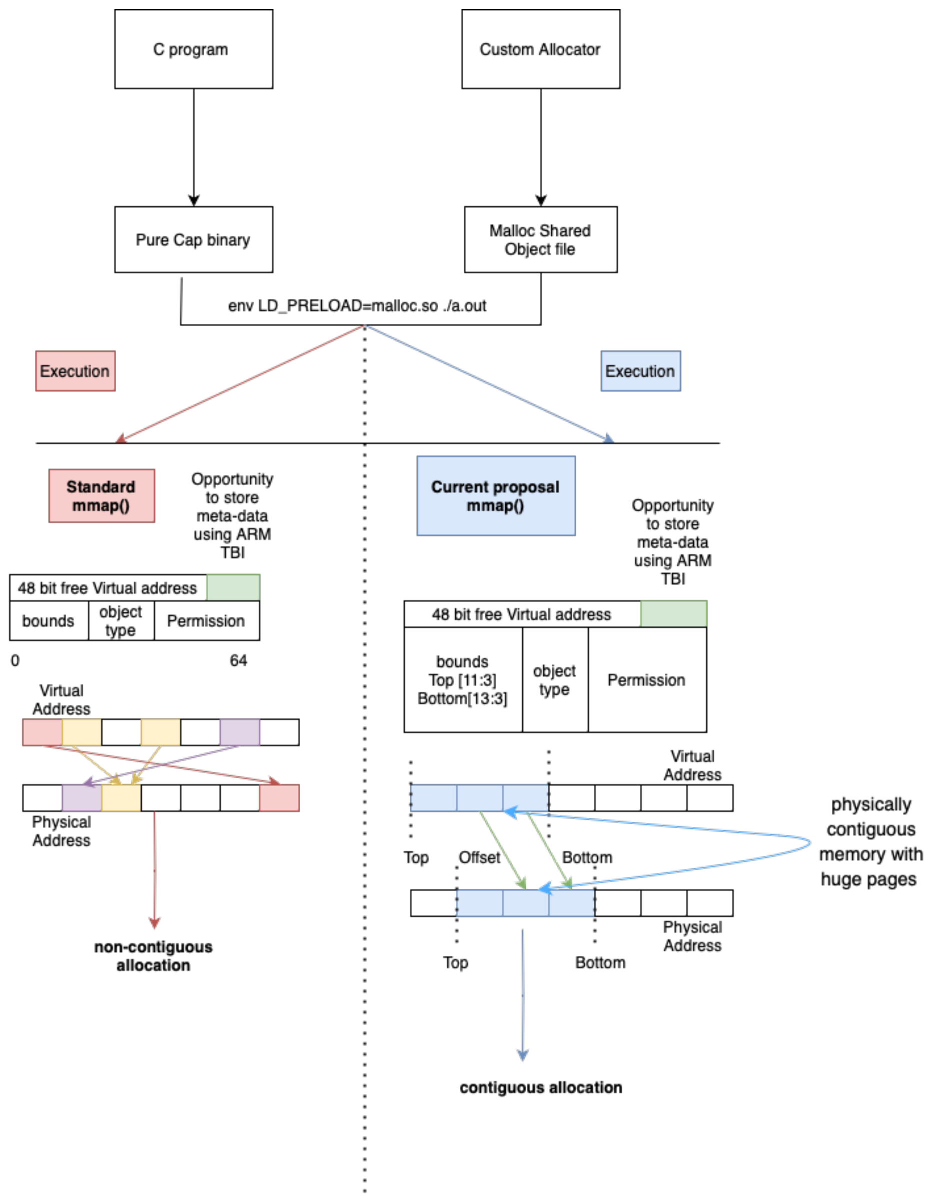

C program
Custom Allocator
Pure Cap binary
Malloc Shared Object file
env LD_PRELOAD=malloc.so ./a.out
Execution
Execution
Standard mmap()
Opportunity to store meta-data using ARM TBI
48 bit free Virtual address
bounds
object type
Permission
0
64
Virtual Address
Physical Address
non-contiguous allocation
Current proposal mmap()
Opportunity to store meta-data using ARM TBI
48 bit free Virtual address
bounds Top [11:3] Bottom[13:3]
object type
Permission
Virtual Address
Top
Offset
Bottom
physically contiguous memory with huge pages
Physical Address
Top
Bottom
contiguous allocation

Fig. 2.1 High overview architecture

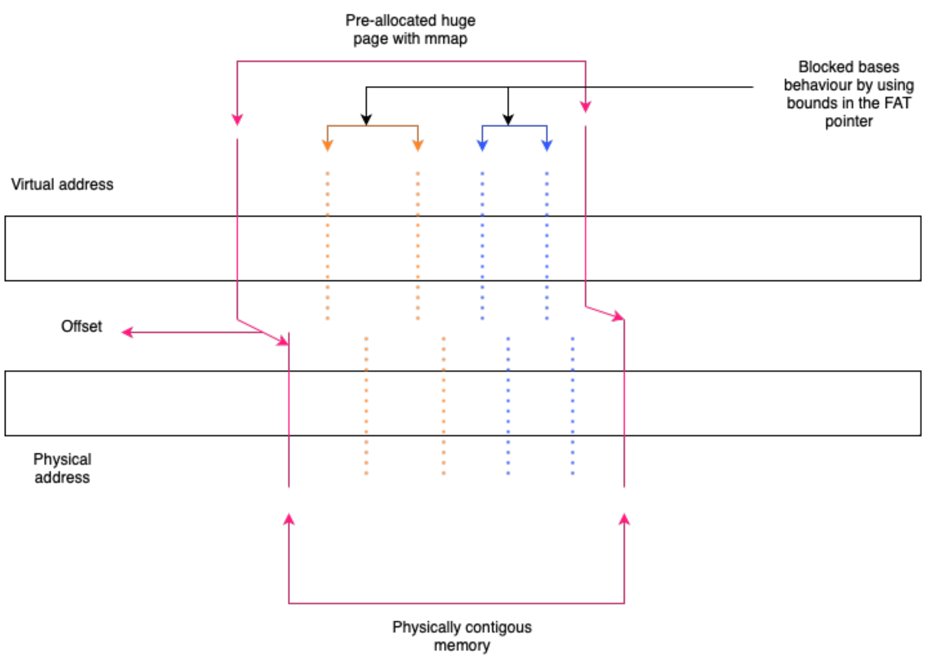

Pre-allocated huge page with mmap
Blocked bases behaviour by using bounds in the FAT pointer
Virtual address
Offset
Physical address
Physically contigous memory

Fig. 2.2 Range of memory

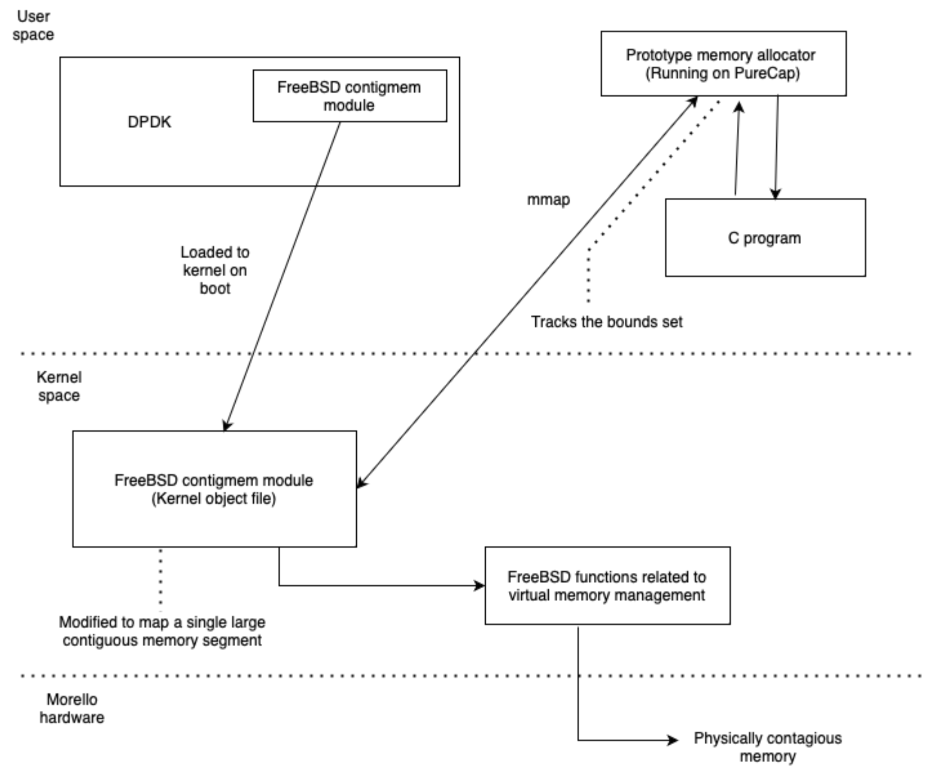

User space
DPDK
FreeBSD contigmem module
Prototype memory allocator (Running on PureCap)
mmap
C program
Loaded to kernel on boot
Tracks the bounds set
Kernel space
FreeBSD contigmem module (Kernel object file)
FreeBSD functions related to virtual memory management
Modified to map a single large contiguous memory segment
Morello hardware
Physically contagious memory

Fig. 2.3 Overview of the software stack

Chapter 3

Evaluation

To evaluate the performance of FAT-Pointer based range addresses a sample implementation we used the morello board with CheriBSD's Benchmark ABI[2] compilation mode for accurate performance recordings. The evaluation is to identify CheriBSD's default memory allocator SnMalloc against the prototype memory allocator using the contribution of this paper in terms of: To assess the performance of FAT-Pointer-based range addressing in the sample implementation, we utilized the Morello board running CheriBSD in Benchmark ABI compilation mode to ensure precise performance measurements. The evaluation focuses on comparing CheriBSD's default memory allocator, SnMalloc, against the prototype memory allocator developed in this study, with particular attention to the following aspects:

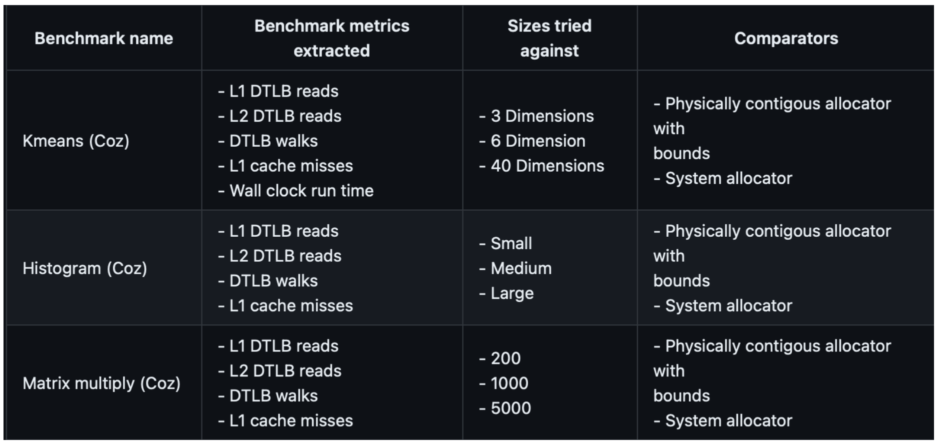
Metric name	type of graph	tool used	x axis	y axis
DTLB L1 read	line graph	Pmcstat	Time	DTLB L1 reads (each second)
DTLB L2 read	line graph	Pmcstat	Time	DTLB L2 reads (each second)
DTLB walk	line graph	Pmcstat	Time	DTLB Walks (each second)
L1 cache miss	line graph	Pmcstat	Time	L1 cache miss (each second)
Wall clock run time	bar graph	time	Benchmarks	Time

3.0.1 Benchmarks used

: To conduct the evaluations, we utilized the COZ[] benchmark suite, a well-regarded tool specifically designed to measure and analyze performance improvements in concurrent programs. The COZ benchmark suite provides a robust framework for identifying bottlenecks and evaluating the performance impact of various optimization techniques. By leveraging COZ, developers can gain precise insights into the efficiency and scalability of their concurrent code, making it an ideal choice for rigorous performance analysis.

From the extensive set of benchmarks provided by COZ, we selected four representative C programs. These programs were chosen based on their relevance to common concurrent programming patterns and their ability to effectively demonstrate the strengths and weaknesses of different optimization strategies. The selected programs cover a range of concurrency scenarios, ensuring a comprehensive evaluation of performance improvements.

By implementing these modifications, we ensured that the selected C programs not only adhered to CHERI's security model but also maintained their functional and performance characteristics. This allowed us to effectively use the COZ benchmark suite to analyze the performance in a CHERI-enhanced environment.



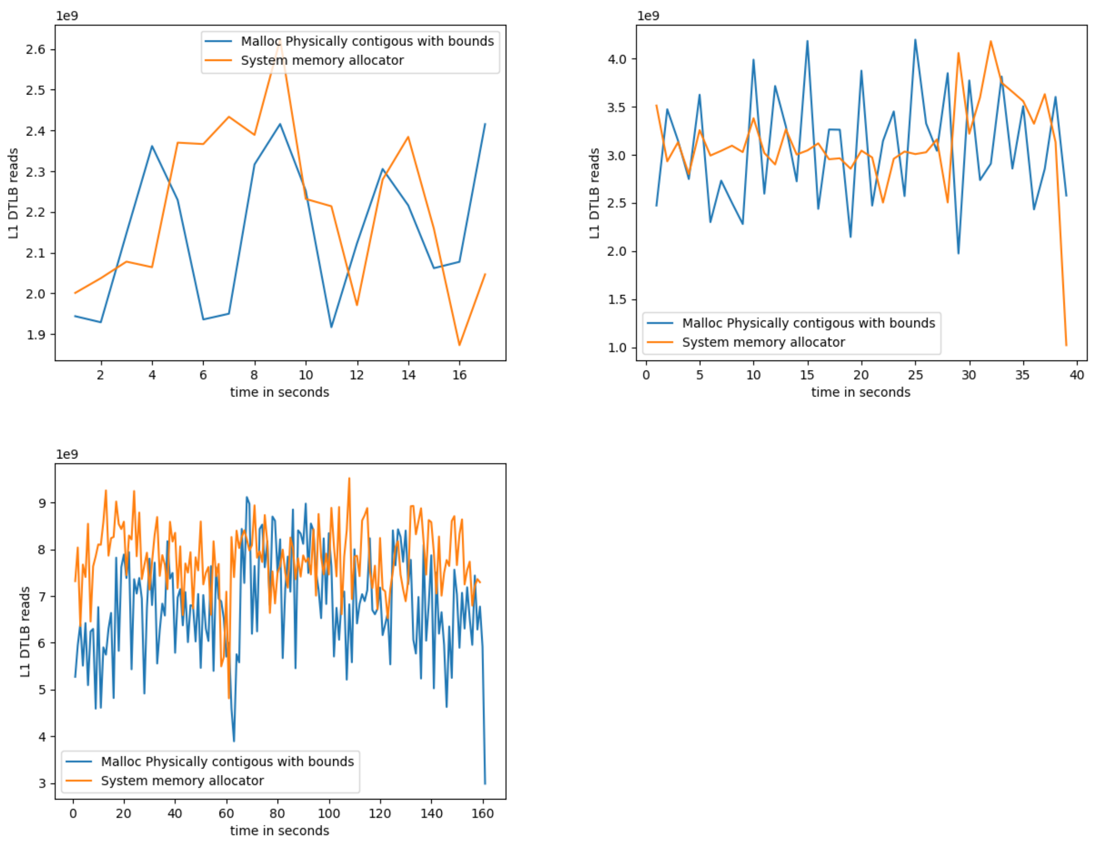
Benchmark name	Benchmark metrics extracted	Sizes tried against	Comparators
Kmeans (Coz)	- L1 DTLB reads - L2 DTLB reads - DTLB walks - L1 cache misses - Wall clock run time	- 3 Dimensions - 6 Dimension - 40 Dimensions	- Physically contigous allocator with bounds - System allocator
Histogram (Coz)	- L1 DTLB reads - L2 DTLB reads - DTLB walks - L1 cache misses	- Small - Medium - Large	- Physically contigous allocator with bounds - System allocator
Matrix multiply (Coz)	- L1 DTLB reads - L2 DTLB reads - DTLB walks - L1 cache misses	- 200 - 1000 - 5000	- Physically contigous allocator with bounds - System allocator

3.0.2 DTLB L1 reads:

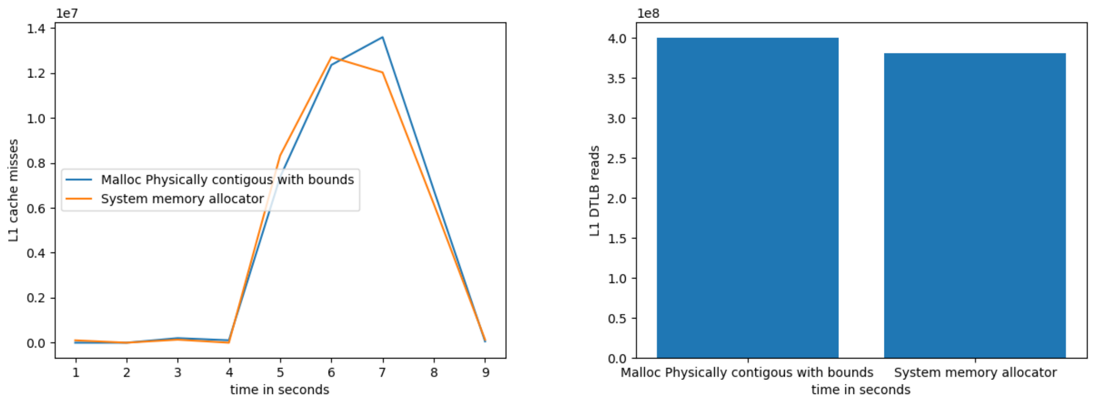
The Graphs above represent the DTLB L1 reads which is a Performance counter from the ARM specs. The counter increments for every Memory-read or Memory-write operation that necessitates an access to the Level 1 data or unified Translation Lookaside Buffer (TLB). Each access to a TLB entry is counted including multiple accesses caused by single instructions.

3.0.3 DTLB L2 reads:

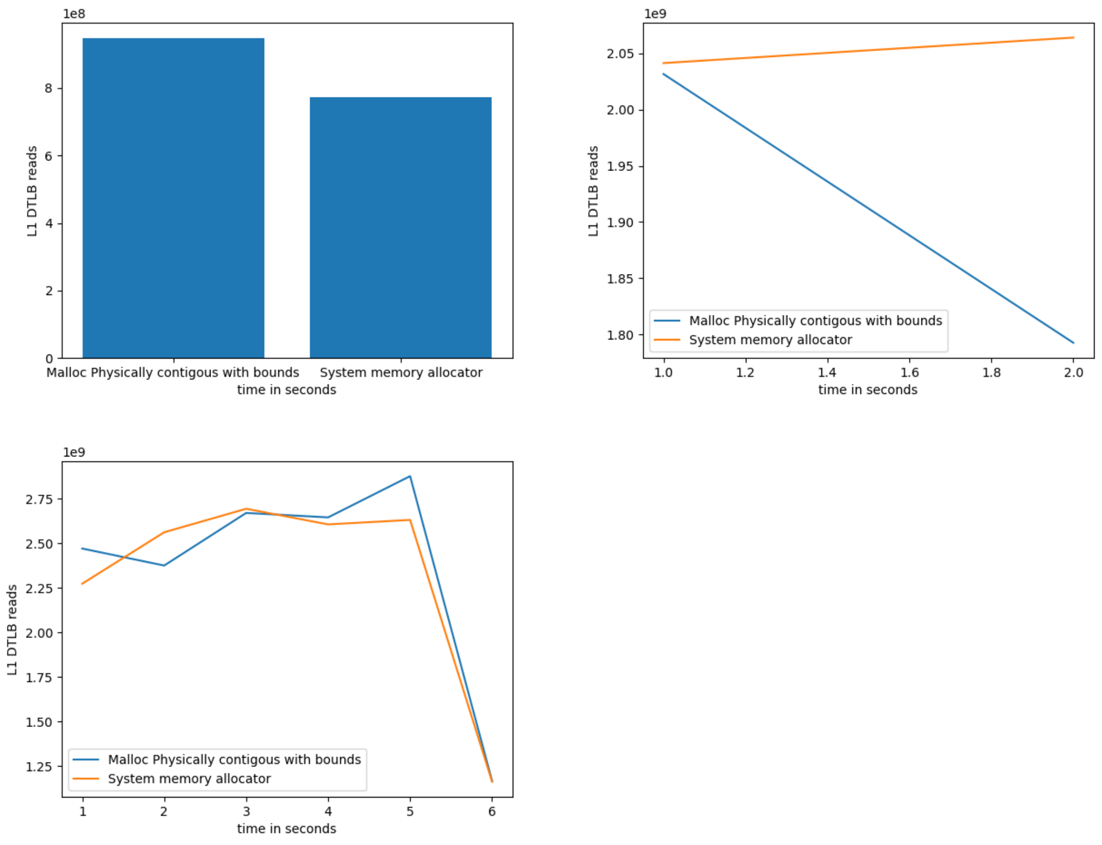
Similar to how L1 TLB reads are counted, DTLB L2 counts every read operation that accesses the Level 2 data or unified TLB. Each time there is a read to an entry in the Level 2 TLB, it is counted by the DTLB L2.



(a) Kmeans DTLB L1 reads

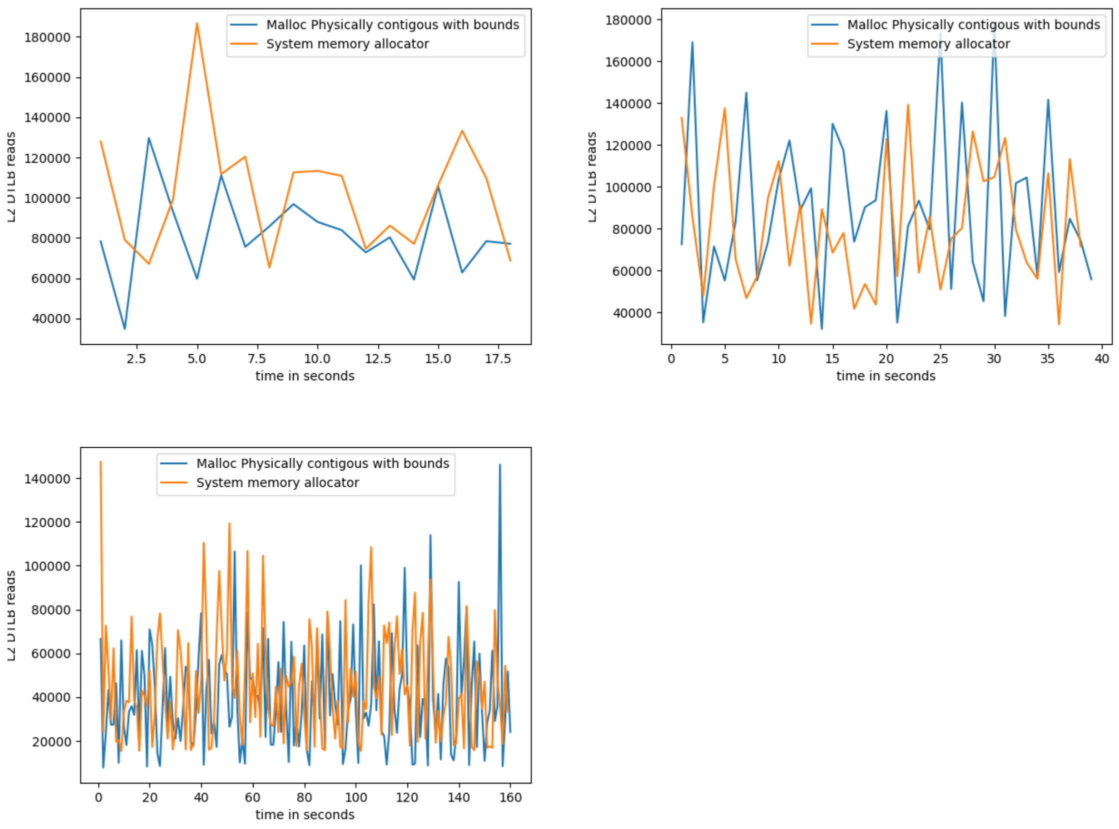


(a) Matrix Multiply DTLB L1 reads

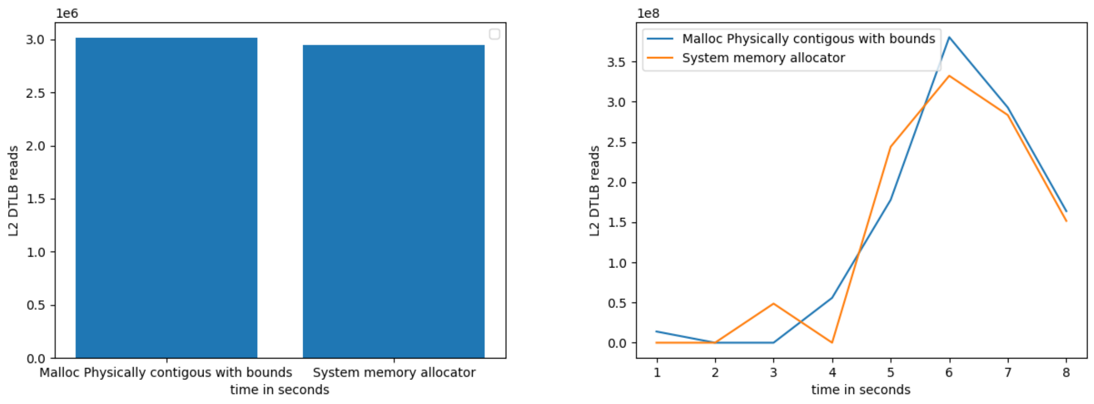


(a)

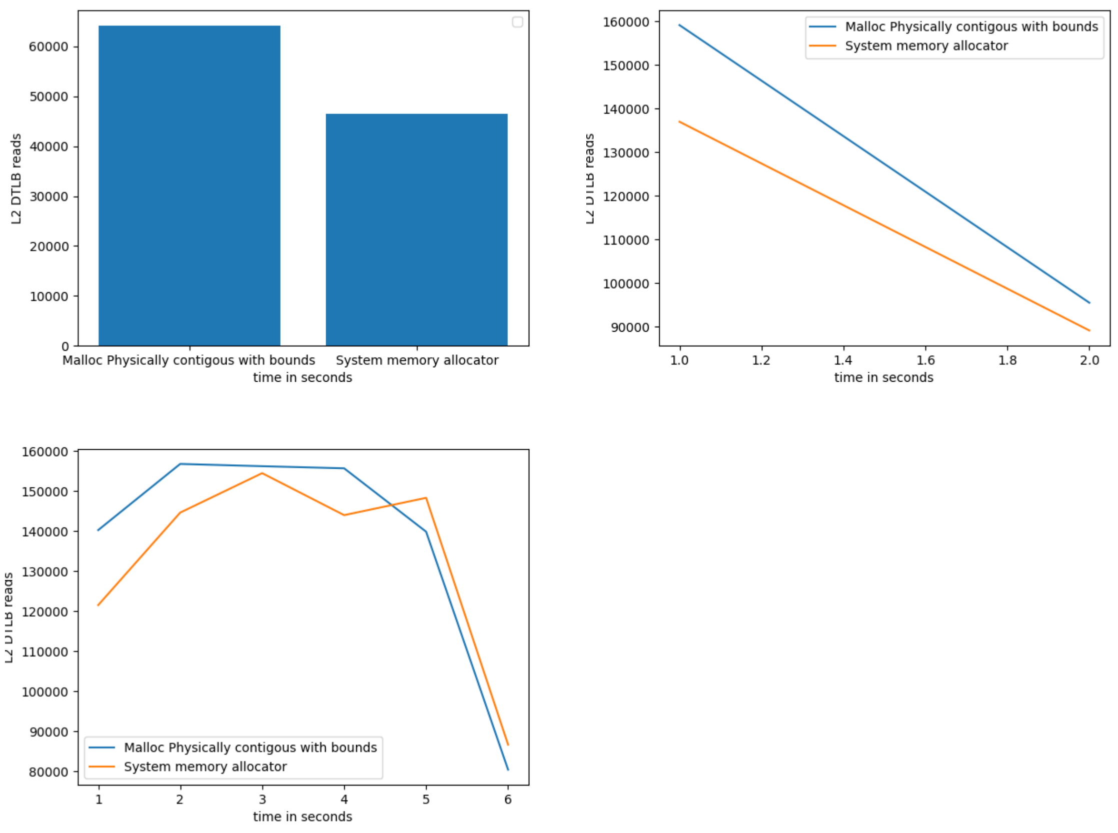
Fig. 3.3 Histogram DTLB L1 reads



(a) Kmeans DTLB L2 reads

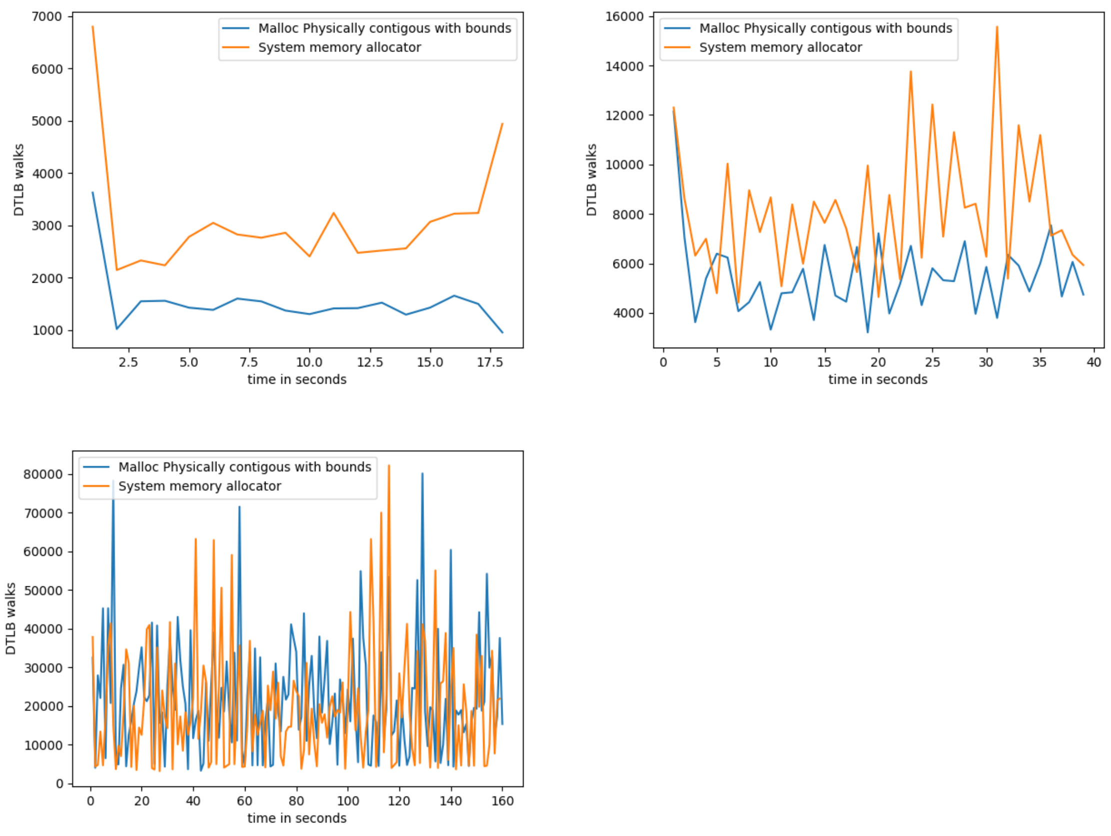


(a) Matrix Multiply DTLB L2 reads



(a)

Fig. 3.6 Histogram DTLB L2 reads



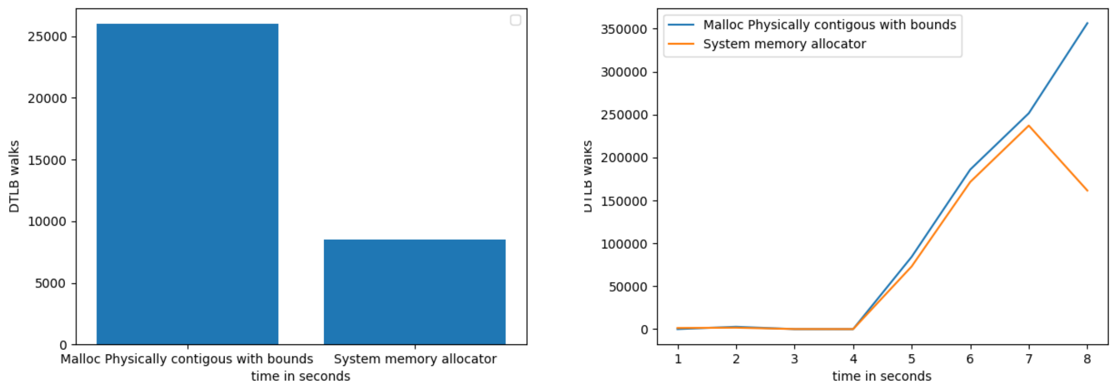
(a) Kmeans DTLB Walks

3.0.4 DTLB walks:

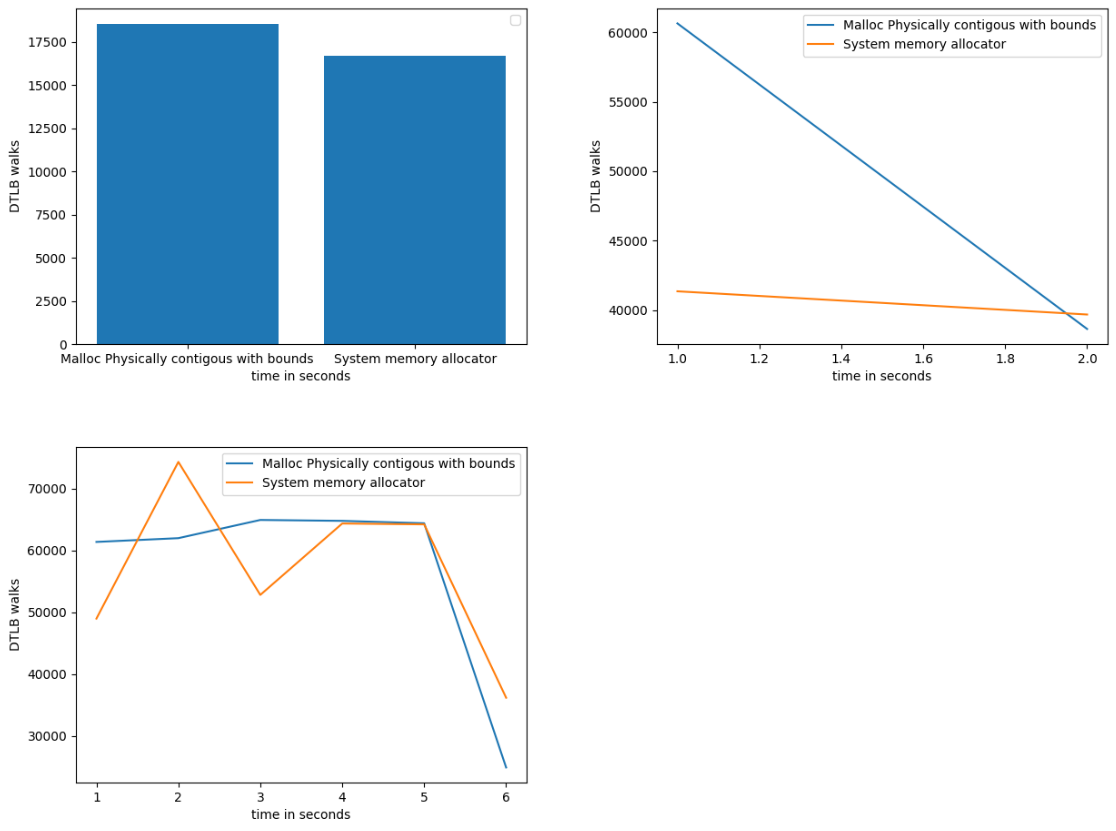
The DTLB walk counter counts each Memory-read operation or Memory-write operation that causes a TLB access to at least the Level 2 data or unified TLB. Each access to a TLB entry is counted including refills of Level 1 TLBs.

3.0.5 L1 cache miss:

L1 cache miss counter counts each Memory-read operation to the Level 1 data or unified cache counted by L1 cache miss counter that incurs additional latency because it returns data from outside of the Level 1 data or unified cache of this PE. The event indicates to software that the access missed in the Level 1 data or unified cache and might have a significant performance impact due to the additional latency compared to the latency of an access that hits in the Level 1 data or unified cache.

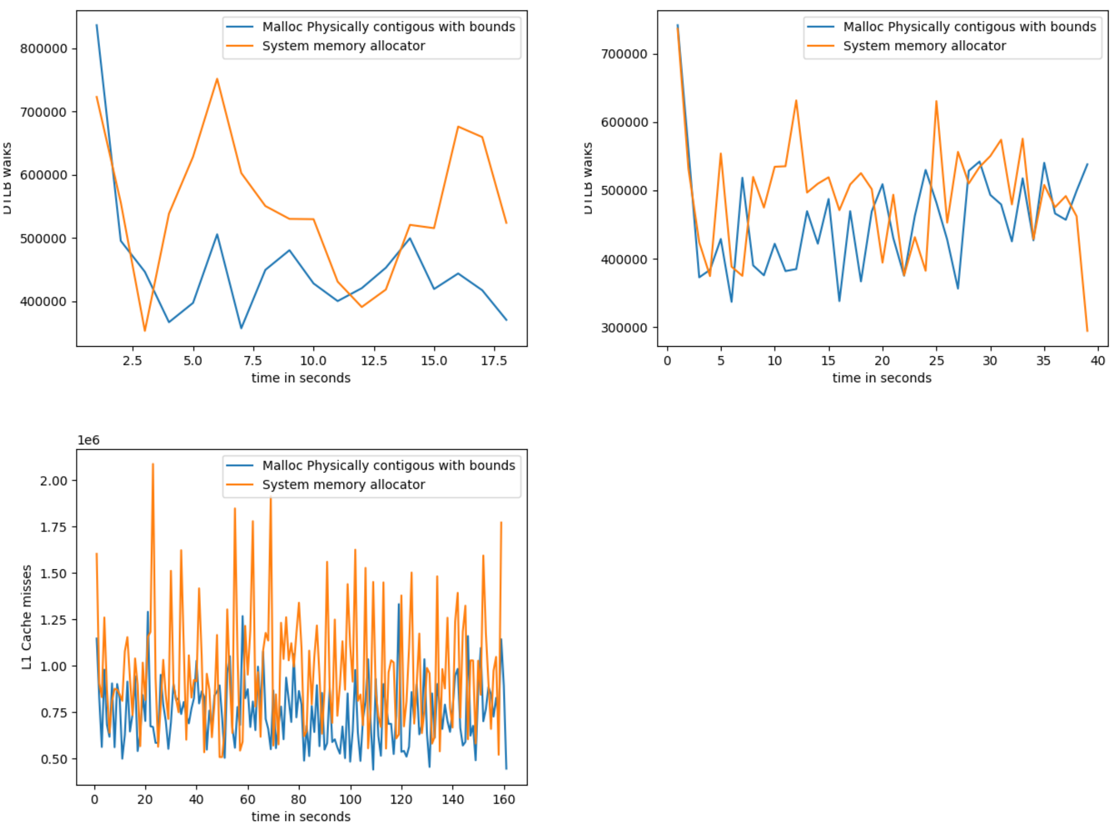


(a) Matrix Multiply DTLB Walks

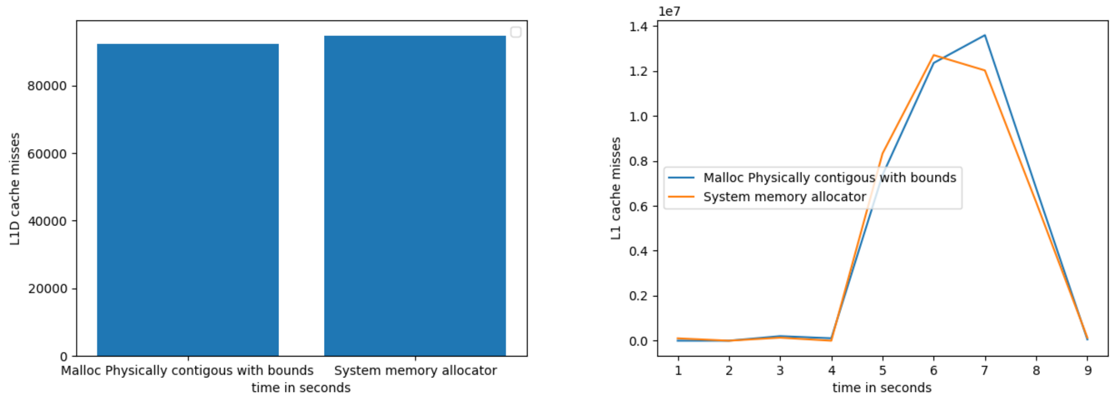


(a)

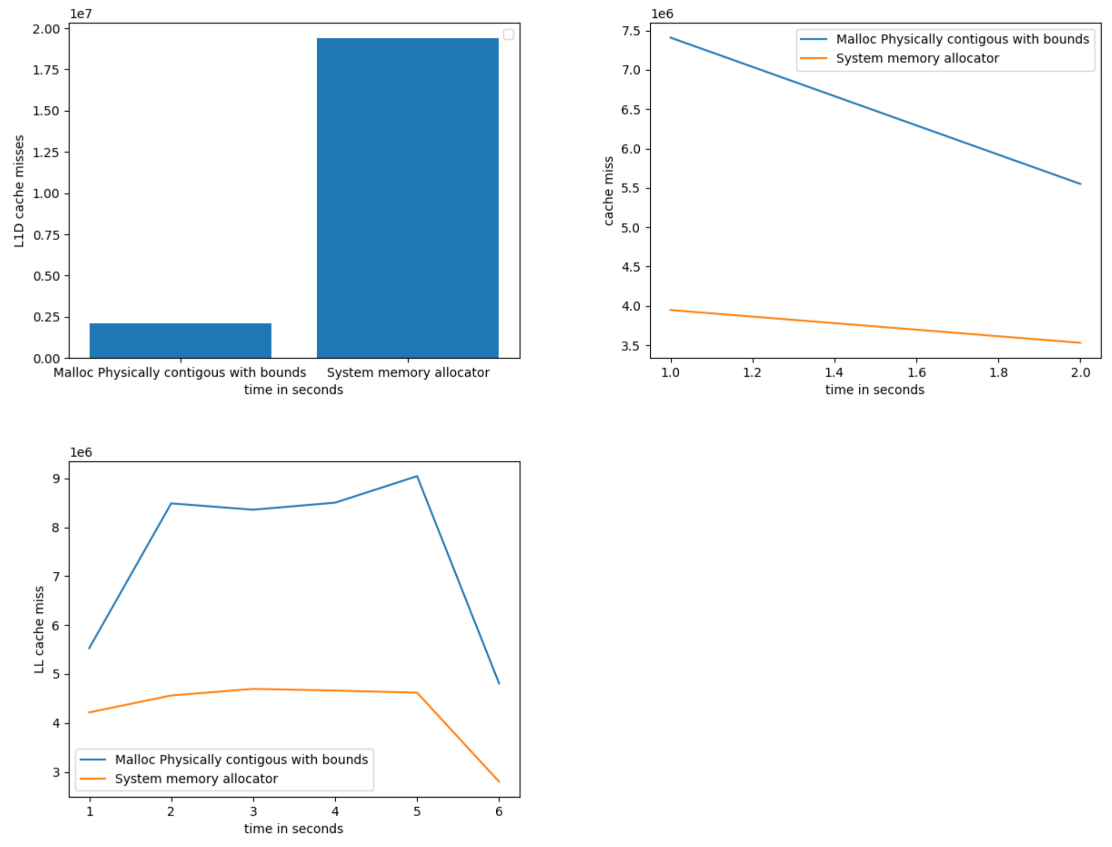
Fig. 3.9 Histogram DTLB Walks



(a) Kmeans L1 cache miss



(a) Matrix Multiply L1 cache miss



(a)

Fig. 3.12 Histogram L1 cache miss

Chapter 4

Future work

The current experimental setup on the ARM Morello board is constrained by the requirement that all memory reads must pass through the Translation Lookaside Buffer (TLB) for address translation. This necessitates frequent TLB lookups, potentially leading to performance bottlenecks. The planned future work aims to address this by leveraging CHERI (Capability Hardware Enhanced RISC Instructions) extensions on the RISC-V architecture, specifically using the Tooba implementation.

Storing Offsets Directly on Pointers

In the current ARM Morello setup, address translations rely on the TLB. The future approach on RISC-V Tooba involves storing the offset directly within the pointer. This is possible due to CHERI's capability model, which supports fine-grained memory protection and can encode bounds within pointers. Utilizing Bounds in CHERI for Block-Based Allocation:

CHERI capabilities allow pointers to carry metadata about memory bounds, providing hardware-enforced memory safety. By encoding the offset and bounds within the pointer, the system can directly access memory without needing intermediate translations via the TLB. This enables the implementation of a block-based allocator that can efficiently manage memory allocations and deallocations within defined bounds. Bypassing the TLB in RISC-V Tooba.

Hardware Modifications

: The Verilog design of the RISC-V processor will be modified to allow certain memory operations to bypass the TLB. This means that when a pointer with encoded offset and bounds is used, the system can directly compute the physical address from the capability information. This modification reduces the dependency on the TLB, decreasing latency and improving performance, especially for frequent memory operations. Transition to a Single-Address-Space Operating System (SASOS).

Concept of SASOS

: In traditional operating systems, there is a clear separation between user space and kernel space. This separation is enforced by memory protection mechanisms and address translation through the TLB. In a Single-Address-Space Operating System, this distinction is removed. Both user applications and the kernel share the same contiguous address space.

Advantages of SASOS with CHERI

:

- Simplified Memory Management : Without the need to switch between user and kernel spaces, memory management becomes simpler and more efficient. The kernel allocator can be the same as the user space allocator, operating on a single, contiguous chunk of memory.
- Unified Allocator: The unified memory allocator can efficiently manage memory for both kernel and user applications, leveraging CHERI's capability-based protection to prevent unauthorized access. This reduces overhead and potential fragmentation issues associated with maintaining separate memory spaces.

References

[1] Arm architecture reference manual for a-profile architecture.

[2] Benchmark ABI - CheriBSD 23.11 new features tutorial.

[3] Capability-based computer systems.

[4] Getting started with CheriBSD 23.11 - getting started with CheriBSD 23.11.

[Basu et al.] Basu, A., Gandhi, J., Chang, J., Hill, M. D., and Swift, M. M. Efficient virtual memory for big memory servers.

[Bi and Wang] Bi, H. and Wang, Z.-H. DPDK-based improvement of packet forwarding. 7:01009. Publisher: EDP Sciences.

[Chen et al.] Chen, D., Tong, D., Yang, C., Yi, J., and Cheng, X. FlexPointer: Fast address translation based on range TLB and tagged pointers. 20(2):1–24.

[Karakostas et al.] Karakostas, V., Gandhi, J., Ayar, F., Cristal, A., Hill, M. D., McKinley, K. S., Nemirovsky, M., Swift, M. M., and Ünsal, O. Redundant memory mappings for fast access to large memories. In *Proceedings of the 42nd Annual International Symposium on Computer Architecture*, pages 66–78. ACM.

[Miller] Miller, M. S. Towards a unified approach to access control and concurrency control.

[Panwar et al.] Panwar, A., Bansal, S., and Gopinath, K. HawkEye: Efficient fine-grained OS support for huge pages. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 347–360. ACM.

[Woodruff et al.] Woodruff, J., Joannou, A., Xia, H., Fox, A., Norton, R. M., Chisnall, D., Davis, B., Gudka, K., Filardo, N. W., Markettos, A. T., Roe, M., Neumann, P. G., Watson, R. N. M., and Moore, S. W. CHERI concentrate: Practical compressed capabilities. 68(10):1455–1469.