Compiling Erlang to CHERI

A PartIII project proposal

J. Fiala (*jf613*), Trinity Hall

Project Supervisor: Dr M. Naylor

Abstract

Capability systems, such as CHERI [1], allow for architectural differentiation between pointers and data and encode additional information alongside pointers. The security and overhead of capabilities employed in C/C++ has widely been studied; however, little work has been done to adapt compilers of higher level languages for such features. This work will aim to modify an existing compiler of a dynamically typed language (Erlang) to make use of the features offered by capabilities. Namely: (1) to mitigate overheads associated with capabilities by replacing dynamic type checks with fast bounds checks; (2) improving garbage collection by cheaply and precisely distinguishing pointers from data; and (3) to implement a secure foreign function interface, allowing the language to securely interface with the language runtime system and primitives, which are written in C.

1 Introduction, approach and outcomes

The idea of architecturally differentiating between pointers and data has been around since the early days of computer architecture [2]. Recent capability systems such as CHERI [1] have been gaining traction, with industry creating prototype chips using this design [3].

The CHERI system combines pointers with additional metadata to form capabilities. The metadata includes: *bounds* to restrict the range of memory accessible by a given capability; *permissions* to control how a capability can be used; *object type* allowing for data to be sealed; lastly a single bit *validity tag* is used to differentiate data and capabilities—this tag is architecturally protected and cannot be modified by software.

It has been demonstrated that the use of capabilities provides strong safety guarantees for programs written in C/C++. However, managed languages will not generally benefit from such guarantees as many of the same protection features are built in at the cost of runtime checks. In particular, dynamically typed languages such as Erlang will perform checks whenever an object is dereferenced—this is required on traditional processors to adhere to the language semantics.

This project proposes to modify an existing compiler, developed here in the Lab, for a subset of Erlang (called Elite [4]) to study the benefits of using CHERI capabilities with a dynamically typed language. Elite is only a small subset of sequential Erlang, thus the aforementioned compiler is correspondingly small and simple[1]—an ideal baseline to build on whilst modifying fundamental components.

Firstly, it should be possible to encode type information within capability bounds, thus delegating dynamic type checks to be automatically carried out in hardware—which raises an exception should such a check fail. Such optimisations should mitigate most of the overhead associated with capabilities.

[1] Yet it still manages to perform rather well; on par with the high performance Erlang (HIPE) compiler in most cases.

Secondly, it should be possible to use CHERI to improve garbage collection. The current compiler compiles to C code and generates a naïve garbage collector (GC) [5] using a simple mark and sweep algorithm incurring a large performance overhead. The GC is conservative as during execution it is not possible to differentiate pointers and data in C. The design space of conservative collectors is limited due to their inability to relocate objects. This project would aim to show how the GC can be greatly improved with the use of capabilities. For example, a compacting collector might be better suited to the frequent allocations made by a functional language.

Thirdly, security properties of the implementation will be explored. Any managed language will have a runtime system, which is generally a large codebase written in C. Though Elite itself should be robust and secure, it will often need to interface with the less secure runtime. At the cost of a large impact on performance, it is possible to implement a secure native code interface such as Java's JNI, however recent work [6] has shown that by using CHERI one can create such a foreign function interface (FFI) with a much smaller overhead whilst retaining all of the necessary security properties. As an extension the project would implement a secure and efficient FFI for the Elite compiler described above.

In summary, the project will produce an Elite-to-CHERI compiler (based on an existing Elite-to-C compiler), and study its security and performance characteristics.

References

[1] R. N. Watson, S. W. Moore, P. Sewell, and P. G. Neumann, "An Introduction to CHERI," tech. rep., University of Cambridge, Computer Laboratory, 2019.

[2] B. E. Clark and M. J. Corrigan, "Application system/400 performance characteristics," *IBM Systems Journal*, vol. 28, no. 3, pp. 407–423, 1989.

[3] Arm Limited, "Arm Architecture Reference Manual Supplement Morello for A-profile Architecture," 2020.

[4] M. Naylor, "Actora." `https://github.com/blarney-lang/actora`, 2019.

[5] "Boehm garbage collector." `https://en.wikipedia.org/wiki/Boehm_garbage_collector`. Accessed 20-November-2020.

[6] D. Chisnall, B. Davis, K. Gudka, D. Brazdil, A. Joannou, J. Woodruff, A. T. Markettos, J. E. Maste, R. Norton, S. Son, M. Roe, S. W. Moore, P. G. Neumann, B. Laurie, and R. N. Watson, "CHERI JNI: Sinking the Java Security Model into the C," *SIGARCH Comput. Archit. News*, vol. 45, p. 569–583, Apr. 2017.

2 Workplan

7th December to 21st December (Christmas vacation)

Read related work regarding use of capabilities in managed languages, in particular dynamically typed ones. Setup and test the existing Elite-to-C compiler and practice writing example code in Elite. Compile the C intermediate to capability machine code using a CHERI compiler.

Milestones: Working Elite-to-'capability machine code' compilation chain. Slight modifications to existing Elite compiler, testing out use with capabilities.

22nd December to 4th January (Christmas vacation)

Decide on final design, doing additional research if necessary. Plan out main modifications to the compiler needed to achieve main goal.

Milestones: Framework setup for work on main goal.

5th January to 18th January (Christmas vacation)

Complete success criteria. That is: use of capabilities to generate more efficient assembly from Elite source. This will likely require additional time to fully understand the existing compiler. In such a case the work can be extended to later weeks, at the cost of reducing the scope of proposed extensions.

Milestones: Code satisfies main criteria. Framework setup for extensions.

19th January to 1st February

Work on implementing a FFI for secure interfacing of the language and runtime code, with an emphasis on ensuring security—researching other related capability FFI work if necessary. If there is time, start collecting performance data for Evaluation.

Milestones: Compiled code runs on CHERI ISA simulator. Additional feature mostly complete.

2nd February to 1st March

If progress is on track, experiment with various GC designs, analysing which is most appropriate. Implement a faster GC (e.g. compacting collector), greatly reducing associated overheads. Scope of GC improvements can be reduced in order to complete outstanding work from previous weeks.

Additionally perform minor improvements and bugfixes, and create an evaluation suite of example Elite code for compilation.

Milestones: All work mostly complete, project in stable final state. Ready to perform Evaluation.

2nd March to 22nd March

Use busy weeks at the end of term to run evaluation code and perform analysis. Only small crucial changes to code at this point.

Milestones: Project and analysis is complete. Ready to start writing Dissertation.

23rd March to 26th April (Easter vacation)

Complete the Introduction and Preparation sections and start work on Implementation chapter.

Milestones: Slightly over half of Dissertation complete. Codebase ready for submission.

26th April to 13th May

Write Evaluation and Conclusions sections. Rerun tests and analysis as required due to any small last minute changes to codebase. Create and run additional evaluation tests if necessary.

Milestones: Dissertation completed and ready for submission if necessary.

14th May to 28th May

Buffer time to be used for any outstanding work. Otherwise can be spent reviewing entire project, and rereading the Dissertation.

Milestones: Submission of polished Dissertation.