The Arm Morello Evaluation Platform—Validating CHERI-Based Security in a High-Performance System & Efficient Tagged Memory

Yash Shah (ys562)

November 2024, Word count: 1646

1 Contributions of the Morello paper

Morello [1] is an experimental platform for prototyping, evaluation, and exploration of Capability Hardware Enhanced RISC Instructions (CHERI) for mass-market adoption. By including a prototype version of the ARMv8.2 architecture, it can run standard ARM software with the complete set of capability extensions. These architectural hardware extensions allow for fine-grained memory safety and software compartmentalization.

This project is also the first example demonstrating the feasibility of a full-scale industrial architecture being formally verified through a machine-checked mathematical proof. This verifies its security properties, and greatly enhances trust in Morello.

2 Contributions of the Efficient Tagged Memory (TM) paper

Tagged memory [2] is a pivotal enabler for security invariants such as capabilities and pointer integrity by distinguishing between untyped data words and the new hardware-enforced type. It identifies a single-bit (using an embedded architecture for additional metadata) tag shadowspace as a common proposed requirement to draw this distinction.

Due to its focus on commodity memory hardware (avoidance of non-standard memory widths), it explores an in-memory table approach to store this metadata. To reduce DRAM traffic to $< 5\%$ for most applications, two key tag access patterns are identified and explored—page-scale spatial locality and each tag bit covering many bits of data (cache amplification factor), leading to the

development of a table cache (with a merged-cache hierarchy). Importantly, a hierarchical tag table compression scheme almost eliminates table-cache pressure for applications not utilizing tagged pointers.

3 Metadata storage

Differing choices are made for metadata storage. Morello repurposes the ECC bits for tag storage which has two main advantages. Firstly, since Morello is not deployed in environments requiring high reliability or availability, the ECC bits are unused and so repurposing them saves key development time and reduces cost. Secondly, it allows for comparison and evaluation against other mechanisms for holding tags (such as TM's table-based approach)—key for an evaluation platform.

In comparison, TM demonstrates feasibility for the commodity scenario noting that ECC is not economical for many applications and may be required for its original purpose. Although having a wider memory would be ideal, this is not the case for commodity hardware with power-of-two widths. It also rejects a dedicated tag storage option—it is hard to justify the cost and complexity of such a small, special dedicated memory. Instead TM relies upon a DRAM table and exploiting the spatial and temporal patterns of tag-access by caching tags, which fits within commodity hardware constraints that TM targets to demonstrate feasibility.

4 Microarchitectural implications

Although both papers take differing approaches for tag-storage for the entire memory, Morello and TM resort to widening of registers and caches respectively to accommodate the tag bit. Unlike DRAM, registers and cache lines are not commodity logic, and so there is little downside here.

Some argue that wider caches waste storage and power as tags can be redundant across large regions of memory. However splitting the tag and data cache is itself redundant as tagged memory logically extends every memory word and is problematic fitting another memory access in the pipeline. Additionally, we would require a separate tag-table for every virtual address space, increasing complexity and likely costing much more power and performance.

Additionally, both designs are impacted by other microarchitectural choices. Since Morello aims to augment an ordinary Neoverse-N1 CPU, significant work went into the creation of a base, bound, and permission decompression scheme which did not affect any critical path on the device. By parallelizing the base-bound decompression with the address-generation arithmetic, the bounds check is timed similarly to a TLB-hit.

TM's optimal tag-cache size is sensitive to LLC size. We see a large difference in DRAM traffic overhead when the same sized tag-cache is used for the small 256 KiB and large 8 MiB LLCs. The relationship between the optimal tag-cache

and LLC size is non-linear (there isn't a fixed multiple of LLC coverage we should aim for), so tuning is required. However, due to the tag cache amplification factor (which itself depends on the amount of metadata held for every word), the tag-cache required is relatively small.

5 Compatibility with legacy software

Morello requires the recompilation of programs with the Cheri LLVM compiler in order for the program to use CHERI versions of various instructions (e.g. load-and-store), which utilize capabilities. However, a CHERI-equipped system may still have to run software which has not been recompiled. Thus, traditional load and store instructions taking integer addresses held in data registers are also supported.

Additionally, the inclusion of an experimental Default Data Capability (DDC) register allows for sandboxing of "legacy" software. The address held in the DDC is the base address of the sandbox, compartmentalizing several instances of the same legacy software.

Although TM does not require recompilation (it is by itself not ABI-breaking), legacy software not utilizing tagged pointers may suffer unnecessarily from table-cache pressure. The hierarchical tag table compression scheme essentially eliminates this problem due to the large grouping factor. Since the leaf table is all zeros for such legacy programs, only the root table needs to be accessed.

6 Additional programmer effort

Enforcement of spatial memory safety with Morello requires relatively little programmer effort. Recompilation and linking with the CHERI LLVM toolchain will implement each address as a capability and use CHERI versions of various instructions. This is possible in part due to Morello's decision to make all 31 architectural registers and stack pointers capable of holding capabilities, simplifying and unconstraining compiler register allocation.

Additionally, the capability registers overlap with the general-purpose registers. This allows the same architectural state to be viewed as 128-bit registers holding capabilities, or 64-bit data values, distinguished by metadata bits with each register. This simplifies ABI implications and reduces hardware complexity.

However, if one wants to utilize more fine-grained software compartmentalization than what is possible through MMU-based approaches utilizing different processes, significant programmer effort may be required. It is unknown whether this can be offset by the performance and security benefits provided through enhanced compartmentalization.

Meanwhile, TM does not directly require additional programmer effort as the shadowspace is hidden memory (which provides both integrity and is naturally

protected since it cannot be named by instructions). However, higher-level features like capabilities, which it facilitates, may require additional work.

7 Future development and critique (Morello)

Morello is not designed to resist speculation and side-channel attacks [3]. However, it is interesting to see how Morello's memory protection and compartmentalization interact with DRAM glitching [4] techniques such as Rowhammer [5], speculative execution attacks such as Spectre and Meltdown [6], as well as analogue attacks like power hammering [7]. Additionally, the formal verification approach taken by Morello can be extended to cover some of these attacks classes, for example by building on prior work for formally verified protections against timing-based attacks [8] like Spectre.

Significant work regarding the software toolchain is essential for future success. This includes fixing ABI-limitations around issues such as protections for thread-local storage [3] and techniques related to software compartmentalization. For example, compartments cannot access outside memory as constrained by the DDC, but data must sometimes need to be shared efficiently (without copying) [9]. Such problems have numerous solutions with different performance, security, and scalability implications—here two approaches could be replacing pointers by capabilities or overlapping shared regions (dropping fine-grained capabilities and relying on a single region of shared data).

Although the physical registers are expanded to 129 bits, the datapaths to memory are not doubled in width. This choice reduces Morello's development time and cost, however also means moving capabilities takes multiple clock cycles. This complicates timing analysis and limits performance comparisons with traditional systems when we perhaps want to measure the impact of moving a large number of capabilities in and out of memory. Doubling the datapath width is important for future iterations of Morello and commercial implementation.

Morello also currently re-purposes ECC bits for metadata storage. Future development could include implementation of efficient tagged memory as described in [2] as the project moves closer to commercialization.

In comparison to TM, Morello is much closer to a fully-realized product. Morello is a complete implementation of ARMv8.2, allowing for thorough evaluation on ARM-compatible software both with and without capabilities.

8 Future development and critique (Tagged Memory)

Direct Memory Access (DMA) is a key tool for high-bandwidth I/O devices to copy data directly to-and-from memory without the CPU's involvement. However, TM makes no mention of DMA support. Currently, since the DMA controller/policy is unaware of the tags, it simply bypasses the tag policy [10]. Even

more alarming is the fact that since the tags are stored as a table in DRAM in TM, the DMA engine can just rewrite them.

This must be addressed in future work by making the DMA controller tag-aware, although there has so far only been limited work [11, 12] regarding this. Additionally, we may need to encrypt the memory to prevent the DMA engine from modifying the tags as demonstrated by AEGIS [13], DataSafe [14], and XOM (Execute-Only Memory) [15].

Further work needs to be conducted reviewing TM's vulnerability to side-channel attacks. The additional caching performed as well as tag-dependent computation may introduce new side-channels [10]. For example, the additional delay due to a tag cache miss may be measurable and exploitable through flush-and-reload [16] and prime-and-probe [17] attacks. Perhaps inspiration could be taken from the HyperFlow [18] processor which used a tagged architecture and mitigates timing side-channels.

The use of formal verification methods (like Sail + Isabelle in Morello) could be crucial in proving not only the safety of tagged architectures like TM against such attacks, but also their correctness when they are placed in more complex out-of-order architectures. TM is tested physically only on an FPGA, but exploring it's interaction with out-of-order hardware is important if it is to be used in real systems.

TM seems more like a feasibility demonstration than a real-world solution. Apart from the fact that issues such as out-of-order architectures and DMA are not addressed, most of the data collected does not come from a real system running TM e.g. TM's DRAM traffic overhead and tag cache access data come from replaying DRAM traces captured from unmodified applications. Having TM run on real hardware like an ARM CPU would allow future work like Morello to directly compare TM with other tagged-memory solutions.

References

[1] Richard Grisenthwaite et al. "The Arm Morello Evaluation Platform—Validating CHERI-Based Security in a High-Performance System". In: *IEEE Micro* 43.3 (2023), pp. 50–57. DOI: 10.1109/MM.2023.3264676.

[2] Alexandre Joannou et al. "Efficient Tagged Memory". In: *2017 IEEE International Conference on Computer Design (ICCD)*. 2017, pp. 641–648. DOI: 10.1109/ICCD.2017.112.

[3] Robert NM Watson et al. *Arm Morello Programme: Architectural security goals and known limitations*. Tech. rep. University of Cambridge, Computer Laboratory, 2023.

[4] Chad Spensky et al. "Glitching Demystified: Analyzing Control-flow-based Glitching Attacks and Defenses". In: *2021 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 2021, pp. 400–412. DOI: 10.1109/DSN48987.2021.00051.

[5] Onur Mutlu, Ataberk Olgun, and A Giray Yağlıkcı. "Fundamentally understanding and solving rowhammer". In: *Proceedings of the 28th Asia and South Pacific Design Automation Conference*. 2023, pp. 461–468.

[6] Nael Abu-Ghazaleh, Dmitry Ponomarev, and Dmitry Evtyushkin. "How the spectre and meltdown hacks really worked". In: *IEEE Spectrum* 56.3 (2019), pp. 42–49.

[7] Kaspar Matas et al. "Power-hammering through Glitch Amplification – Attacks and Mitigation". In: *2020 IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*. 2020, pp. 65–69. DOI: 10.1109/FCCM48280.2020.00018.

[8] Swarn Priya. "Formally computer-verified protections against timing-based side-channel attacks". PhD thesis. Centre Inria d'Université Côte d'Azur STAMP, 2023.

[9] John Alistair Kressel, Hugo Lefeuvre, and Pierre Olivier. "Software Compartmentalization Trade-Offs with Hardware Capabilities". In: *Proceedings of the 12th Workshop on Programming Languages and Operating Systems*. PLOS '23. Koblenz, Germany: Association for Computing Machinery, 2023, pp. 49–57. ISBN: 9798400704048. DOI: 10.1145/3623759.3624550. URL: https://doi.org/10.1145/3623759.3624550.

[10] Samuel Jero et al. "Tag: Tagged architecture guide". In: *ACM Computing Surveys* 55.6 (2022), pp. 1–34.

[11] Joël Porquet and Simha Sethumadhavan. "WHISK: An uncore architecture for Dynamic Information Flow Tracking in heterogeneous embedded SoCs". In: *2013 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*. 2013, pp. 1–9. DOI: 10.1109/CODES-ISSS.2013.6658991.

[12] Christian Pilato et al. "Tainthls: High-level synthesis for dynamic information flow tracking". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38.5 (2018), pp. 798–808.

[13] G Edward Suh et al. "AEGIS: Architecture for tamper-evident and tamper-resistant processing". In: *ACM International Conference on Supercomputing 25th Anniversary Volume*. 2003, pp. 357–368.

[14] Yu-Yuan Chen, Pramod A Jamkhedkar, and Ruby B Lee. "A software-hardware architecture for self-protecting data". In: *Proceedings of the 2012 ACM conference on Computer and communications security*. 2012, pp. 14–27.

[15] David Lie et al. "Architectural support for copy and tamper resistant software". In: *Acm Sigplan Notices* 35.11 (2000), pp. 168–177.

[16] Yuval Yarom and Katrina Falkner. "{FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack". In: *23rd USENIX security symposium (USENIX security 14)*. 2014, pp. 719–732.

[17] Daimeng Wang et al. “Papp: Prefetcher-aware prime and probe side-channel attack”. In: *Proceedings of the 56th Annual Design Automation Conference 2019*. 2019, pp. 1–6.

[18] Andrew Ferraiuolo et al. “HyperFlow: A processor architecture for non-malleable, timing-safe information flow security”. In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 1583–1600.