CHERI-RISCV Extension Progress

Robert N. M. Watson[1], Alex Richardson[2], Jessica Clarke[1], Brooks Davis[3], Lawrence Esswood[2], Ben Laurie[2], Simon W. Moore[1], Peter Rugg[1], Alexandre Joannou[1], and Peter Sewell[1]

[1] University of Cambridge

[2] Google

[3] SRI

Abstract

The CHERI ISA extensions for security have been under development for the last 13 years and have seen implementations in the MIPS, RISCV and ARM ISAs. It has recently seen its first commercial RISCV implementation and several companies are showing interest in the technology. The CHERI-RISCV SIG aims to capture a minimum useful specification for CHERI-enhanced RV32 and RV64 ISAs. We aim to support the commercial implementation of a CHERI-enabled single-core RV32 or multicore RV64 configuration. For 64-bit, the ISA would support virtual memory suitable to use in lower-end computing environments that run operating systems such as Linux, FreeBSD, and seL4. However, our current focus is on vertically integrated hardware-software 32-bit stacks built by early adopters to provide a consistent minimal baseline that future specifications can extend.

Standardization scope

The starting point for this work is the SRI/Cambridge baseline CHERI-RISC-V 64-bit prototype specification[1],[2],[3], for which multiple FPGA-based cores, a formal Sail model, and complete software stack including CHERI LLVM toolchain and multiple operating-system adaptations have been developed. The aim in this work is to select and standardize a mature subset of that specification, while making necessary adaptations such as formal opcode allocations, maturing the capability format, and laying necessary groundwork to ensure that future extensions can build on the initial version. This work will learn from 32-bit work including at SRI/Cambridge and also the recently released Microsoft CHERIoT embedded microcontroller ISA specification[4].

CHERI aspects considered

Some aspects of the CHERI model are extremely mature (e.g., those relating to C/C++ memory safety), whereas others are the subject of ongoing research (e.g., software compartmentalization based on otypes). The initial focus of the standardization effort is around the more mature aspects, with the intention of integrating well with maturing elements of the CHERI design as they proceed. The aspects currently being considered are:

- 128-bit capabilities over a 64-bit baseline ISA, and 64-bit capabilities over a 32-bit baseline ISA, with a specific architectural format and a clear path to future expansion

- CHERI features to support C/C++ memory protection including subobject bounds and efficient stack alignment

- CHERI features to support sealed entry (sentry) protection of control-flow pointers

- CHERI features to support safe, capability-aware exception handling

- CHERI features to support efficient temporal memory safety

- Tagging behavior for memory

- Concurrency behavior for memory

- Opcode assignments for instructions implied by the above

It may be that multiple standard documents capture different aspects of the work – e.g., a baseline spatial memory-safety specification, an extension for domain transition, and an extension for temporal memory safety. Specific function software objectives include the implementation of:

- Safe use of an entirely unmodified RV32/64 software stack

[1] CHERI ISAv8: https://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-951.html
[2] CHERI specification working version GitHub repository: https://github.com/CTSRD-CHERI/cheri-specification
[3] Automatically built drafts from the cheri-specification repository: https://github.com/CTSRD-CHERI/cheri-specification/tags
[4] CHERIoT: Rethinking security for low-cost embedded systems: https://www.microsoft.com/en-us/research/uploads/prod/2023/02/cheriot-63e11a4f1e629.pdf

- CHERI RV32/64 firmware hosting RV32/64 or CHERI RV32/64 kernel

- CHERI RV32/64 kernel hosting RV32/64 or CHERI RV32/64 userlevel

"CHERI RV32/64" above does not imply "pure-capability" or "hybrid" language or code generation/linkage models – just the use of CHERI-aware ISA features. Software stacks might be compiled as conventional RV64 machine code using inline assembly for CHERI-related functions, employ the CHERI Hybrid C/C++ model to achieve similar goals but with stronger language and toolchain support, or use capabilities ubiquitously for pure-capability CHERI C/C++ and code generation. ABI specification activities will also aim to support all of these use cases.

We will aim to converge to the greatest extent possible with a potential concurrent effort to standardize CHERI microcontroller support – e.g., by considering similar encoding choices for capability permissions, instruction selections, and so on.

CHERI aspects to initially deferred

Some aspects of CHERI remain areas of active research, with architectural and microarchitectural prototypes but not yet substantial software ecosystem experience. As these mature alongside standardization efforts, they may be included in an eventual standard but are not committed to at this point in our work:

- Sealed capability pairs with object types

- Relocating / offsetting PCC (Program Counter Capability) and DDC (Default Data Capability) – initially PCC/DDC will only perform bounds & permission checks.

- Specialist capability conversion instructions (CFromPtr and CToPtr) used to optimize certain types of pointer manipulation in Hybrid C; these are likely to be deprecated in CHERI

- MMU support for page capability-dirty tracking

- MMU support for load-barrier temporal memory safety

- Morello-specific domain-transition mechanisms such as executive/restricted modes and indirect sealed jumps

- Local-global mechanisms

- One vs. multiple capability roots and formats; as the proposed standardization effort would introduce only memory capabilities, this design aspect can be deferred

- The final bit layout of a capability's non-address word

- The use of DRAM metadata / ECC bit-stealing or CHERI tag cache/controllers

- As-user CHERI memory instructions

Known CHERI issues to resolve before / during standardization

Before completing standardization, there a number of known (sometimes minor) issues to resolve:

- Current opcode assignments are from the local extensions space, and this will need to change

- The CSR model needs refinement or replacement

- Whether and how to bank DDC

- The Access_System_Registers (ASR) permission may not be sufficient, especially if there are notions of privileges that it controls in more than one ring. This can lead to confused-deputy problems between rings if ASR needs to be assigned to less privileged rings

- There hasn't been a proper quantitative measurement/analysis/optimization cycle on the instruction encodings, including compressed encodings

- There hasn't been a co-designed optimization cycle between the ISA and ABI

- Tune bounds precision

- How many "reserved bits" can reasonably be set aside at this point in standardization in order to enable future extensions – e.g., for otypes (CHERI object types, used in some forms of domain transition) – and what those extensions might require

Projected performance overhead

Early performance analysis by the University of Cambridge shows promise. Code generation improvements are ongoing and we will present an update on the current state of performance evaluation in the presentation.