The Arm Morello Evaluation Platform-Validating CHERI-based Security in a High-performance System

Richard Grisenthwaite, Arm Ltd, Cambridge, UK
Graeme Barnes, Arm Ltd, Cambridge, UK
Robert N. M. Watson, University of Cambridge, Cambridge, UK
Simon W. Moore, University of Cambridge, Cambridge, UK
Peter Sewell, University of Cambridge, Cambridge, UK
Jonathan Woodruff, University of Cambridge, Cambridge, UK

Abstract— Memory safety issues are a persistent source of security vulnerabilities, with conventional architectures and the C/C++ codebase chronically prone to exploitable errors. The CHERI research project has explored a novel architectural approach to ameliorate such issues using unforgeable hardware capabilities to implement pointers.

Morello is an Arm experimental platform for evaluation of CHERI in the Arm architecture context, to explore its potential for mass-market adoption. This paper describes the Morello Evaluation Platform; covering the motivation; the functionality of the Morello architectural hardware extensions, their potential for fine-grained memory safety and software compartmentalization; their formally proven security properties; their impact on the micro-architecture of the high-performance out-of-order multi-processor Arm Morello processor; and the software enablement program by Arm, University of Cambridge, and Linaro. Together, this allows a wide range of researchers in both industry and academia to explore and assess the Morello platform.

Introduction

Arm believes that security is the greatest challenge that computing needs to address to meet its full potential. Arm technology is used in products that are transforming every industry by enabling access to data and communications, and by extracting information and meaning from that data. This transformation continues in our society wherever the application of computing resources can make people's lives easier and more connected. Unfortunately, this increasing reliance on computing has created unprecedented opportunities for criminals, as can be seen in the ever-growing cost of cybercrime. In addition, the growing reliance of national infrastructure on technology means that computer security is part of National Security. Given this context, seems likely that the boundaries of the computing revolution will be determined by the security of our computing systems.

There is ample evidence that memory safety issues such as buffer overflows and use-after-free have been a persistent source of vulnerabilities for many years, and this continues in many ecosystems 1,2. While languages such as Rust offer the prospect of more inherent memory safety, the reality is that there is a huge body of C and C++ code being used, written, and adapted every day, and there are many undetected vulnerabilities waiting to be exploited. Arm has introduced the Memory Tagging Extensions in recent years to provide a mechanism to help identify memory safety issues, and these have demonstrated that ordinary code has a great number of latent memory safety errors.

CHERI Architecture

The University of Cambridge and SRI International, supported by DARPA, have been developing CHERI[4], a novel architectural approach to addressing memory safety issues, since 2010. Arm began collaborating with the CHERI project in 2014 to understand and develop this technology with a view to potentially incorporating it into the Arm Architecture in the future to be a foundation for improved security.

The fundamental CHERI concepts are independent of the underlying architecture and are designed to extend existing functionality. The original CHERI prototype extended MIPS, and later work extended RISC-V. This paper describes an experimental prototype extension to the 64-bit Arm architecture, as part of the Morello program.

Overview of CHERI capabilities

The basic premise behind CHERI-based architectures is to introduce a new architectural concept, called a "capability" alongside the existing architectural concepts of data and addresses. A capability is designed primarily to hold an address and to atomically combine that with the permissible address range that can be accessed, along with a set of permissions such as read, write, and execute. The architecture defines these as a single 128-bit concept that is distinct from the normal data and address concepts of the architecture. Capabilities are distinguished from other addresses and data by the use of an additional meta-data bit, usually referred to simply as a tag. The architecture requires that general-purpose memory can hold both capabilities and other data, and therefore each 128-bit quantity (aligned to 128 bits) held in memory has a meta-data tag associated with it. Mechanisms for storing the meta-data are described later in this paper. In keeping with a RISC philosophy, capabilities can be held and acted upon in 128-bit registers in the processor architecture, and the register file holds a meta-data tag bit for each register entry.

This fundamental distinction between capabilities and other data is an essential property of the CHERI approach, and is used to enforce the architectural principle that capabilities are unforgeable for most software. This lets software constrain the execution of code in fine-grained and scalable ways.

Unlike most other hardware capability systems, CHERI capabilities are design to be used alongside traditional memory protection mechanisms, including virtual memory.

CHERI-based architectures add various classes of instructions using capabilities, which can be broadly categorized as follows:

- A set of load and store instructions that use capabilities for the base address and can apply an offset to them, and which check that their memory accesses are performed within the allowed range of the capability and with its permissions. A failure to adhere to permissions causes a precise data abort in a similar way to traditional memory management data aborts.
- A set of specialized arithmetic and logical instructions that operate on capabilities for use by later loads or stores. These new instructions obey certain rules, for example, the address can be adjusted within limits defined by the bounds of the capability and cannot be taken arbitrarily out of range. The instructions allow the bounds or permissions of an capability to be decreased to allow the creation of sub-objects from an original object – but the bounds or permissions of a capability cannot be increased without a capability that gives you that right. This gives a monotonicity of privilege that is the essence of the compartmentalization that can be built using

CHERI. Attempting to manipulate a capability outside these limits results in it being transformed into ordinary data, with tag cleared.

- The program counter is enhanced to become a capability, so applying permission and range checks to direct branches. Indirect branches now take a capability as an argument, to update the program counter capability bounds and permissions.
- Traditional loads and stores that take integer addresses held in normal data registers are also supported to enable easier porting of legacy code to systems that use capabilities; the architecture introduces an additional register, the Default Data Capability, to apply permissions and access bounds to these, and also potentially an offset to the register-held address. This allows the creation of sandboxes for legacy software that has not been recompiled.

Various more specialist instructions working with capabilities are also added to support a range of different models – these are not discussed in this paper, but can be found in the CHERI ISAv8[3] and Morello specifications.

Software models for using CHERI capabilities

These new architectural primitives can be used in two ways, which are not mutually exclusive.

The first is to enforce programming-language-level fine-grained memory safety properties. The encapsulation of the bounds of an object with its address in a capability makes the enforcement of spatial memory safety straightforward – for example, recompiling and linking for CHERI will implement every address in the code with a capability, with appropriate bounds for stack and heap allocations. This can be achieved with minimal porting effort for software, and significant work has gone into supporting the de-facto uses of the C language and the relationship to the ISO standard.[5]

CHERI can also be used as the basis for efficient temporal memory safety, as the metadata that distinguishes capabilities from other data allows a quarantine and garbage collection-like revocation approach when freeing memory.[5]

The main cost of using the CHERI technology this way comes from the larger cache and memory footprint of 128-bit capabilities, which various benchmarks have shown to be in the range of 0-5%, though on some very pointer heavy workloads, it can be more than that.[6]

Microsoft Security Research Centre did a survey of Microsoft's 2019 critical vulnerabilities that required a software update, analyzing which would have been deterministically mitigated by using CHERI memory safety. They found that CHERI, when combined with allocation initialization and capability revocation, would have deterministically mitigated at least two-thirds of those issues.[7] This not only greatly exceeds the mitigation rate that has been seen from other technologies, but does so in a deterministic and secrets-free manner, making it more robust against attacks than many other mitigation technologies.

The second way to use a CHERI-based architecture is to use CHERI capabilities to construct much more fine-grained software compartmentalization than can be achieved with traditional MMU-based approaches (where software is compartmentalized by using different processes), as seen for example in content separation in a browser. With more fine-grained compartmentalization, software can be built to be more robust by limiting the damage that can be done by a single exploited vulnerability, and so increase the overall security of the system. Prototyping has shown that CHERI can reduce the overhead of switching between compartments by orders of magnitude compared with a traditional process switch.[8] CHERI's primitives are flexible, and potential models include accelerating current domain-switching costs (e.g., between processes) and also new models (e.g., sandboxing libraries).

Using fine-grained compartmentalization to increase security will take more work for software developers than recompiling and porting to use CHERI memory safety, as programmers are not used to having the ability to create fine-grained compartments, the traditional process-based approach being too expensive. While small-scale studies have shown that significant benefits can be achieved in research environments, it is an open question whether this scales to industrial-scale production workloads, in a way that provides a sufficiently compelling benefit to software to justify the commercial deployment of a CHERI-based architecture. The benefit from such a technology can be measured in terms of either a substantial increase in security as a reason to prefer using this product, or in terms of a performance improvement for the same level of compartmentalization as is achieved today, due to the far faster switching times from avoiding full process switches.

The Morello Prototype System

To enable software prototyping and evaluation for CHERI technology at large scale, Arm has developed the Morello prototype system – this was done in close collaboration with its partners: the Universities of Cambridge and Edinburgh, and Linaro, and with funding support from UKRI under the Digital Security by Design program.

Morello includes a prototype version of the Armv8.2 architecture, capable of running standard Arm software, with a full set of capability extensions. Because this architecture was designed to enable a wide range of prototyping, various experimental features were added to the architecture whose value needs to be assessed. For example, multiple viable capability-based domain-transition mechanisms are included in the prototype, for comparison, but fewer such mechanisms would be expected in a production architecture. The presence of these features allows comparison between code using those features, and code not using them. The full Morello architecture is published by Arm.[10]

This prototype architecture has been built, in a new experimental high performance out-of-order CPU based on the Neoverse-N1 processors that are shipping today. The original Neoverse-N1 processors are 4-way decode, 8-way issue, 11 stage pipeline systems with a 128-entry re-order buffer and three levels of cache, and these parameters remained the same in the experimental Morello CPUs with capabilities added. The Morello CPU runs at 2.5GHz in the SoC which was fabricated on a TSMC N7 process, and the overall SoC has an area of 109.9mm^2.

The system-on-chip designed as part of the Morello program has four of these CPUs together with a Mali G76 GPU and PCIe, CCIX and PCIe interfaces, in an SoC that is compliant with the Base System Architecture suitable for either Server or Client workloads.

The demonstrator board of this SoC includes 16GByte of RAM and a host of additional features for prototyping, as shown in Figure 1.

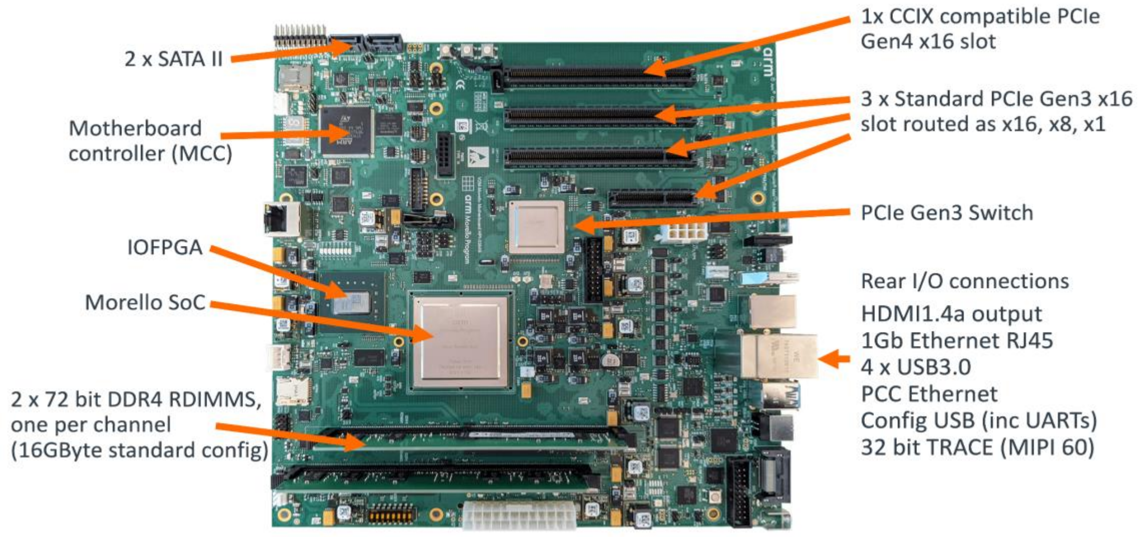
2 x SATA II
Motherboard controller (MCC)
IOFPGA
Morello SoC
2 x 72 bit DDR4 RDIMMS, one per channel (16GByte standard config)
1x CCIX compatible PCIe Gen4 x16 slot
3 x Standard PCIe Gen3 x16 slot routed as x16, x8, x1
PCIe Gen3 Switch
Rear I/O connections
HDMI1.4a output
1Gb Ethernet RJ45
4 x USB3.0
PCC Ethernet
Config USB (inc UARTs)
32 bit TRACE (MIPI 60)

Figure 1: The features of the Morello demonstrator board

All the other standard documentation and support deliverables have also been produced to allow people to experiment with these systems. We are making these available to a variety of partners, including Microsoft, Google, and a number of universities, research labs, and smaller companies, under the guidance of UKRI and the UK Digital catapult.

The UK Digital Security by Design program has also provided funding for universities and smaller companies to experiment with the Morello systems, while larger companies have committed to do their own Morello research. Only a small number of Morello boards have been built, as this is a prototyping system, not a commercial product. Around 450 boards have been delivered to a community of around 50 institutions at the time of writing.

Implications of Capabilities on the CPU micro-architecture

The CPU microarchitecture has to be extended to support a register file with a set of capabilities as an alternative to the normal 64-bit general-purpose integer registers. In the Morello architecture, the capability registers overlap with the general-purpose registers, such that the same architectural state could be viewed as a 128-bit register designed to hold capabilities or as a 64-bit general-purpose data value, along with the meta-data tag bit associated with each 128-bit register to indicate whether it holds a capability or other data. The choice to overlap the register files was made because there was not sufficient value in having separate architectural registers to hold capabilities to justify the additional hardware complexity, and having a single register file simplifies the implications to software ABIs. The Morello prototype architecture made all 31 architectural registers (and the stack pointers) able to hold capabilities to allow most flexibility in prototyping and would avoid constraining compiler register allocation. A commercial implementation might choose to have fewer registers able to hold capabilities, and the implications of this can be measured using the Morello prototype architecture.

All of the physical registers in the design that are used by the renaming system to map to the architectural registers were expanded to 129 bits wide (including the meta-data tag bit), and much of the micro-architecture was enhanced to have the 129-bit capability as a first-class concept. However, in the Morello prototype, the datapaths to memory were not doubled in width, so instead multiple cycles were taken to move capabilities – this ensured that performance and area comparisons would

be realistic in terms of the amount of physical bandwidth available. This choice reduced the impact on much of the memory system, reducing the development time and cost of the prototype CPU. This decision would need to be revisited for a commercial implementation.
The CPU caches were enhanced so that alongside every 128 bits of data there is also a meta-data tag, and the system buses were expanded to carry the tag alongside the data.

One of the experimental capability features included in the Morello architecture is that the Default Data Capability (DDC) can apply an offset to the general-purpose register-based address calculations. The usage model behind this is to allow the compartmentalization of several instances of the same legacy code in a set of different sandboxes, where only one instance is used at a time by a given CPU. The base address held in the DDC at any moment in time is therefore the base address of the sandbox, and all other addresses within the sandbox are relative to this base address. Moving between sandboxes involves some executive software changing the DDC.

While this is an architecturally interesting concept, it has a significant impact on an implementation's address generation stage, as it changes the normal two-way addition of base address and offset for each memory calculation into a three-way addition of DDC, register base address, and offset. In many microarchitectures, the address generation path, and its subsequent lookup into the first level of TLB, is a very sensitive timing path, and so the expansion of this to include another addition, albeit done in a carry-save format, is a controversial issue. The Morello prototype included this functionality to allow evaluation of this feature and usage model, and measurement of the execution cycle-count, compared with generating the address using multiple instructions.

The main change to the load store unit is that loads and stores need to check that the memory accesses are in range and have the right permissions. Most of those checks are address-based and run in parallel to the normal memory-management function of the TLB and have similar timing, so the determination that the access is permissible is available slightly earlier in the micro-architecture than the result of the normal TLB-based permission checks. It is not acceptable to speculate using data from loads that fail the capability checks to avoid speculation issues similar to those seen in Meltdown.

However, the Morello architecture defines a particular capability-based permission check that generates a memory fault if a 128-bit value that is tagged as a capability is stored to a memory location that does not permit capabilities to be stored to it. This permission fault is dependent on the value of the meta-data tag of the data being stored, a micro-architecturally novel concept that added complexity and performance cost. The mechanisms for using capabilities for temporal memory safety are reliant on the ability to generate this fault, so it was included as part of the Morello prototype.

The Morello architecture also added a requirement for atomic load-store instructions, that are typically used on a pair of addresses, to support pairs of capabilities. The architecture requires that the implementation can perform an atomic compare-and-swap of a 256-bit quantity and the two additional meta-data tags. This was a new requirement in terms of the atomic access support of the design.

A key innovation in the CHERI architecture is that the base, bounds, and permission information of a capability is compressed into only 64 bits of additional state, which means that when the address checks are performed, the base and bounds need to be uncompressed to allow them to be compared with the result of the address calculation. Much work has gone into developing a scheme that can be quickly decompressed when necessary, so as not to impact any critical paths on the device[10]. In the memory access path, the decompression of the base and bounds is done as a pair of expansions in parallel with the address generation arithmetic, and that means that the bounds check is timed in a very similar way to the normal TLB hit case.

The details of the expansion are in Figure 2, based on the format of the capability described in the Morello architecture documentation[9]. This requires two shifters, an adder, and three comparators for each of the base and limit. Fortunately, it was possible to implement it in the full cycle of the Morello CPU without impacting the overall critical timing path of the design.

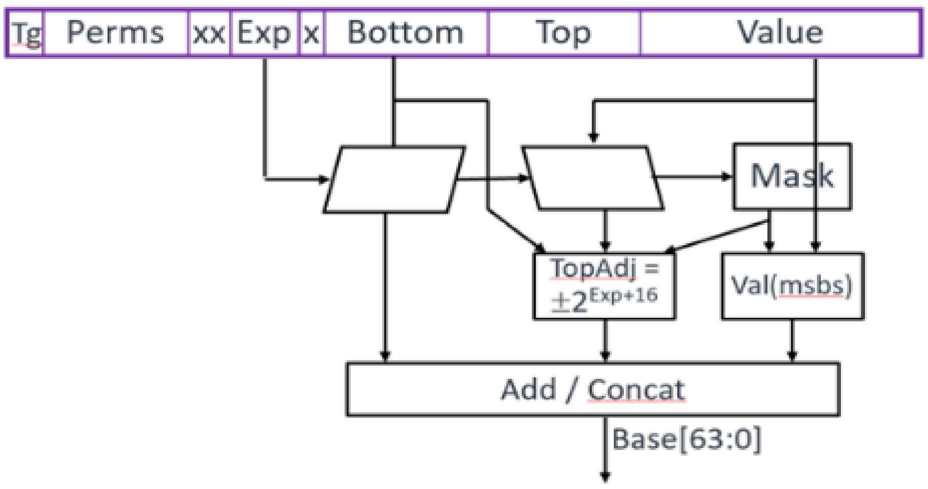

Figure 2: capability expansion logic

A consequence of the compression scheme is that not all combinations of an address, base, and bounds are representable[11], which means that if the address part of a capability is taken too far out of range as part of arithmetic operations, the bounds cannot be represented. Fortunately, doing that on a C pointer is illegal. When any arithmetic is performed on a capability, a hardware representability check clears the tag if the address has been pushed too far out of range, and the bounds become meaningless.

The compression scheme design ensures that this check can be done without decompressing capabilities to avoid hitting the critical path for arithmetic. There are a small number of integer operations on capabilities that need to have the capability decompressed, but these occur sufficiently infrequently that they can take multiple cycles.

The fact that the PC becomes a capability in the Morello architecture introduces questions about how to handle the indirect prediction of PC capabilities. The highest performance solution would be to expand the indirect branch prediction targets to hold 129-bit capabilities, allowing changes to the base/bounds of the PC to be predicted, although this has a non-trivial area cost. Two other options exist: one is to predict that the base/bounds do not change (relying on the fact that most indirect branches do not change the PC base/bounds), and the other is to simply stall until the new bounds are known. The Morello implementation stalls in some circumstances until the new bounds are known, and this has led to undesirable performance issues that have had to be worked around in the compiler. A commercial implementation of a capability architecture will need to use one of the other approaches, though the issue can be worked around in a prototyping environment such as Morello. The Morello prototype did not address Spectre-type speculation attacks, where predictors would need to be tagged with context information.

The Morello prototype implements two ways of holding the meta-data tags in memory, to allow performance comparison between them as part of the prototyping. The first approach is to carve out a <1% area of the DRAM to hold the tags and to split all cache misses into two memory accesses, one for the data and the other for the tag. In the Morello system, this is augmented with a caching structure to minimize the number of meta-data accesses to memory. The second is to have the meta-data explicitly held alongside the memory by repurposing the bits normally used for ECC. For an experimental platform like Morello, the ECC information is not used as the systems will not be deployed in an environment with high RAS requirements. Repurposing the ECC data to hold tags makes it possible to directly measure the performance overhead that comes from the other mechanism for holding tags.

Morello software infrastructure and experiments

Hardware is only part of the Morello system. The CHERI LLVM compiler, adapted to Morello by Arm, implements the CHERI C and C++ variants of these popular memory-unsafe programming languages. They support the protection of language-level resources such as stack and heap allocations, and also of sub-language structures such as dynamic program linkage and global variable access.

The CheriBSD operating system, a heavily CHERI-adapted version of the open-source FreeBSD UNIX, provides a rich demonstration of how extensive CHERI integration can enable kernel and user-level memory safety, and scalable software compartmentalisation, while still supporting legacy 64-bit Arm applications. Arm has embarked on a port of the gcc compiler and Linux operating system to Morello, and is integrating research features into these software stacks as they mature.

The X11 KDE-based desktop environment was ported in 3 months by a single engineer having to make changes in less than 0.03% of the 6 million lines of code - these changes give an assessed vulnerability mitigation rate of 73.8%[11]. The system is capable of running a complete suite of legacy FreeBSD applications, and has been used to give presentations about the Morello system.

Formal proofs of the Morello security properties

CHERI aims to provide strong deterministic guarantees of certain architectural security properties, to provide a solid foundation for software to use in constructing more secure systems. It is thus important to establish high assurance that the detailed Morello architecture design provides the intended security (as otherwise any conforming hardware implementation would build in vulnerabilities), but this cannot be done by conventional testing.

Accordingly, a major strand of work of the Morello program was to develop and use formal methods. Teams at the Universities of Cambridge and Edinburgh took the description of the Morello architecture (62K lines of specification), translated it into a mathematical definition (via the Sail language), precisely stated the fundamental intended security property (reachable capability monotonicity, i.e. that the available capabilities cannot be increased during normal execution of arbitrary code), and proved that this holds of the full sequential architecture specification, using the Isabelle Proof Assistant. The formal semantics was also used for high-coverage test generation.

This gives us machine-checked mathematical proofs of security properties of a full-scale industry architecture, at design-time. To the best of our knowledge, this is the first demonstration that that is feasible, and it significantly increases confidence in Morello. Moreover, in doing the proof, three security issues were found before tapeout.[12]

Conclusion

The creation of the Morello platform is an important step of the Digital Security by Design program to investigate the applicability of CHERI capabilities in an industrial context. The platform will allow software developers to experiment with the concepts of CHERI for software that is commonly used for the Arm architecture – while also building extensive practical experience in CHERI microarchitecture applied to a high-performance industrial-scale hardware design. This provides a valuable opportunity to understand this new technology and will help refine the details of an architecture that we would make commercially available in future products. It also provides a real vehicle to allow red teams to evaluate the technology and to allow a wider range of software users to try using the technology. For example, the existence of Morello has created excitement in the Rust community, and has inspired academic projects looking at its use in Java and JavaScript runtimes.

The essential CHERI technologies are not patented, and it is recognized by the authors that adoption of the CHERI concepts by many software ecosystems will require adoption of the CHERI technology in multiple architectures. One of the aims of the Morello program is to act as a showcase for the technology, so that where compelling benefits can be demonstrated in the Arm ecosystem, this can encourage the adoption of the technology by other architectures. This is an explicit aim of the program.

It is expected that the software evaluations and prototyping using the Morello platform, which have already started, will continue over the next years, as it will take time to develop the compelling arguments for the use of this technology. If we have such compelling arguments then Arm anticipates that a form of this technology will be made available in future versions of the Arm architecture. The investigation phase made possible by Morello is an essential part of the process that is necessary to properly explore this very promising new approach to the vital area of security.

Acknowledgements

We acknowledge the contribution of the following people from the University of Cambridge, SRI International, and other institutions: Peter G. Neumann, Hesham Almatary, Jonathan Anderson, Alasdair Armstrong, Peter Blandford-Baker, Rosie Baish, John Baldwin, Hadrien Barrel, Thomas Bauereiss, Ruslan Bukin, Brian Campbell, David Chisnall, Jessica Clarke, Nirav Dave, Brooks Davis, Lawrence Esswood, Nathaniel W. Filardo, Franz Fuchs, Dapeng Gao, Khilan Gudka, Brett Gutstein, Alexandre Joannou, Mark Johnston, Robert Kovacsics, Ben Laurie, A. Theo Markettos, J. Edward Maste, Alfredo Mazzinghi, Alan Mujumdar, Prashanth Mundkur, Steven J. Murdoch, Edward Napierala, George Neville-Neil, Kyndylan Nienhuis, Robert Norton-Wright, Philip Paeps, Lucian Paul-Trifu, Allison Randal, Ivan Ribeiro, Alex Richardson, Michael Roe, Colin Rothwell, Peter Rugg, Hassen Saidi, Thomas Sewell, Stacey Son, Ian Stark, Domagoj Stolfa, Andrew Turner, Munraj Vadera, Konrad Witaszczyk, Jonathan Woodruff, Hongyan Xia, and Bjoern A. Zeeb

Within Arm we acknowledge the particular contributions of Lee Smith in the evaluation of CHERI technology, of Lee Eisen, Mark Inskip, and Karthik Muthusamy in leading the Morello design, and those of the over 100 people who have worked on the Morello program.

This work was supported in part by the Innovate UK project Digital Security by Design (DSbD) Technology Platform Prototype, 105694.

The initial development of CHERI was supported by the Defense Advanced Research Projects Agency (DARPA) and the Air Force Research Laboratory (AFRL), under contract FA8750-10-C-0237 (“CTSRD”), with additional support from FA8750-11-C-0249 (“MRC2”), HR0011-18-C-0016 (“ECATS”), FA8650-18-

C-7809 (“CIFV”), and HR001122C0110 (“ETC”) as part of the DARPA CRASH, MRC, and SSITH research programs. The views, opinions, and/or findings contained in this paper are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

We also acknowledge the EPSRC REMS Programme Grant (EP/K008528/1), the EPSRC CHaOS grant (EP/V000292/1), the ERC ELVER Advanced Grant (789108), the Isaac Newton Trust, the UK Higher Education Innovation Fund (HEIF), Thales E-Security, Microsoft Research Cambridge, Arm Limited, Google, Google DeepMind, HP Enterprise, and the Gates Cambridge Trust.

References

1. Matt Miller. Trends, challenges and strategic shifts in the software vulnerability mitigation landscape. BlueHat IL, Tel Aviv, Israel, February 2019
2. Chromium,https://www.chromium.org/Home/chromium-security/memory-safety/
3. Robert N. M. Watson, Peter G. Neumann, Jonathan Woodruff, Michael Roe, Hesham Almatary, Jonathan Anderson, John Baldwin, Graeme Barnes, David Chisnall, Jessica Clarke, Brooks Davis, Lee Eisen, Nathaniel Wesley Filardo, Richard Grisenthwaite, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, Simon W. Moore, Steven J. Murdoch, Kyndylan Nienhuis, Robert Norton, Alexander Richardson, Peter Rugg, Peter Sewell, Stacey Son, Hongyan Xia. Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 8), Technical Report UCAM-CL-TR-951, Computer Laboratory, October 2020
4. David Chisnall, Colin Rothwell, Robert N.M. Watson, Jonathan Woodruff, Munraj Vadera, Simon W. Moore, Michael Roe, Brooks Davis, and Peter G. Neumann. Beyond the PDP-11: Architectural support for a memory-safe C abstract machine, Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2015), Istanbul, Turkey, March 2015
5. Nathaniel Wesley Filardo, Brett F. Gutstein, Jonathan Woodruff, Sam Ainsworth, Lucian Paul-Trifu, Brooks Davis, Hongyan Xia, Edward Tomasz Napierala, Alexander Richardson, John Baldwin, David Chisnall, Jessica Clarke, Khilan Gudka, Alexandre Joannou, A. Theodore Markettos, Alfredo Mazzinghi, Robert M. Norton, Michael Roe, Peter Sewell, Stacey Son, Timothy M. Jones, Simon W. Moore, Peter G. Neumann, and Robert N. M. Watson. Cornucopia: Temporal Safety for CHERI Heaps. In Proceedings of the 41st IEEE Symposium on Security and Privacy (Oakland 2020). San Jose, CA, USA, May 18-20, 2020
6. Brooks Davis, Robert N. M. Watson, Alexander Richardson, Peter G. Neumann, Simon W. Moore, John Baldwin, David Chisnall, Jessica Clarke, Nathaniel Wesley Filardo, Khilan Gudka, Alexandre Joannou, Ben Laurie, A. Theodore Markettos, J. Edward Maste, Alfredo Mazzinghi, Edward Tomasz Napierala, Robert M. Norton, Michael Roe, Peter Sewell, Stacey Son, and Jonathan Woodruff. CheriABI: Enforcing Valid Pointer Provenance and Minimizing Pointer Privilege in the POSIX C Run-time Environment. In Proceedings of 2019 Architectural Support for Programming Languages and Operating Systems (ASPLOS’19). Providence, RI, USA, April 13-17, 2019
7. Nicolas Joly, Saif ElSherei, and Saar Amar. Security analysis of CHERI ISA, Microsoft Security Response Center (MSRC), October 2020.
8. Robert N. M. Watson, Jonathan Woodruff, Peter G. Neumann, Simon W. Moore, Jonathan Anderson, David Chisnall, Nirav Dave, Brooks Davis, Khilan Gudka, Ben Laurie, Steven J. Murdoch, Robert Norton, Michael Roe, Stacey Son, and Munraj Vadera. CHERI: A Hybrid Capability-System Architecture for Scalable Software Compartmentalization, Proceedings of the 36th IEEE Symposium on Security and Privacy ("Oakland"), San Jose, California, USA, May 2015
9. Arm. Architecture Reference Manual Supplement: Morello for A-profile Architecture. Arm Limited. 2020.
10. Jonathan Woodruff, Alexandre Joannou, Hongyan Xia, Anthony Fox, Robert Norton, Thomas Bauereiss, David Chisnall, Brooks Davis, Khilan Gudka, Nathaniel W. Filardo, A. Theodore

Markettos, Michael Roe, Peter G. Neumann, Robert N. M. Watson, and Simon W. Moore. CHERI Concentrate: Practical Compressed Capabilities. In IEEE Transactions on Computers, 10.1109/TC.2019.2914037, IEEE, 2019

11. Robert N. M. Watson, Ben Laurie, and Alex Richardson. Assessing the Viability of an Open-Source CHERI Desktop Software Ecosystem, Technical Report, Capabilities Limited, 17 September 2021.
12. Thomas Bauereiss, Brian Campbell, Thomas Sewell, Alasdair Armstrong, Lawrence Esswood, Ian Stark, Graeme Barnes, Robert N. M. Watson, and Peter Sewell. Verified Security for the Morello Capability-enhanced Prototype Arm Architecture, 31st European Symposium on Programming (ESOP 2022), May 2022.

Richard Grisenthwaite is Chief Architect and Fellow at Arm Ltd and is responsible for the development of all aspects of the Arm architecture. He has a BA in Electrical and Information Sciences from the University of Cambridge. He can be contacted at Richard.Grisenthwaite@arm.com

Graeme Barnes is a Lead ISA Architect and Fellow at Arm Ltd and is responsible for leading the development of various Arm instruction set architecture features. He has a BA in Computer Science from the University of Cambridge. He can be contacted at Graeme.Barnes@arm.com.

Robert N. M. Watson is Professor of Systems, Security, and Architecture at the University of Cambridge. He has led the development of the CHERI protection model, ISA extensions, software models, compiler and language extensions, operating systems, and applications. He received his PhD from the University of Cambridge. He can be contacted at Robert.Watson@cl.cam.ac.uk.

Simon W. Moore is Professor of Computer Engineering at the University of Cambridge. His research interests are in computer architecture and microarchitecture with a particular focus on security. He received his PhD from the University of Cambridge. He can be contacted at Simon.Moore@cl.cam.ac.uk.

Peter Sewell is Professor of Computer Science at the University of Cambridge. His research interests include the development and use of mathematically rigorous semantics in mainstream engineering, especially the semantics of architectures and programming languages. He received his PhD from the University of Edinburgh. He can be contacted at Peter.Sewell@cl.cam.ac.uk.

Jonathan Woodruff is a Senior Research Associate at the University of Cambridge specializing in capability processor design including both cores and memory hierarchy. He received his PhD from the University of Cambridge. Email is Jonathan.Woodruff@cl.cam.ac.uk.