Thesis Proposal: Performance and Microarchitectural Evaluation of the CHERI Architecture

Martin Fink

October 2024

Details

Type:	MA/BA/GR
Timeframe:	Wintersemester 2024
Keywords:	CHERI, Evaluation, Microarchitectural Analysis, Morello

1 Introduction

Despite ongoing efforts, memory vulnerabilities remain a relevant topic. Several studies have shown that in large software projects, memory safety bugs make up between 70 % and 75 % of their high-impact security vulnerabilities [4, 8, 9]. Software-based approaches either require modifications to source code or incur high performance overheads [7, 6, 3, 5, 1]. The CHERI (Capability Hardware Enhanced RISC Instructions) architecture implements memory safety primitives at the hardware-level by extending 64-bit pointers to 128+1 bits, including a validity bit, bounds, and permissions. On memory accesses, the hardware checks the bounds and permissions, promising better performance compared to existing software-based solutions. Still, this protection comes at a performance cost.

2 Objective of the Thesis

In this thesis/guided research, you will perform a detailed microarchitectural analysis of Arm's Morello [2] board, which implements the CHERI architecture. You will implement a framework to measure and investigate several performance and architectural metrics, including:

- Instruction cycles and latencies of CHERI instructions.
- Microarchitectural details such as the cache and pipeline behavior when handling capabilities.

- The cost of performing CHERI access checks on memory operations.
- Potential trade-offs between security benefits and performance overhead.

References

[1] Periklis Akritidis et al. "Baggy Bounds Checking: An Efficient and Backwards-Compatible Defense against Out-of-Bounds Errors." In: *USENIX Security Symposium*. Vol. 10. 2009, p. 96.

[2] Richard Grisenthwaite et al. "The Arm Morello Evaluation Platform—Validating CHERI-Based Security in a High-Performance System". In: *IEEE Micro* 43.3 (May 2023), pp. 50–57. ISSN: 1937-4143. DOI: 10.1109/MM.2023.3264676. URL: https://ieeexplore.ieee.org/abstract/document/10123148 (visited on 10/04/2024).

[3] Trevor Jim et al. "Cyclone: a safe dialect of C." In: *USENIX Annual Technical Conference, General Track*. 2002, pp. 275–288.

[4] *Memory Safety*. Accessed on March 14, 2024. URL: https://www.chromium.org/Home/chromium-security/memory-safety/ (visited on 03/14/2024).

[5] Santosh Nagarakatte et al. "SoftBound: Highly compatible and complete spatial memory safety for C". In: *Proceedings of the 30th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2009, pp. 245–258.

[6] George C Necula, Scott McPeak, and Westley Weimer. "CCured: Type-safe retrofitting of legacy code". In: *Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2002, pp. 128–139.

[7] Konstantin Serebryany et al. "AddressSanitizer: A fast address sanity checker". In: *2012 USENIX annual technical conference (USENIX ATC 12)*. 2012, pp. 309–318.

[8] Gavin Thomas. *A proactive approach to more secure code*. Accessed on March 14, 2024. URL: https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code/ (visited on 07/16/2019).

[9] Jeff Vander Stoep and Chong Zhang. *Queue the Hardening Enhancements*. Accessed on March 14, 2024. URL: https://security.googleblog.com/2019/05/queue-hardening-enhancements.html (visited on 05/09/2019).