CHERI Performance Optimization

Carl Shaw

Codasip

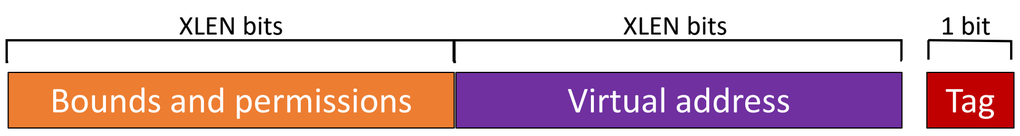
→ What is CHERI?

CHERI (Capability Hardware Enhanced RISC Instructions) is a memory access protection extension. It extends the RISC-V ISA to allow for the replacement of pointers with "capabilities", which are protected objects that encapsulate an integer memory address along with usage rights. In CHERI these rights include the allowed upper and lower memory bounds and access permissions. Every memory access is constrained by the capability used for each read or write.

The ability to distinguish pointers from integers and constrain their memory access can be used to enforce memory safety and prevent memory corruption security vulnerabilities, which account for up to 70% of all security vulnerabilities.

→ The impact of CHERI-RISC-V

Capabilities in CHERI-RISC-V double the architectural length of a pointer to 2 x XLEN bits plus an additional out-of-band tag bit used to enforce validity. Enlarged pointers (stored in double width registers) will have an impact on the memory system, particularly caches.



Additional instructions also need to be added to software to operate on capabilities (e.g. increment or set bounds). These are normally added automatically during compilation.

In this poster we present our ongoing software optimization work to ensure that CHERI has minimal performance impact.

→ Methodology

We use the Codasip X730 64-bit application core instantiated on an AMD/Xilinx Ultrascale+ FPGA to measure the impact of CHERI and test optimizations. The X730 implements the latest version of the proposed Zcherihybrid extension, allowing both CHERI-enhanced and normal RISC-V code to run and be compared.

→ Optimizing for CHERI

To optimize software for CHERI we try to make use of:

- The security provided by CHERI
- The extra information in capabilities
- The extended hardware registers and instructions

The CHERI security properties allow the removal of software security mechanisms from applications such as stack-protector or shadow stacks that can impact performance by >10% [1]

[1]Thurston Dang, Petros Maniatos and David Wagner. The Performance Cost of Shadow Stacks and Stack Canaries, ASIA CCS '15: Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security Pages 555 - 566

→ Optimizing C library string functions

String functions, e.g. memcpy(), are an obvious target for optimization. CHERI memory copies must preserve capabilities, so CHERI capability load and store instructions must be used to prevent clearing a validity tag. These instructions allow copying of XLEN*2 data bits at a time, increasing throughput.

Optimized string functions on standard RISC-V can deliberately load words at a time from memory to improve performance, running off the end of a string array. This is not allowed in CHERI due to the bounded accesses. Instead, bounds information in capabilities pointing to strings must be used to determine limits. Again, capability loads can help recover performance.

Some example results on X730:

Test	Unoptimized	Optimized
Dhrystone benchmark (heavy user of strcmp())	-14% compared to standard RISC-V code	-5% compared to standard RISC-V code
memcpy()		**5x faster** than the unoptimized CHERI memcpy()

Optimized string functions can be re-used in many places. For example, as well as adding them to the C library in Linux, we also use them within the kernel. We have used the memcpy() optimizations to optimize copy_to_user() and copy_from_user() to boost Linux performance.

→ Compiler optimizations

The compiler is another target for optimization. The CHERI LLVM compiler up to this point has focused on correctness and not on performance. We are working to re-enable optimization passes (taking capabilities into account where necessary) and also use the additional functionality and features of CHERI to improve code.

For example, if a register is not being used to store a capability, then the top XLEN bits can be used to temporarily store the contents of another non-capability register, reducing stack usage and cache impact.

→ Further information

We expect performance to continue to improve as CHERI software matures.

Our work is being continuously merged into the open-source repositories at https://cheri-alliance.org.

CHERI Alliance Founder