Mapping Unikernels with TAG based architectures

Akilan Selvacoumar

Mathematics and Computer Sciences
Heriot Watt University

Year 1 progression report of:
Doctor of Philosophy

December 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
December 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

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 ?
- Using TAG standard memory with interleaving for speeding up types for dynamic languages and execution of parallelization on Mult-kernels with each core running a Uni-kernel.

]

Chapter 3

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).

3.1 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.

3.1.1 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" [26]. Unikernels is defined as "Unikernels are specialized, single-address-space machine images constructed by using library operating systems." [Uni]. 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[31] 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

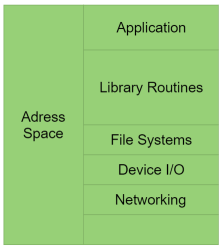

Fig. 3.1 Unikernel application stack [7]

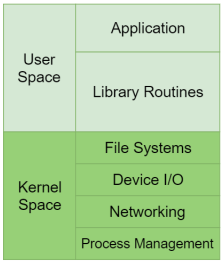

Fig. 3.2 Normal application stack [7]

enabling 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.

3.1.2 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

3.1.3 Implementations

Unikraft [20]

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 separation to avoid costly context switching.
- Static linking: Compiler features such as dead code elimination and link time optimization 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 components:

- Micro libraries: Micro-libraries are software components which implement one of the core Unikraft APIs.
- Build system: The 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 in Unikraft:

- 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 bare-metal.

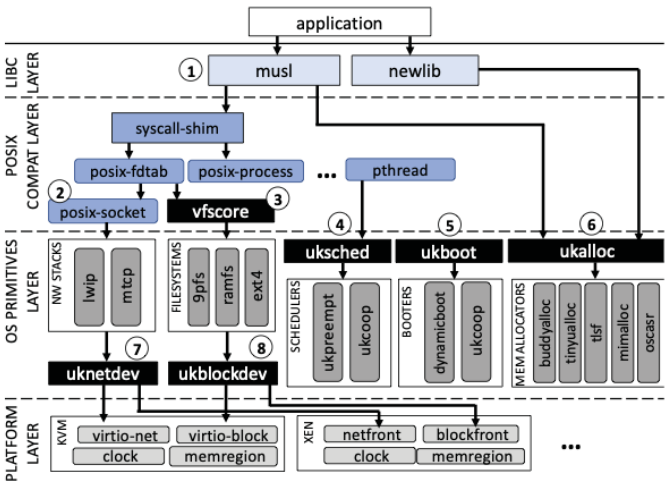

Fig. 3.3 Unikraft application stack [20]

OSv

OSv[Kivity et al.] 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 [16], which protects the mutex's internal data structures with atomic operations in a lock-free fashion.
- Network Channels: In OSv almost all packet processing is performed in an application thread. Upon packet received, a simple classifier associates it with a channel, which is a single producer/single consumer queue for transferring packets to the application thread.

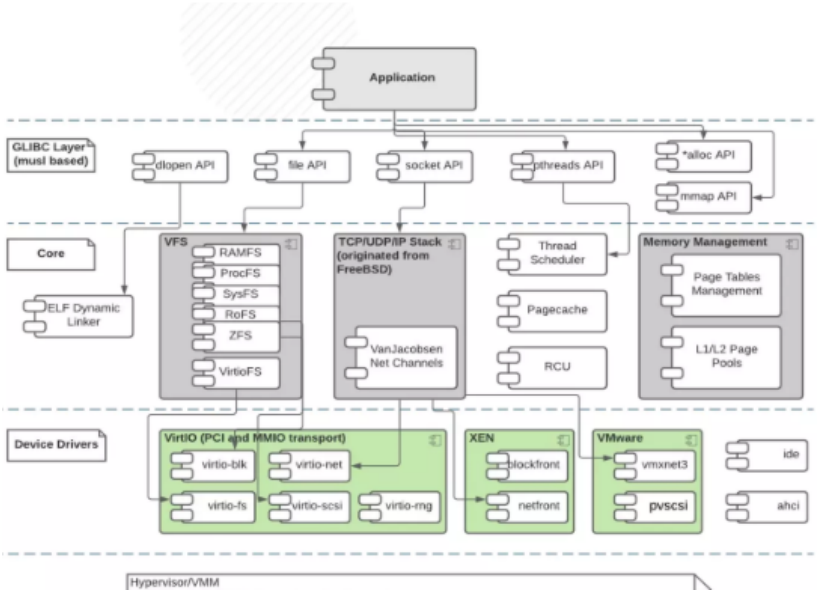

Fig. 3.4 OSv application stack [ScyllaDB]

HermitCore

HermitCore[24] 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.

The following are derived projects from the hermit-core project:

- HermitTux [29] : It is a linux binary compatible Unikernel that can run native linux executables.
- RustyHermit [23]: Implementation of the Hermit core Unikernel in Rust.
- Lib-hermitMPK [38] : Providing support for IntelMPK for RustyHermit to isolate the unsafe parts of the kernel and application with proven performance similar RustyHermit without the memory protection.

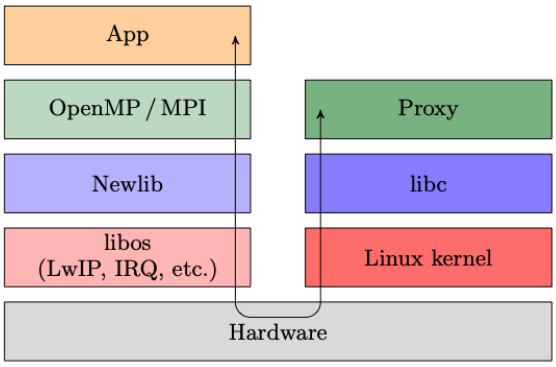

Fig. 3.5 HermitCore Software stack [24]

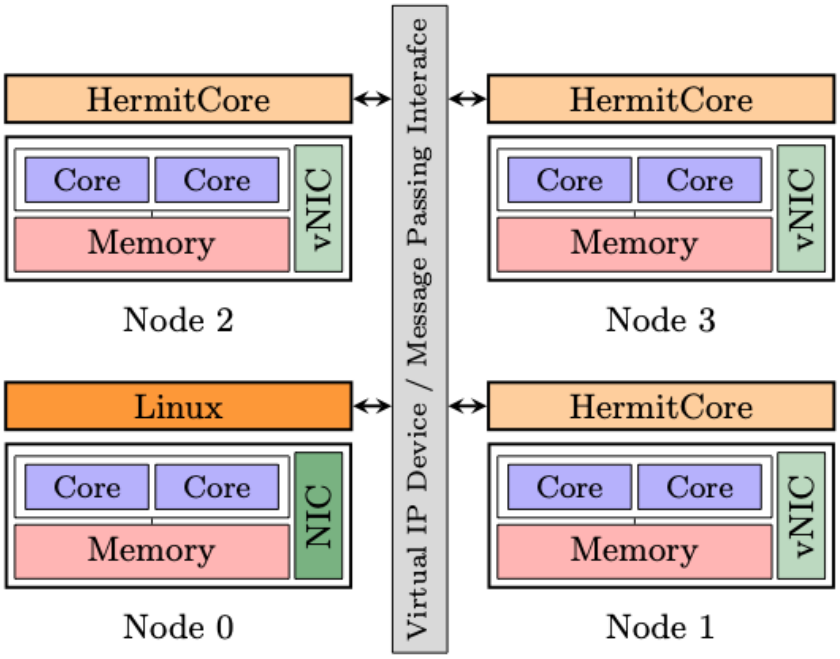

Fig. 3.6 A NUMA system with one satellite kernel per NUMA node [24]

RKOS

RKOS[Marheine] 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[Martins et al.] 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.

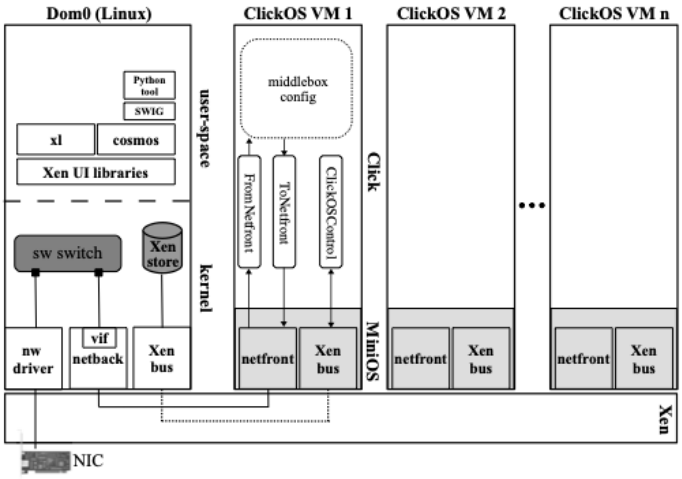

Fig. 3.7 ClickOS architecture [Martins et al.]

NanoOS [Nan]

Nanos is a Unikernel implementation designed to run micro services on the Cloud, it runs on top of a Qemu Hypervisor and 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 the 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[11] 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 VMs exits by keeping the number of protected instructions very low.

Performance Evaluations conducted:

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

Azelea

Azalea[Aze] 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 the 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, FWQ, Hour Glass) [12]
- IO offload acceleration [17]

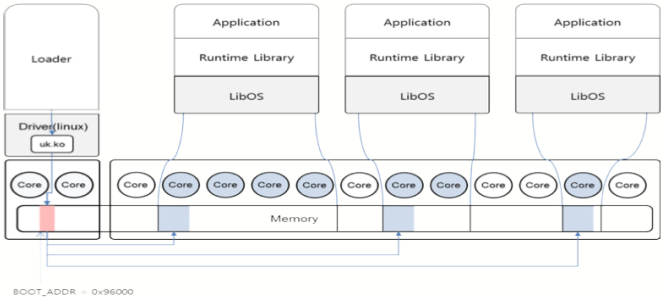

Fig. 3.8 Azelea-unikernel in a single KNL [Aze]

HaLVM

HaLVM(Haskell Lightweight Virtual machine) is an unikernel implementation based on Xen hypervisor (i.e type 1 hypervisor). HaLVM is implemented using Haskell. HaLVM is suitable for small, single-use and low-dependence programs. There was only 1 published work was a paper on analyzing parallel programs model for HaLVM[13].

Parallel Model	Unit of Parallelism	Running	Scalability
Eval monad	Spark	Yes	No
`forkIO`	green thread	Yes	No
`forkOS`	OS thread	No	-
Cloud Haskell	process	No	-
IVC	VM	Yes	Yes

Fig. 3.9 Performance Evaluations conducted(parallel model) [13]

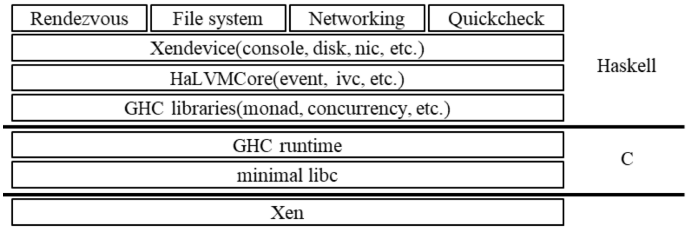

Fig. 3.10 HaLVM architecture [13]

Mirage

Mirage[Madhavapeddy et al.] produces an Unikernel by compiling and linking OCaml to an Xen VM image. The objective was to combine static type-safety with a single address-space layout. Using Mirage it is possible to use libraries such as networking, storage and concurrency that works under unix during development, when compiled to production becomes operating system drivers.

Mirage takes advantage of Ocaml for the following reasons:

- Static type checking
- Automatic memory management
- Modules
- Metaprogramming

Performance Evaluations conducted:

- Boot time.
- Thread performance.
- Throughput.
- Sessions per for a sample dynamic web application.

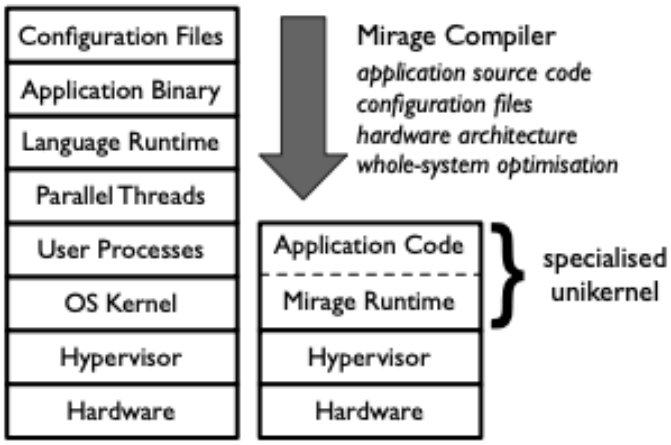

Fig. 3.11 Azelea-unikernel in a single KNL [Aze]

3.1.4 Unikernel analysis

The following section consists of analysis of the Uni-kernels implementations surveyed in the current literature. The analyses is based on:

- Best suitable implementations for various platforms supported ?
- How do each of them handle parallel applications ?

Table 3.1 Analyzing various Uni-kernel implementations

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.	- Macro Benchmarks (Memcached, SPECjvm2008) - Micro Benchmarks (Network performance, JVM ballon, context switches)
NanoOS	C, C++, Go, Java, Node js, Python, 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 - Chaining - Scaling out
IncludeOS	C++	KVM, VirtualBox, ESXi, OpenStack	- Bootup times - Memory Performance
Azelea	C	Bare-metal	- OS Noise (FTQ, FWQ, Hour Glass) - IO offload acceleration

Best suitable implementations based on platforms(i.e targets) supported ?

This refers to which Uni-kernel implementation would be preferred based on the various targets supported, this is based on table 3.1. Based on the number of targets supported Unikraft has the most amount of targets supported. Since the the research goals (//todo refer research goals) for using Uni-kernels is to run on bare-metal as a major requirement (This is because of the way multi-kernels work 3.2).Unikraft would be suitable for testing a multi-kernel environment, but porting to bare-metal would be an important step along the way. Hermit-core would be suitable since it does support running on bare-metal and runs on a hypervisor (i.e KVM and uhyve).

Multi-core

Unikraft does not currently support Multi-core mode yet. By default it uses the library uklock which synchronization primitives such as Mutexes and semaphores. If multi-core was supported primitives such as spin-locks and RCUs would be supported.

OSv supports running application in multiple cores. OSv thread scheduler is lock-free, preemptive, tick-less, fair, scalable and efficient.

- Lock-free: The scheduler keeps separate run-queue on CPU. Sleeping threads are not listed on any run-queue. Separate run queues leads to a situation where one CPUs queue has more runnable threads than another CPUs queue, this impacts the scheduler. This is solved by a load balancer thread on each CPU.
- Preemptive: OSv supports preemptive multi-tasking. According to the paper[**?**] this feature is useful for maintaining per-CPU variables and RCU locks.
- Tick-less: OSv uses a high resolution clock, scheduler accounts to each thread the exact time it consumed, this is in-contrast to approximating ticks.
- Fair: On each reschedule, the scheduler must decide which of the CPUs runnable threads should run next and for how long. OSv scheduler calculates the exponentially-decaying moving average of each thread's recent run time. The scheduler decides the next runnable thread with the lowest moving-average runtime.
- Scalable: OSv scheduler has O(log N) complexity in the number of runnable threads on each CPU.

- Efficient: Apart from the scheduler scalability, OSv employs additional techniques to make the scheduler and context switches more efficient. OSv single address space means there is no need to switch page tables and or flush the TLB on context switches. This means that context switches are significantly cheaper than the standard multi-process operating system.

HermitCore (i.e currently called RustyHermit) supports multi-threaded and multiprocessing applications. The scheduler does not support load balancing this is because explicit thread placing is preferred over automatic strategies. The scheduling overhead is also minimized by employing a dynamic timer (i.e the kernel does not interrupt computational threads which runs on particular cores and due to this a timer is not needed).

RKOS supports concurrency and multi-threading. The threads are preemptive and scheduled non-cooperatively. Preemptive multitasking was selected because it was largely used with existing systems.

Azelea Unikernel supports multi threaded applications. Each core uses a queue to manage multiple threads and with a round robin scheduler.

3.2 Multi-kernels Survey

The following is the survey for Multi-kernels. The introduction is based on the first paper published on Multi-kernels [10], follows up with a survey on various implementations and with an analysis section of various multi-kernel implementation.

3.2.1 Introduction to Multi-kernels

"A multikernel operating system treats a multi-core machine as a network of independent cores, as if it were a distributed system" [5]. It implements interprocess communications as message-passing. The design of multi-kernels can be stated as the following:

- Inter-core communication is explicit.
- OS Structure is hardware neutral.
- State is view as replicated instead of shared.

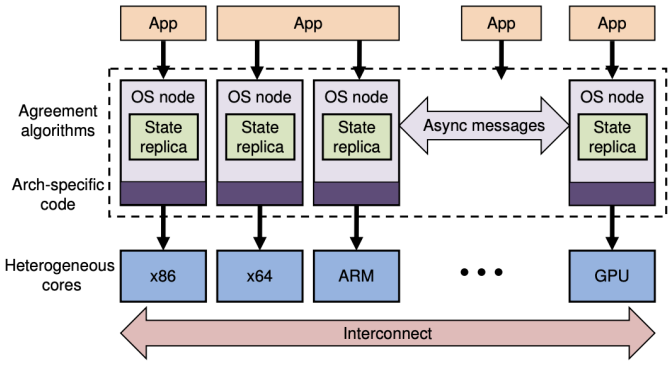

Fig. 3.12 The Multi-kernel model [10]

Benefits of Multi-kernels

The following below highlights the major characteristics of Multi-kernels.

- Ability to handle diverse set of cores:
- Interconnect matters:
- Messages cost less than shared memory:

3.2.2 Implementation

The following section mentions about the Multi-kernel implementations.

Barrelfish

Barrelfish[10] is a multi-kernel operating system that consists of a small kernel running on each core. The kernels share no memory (even on machines with cache-coherent shared RAM). A CPU driver in Barrelfish represents a kernel when is ran on a given core. In a heterogeneous system the CPU driver would different based on the architecture of the core.

3.2.3 Popcorn Linux

Popcorn[Barbalace et al.] linux is a replicated-kernel OS based on Linux. Popcorn boots up multiple instances of Linux kernels on a multi-core hardware. Popcorn linux was evaluated based on the NAS benchmark [8]. Popcorn linux uses a customized compiler based off LLVM which translates C/C++ applications into machine code for runtime execution and migration across multiple ISAs. Papers and sub projects derived from popcorn Linux:

- Aparapi: Applying Source Level Auto-Vectorization
- AIRA: A Framework for Flexible Compute Kernel Execution in Heterogeneous Platforms
- HEXO: Offloading HPC Compute-Intensive Workloads on Low-Cost, Low-Power Embedded Systems.
- H-Container: Enabling Heterogeneous-ISA Container Migration in Edge Computing.
- HeterSec: Software diversification using ISA heterogeneity

3.2.4 FusedOS

FusedOS was one of the first to combine linux with a LWK(Light weight kernel). FusedOS was assuming heterogeneous hardware architecture that consists of both a light weight and full weight cores. The full cores runs linux and is also responsible to partition hardware resources between itself and LWKs. To execute an application the LWK requests hardware resources (i.e light weight cores and memory) from the FWK(Full weight Kernel, This refers to the linux kernel). The system calls are generated by the application and are forwarded to Linux which is then handled with LWK process.

3.2.5 IHK/McKernel

IHK/McKernel is a multi-kernel approach which runs Linux and LWKs side by side on compute nodes. A low-level software infrastructure which is present at the heart of the stack which is called Interface for Heterogeneous Kernels (IHK). By using IHK it is possible to dynamically partition resources in a many-core environment. An IKC (Inter-Kernel communication) layer is also introduced upon which the system call delegation is implemented. McKernel is a light weight kernel written from scratch and designed for HPC. McKernel retains a binary compatible ABI with Linux. It supports multi-threading with a simple round robin cooperative scheduler.

3.2.6 FFMK

FFMK (Fast and fault tolerant Microkernel based system) which is designed for Exascale computing. It investigates the feasibility of a Microkernel based hybrid OS designed for HPC.It relies on a L4 microkernel and a para-virtualized Linux instance (i.e L4Linux[Lackorzynski]). The idea of FFMK is to run HPC application directly on L4 with transparent access to linux

features by using L4Linux. The L4Linux user process can be decoupled from the linux kernel and moved to another core if required (i.e by using the L4 Thread).

3.3 TAG based architecture survey

The following was a survey conducted on existing TAG based implementations and the recent survey based on TAG based architectures [6] published in 2022 was a good staring point to understand about various implementations of TAG based architectures with the high level merits 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).

3.3.1 Introduction to TAG based architectures

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

3.3.2 Implementations

According to the TAG based architecture survey [6] there are 37 published efforts on TAG based architectures over the past decade and 20 published efforts preceding that. The following below are relevant papers in relation to the research questions:

Timder V

Timber V[40] 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 individual 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.

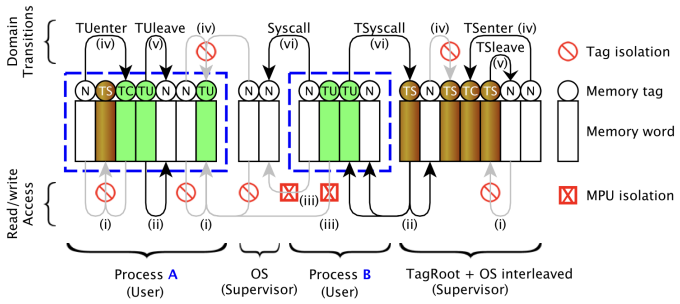

Fig. 3.13 TimberV TAG interleaved on flat physical memory[40]

ARM MTE

The ARMv8.5-Memory Tagging Extension (MTE)[4] 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.

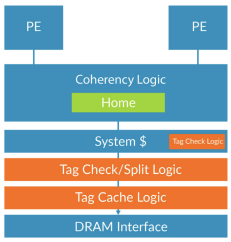

Fig. 3.14 Example of an ARM MTE-based system [4]

D-RI5CY

D-RISCY[30] 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.

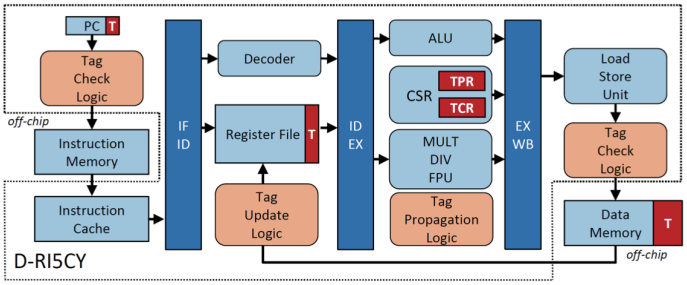

Fig. 3.15 Block diagram of the D-RI5CY processor. In red and pink the DIFT components. [30]

HyperFlow

Hyperflow[15] 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.

SDMP

SDMP[32] 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.

Typed Architecture

This paper introduces Typed Architectures[18], 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.

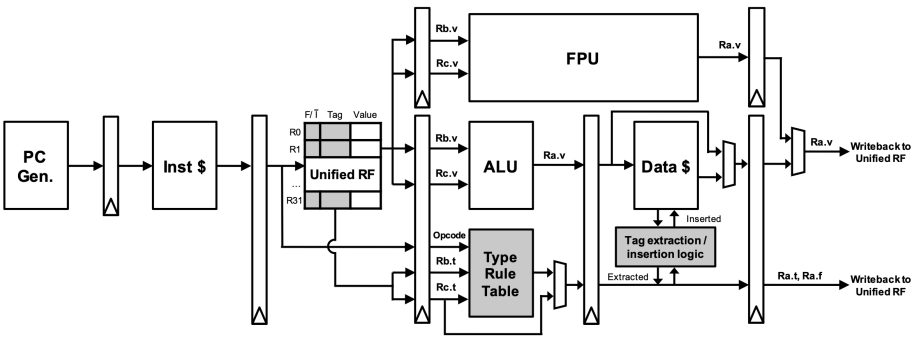

Fig. 3.16 Pipeline structure augmented with Typed Architecture [18]

Dover

Dover[37] 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.

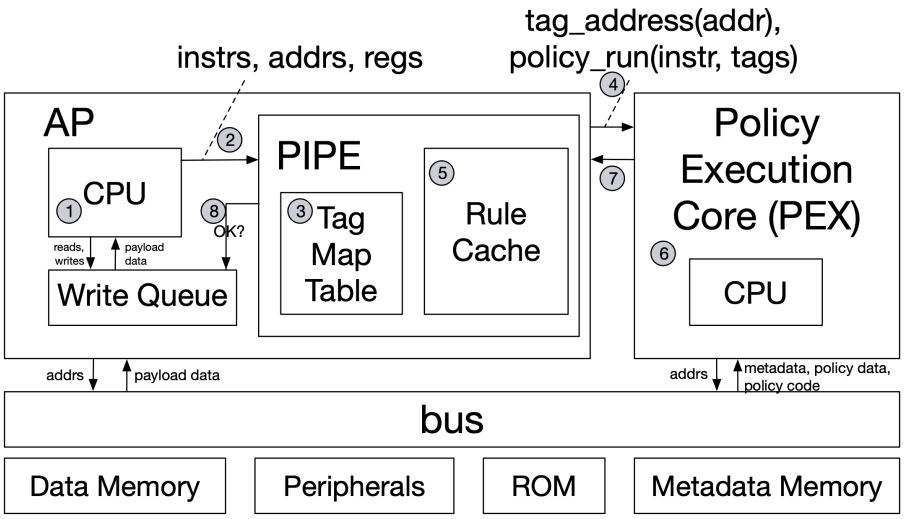

Fig. 3.17 High level overview of Dover Architecture [37]

x

CHERI [39]

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.

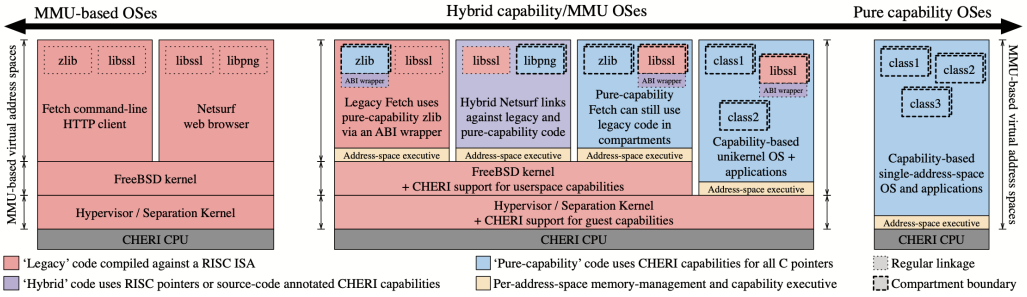

Fig. 3.18 Spectrum of Hardware-software architectures, from conventional MMU-based virtualization and OS process models to single address-space capability system [39]

Low-Fat Pointers

Low-Fat Pointers[21] adds hardware-managed tags to the pointer. This, in turn, allows the pointers to be used as capabilities to facilitate fine-grained access control and fast security domain crossing. The dedicated checking hardware runs in parallel with the processor's normal data-path so that the checks do not slow down processor operation (0% runtime overhead).The following paper has a gate-level implementations of the logic for updating and validating these compact fat pointers and show that the hardware requirements are low and the critical paths for common operations are smaller than processor(i.e ALU operations). The contribution of the following project include:

- Design and evaluation of a new, compact fat-pointer encoding and implementation (BIMA).
- Hardware that enforces the BIMA bounds checking and update, making the fat pointers unforgeable and non-bypass able.
- Pipeline organization that allows the BIMA encoding to run just as fast as the baseline processor without spatial safety checking.

HardBound

HardBound[Devietti et al.] focuses on an architectural hardware bounded pointer primitive that supports hardware and software enforcements for memory safety in C programs. The C pointer representation is left intact but the bounds information is maintained separately and invisibly by the hardware. This means the bounds are initialized by software and is then propagated and transparently maintained by hardware (which automatically checks a pointer bound before it's dereferenced). The paper combined intra-procedural compiler instrumentation and hardware bounded pointers to enable a low overhead approach to enforce complete spatial memory safety in unmodified C programs. Based on the experiments conducted on the following paper the runtime overhead was between 5% to 9%. The following does not provide full type safety, handling dangling pointers and uninitialized memory reads. The contribution of the following project include:

- A hardware bounded pointer primitive and accompanying complier transformation that when combined enforce spatial safety for C programs. This is to minimize changes to the compiler infrastructure and to retain compatibility with legacy C code.
- Efficient implementation of hardware bounded pointers: This means using a compressed metadata encoding, the entire base and metadata for bounds are stored in a reserver portion of virtual memory. The hardware encodes the bounded pointer metadata by using just a few bits. These bits can be stored either in memory or unused bits in the pointer itself.
- Experimentally evaluating functional correctness and performance of the approach in this paper.

Chapter 4

Year 1 Activity

This is section will be split into the timeline of activities of year 1.

4.0.1 Poster SISCA PhD Conference

The PhD symposium was held in in Glasgow Caledonian University for 2 days. A poster by the title "Benchmarking Unikernels with distributed map reduce"[36]. The objective for attending this conference was to socialize with other PhD students in Scotland by also presenting one of the plans of the initial experiments.

Submission type:

1. Poster: "Benchmarking Unikernels with distributed map reduce"[36]

4.0.2 Europar PhD symposium and poster session

The Europar PhD conference was held in the university of Glasgow. The title of the symposium paper being "Benchmarking Parallelism in Unikernels"[34]. This is expected to be published in springer proceeding of Europar 2022.

Submission type:

1. Poster: "Benchmarking Parallelism in Unikernels"[35]
2. PhD Symposium paper: "Benchmarking Parallelism in Unikernels"[34]

Chapter 5

Research Timeline

The following chapter talks about the research activity timeline conducted for a duration of 2 and half years. There will 2 sections and with a base planner of the activities conducted on the following years. The plan is a subject to change based on any deviation which will be attempted to be covered in the risk analysis section. The recent tasks are provided in depth in contrast to later tasks which will be more open ended as it reliant to the results from preceding tasks.

5.0.1 Year 2

This section is split by a month by month planner to help keep tracks of tasks.

January 2023

The high level overview being that most of the setups for the upcoming experiments are complete.

1. Review of year 1 report submitted
2. Setup test cluster for testing popcorn linux
3. Setup popcorn linux on the test cluster
4. Start running existing popcorn linux benchmarks
5. Setup RustyHermit and HermitCore independently with test application
6. Setup Cheri on a QEMU emulator and run a sample C program

February 2023

1. Deep dive understanding to HEXO fork of popcorn linux source code vice.
2. Setup HEXO offload tasks to a uni-kernel on an external machine using hermit-core and test of 2 more external devices to benchmark the scheduler used by HEXO linux.
3. Start working more on porting HermitTux to RustyHermit and look into development of Unikraft for support of Cheri and switch into the Unikraft for further development if there is full support for Cheri.
4. Start drafting a conference paper or journal based on improvements to the HEXO papers scheduler and support to either RustyHermit or Unikraft.

March 2023

1. Start porting either RustyHermit or Unikraft to support the Cheri Architecture. (The following sub-section is assuming RustyHermit is selected).
   (a) Making a rust based clone of Hermitux on the rust based rusty-hermit.
   (b) Merging certain C libraries from CheriBSD (or if possible rewriting the C libraries in Rust directly) with the RustyHermit kernel.
2. Investigating having Popcorn modified LLVM C/C++ features with the GHC Haskell compiler.

April 2023

1. Continue on making rust based clone of Hermitux on the rust based rusty-hermit and make decision if it's worth going on.
2. Continue work on the conference paper based on the improvements of the HEXO paper.
3. Get access to the ARM based Cheri Morello

May 2023

1. Finalize conference paper/journal paper for improvements based on the HEXO paper.
2. Start working on merging certain C libraries from CheriBSD (or if possible rewriting the C libraries in Rust directly) with the RustyHermit kernel.

June 2023

1. Catch up on the pending tasks not completed listed above.

July 2023

1. Starting on writing completed research experiments to the thesis.

August 2023

1. Summer break

September 2023

1. Start modifying Popcorn Linux for building parts of a program to a TAG based architecture.
2. Start looking into ways to find out which parts of a program should be executed on a TAG based architecture (todo reference the paper from popcorn linux on enclaves).
3. Start drafting proposals that could be used to potentially take the above features of popcorn linux and make a clone of base features which can be used in the GHC Haskell compiler.

October 2023

1. Continue work on implementing Cheri with popcorn linux using Uni-kernels.
2. Start building a test framework to test Cheri with popcorn linux.

November 2023

1. Reiterate through the literature review and add more background context based on the implementation and experiments completed.
2. Start working on proposal drafted for adding popcorn linux features to the Haskell GHC compiler.

December 2023

1. Catch up on pending tasks.
2. Create benchmark suite for the experiments conducted throughout the year.
3. Christmas and new year break.

January 2024

1. Starting writing a conference paper which combines:
   (a) Multi-kernel approach with a functional language such as Haskell.
   (b) With a scheduler such from the HEXO paper with modification to run on TAG based architecture.

February 2024

1. Continue work on the conference paper and complete the draft by the month end.

March 2024

1. PhD writing period begin.

September 2024

1. PhD writing period end and Phd thesis draft ready.

Chapter 6

Conclusion

References

[Aze] Azalea-Unikernel: Unikernel into Multi-kernel Operating System for Manycore Systems.

[Nan] The Book — Nanos.org.

[Uni] Unikernels - Rethinking Cloud Infrastructure.

[4] (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].

[5] (2022). Multikernel. Page Version ID: 1109168202.

[6] (2022). TAG: Tagged Architecture Guide | ACM Computing Surveys — dl.acm.org. https://dl.acm.org/doi/abs/10.1145/3533704. [Accessed 20-Oct-2022].

[7] (2022). Unikernel and Immutable Infrastructures. original-date: 2018-05-11T13:37:54Z.

[8] Bailey, D., Barszcz, E., Dagum, L., and Simon, H. (1993). NAS parallel benchmark results. *IEEE Parallel & Distributed Technology: Systems & Applications*, 1(1):43–51. Conference Name: IEEE Parallel & Distributed Technology: Systems & Applications.

[Barbalace et al.] Barbalace, A., Ravindran, B., and Katz, D. Popcorn: a replicated-kernel OS based on Linux. page 16.

[10] Baumann, A., Barham, P., Dagand, P.-E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A., and Singhania, A. (2009). The multikernel: a new OS architecture for scalable multicore systems. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles - SOSP '09*, page 29, Big Sky, Montana, USA. ACM Press.

[11] Bratterud, A., Walla, A.-A., Haugerud, H., Engelstad, P. E., and Begnum, K. (2015). IncludeOS: A Minimal, Resource Efficient Unikernel for Cloud Services.

[12] Cha, S.-J., Jeon, S. H., Jeong, Y. J., Kim, J. M., and Jung, S. (2021). OS noise Analysis on Azalea-unikernel. In *2021 23rd International Conference on Advanced Communication Technology (ICACT)*, pages 81–84. ISSN: 1738-9445.

[13] Cheon, J., Kim, Y., Hur, T., Byun, S., and Woo, G. (2020). An analysis of haskell parallel programming model in the halvm. *Journal of Physics: Conference Series*, 1566:012070.

[Devietti et al.] Devietti, J., Blundell, C., Martin, M. M. K., and Zdancewic, S. HardBound: Architectural Support for Spatial Safety of the C Programming Language. page 12.

[15] 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.

[16] Gidenstam, A., Papatriantafilou, M., and Tsigas, P. (2005). Allocating Memory in a Lock-Free Manner. In Brodal, G. S. and Leonardi, S., editors, *Algorithms – ESA 2005*, Lecture Notes in Computer Science, pages 329–342, Berlin, Heidelberg. Springer.

[17] Jeong, Y., Kim, J., Jeon, S., Cha, S.-J., Lee, Y., Woo, Y., and Jung, S. (2020). Azalea unikernel IO offload acceleration. pages 1377–1380.

[18] 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.

[Kivity et al.] Kivity, A., Laor, D., Costa, G., Enberg, P., Har'El, N., Marti, D., and Zolotarov, V. OSv— Optimizing the Operating System for Virtual Machines. page 13.

[20] Kuenzer, S., Bădoiu, V.-A., Lefeuvre, H., Santhanam, S., Jung, A., Gain, G., Soldani, C., Lupu, C., Teodorescu, S., Răducanu, C., Banu, C., Mathy, L., Deaconescu, R., Raiciu, C., and Huici, F. (2021). Unikraft: fast, specialized unikernels the easy way. In *Proceedings of the Sixteenth European Conference on Computer Systems*, EuroSys '21, pages 376–394, New York, NY, USA. Association for Computing Machinery.

[21] Kwon, A., Dhawan, U., Smith, J. M., Knight, T. F., and DeHon, A. (2013). Low-fat pointers: Compact encoding and efficient gate-level implementation of fat pointers for spatial safety and capability-based security. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, CCS '13, page 721–732, New York, NY, USA. Association for Computing Machinery.

[Lackorzynski] Lackorzynski, A. L4Linux Porting Optimizations. page 58.

[23] Lankes, S., Breitbart, J., and Pickartz, S. (2019). Exploring rust for unikernel development. In *Proceedings of the 10th Workshop on Programming Languages and Operating Systems*, PLOS'19, page 8–15, New York, NY, USA. Association for Computing Machinery.

[24] Lankes, S., Pickartz, S., and Breitbart, J. (2016). HermitCore: A Unikernel for Extreme Scale Computing. In *Proceedings of the 6th International Workshop on Runtime and Operating Systems for Supercomputers*, ROSS '16, pages 1–8, New York, NY, USA. Association for Computing Machinery.

[Madhavapeddy et al.] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S., and Crowcroft, J. Unikernels: Library Operating Systems for the Cloud. page 12.

[26] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S., and Crowcroft, J. (2013). Unikernels: library operating systems for the cloud. *ACM SIGARCH Computer Architecture News*, 41(1):461–472.

[Marheine] Marheine, P. H. RKOS: UNIKERNEL DESIGN FOR SAFETY AND PERFORMANCE. page 71.

[Martins et al.] Martins, J., Ahmed, M., Raiciu, C., Olteanu, V., Honda, M., and Huici, F. ClickOS and the Art of Network Function Virtualization. page 16.

[29] Olivier, P., Chiba, D., Lankes, S., Min, C., and Ravindran, B. (2019). A binary-compatible unikernel. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments - VEE 2019*, pages 59–73, Providence, RI, USA. ACM Press.

[30] 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.

[31] Porter, D. E., Boyd-Wickizer, S., Howell, J., Olinsky, R., and Hunt, G. C. (2011). Rethinking the library OS from the top down. *ACM SIGPLAN Notices*, 46(3):291–304.

[32] 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.

[ScyllaDB] ScyllaDB. OSv Unikernel — Optimizing Guest OS to Run Stateless and Serverless A. . . .

[34] Selvacoumar, A. (2022a). Benchmarking Parallelism in Unikernels. original-date: 2022-12-14T20:31:10Z.

[35] Selvacoumar, A. (2022b). Benchmarking Parallelism in Unikernels. original-date: 2022-12-14T20:31:10Z.

[36] Selvacoumar, A. (2022c). PhD Activity. original-date: 2022-12-14T20:31:10Z.

[37] 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.

[38] Sung, M., Olivier, P., Lankes, S., and Ravindran, B. (2020). Intra-unikernel isolation with intel memory protection keys. In *Proceedings of the 16th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments*, VEE '20, page 143–156, New York, NY, USA. Association for Computing Machinery.

[39] 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.

[40] 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.