Contents

1 Plan

This document outlines the proposed PhD research plan for the forthcoming academic year, building upon the outcomes and insights gained during the preceding year.

1.1 Current experiments:

- (Experiment 1)FAT allocator with huge pages (EuroSys26 paper draft)

- (Experiment 2)FAT allocator to bypass TLB. (Proposal of ongoing expirement)
- (Experiment 3)Allocator evaluation based on stripping instruction calls from larger allocators. (Appendix)

1.2 Experiment cancelled:

1. Uni-Kernel Development (Reason): Work scaled down to fit within the allocated within PhD timeframe and instead introduced an extension of the FAT allocator as the third experiment.

1.3 Link to the Previous PhD Plan

- `https://github.com/Akilan1999/phd-thesis/releases/download/Year2/thesis.pdf`

1.4 Summary of the Previous Plan

1.4.1 Phase 1: FAT-Pointer Mechanism (JulySeptember 2024)

1. 1st to 15th July 2024
    - Investigated causes of L1 TLB misses associated with contiguous memory allocation.
    - Executed performance benchmarking using COZ[1] on selected C programs.
    - Ported the kernel module to support SnMalloc, the default allocator in CheriBSD.
2. 15th to 30th July 2024
    - Conducted benchmarking using the SPEC and XSBench suites.
    - Performed comparative analysis with both baseline and modified SnMalloc implementations.
3. August 2024
    - Initiated drafting of a paper for submission to EuroSys, focusing on the FAT-Pointer memory allocator (Current: On review stage).
4. September 2024

- Compiled and structured thesis chapter related to the FAT-Pointer architecture.
- Finalised and submitted the EuroSys paper.

1.4.2 Phase 2: RISC-V Integration (October 2024 May 2025)

1. October to December 2024

   - Modified the Bluespec implementation to enable TLB bypass in the memory access pipeline (Current: On progress).
   - Configured the experimental platform and evaluation toolchain (Current: On progress).

2. January to February 2025

   - Undertook experimental evaluation of the FAT-Pointer system on RISC-V (Toooba) (Current: Todo).
   - Commenced drafting of a technical paper for ISMM based on RISC-V integration results (Current: Todo).

3. March to May 2025

   - Addressed outstanding tasks and technical backlog.
   - Continued development of the corresponding thesis chapter.

1.4.3 Phase 3: Uni-Kernel Deployment (May 2025 September 2026)

1. May to December 2025

   - Ported the memory allocator to a CHERI-enabled Uni-Kernel environment (Current: Cancelled)
   - Designed and implemented a unified memory allocator to support both kernel and user-level allocations (Current: Cancelled)
   - Initiated drafting of a manuscript targeted at OSDI (Current: Cancelled)

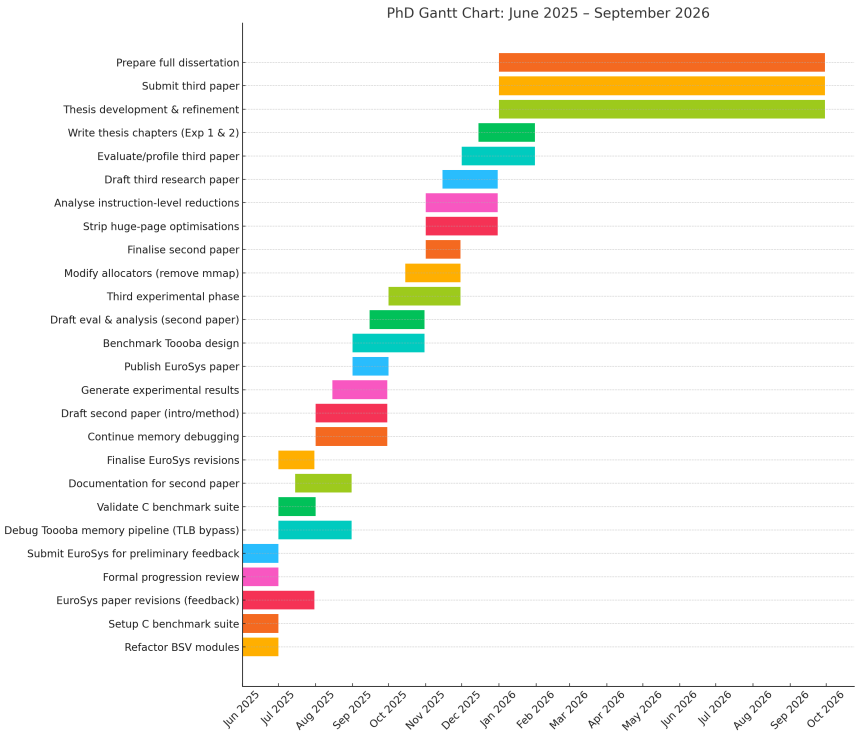
2. January to September 2026

   - Finalised documentation and submission of the PhD thesis (Current: New plan to still keep these dates unchanged)

- Submitted third research paper based on extended evaluation (Current: Replaced with "Allocator evaluation based on stripping instruction calls from larger allocators")

1.5 Current Research Plan

This section outlines a comprehensive timeline of research activities and academic milestones to be undertaken between June 2025 and September 2026 as part of the PhD to be focused on the CHERI Toooba architecture[2]. It includes the refactoring of BlueSpec[3] SystemVerilog[4] modules and the development of a bare-metal C benchmark suite. The work involves in-depth debugging of the Toooba memory subsystem, the design and evaluation of in-depth analyses of the FAT allocator and performance analysis at the instruction level. Alongside these technical efforts, the timeline reflects the structured drafting of academic publications and the ongoing development of the PhD thesis.



1.5.1 June to July 2025

- Refactor BlueSpec SystemVerilog (BSV)[4] modules within the CHERI Toooba architecture.
- Set up a bare-metal C benchmark suite for execution on the Bluespec simulation platform.
- Incorporate supervisory team feedback into revisions of the EuroSys paper.
- Undertake formal progression review requirements.
- Submit EuroSys paper to the CHERI research team at the University of Glasgow for preliminary feedback.

1.5.2 July to August 2025

- Engage in extensive debugging of the Toooba memory pipeline, specifically targeting the TLB bypass path.
- Finalise and validate the C benchmark suite for the Toooba evaluation.
- Begin technical documentation of the Toooba workflow, to support a 2nd publication.
- Conclude revisions to the EuroSys paper by the end of July.

1.5.3 August to September 2025

- Continue debugging efforts within the Toooba memory subsystem.
- Draft the abstract, introduction, and methodology sections of the second research paper.
- Aim to generate preliminary experimental results for inclusion in the evaluation for the 2nd experiment.

1.5.4 September to October 2025

- Publish the EuroSys paper detailing the FAT-Pointer allocator.
- Commence benchmarking of the Toooba design.
- Simultaneously draft the evaluation and analysis sections of the 2nd paper.

1.5.5 October to November 2025

- Initiate 3rd experimental phase, aimed at deeper evaluation of prior experiments.
- Modify memory allocators (TcMalloc and Mesh) to remove reliance on system ‘mmap‘ call.

1.5.6 November to December 2025

- Finalise 2nd paper for peer review.
- Strip away huge-page-specific optimisations from JeMalloc, TcMalloc, and Mesh.
- Analyse instruction-level reductions and performance implications.
- Commence drafting of the 3rd research paper, building on contributions from the EuroSys paper.

1.5.7 December 2025 to January 2026

- Conduct evaluation and profiling for the 3rd paper.
- Commence thesis chapter write-up for Experiments 1 and 2.

1.5.8 January to September 2026

- Continue thesis development and refinement across all experimental chapters.
- Finalise and submitted 3rd paper for peer review.
- Prepare complete PhD dissertation for submission.

References

[1] C. Curtsinger and E. D. Berger, “Coz: finding code that counts with causal profiling,” in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP ’15. New York, NY, USA: Association for Computing Machinery, 2015, p. 184197. [Online]. Available: https://doi.org/10.1145/2815400.2815409

[2] P. Rugg, "Efficient spatial and temporal safety for microcontrollers and application-class processors," 2022. [Online]. Available: https://www.repository.cam.ac.uk/handle/1810/353468

[3] R. S. Nikhil and Arvind, "What is bluespec?" *SIGDA Newsl.*, vol. 38, no. 23, p. 1, Dec. 2008. [Online]. Available: https://doi.org/10.1145/1862867.1862868

[4] O. Arcas-Abella and N. Sonmez, "Bluespec SystemVerilog," in *FPGAs for Software Programmers*, D. Koch, F. Hannig, and D. Ziener, Eds. Cham: Springer International Publishing, 2016, pp. 165–172. [Online]. Available: https://doi.org/10.1007/978-3-319-26408-0_9

[5] "Arm Architecture Reference Manual for A-profile architecture." [Online]. Available: https://developer.arm.com/documentation/ddi0487/latest

[6] A. Hunter, C. Kennelly, P. Turner, D. Gove, T. Moseley, and P. Ranganathan, "Beyond malloc efficiency to fleet efficiency: a hugepage-aware memory allocator," in *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*. USENIX Association, Jul. 2021, pp. 257–273. [Online]. Available: https://www.usenix.org/conference/osdi21/presentation/hunter

1.6 Appendix

1.7 (Experiment 3)Allocator evaluation based on stripping instruction calls for larger allocators

1.7.1 Box 1 (Transparent huge pages)

The diagram 1 highlights three specific implementations, the first of which is the standard Transparent Huge Pages (THP) mechanism employed by modern memory allocators. THP initially focuses on handling smaller memory allocations. As the volume of allocations increases, it employs a strategy that consolidates these smaller allocations into contiguous memory regions. Once aggregated, these regions are subsequently converted into larger memory pages, typically of size 4MB, as seen in allocators like jemalloc. This approach optimises memory management by reducing fragmentation and improving performance through the use of larger page sizes.

This approach, however, introduces additional overhead, including the operations required to consolidate smaller allocations and modify Translation Lookaside Buffer (TLB) entries. These modifications can initially

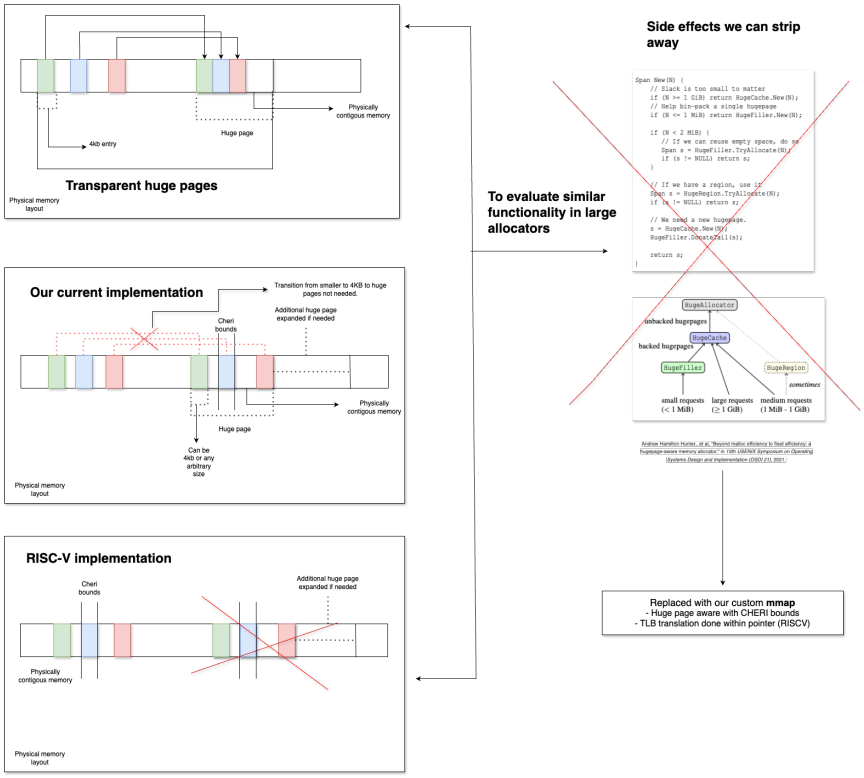


Figure 1: Deprecating the use of THP with CHERI bound aware embedded mmap.

increase the likelihood of TLB misses, as the process of grouping and reorganizing memory allocations temporarily disrupts the efficiency of TLB utilization. It is only after the successful creation of the huge page that the benefits materialize, leading to a reduction in TLB misses due to the improved alignment of memory access patterns with the larger page size.

1.7.2 Box 2 (Our current implementation)

Box 2 outlines the current implementation, which involves the pre-allocation of huge pages and leverages CHERI (Capability Hardware Enhanced RISC Instructions) bounds to meticulously track each allocation within these pages. This approach enables a single TLB entry, combined with the precise bounds defined by CHERI capabilities, to facilitate block-based memory management within physically contiguous regions. By enforcing strict bounds on pointers, the system ensures that each pointer can only access memory within its explicitly defined region.

Another critical aspect of this implementation is the ability to define bounds of dynamic sizes, which stands in contrast to traditional approaches that rely on fixed-size page entries. Fixed-size entries inherently require multiple TLB entries, regardless of the actual memory usage, leading to inefficiencies. In the current approach, when the allocated memory within a huge page reaches its capacity, a new huge page is allocated. However, a notable limitation of this method is its dependence on the maximum huge page size supported by the underlying processor architecture. In this case, the system is constrained by the huge page size defined by the CHERI-enhanced ARM v8.1[5] architecture. While this approach offers flexibility in memory allocation and reduces the need for multiple TLB entries, it is ultimately bounded by the hardware's architectural specifications.

1.7.3 Box 3 (RISC-V implementation)

The third approach, as outlined in Box 3, deviates from the use of huge pages and does not require memory to be physically contiguous. In this model, each pointer is designed to store comprehensive metadata at the pointer necessary for the translation from virtual to physical addresses. This metadata enables the system to manage memory allocations without the constraints of physical contiguity, thereby offering greater flexibility in memory utilization. By embedding translation information directly within the pointers, this approach eliminates the need for large, contiguous memory regions and allows for more granular and dynamic memory management.

1.7.4 Building up from the work of Box 2 and Box 3 (Side effects we can strip away)

From a high-level perspective, the primary distinction between Box 2 and Box 3 lies in the use of huge pages in the former and their absence in the latter. Both approaches share the common advantage of reducing the number of instructions required in modern memory allocators by eliminating the need for transitioning between smaller and larger pages. This simplification streamlines memory management processes. However, this document has not yet provided a detailed breakdown or quantitative analysis of the specific performance implications or trade-offs.

As illustrated to the right of the diagram, a sample snippet of TC malloc from the paper "Beyond malloc Efficiency to Fleet Efficiency"[6] is provided. In the proposed approach, the entire span function, which is essential in TC malloc, would become unnecessary. Additionally, the approach offers the advantage of simplifying memory management by integrating mmap directly within the allocator. This integration eliminates the need for separate mechanisms to handle memory mapping.

1.7.5 Evaluation:

- The number of instructions that can be eliminated from a page-aware memory allocator by adopting the proposed approach.
- A comparative analysis of the memory allocator's performance using wall-clock runtime measurements, both with and without the modified mmap implementation.
- While CHERI Purecap introduces additional instructions, such as bounds checks, the overall approach aims to determine whether it reduces the total number of instructions when compared to a traditional ARMv8 Clang program using the same allocator. This involves evaluating the trade-offs between the overhead of CHERI-specific instructions and the potential reductions in allocator-emitted instructions.