Mapping Unikernels with TAG based architectures

Akilan Selvacoumar

Mathematics and Computer Sciences
Heriot Watt University

Year 1 progression report of:
Doctor of Philosophy

November 2022

I would like to dedicate this thesis to my loving parents ...

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements. This dissertation contains fewer than 65,000 words including appendices, bibliography, footnotes, tables and equations and has fewer than 150 figures.

Akilan Selvacoumar
November 2022

Acknowledgements

And I would like to acknowledge ...

Abstract

This is where you write your abstract ...

Table of contents

List of figures

List of tables

Chapter 1

Introduction

Chapter 2

Motivation

Chapter 3

Research Questions

The following section talks about research questions:

- Which areas can Unikernels provide performance gains for TAG based architectures in comparison to the same implementation built using a monolithic OS?
- Does using Enclaves inside Unikernels provide a isolation mechanism within Unikernels, maintain lightweightness characterestics, and what would be the performance differnence between using Intel SGX and a open source implementation such as Timber V?
- Due to lesser depencies in Unikernels does that mean lesser TAG policies are required for the appication ?
- Can Unikernel provide sufficient performance in such a way that a dedicated processor is not required for processing TAGS ?
- Does Unikernels with TAGS provide a secure and elastic environment ?

]

Chapter 4

Literature Review

The literature review is split into 3 sections. The first section talks about the papers surveyed for Unikernels and the 2nd section talks about papers surveyed for TAG based architectures and the third sections talks about the possible incentives of combining them both which helps answer the research questions stated (TODO: Add reference to research question section).

4.1 Unikernels Survey

The following section is the Uni-kernel Survey which starts with the Introduction of Unikernels, Types of Uni-kernels, Various Uni-kernels implementations and analysis of the various Uni-kernel implementations.

4.1.1 Introduction to Unikernels

Unikernel is a relatively new concept that was first introduced around 2013 by Anil Madhavapeddy in a paper titled "Unikernels: Library Operating Systems for the Cloud" (todo reference). Unikernels is defined as "Unikernels are specialized, single-address-space machine images constructed by using library operating systems." (todo reference). Specialized indicates that an Unikernel holds a single application. Single address indicates that Unikernels does not have separation between the user and kernel address space.

Library Operating Systems

Library operating system is an method of constructing an operating system where the kernel modules required by an application is executed in the same address space as the application. The original goal of Library operating systems was to improve performance by enabling

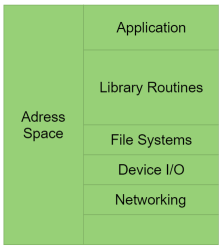

Fig. 4.1 Unikernel application stack

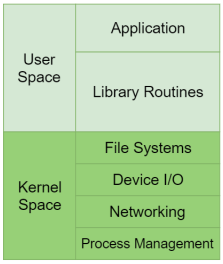

Fig. 4.2 Normal application stack

applications to manage resources according to their own needs, thereby allowing a high level of customizability. One of the major drawbacks for Library OS was support for various device drivers written for specific hardware.

Nowadays, however, virtualization already provides an abstraction of the underlying hardware by exposing virtualized hardware drivers. This allows library OS implementations to support the generic virtual driver as opposed to attempting to support various hardware drivers.

4.1.2 Types of Unikernels

Clean slate (Specialized and purpose-built unikernels)

Designed to utilize all the modern features of software and hardware, without worrying about backward compatibility. They are not POSIX-compliant.

- Halvm (TODO survey)
- MirageOS (TODO survey)

Legacy (Generalized "fat" unikernels)

Designed to run unmodified applications in an Unikernel, which make them bulky in comparison to the clean slate approach. Designed to be POSIX compliant. The following below are the ones surveyed in the following paper:

- Unikraft
- OSv
- HermitCore
- RKOS
- Azelea
- IncludeOS
- ClickOS
- NanoOS

4.1.3 Implementations

Unikraft

Unikraft is a uni-kernel implementation that claims to be a micro library OS. The major features of Unikraft is:

- Single address space: Intended to target single applications.
- Fully modular system: All drivers and platform libraries can be easily removed.
- Single protection level: No kernel and user space seperation to avoid costly context switching.
- Static linking: Compiler features such as dead code elimination and link time optmization supported.
- POSIX support: Support for legacy applications while still allowing for specialization.
- Platform abstraction: The ability to run on different Hypervisors/VMs.

To reach for the principal of modularity. Unikraft consists of 2 major componenents:

- Micro libraries: Micro-libraries are software components which implement one of the core Unikraft APIs.
- Build system: he build system then compiles all of the micro-libraries, links them, and produces one binary per selected platform.

In terms of performance the following was evaluated:

- Resource Efficiency (Smaller is Better): Overall, the total VM boot time is dominated by the VMM, with Solo5 and Firecracker being the fastest (3ms), QEMU microVM at around 10ms and QEMU the slowest at around 40ms.
- Filesystem Performance: Unikraft achieves lower read latency and lower write latency with different block sizes and are considerably better than ones from the Linux VM.
- Application Throughput: Unikraft is around 30%-80% faster than running the same app in a container, and 70%-170% faster than the same app running in a Linux VM. Surprisingly, Unikraft is also 10%-60% faster than Native Linux in both cases.
- Performance of Automatically Ported Apps: The results show that the automatically ported app is only 1.5% slower than the manually ported version, and even slightly faster than Linux baremetal.

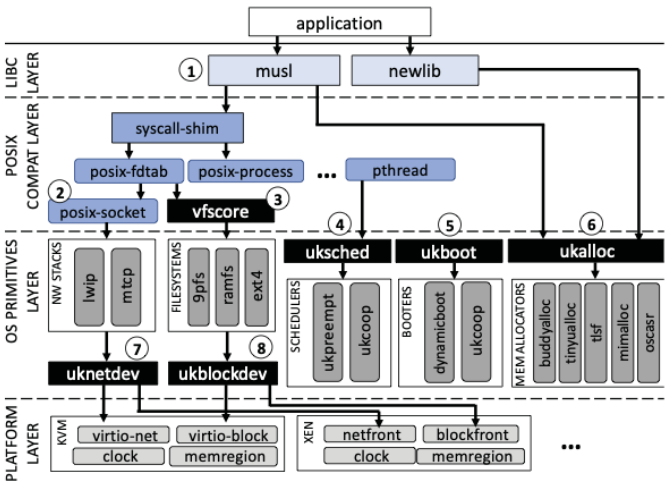

Fig. 4.3 Unikraft application stack

OSv

OSv is an Unikernel that runs existing Linux cloud applications on various hypervisors and machine architectures. OSv runs on 64-bit x86 and ARM architectures and supports KVM/Qemu, VMware, Xen and VirtualBox hypervisors.OSv demonstrates up to 25% increase in throughput and 47% decrease in latency. By using non-POSIX network APIs, it can further improve performance and demonstrate a 290% increase in Memcached throughput. OSv is designed as a drop-in replacement for applications that use a supported subset of the Linux application binary interface (ABI). The following below is the design of OSv:

- Memory Management: OSv uses virtual memory like general purpose OSs.OSv supports demand paging and memory mapping via the mmap API.
- No Spinlocks: The mutex implementation is based on a lock-free design by Gidenstam & Papatriantafilou (add reference), which protects the mutex's internal data structures with atomic opera- tions in a lock-free fashion.
- Network Channels: OSv almost all packet processing is performed in an application thread. Upon packet receipt, a simple classifier associates it with a channel, which is a single producer/single consumer queue for transferring packets to the application thread.

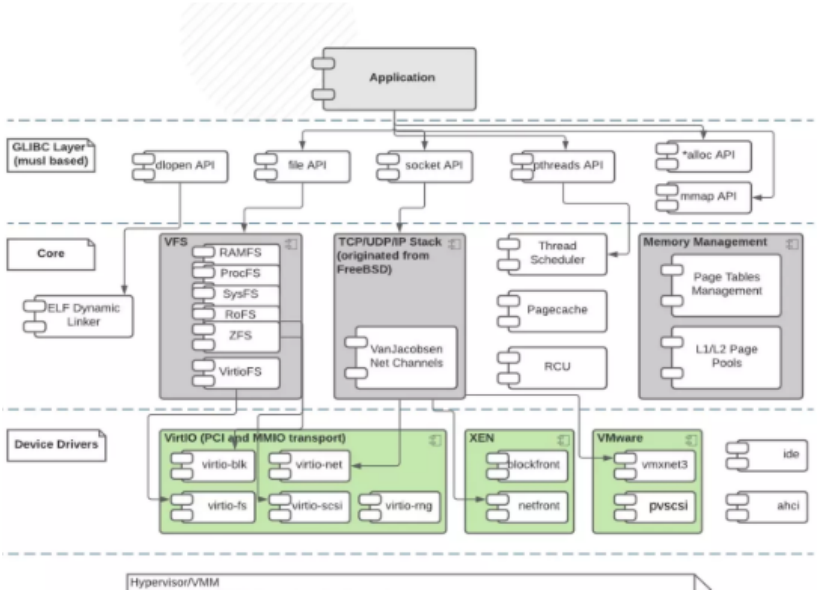

Fig. 4.4 OSv application stack

HermitCore

HermitCore is an Unikernel implementation designed for HPC. The kernel extends the multi-kernel approach with the advantages of a Unikernel.The focus of HermitCore is the mapping of the hardware to the software structure rather than full support of the Linux API.In a HermitCore system, each NUMA node runs its own HermitCore instance managing all its resources. The aims for Hermit core are the following:

- Reduction of OS noise.
- Predictable runtimes.
- Maintainability, extensibility, and flexibility.
- Abstraction of hardware details.
- Support for common HPC programming models (e. g., OpenMP, MPI).
- Simple integration into existing software stacks of compute centers.

Benchmarks conducted:

- Operating System Micro-Benchmarks.
- Hourglass Benchmark (For OS Noise).
- Inter-kernel Communication Benchmark.
- OpenMP Micro-Benchmarks.

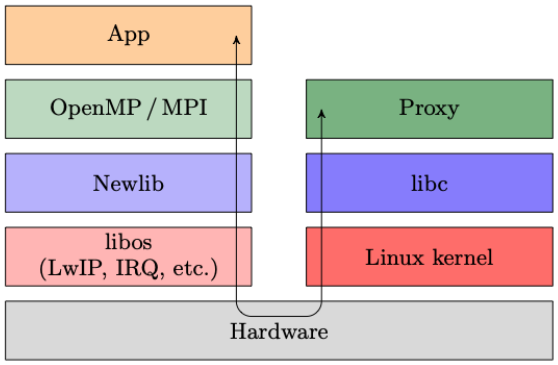

Fig. 4.5 HermitCore Software stack

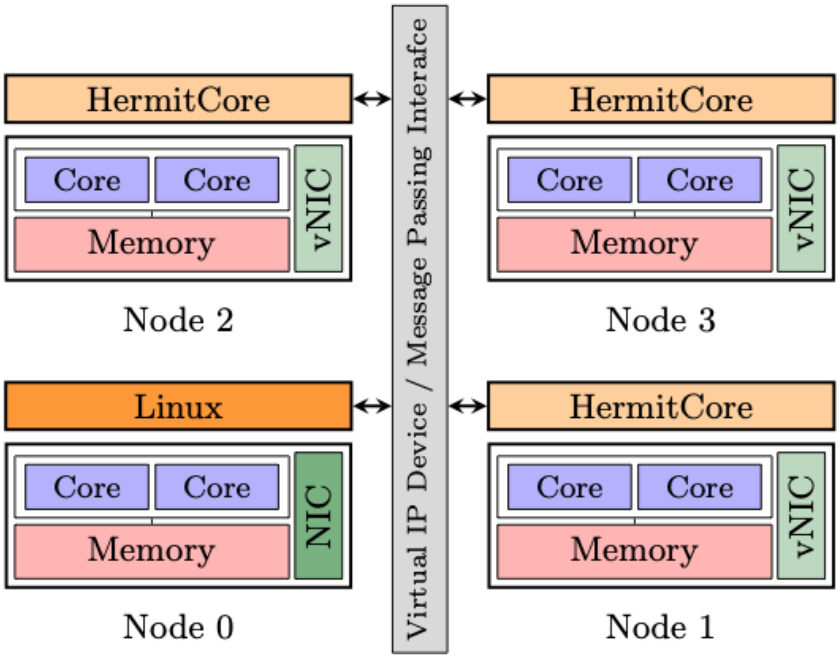

Fig. 4.6 A NUMA system with one satellite kernel per NUMA node

RKOS

RKOS is an unikernel implemented in Rust which offers safety guarantees comparable to implementations which depend on complex runtime libraries while being capable of providing predictable application performance demanded by real-time applications in a relatively simple implementation. Design decisions for RKOS are as follows:

- Mutual trust between components allows a shared, uniform address space.
- Virtualized runtime environments have uniform hardware configuration.

Performance Evaluations conducted:

- Run time memory footprint
- Binary size

ClickOS

ClickOS is an unikernel optimized for middleboxes that runs exclusively on the Xen hypervisor with small virtual machine memory footprint overhead (5 MB), fast boot times (under 30 milliseconds), and high performance networking capabilities.ClickOS adds only a 45 microsecond delay per packet. When compared to a general purpose Linux also running on

Xen, ClickOS network throughput is up to 1.5x times higher for MTU-sized packets and as much as 13.6x times higher for minimum-sized packets.

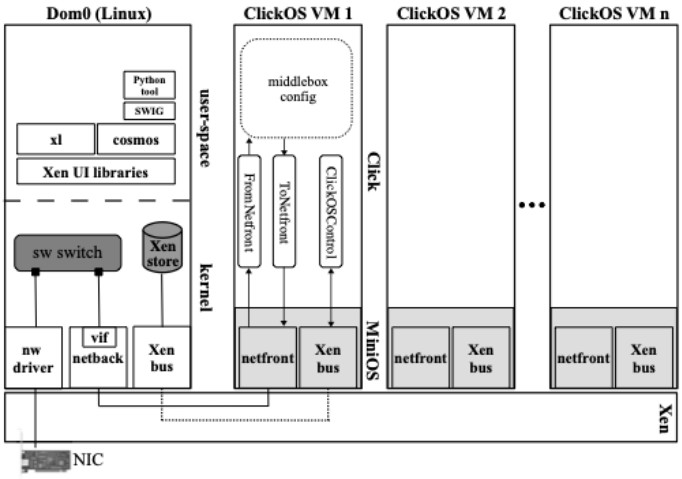

Fig. 4.7 ClickOS architecture

NanoOS

Nanos is a Unikernel implementation designed to run micro services on the Cloud it runs on top a Qemu Hypervisor. It has it's own Orchestrator written in Go called OPS. Nanos employs various forms of security measures found in other general purpose operating systems including ASLR and respects page protections that compilers produce.

ASLR:

- Stack Randomization
- Heap Randomization
- Library Randomization
- Binary Randomization

Page Protections:

- Stack Execution off by Default
- Heap Execution off by Default
- Null Page is Not Mapped

- Rodata no execute
- Text no write
- SMEP
- UMIP

Performance Evaluations conducted:

- Bootup Times.
- Requests per second.

IncludeOS

IncludeOS is a single tasking library operating system for cloud services which is written from scratch in C++. Key features include: extremely small disk and memory footprint, efficient asynchronous I/O, OS-library where only what your service needs gets included. In the test case the bootable disk image consisting of a simple DNS server with OS included is shown to require only 158 kb of disk space and to require 5-20% less CPU-time. The contributions of IncludeOS are:

- Extreme resource efficiency and footprint.
- Efficient deployment process.
- Virtualization platform independence.

The proposed benefits of IncludeOS in comparison to Linux Kernels are:

- Extremely small disk and memory footprint.
- No host or software dependencies, other than virtual x86 hardware, and standard virtio for networking
- No system call overhead (The OS and the services are in the same binary, and the system calls are simple function calls(i.e without passing any memory protection barriers)).
- Reduced number of VM exits by keeping the number of protected instructions very low.

Performance Evaluations conducted:

- Bootup times
- Memory performance (i.e The Stream Benchmark)

Azelea

Azalea is a multi-kernel OS, which consists of Unikernels and a full kernel. Azelea Unikernel provides scalability and parallel performance. The full kernel provides compatibility with POSIX APIs that Unikernel cannot handle. The Full kernel is combined with the Unikernel for side by side partitioning. The Azelea Unikernel is a library OS which consists of the following:

- Kernel Functions
- Run time libraries
- Application

A server can run multiple Azelea-unikernels with the number of cores and memory allocated. The Linux install which is a part of the server acts as a driver and that loads each Unikernel or supports communication between other nodes. The contributions of Azelea Uni-kernels are:

- Lightweight kernel.
- Compatibility with legacy application (i.e Support for statically build Linux binaries).
- I/O offloading (i.e FWK(Full weight kernel) handles all the I/O offloading so that applications can be executed without any interference).

Performance Evaluations conducted:

- OS Noise (FTQ(todo: reference), FWQ(todo: reference), Hour Glass(todo: reference))
- IO offload acceleration

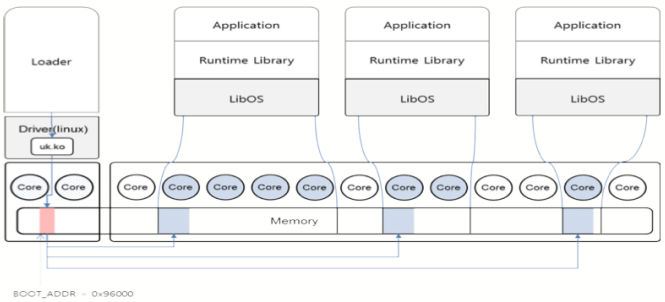

Fig. 4.8 Azelea-unikernel in a single KNL

Unikernel	*Languages supported*	*Targets*	**Performance evaluation**
Unikraft	*C, C++, Rust, Go, Python*	*KVM, Xen, Linux Userspace, Solo5, VMware, HyperV*	- Resource Efficiency - Filesystem Performance - Application throughput. - Performance of Automatically ported apps.
OSv	Java, C, C++, Node, Ruby, Go	Virtual Box, EXSi, KVM and HyperV.	
NanoOS	C, C++, Go, Java, Node js, Python, Rust, Rust, Ruby, and PHP	KVM, XEN,ESXi and Hyper V	- Boot Up times - Request per second
HermitCore	Rust, C,C++, Go and Fortran	uhyve, KVM and bare metal	- Operating system micro benchmark - Hourglass benchmark - Inter-kernel communication benchmark - OpenMP micro benchmark
RKOS	Rust	Bare metal	- Run time memory footprint - Binary size
ClickOS	C++	Xen	- ClickOS Switch - Memory Footprint - Boot times - Delay (When processing packets) - Throughput (Amount of packets ClickOS can handle) - State Insertion - Chaning - Scaling out
IncludeOS	C++	KVM, VirtualBox, ESXi, OpenStack	- Bootup times - Memory Performance

4.1.4 Unikernel analysis

The following section consists of analysis of the Uni-kernels implementations surveyed in the current literature:

4.2 TAG based architecture survey

The following was a survey conducted on existing TAG based implementations and the recent survey based on TAG based architectures [2] published in 2022 was a good staring point to understand about various implementations of TAG based architectures with the high level metrits and limitations. The following section provides our own version of the Survey to help decide the best implementations to answer the research questions (//TODO reference research questions chapter).

Before deep diving into TAG based architecture implementations it is important to answer what is a TAG based architecture ? and the high level of various categories of various TAG based architectures.

Tagged architectures are a prominent class of hardware security primitives that augment data and code words with tags. The tags, which function as the security metadata about memory, are created before the program is loaded. Then, at runtime, the hardware enforces security policies on the tags to provide safety guarantees. The advantage being tags automate the secure and efficient management of security metadata.

Tags policies as designed to address mostly:

- Type and memory corruption
- Integer overflows
- Thread safety
- Buffer overflows

TAG policies can be categorized into 5 main categories which is:

- Information-low control (IFC) policies
- Dynamic information-low tracking (DIFT) policies
- Capability models
- Programmable architectures

According to the TAG based architecture survey [2] there are 37 published efforts on TAG based architectures over the past decade and 20 published efforts preceding that.

4.2.1 Timder V [9]

It is a tagged memory architecture for flexible and efficient isolation of code and data on small embedded systems. The TAG isolation is augmented with a memory protection unit to isolate induvidual processes. Timber V is compatible with existing code. The contributions of the paper are:

- Efficient tagged memory architecture for isolated execution on low-end processors.
- Concept introduced called stack interleaving that allows efficient and dynamic memory management.

- Lightweight shared memory between enclaves.
- Efficient shared MPU (i.e Memory Protection Unit) design.

4.2.2 ARM MTE [1]

The ARMv8.5-Memory Tagging Extension (MTE) aims to increase the memory safety written for unsafe languages without requiring source code changes and in certain cases without recompilation. It generally focuses on the bounds checking use case, Though it provides limited tags which means it can only provide probabilistic overflow detection. It is one of the latest commercial incarnations of memory-safety-focused tagged architectures.

4.2.3 D-RI5CY [5]

It provides a design a design and implementation of a hardware dynamic information flow tracking (DIFT) architecture for RISC-V processor cores. The paper presents a low overhead implementation of DIFT that is specialized for low-end embedded systems for IOT applications. The following are high level contributions:

- Design f D-RI5CY, A DIFT-protected implementation of the RI5CY processor core. The paper implements the modification of the DIFT TAG propagation and TAG checking mechanism in a way that is transparent to the execution of the regular instructions.
- Concept introduced called stack interleaving that allows efficient and dynamic memory management.
- Lightweight shared memory between enclaves.
- Efficient shared MPU (i.e Memory Protection Unit) design.

4.2.4 HyperFlow [3]

It is a design and security implementation that offers security assurance because it is implemented using a security-typed hardware description language. It allows complex information flow policies to be configured at run time. The paper introduces ChiselFlow, a new secure hardware description language. The contribution of the paper includes:

- Processor architecture and implementation designed for timing-safe information flow security.

- Complete RISC-V instruction set extended with instructions for information flow control.
- Verified at design time with a hardware description language.
- Novel representations of lattices that can be implemented in hardware efficiently.

HyperFlow implements a nonmalleable IFC policy using tags. To eliminate timing side channels, the processor tracks the tag of the currently executing code and lushes caches, TLB, branch predictor, and other micro-architectural state on changes in the conditionality or integrity tag of the running code. The modifications to avoid timing side channels seem more extensive than those to add tags. The authors report overheads in cycles per instruction of between 1% and 69%, largely due to padding the multiply operation to the worst-case number of cycles.

4.2.5 SDMP [6]

This paper focuses on designing metadata tag based stack-protection security policies for general purpose tagged architecture. The policies specifically exploit the natural locality of dynamic program call graphs to achieve cache-ability of the metadata rules that they require. The simple Return Address Protection policy has a performance overhead of 1.2% but just protects return addresses. The two richer policies present, Static Authorities and Depth Isolation, provide object-level protection for all stack objects. When enforcing memory safety, The Static Authorities policy has a performance overhead of 5.7% and the Depth Isolation policy has a performance overhead of 4.5%. The contribution of the paper includes:

- The formulation of a range of stack protection policies within the SDMP model.
- Three optimizations for the stack policies: Lazy Tagging, Lazy Clearing and Cache Line Tagging.
- The performance modeling results of the policies on a standard benchmark set, including the impact of the proposed optimizations.

4.2.6 Typed Architecture [4]

This paper introduces Typed Architectures, a high-efficiency, low-cost execution substrate for dynamic scripting languages, where each data variable retains high-level type information at an ISA level. Typed Architectures calculate and check the dynamic type of each variable implicitly in hardware, rather than explicitly in software. Typed Architectures provide

hardware support for flexible yet efficient type tag extraction and insertion, capturing common data layout patterns of tag- value pairs. The evaluation using a fully synthesizable RISC-V RTL design on FPGA shows that Typed Architectures achieve mean speedups of 11.2% and 9.9% with minimum speedups of 32.6% and 43.5% for two production- grade scripting engines for JavaScript and Lua. The contribution of the paper includes:

- ISA extension to efficiently manage type tags in hardware, which can be flexibly applied to multiple scripting languages and engines.
- Design and implement the Typed Architecture pipeline, which effectively reduces the overhead of dynamic type checking at low hardware cost.
- Prototype the proposed processor architecture using a fully synthesizable RTL model to execute two production-grade scripting engines with large inputs on FPGA (executing over 274 billion instructions in total) and provide a more accurate estimate of area and power using a TSMC 40nm standard cell library.

4.2.7 Dover [7]

It is a secure processor that extends the conventional CPU with a Policy EXecution co-processor (PEX). PEX maintains metadata of every word assessable by the application processor. PEX enforces software-defined policies at the granularity of each instruction executed by the AP(i.e application process) CPU. Hardware interlocks enforce strict separation between code and data for user-land and policy-related. The Dover system has a dover specialized kernel and modifications to the GCC toolchain which can implement a wide range security and safety policies on top existing C based applications.

4.2.8 CHERI [8]

CHERI (Capability Hardware Enhanced RISC Instructions) extends conventional processor Instruction-Set Architectures (ISAs) with architectural capabilities to enable fine-grained memory protection and highly scalable software compartmentalization. CHERI is a hybrid capability architecture that can combine capabilities with conventional MMU(i.e Memory Management Unit) based systems. The contribution of the following project include:

- ISA changes to introduce architecture capabilities.
- New microarchitecture proving that capabilities can be implemented efficiently in hardware. Support for efficient tagged memory to protect capabilities and compress capabilities to reduce memory overhead.

- Newly designed software construction model for that uses capability to provide fine grain memory protection and scalable software compartmentalization.
- Language and Compiler extension to use capabilities for C and C++.
- OS extensions to use (and support application use of) fine-grained memory protection (spatial, referential, and (non-stack) temporal memory safety) and abstraction extensions to support scalable software compartmentalization.

Chapter 5

Expirements

Chapter 6

Research Goals

Chapter 7

Research Timeline

Chapter 8

Conclusion

References

[1] (2019). 1 Armv8.5-A Memory Tagging Extension. https://developer.arm.com/-/media/Arm%20Developer%20Community/PDF/Arm_Memory_Tagging_Extension_Whitepaper.pdf?revision=ef3521b9-322c-4536-a800-5ee35a0e7665&la=en&hash=D510ED84099D3B8AA34723AC110D48E3A28FA8D6. [Accessed 20-Oct-2022].

[2] (2022). TAG: Tagged Architecture Guide | ACM Computing Surveys — dl.acm.org. https://dl.acm.org/doi/abs/10.1145/3533704. [Accessed 20-Oct-2022].

[3] Ferraiuolo, A., Zhao, M., Myers, A. C., and Suh, G. E. (2018). Hyperflow: A processor architecture for nonmalleable, timing-safe information flow security. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS '18, page 1583–1600, New York, NY, USA. Association for Computing Machinery.

[4] Kim, C., Kim, J., Kim, S., Kim, D., Kim, N., Na, G., Oh, Y. H., Cho, H. G., and Lee, J. W. (2017). Typed architectures: Architectural support for lightweight scripting. *SIGARCH Comput. Archit. News*, 45(1):77–90.

[5] Palmiero, C., Di Guglielmo, G., Lavagno, L., and Carloni, L. P. (2018). Design and Implementation of a Dynamic Information Flow Tracking Architecture to Secure a RISC-V Core for IoT Applications. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–7. ISSN: 2377-6943.

[6] Roessler, N. and DeHon, A. (2018). Protecting the stack with metadata policies and tagged hardware. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 478–495.

[7] Sullivan, G. T., DeHon, A., Milburn, S., Boling, E., Ciaffi, M., Rosenberg, J., and Sutherland, A. (2017). The dover inherently secure processor. In *2017 IEEE International Symposium on Technologies for Homeland Security (HST)*, pages 1–5.

[8] Watson, R. N., Woodruff, J., Neumann, P. G., Moore, S. W., Anderson, J., Chisnall, D., Dave, N., Davis, B., Gudka, K., Laurie, B., Murdoch, S. J., Norton, R., Roe, M., Son, S., and Vadera, M. (2015). Cheri: A hybrid capability-system architecture for scalable software compartmentalization. In *2015 IEEE Symposium on Security and Privacy*, pages 20–37.

[9] Weiser, S., Werner, M., Brasser, F., Malenko, M., Mangard, S., and Sadeghi, A.-R. (2019). TIMBER-V: Tag-Isolated Memory Bringing Fine-grained Enclaves to RISC-V. In *Proceedings 2019 Network and Distributed System Security Symposium*, San Diego, CA. Internet Society.