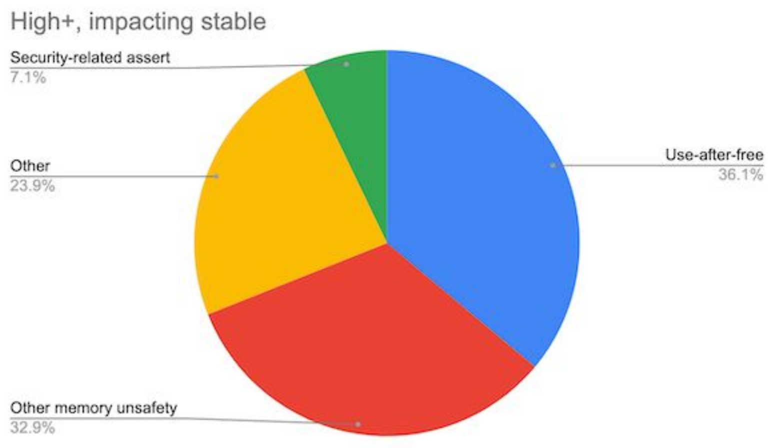
Microarchitectural Analysis of CHERI on the Morello Platform

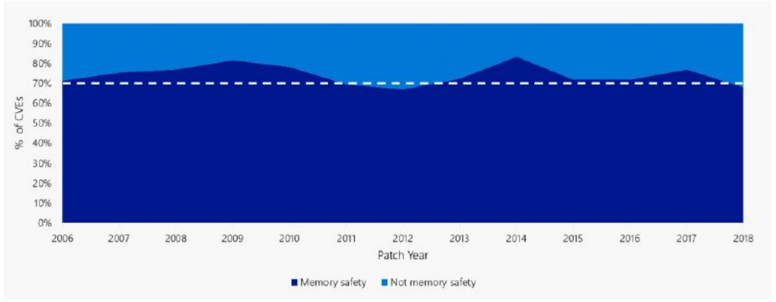
Yude Jiang
Advisor: Martin Fink, Dr. Masanori Misono
Chair of Computer Systems
https://dse.in.tum.de/

28.11.2024 – 28.03.2025

Memory safety bugs are a critical problem in Software



[1] Chromium high-severity security bugs



[2] MSRC Microsoft CVEs

Possible Solution:

Hardware-based memory protection mechanisms

[1] https://www.chromium.org/Home/chromium-security/memory-safety
[2]https://msrc.microsoft.com/blog/2019/07/a-proactive-approach-to-more-secure-code

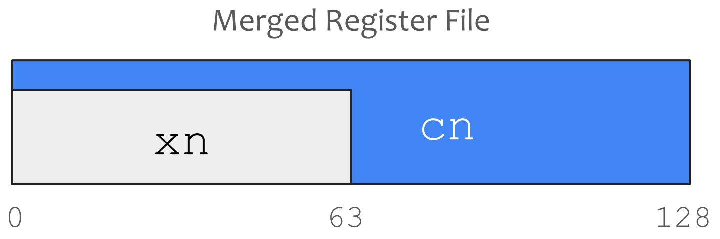
Background: CHERI

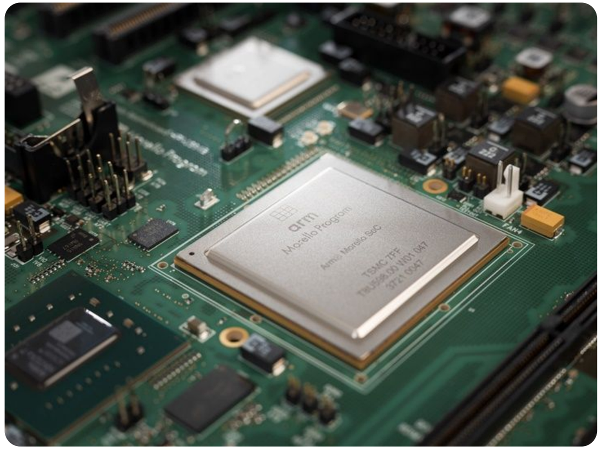
Capability Hardware Enhanced RISC Instructions

- **Capabilities:**
  Unforgeable, permission-tagged, and bounds-checked references that replace raw pointers
- **Memory Safety Enforcement:**
  Hardware checks on capability bounds and validity prevent spatial and temporal memory vulnerabilities
- **Execution Models:**
  - **Hybrid:** Legacy and capability-aware code coexist
  - **Purecap:** All pointers are capabilities - full fine-grained memory safety

Background: Morello

- Research Platform provided by Arm
- Extends the Armv8.2 ISA with CHERI
- Uses a merged register file for capability registers
- Hardware prototype based on Neoverse N1 shipped in 2022




[1] Morello SOC on board

[1]https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/creating-the-morello-technology-demonstrator

Problem Statement

Problem:

Limited performance data on CHERI/Morello hinders accurate evaluation of its security-performance tradeoff.

Solution:

Perform microbenchmark-based instruction analysis to expose latency and throughput characteristics that inform optimization.

Research Objectives

RO1: What is the effect of Capabilities on memory instructions?

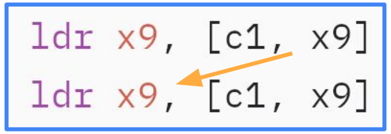
RO2: Does bounds checking introduce overhead?

RO3: What are the Latency and Throughput of Morello instructions?

Methodology

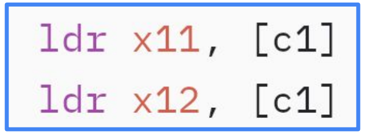
We measure the following metrics for a given Instruction:

- **Latency:** How long an instruction takes to execute
  - Measured by introducing dependencies between instructions
  - Indicates how long an instruction stalls dependent operations
- **Throughput:** How many instructions can be executed in one cycle
  - Measured by eliminating dependencies between instructions
  - Indicates how effectively the CPU can pipeline and overlap executions



```
ldr x9, [c1, x9]
ldr x9, [c1, x9]
```

Dependent instructions



```
ldr x11, [c1]
ldr x12, [c1]
```

Independent instructions

Memory Instructions

Nocap results (Baseline)

Instruction	Throughput (i/c)	Latency (c/i)
LDR	1.986	4.010
STR	1.972	-
LDP	0.996	4.008
STP	0.994	-
LDAR	1.990	-
STLR	1.985	-

Purecap results

Instruction	Throughput (i/c)	Latency (c/i)
LDR	1.990	4.009
STR	1.981	-
LDP	0.996	-
STP	0.996	-
LDAR	1.993	-
STLR	1.990	-
LDR_CAP	1.990	4.005
LDP_CAP	0.995	4.009
STR_CAP	0.997	-
STP_CAP	0.499	-

No performance difference between memory addressing modes

Memory Instructions

Nocap results (Baseline)

Instruction	Throughput (i/c)	Latency (c/i)
`LDR`	1.986	4.010
`STR`	1.972	-
`LDP`	0.996	4.008
`STP`	0.994	-
`LDAR`	1.990	-
`STLR`	1.985	-

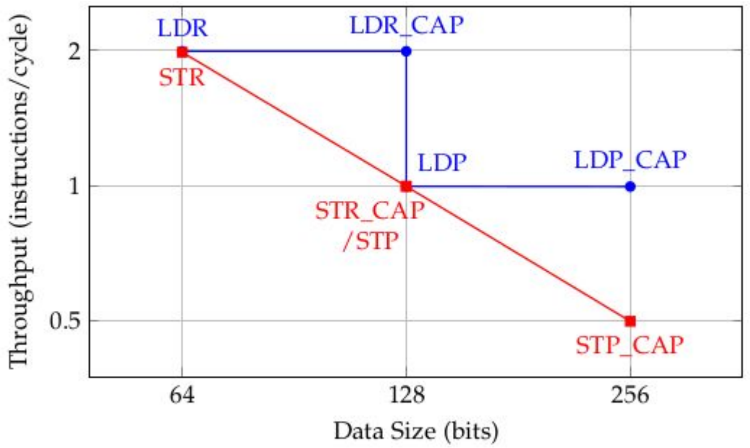
Purecap results

Instruction	Throughput (i/c)	Latency (c/i)
`LDR`	1.990	4.009
`STR`	1.981	-
`LDP`	0.996	-
`STP`	0.996	-
`LDAR`	1.993	-
`STLR`	1.990	-
`LDR_CAP`	1.990	4.005
`LDP_CAP`	0.995	4.009
`STR_CAP`	0.997	-
`STP_CAP`	0.499	-

No performance difference between memory addressing modes

No performance penalty for instructions enforcing access order

Memory Instructions



Instruction throughput vs data size

Instruction	Offset	In-place
STR	1.981	→ 1.984
STP	0.996	→ 0.333
STR_CAP	0.997	→ 0.333
STP_CAP	0.499	→ 0.166

Offset/in-place store throughput comparison

Load instructions scale based on if they are single or paired

Store instructions scale linearly with the stored datasize

Store throughput decreases 3x when performing in-place stores sized over 64 bits

Capability Instructions

Instruction	Throughput (i/c)	Latency (c/i)
CVT_TOCAP	0.998	2.006
CVT_TOPTR	0.997	2.007
CVTP_TOCAP	0.997	2.004
CVTP_TOPTR	0.998	2.005
CVTD_TOCAP	0.998	2.006
CVTD_TOPTR	0.997	1.004
CFHI	2.800	1.005
CTHI	2.818	1.007

Most conversion instruction perform the same

Capability Instructions

Instruction	Throughput (i/c)	Latency (c/i)
`CVT_TOCAP`	0.998	2.006
`CVT_TOPTR`	0.997	2.007
`CVTP_TOCAP`	0.997	2.004
`CVTP_TOPTR`	0.998	2.005
`CVTD_TOCAP`	0.998	2.006
`CVTD_TOPTR`	0.997	1.004
`CFHI`	2.800	1.005
`CTHI`	2.818	1.007

Most conversion instruction perform the same

`CVTD_TOPTR` has half the latency of the others

Capability Instructions

Instruction	Throughput (i/c)	Latency (c/i)
`CVT_TOCAP`	0.998	2.006
`CVT_TOPTR`	0.997	2.007
`CVTP_TOCAP`	0.997	2.004
`CVTP_TOPTR`	0.998	2.005
`CVTD_TOCAP`	0.998	2.006
`CVTD_TOPTR`	0.997	1.004
`CFHI`	2.800	1.005
`CTHI`	2.818	1.007

Most conversion instruction perform the same

`CVTD_TOPTR` has half the latency of the others

`CFHI` and `CTHI` are highly performant

Addressing the ROs

RO1: What is the effect of Capabilities on memory instructions?

- Loading capabilities is just as performant as loading integer addresses
- Storing capabilities is less performant due to store buffer limitations

RO2: Does bounds checking introduce overhead?

- No performance difference between addressing modes

RO3: What are the Latency and Throughput of Morello instructions?

- Most conversion instructions have high latency of around 2 cycles/instruction
- Throughput of 1 instruction/cycle indicates effective pipelining or multiple execution units

Conclusion

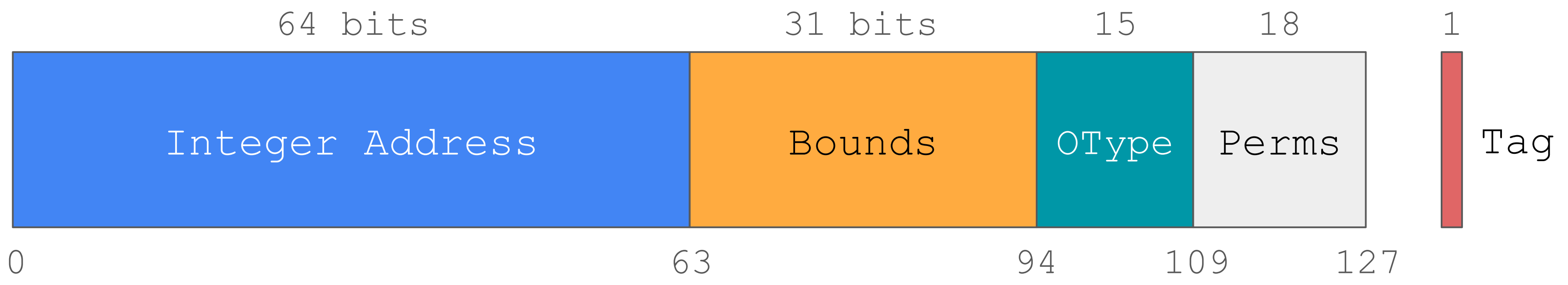
- **What did we do?**

  We performed microbenchmark based analysis of the Morello CPU to expose its Latency and Throughput characteristics.

- **What did we find?**
  - Capability loads are efficient
  - Stores and conversions can introduce overhead
  - Bounds checking is inexpensive
- **What could be done in the future?**
  - Methodology extension
  - Application level analysis

Backup

Background: CHERI Capabilities



Integer Address:	Conventional 64-bit integer address
Bounds:	Compressed addresses which describe accessible memory area
Object Type:	Indicates if and how a capability is sealed
Permissions:	Indicates how a capability can be used
Validity Tag:	Tracks if the capability is valid

Background: CHERI Software Modes

	Pure-capability Mode	Hybrid Mode
Memory access via capabilities	✓	✓
Memory access via integer addresses	✗	✓*
Fine-grained memory safety	✓	✗
Compatibility with legacy code	✗	✓

* Bound/Permission checks are performed against the Default Data Capability (DDC)

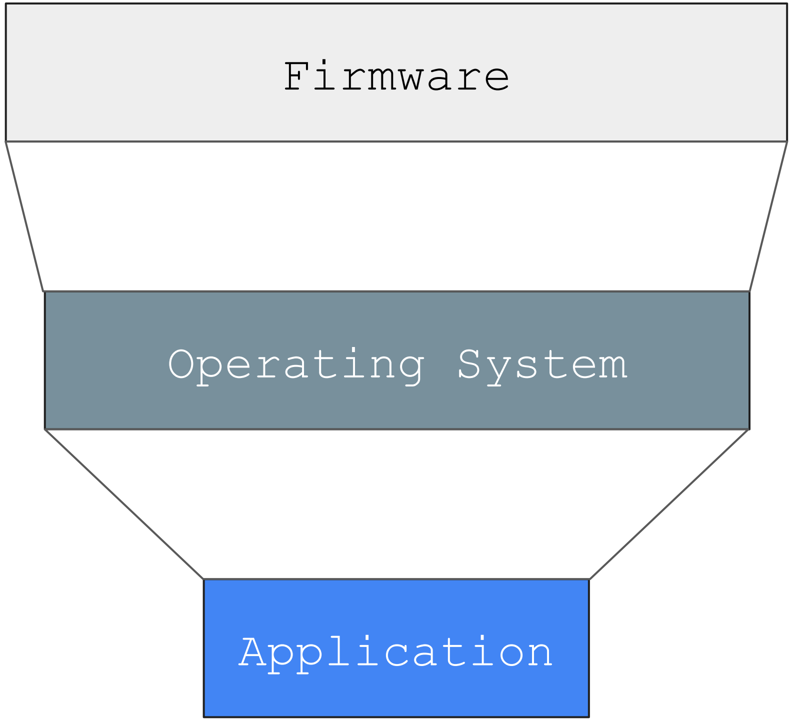
Background: CHERI Architectural Rules

Provenance Validity:

A valid capability can only be derived from another valid capability.

Capability monotonicity:

A capability cannot have greater permissions or bounds than the capability it was derived from.



Methodology

```
ldr_latency_loop:
    mov x9, #0
    ldrl_loop:
    cbz x0, ldrl_end
        sub x0, x0, #1
        ...
        ldr x9, [c1, x9]
        ldr x9, [c1, x9]
        ...
    b ldrl_loop

    ldrl_end:
        ret
```

```
ldr_throughput_loop:
    cbz x0, ldrt_end
        sub x0, x0, #1
        ...
        ldr x11, [c1]
        ldr x12, [c1]
        ...
    b ldr_throughput_loop

    ldrt_end:
        ret
```