Stramash: A Fused-Kernel Design Operating System for Cache-Coherent, Heterogeneous-ISA Platforms

Tong Xing
The University of Edinburgh
Edinburgh, UK
tong.xing@ed.ac.uk

Cong Xiong[*]
Imperial College London
London, UK
cong.xiong24@imperial.ac.uk

Tianrui Wei
UC Berkeley
Berkeley, USA
tianruiwei@berkeley.edu

April Sanchez[†]
Google
Sunnyvale, USA
aprilsanchez@ucsb.edu

Binoy Ravindran
Virginia Tech
Blacksburg, USA
binoy@vt.edu

Jonathan Balkind
UC Santa Barbara
Santa Barbara, USA
jbalkind@ucsb.edu

Antonio Barbalace
The University of Edinburgh
Edinburgh, UK
antonio.barbalace@ed.ac.uk

Abstract

We live in the world of heterogeneous computing. With specialised elements reaching all aspects of our computer systems and their prevalence only growing, we must act to rein in their inherent complexity. One area that has seen significantly less investment in terms of development is heterogeneous-ISA systems, specifically because of complexity. To date, heterogeneous-ISA processors have required significant software overheads, workarounds, and coordination layers, making the development of more advanced software hard, and motivating little further development of more advanced hardware.

In this paper, we take a fused approach to heterogeneity, and introduce a new operating system (OS) design, the fused-kernel OS, which goes beyond the multiple-kernel OS design, exploiting cache-coherent shared memory among heterogeneous-ISA CPUs as a first principle – introducing a set of new OS kernel mechanisms. We built a prototype fused-kernel OS, Stramash-Linux, to demonstrate the applicability of our design to monolithic OS kernels. We profile Stramash OS components on real hardware but tested them on an architectural simulator – Stramash-QEMU, which we design and build. Our evaluation begins by validating the accuracy of our simulator, achieving an average of less than 4% errors. We then perform a direct comparison between our fused-kernel OS and state-of-the-art multiple-kernel OS designs. Results demonstrate speedups of up to 2.1× on NPB benchmarks. Further, we provide an in-depth analysis of the differences and trade-offs between fused-kernel and multiple-kernel OS designs.

[*]Cong Xiong worked on this project when at The University of Edinburgh
[†]April Sanchez worked on this project when at UC Santa Barbara

CCS Concepts: • **Computer systems organization → Heterogeneous (hybrid) systems**; • **Software and its engineering → Operating systems**; **Simulator / interpreter**.

Keywords: Operating Systems, Multiple-Kernel, Fused-kernel, Simulation, Heterogeneous, Cache Coherent, Shared Memory

ACM Reference Format:
Tong Xing, Cong Xiong, Tianrui Wei, April Sanchez, Binoy Ravindran, Jonathan Balkind, and Antonio Barbalace. 2025. Stramash: A Fused-Kernel Design Operating System for Cache-Coherent, Heterogeneous-ISA Platforms. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '25), March 30-April 3, 2025, Rotterdam, Netherlands.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3676641.3716275

1 Introduction

Heterogeneous-ISA systems have garnered significant interest in recent years by providing a "lighter" and more programmable approach to heterogeneity than the adoption of accelerators, making it possible to run existing general-purpose code with greater efficiency than running with a single instruction set [7, 8, 26, 60]. However, building and running these systems presents inherent challenges due to the various fundamental mismatches between hardware and software that come with integrating different ISAs. To date, commercially available platforms focused on loosely-coupled systems without hardware cache coherence, for example, those based around PCIe. As a result, heterogeneity in ISA is still considered a liability rather than an opportunity, with its introduction leading to overheads and system-level complexity rather than an introduction of ISA-generic capabilities that provide for greater efficiency. Instead, today's heterogeneous-ISA platforms run a separate software stack per CPU [9, 58]. Applications running thereon are perceived to be in a distributed system and hence cannot leverage per-platform optimizations.

New HW landscape. The hardware landscape is rapidly changing. There are several PCIe extensions that consider cache-coherent shared memory (including CXL [25], OpenCAPI [48] and CCIX [20]), making tighter interconnection of heterogeneous processing units an inevitable trend. Additionally, hardware research platforms have begun the move toward greater integration, providing cache-coherent heterogeneous-ISA processor designs [7, 36] or environments where such processors could be prototyped [2], but rather than offering rapid prototyping, their FPGA/ASIC orientation requires long development cycles. As such, these solutions have left open the question of what their accompanying software systems should look like. While we have a number of research operating systems to adopt strategies from [11, 13, 15, 17, 22, 38, 43, 47], none of these had the capability of leverage on cache-coherent shared memory, which we argue will make possible a large suite of performance improvements. Indeed, there exist commercial SoCs with some levels of heterogeneity, such as Intel Alder Lake or Arm big.LITTLE. Still, their heterogeneity is more focused on power efficiency than instruction semantics – single-ISA heterogeneity. In fact, such SoCs simply run traditional OSes for homogeneous-ISA multicores, e.g., vanilla Linux.

A New OS Design. With the emergence of platforms with cache-coherent shared memory amongst heterogeneous-ISA processors, sketched in Figure 1, which sound like shared memory multiprocessor systems but with heterogeneous cores, a naïve question is if also those can run classic SMP OSes? Classic SMP OSes are compiled to run on CPUs of the same ISA, so they cannot run amongst heterogeneous-ISA CPUs. Multiple-kernel OSes address ISA-heterogeneity, but existing designs are shared-nothing – i.e., designed to avoid using shared memory between kernel instances, for scalability [15], for heterogeneity, or because cache coherency simply did not exist before [11]. With the introduction of cache-coherent shared memory, a new OS design that exploits it together with heterogeneous-ISA is sought: the fused-kernel OS. Note we do not claim that heterogeneous-ISA platforms provide better efficiency, we recognize that their effectiveness depends on application- and platform-specific factors, which have yet to be fully demonstrated. Instead, our primary goal is to enable these emerging platforms to run applications as efficiently as possible by minimizing OS-related overheads.

We implemented the fused-kernel OS design atop Linux, in Stramash-Linux, starting from an academic multiple-kernel OS, Popcorn-Linux [11], to demonstrate the feasibility of our design and its applicability to traditional monolithic OSes. While a multiple-kernel OS enables applications to share state, a fused-kernel OS enables applications and kernel code to share (some or all) state among different kernel instances [12].

Stramash-Linux. Stramash-Linux is designed to fully exploit cache-coherent shared memory with the fused-kernel OS design. First and foremost, we nearly eliminate inter-kernel messaging, preferring higher-performance communication via cache-coherent shared memory, including sharing OS' data structures among kernel instances. Based on this, we introduce approaches to make OS services on different kernel instances share data effectively, including locking, because shared data cannot always be shared as-is but may need to be in architecture-dependent formats, e.g. page table. While shared data access is the core of the fused-kernel OS design, prototyping Stramash-Linux within the Linux kernel required the introduction of new mechanisms to reduce OS overhead.

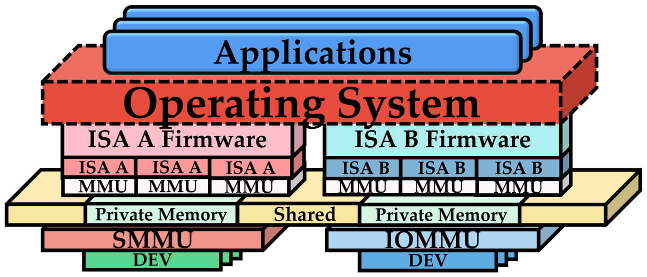

Figure 1. Stramash's target hardware and software model

Hardware Simulator. While we await the envisioned emerging hardware and academic prototypes that remain ill-suited to OS development, we implemented our own hardware simulator, Stramash-QEMU, by extending the industry-hardened QEMU with a cache simulator. By staying in the software realm, Stramash provides for high-speed prototyping exploiting widely-adopted ISAs like x86 and Arm, in contrast to prior hardware-oriented prototyping platforms like BYOC [7] which rely on outdated or less-adopted ISAs.

Stramash-QEMU "fuses" together multiple QEMU instances with a memory system simulator as the first in a new class of coherent, heterogeneous-ISA simulators. The software flexibility provided by Stramash-QEMU enables the investigation of architecture- and platform-level changes of particular interest to architects, as well as research into operating systems. In this paper, we introduce our fused approach to shared memory and interrupt delivery, among others.

Key Results. With our fused-kernel OS, Stramash-Linux, we demonstrate up to 2.1× speedup over the state-of-the-art multiple-kernel OS Popcorn-Linux. We also validate that Stramash-QEMU can achieve performance measurements within 4% on average vs bare metal. Stramash-Linux and Stramash-QEMU source code can be found at https://github.com/systems-nuts/Stramash-AE/

Contributions. Briefly, our contributions include:

- The fused-kernel OS design, a shared-mostly multiple-kernel OS that minimises communication overhead among kernel instances, and a prototype of it based on Linux, Stramash-Linux;
- The fused-simulator design, combining single-ISA simulators into a single multi-ISA simulation platform providing cache-coherent shared memory, and its implementation based on QEMU, Stramash-QEMU;
- The validation of Stramash-QEMU using two physical AArch64 plus x86-64 platforms;
- A quantitative comparison of Stramash-Linux versus Popcorn-Linux on Stramash-QEMU.

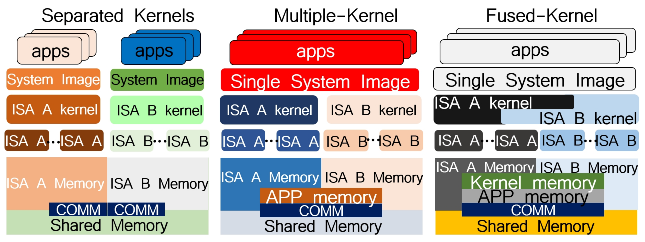

Figure 2. Comparison of OS designs for heterogeneous-ISA cache-coherent shared memory platforms.

Limitations and Future Work. This paper focuses on a new OS design, its implementation into a state-of-the-practice OS (Linux), and the supporting simulation environment. We deliberately do not study the scalability of this approach – which is not possible to do efficiently today with a software simulator alone. Currently, we only target x86-64 and AArch64. Extending that to other 64-bit ISAs is mainly an engineering effort, due to the same bit-width. However, targeting diverse bit-width ISAs requires further research, which is beyond our current scope. Likewise, Popcorn's compilers [44, 49] have only been used for pairs of ISAs of the same bit-width so far.

Moreover, security and fault-tolerance, while part of our design (see below), are future work. Indeed, security and fault-tolerance related specifications may exist, like CXL's IDE and RAS, but we believe it is not worth speculating on the actual security or failure behaviors without a hardware prototype.

2 Background & Related Work

There are a variety of architectural studies which have shown performance, efficiency, and security benefits to adopting heterogeneous-ISA system designs [14, 55, 59, 60]. In the systems realm, a number of new operating system [15, 17, 21, 38, 47] designs have been proposed that leverage existing (or minimally modified) hardware to exploit the architectural benefits and enhance system performance.

Systems Software. Heterogeneous-ISA platforms with general purpose-CPUs, with or without shared memory, are not new [7, 57, 58, 63], e.g., an x86 host CPU featuring a Xeon Phi, or Arm-based smartNIC/SSD. The most common approach to run software on such platforms is to run a separate software stack – including at least the OS/runtime and applications, per island of homogeneous-ISA CPUs, "separated kernel" in Figure 2. Applications must be rewritten to run in a distributed (or offloaded) system where communication happens explicitly via message passing. This approach cannot exploit cache-coherent shared memory even when available.

To more effectively support application execution among heterogeneous-ISA cores, new systems software is needed. Applications must be compiled to support live execution migration, and a runtime or OS is needed to provide the same execution environment on source and destination CPUs. Both runtime and OS solutions have been proposed, in this paper we focus on the OS ones and not on runtime ones like H-Container [10, 62] (C-based), and PadMig [31] (Java-based).

In the last decade, several works proposed new OS architectures to provide the abstraction of a single system among different single-ISA or multiple-ISA heterogeneous CPU platforms, including Helios [47], Barrelfish [15], K2 [38], Popcorn [11], and Flick [21]. Applications running atop these enjoy the same OS interface and services among diverse CPUs, and applications may either spawn threads or migrate threads to a CPU of a different ISA. All works we are aware of achieve this by using multiple, communicating kernel instances, where CPUs of different ISAs run different OS kernel instances, "multiple-kernel" in Figure 2.

Different from SMP OSes, where a single kernel instance runs among all CPU cores, which share all kernel data structures (shared everything [16]), multiple-kernel OSes tend to either do not share anything (shared nothing) [11, 15, 21, 47], or share a few kernel data structures (shared something) [37]. Note that shared nothing multiple kernels do share memory as an optimisation [15, 35]. Anyhow, the level of data sharing in kernel space is unrelated to the level of sharing in user space. In fact, shared-nothing multiple kernels do provide applications with consistent shared memory when applications run amongst kernel instances, either by distributed shared memory (DSM) [11, 53] or using hardware remapping [21].

Two main factors lead the evolution of multiple-kernel OSes to shared nothing/something. Firstly, the potentially poor scalability of cache-coherent shared memory interconnects [15], or their high power draw [38]. Secondly, the absence of hardware with heterogeneous-ISA CPU cores on cache-coherent shared memory, which hindered further OS research, making our work the first of its kind. Previous projects leverage either non-coherent domains [11, 17, 38], or domains connected via high-latency buses with snooping [21].

Prototyping Environments. Early heterogeneous-ISA system prototypes have largely exploited existing hardware. A number of systems used existing x86 server hardware with PCIe non-transparent bridges to connect processors of other ISAs [11, 17]. Others exploited existing systems-on-chip which provided limited heterogeneous-ISA capability (e.g., Arm+Thumb) [38]. While these platforms enable full-speed execution of proposed OS designs, they do not feature cache-coherent memory, necessitating software consistency layers that degrade performance and increase OS complexity – particularly for research prototypes building on top of Linux.

Recently, BYOC introduced a cache coherent heterogeneous-ISA prototyping environment [7]. BYOC uses FPGA emulation to provide high-speed prototyping which could eventually be realised in silicon. The system supports several ISAs (SPARC v9, RISC-V, i486), but does not feature 64 bit x86 or Arm ISAs, which would be of most interest to OS developers and users. Further, prototyping an architectural change requires a full, FPGA-ready implementation, which is a high bar to entry. Towards the end of this work, AMD announced an

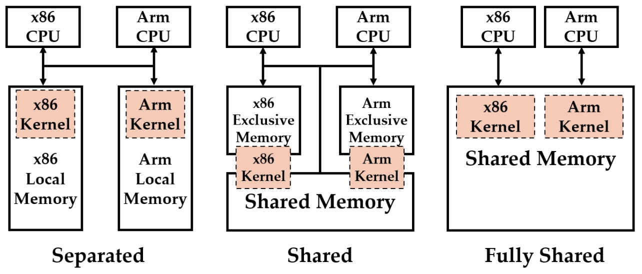

Figure 3. Memory configurations and OS kernel placements.

Embedded series of SoC that has Ryzen and Xilinx FPGA [4], where the FPGA could be programmed as RISC-V; SOPHON released SG200X SoCs featuring RISC-V and Arm cores [56]. Both platforms featuring CC shared memory between cores.

We take a different approach to prototyping by fusing multiple instances of QEMU with an extension of the Cache plugin memory system simulator. This enables OS, architecture, and platform prototyping in software at a relatively high speed without the need for full hardware implementation. With QEMU, we can also exploit proprietary ISAs like 64-bit x86 and Arm. Our work is the first we are aware of to fuse multiple QEMU instances of different ISAs together with shared memory backed by an architectural cache simulator.

3 Hardware Model

Stramash targets emerging platforms with cache-coherent heterogeneous-ISA CPUs integrated into an SoC (like BYOC or SOPHON's SG200X) or interconnected via emerging cache-coherent buses (like CXL). Hence, we considered at least 3 memory hardware configurations: Separated, Shared, and Fully Shared, depicted in Figure 3. In the Separated configuration, each CPU group has its own memory, with coherence managed by a Last Level Cache (LLC), like in NUMA. In the Shared configuration, each CPU group has access to a private memory, and all CPUs can access a cache-coherent shared memory, like CXL 3.0 [23]. In the Fully Shared configuration, there is a single, shared memory for all processors, although it may be mapped to different addresses, like the academic research project OpenPiton [8]. Each processor is capable of sending Inter Processor Interrupts (IPI) to any other processor in the system. All MMIO devices are accessible by all processors. Regarding memory consistency, we assume all processors abide by the strongest memory consistency model of all ISAs (Arm already supports running in TSO mode). This assumption is justified by standards like CXL 3.0 that add a MESI cache coherency protocol for inter-host shared memory – where hosts can be of any ISA. Systems supporting heterogeneous consistency models could take advantage of tools like ArMOR [40] to defend against consistency model mismatches.

Stramash leverages cache coherent shared memory for communication, and IPI for notification. Additionally, for inter-CPU-group communication, Stramash may rely on custom communication devices (e.g., hardware FIFOs or spinlocks [57]) in embedded SoCs – not in this paper, or the use of built-in switching functionality in an Ethernet NIC [18, 45], i.e., sending network packets between ISAs – used herein.

4 Design Principles

With the goal of enabling existing applications to exploit cache coherent heterogeneous-ISA CPUs platforms with maximum performance (with minimal OS/runtime overheads) the fused-OS is based on the following design principles:

- Run applications as-is, or with minimal modifications, to support legacy;
- Target traditional widely-used OS design – monolithic OSes, to attract a large user community, while remaining applicable to other OS designs;
- Generality, to support a wide variety of ISAs, and diverse heterogeneous-memory configurations;
- Minimize data movement; hence, avoid copies across ISA boundaries as much as possible;
- Security, to ensure that the security of each kernel instance is not undermined by the kernel-level sharing.

For our architecture simulator – developed to validate our prototype fused-kernel OS, similar principles apply:

- Exploit, and eventually extend, existing projects, reusing designs and APIs;
- Based on traditional widely-used simulators/emulators to attract a larger user community;
- Fast prototyping, easy configuration – while supporting different memory hierarchies;
- Fast execution speed, with ability to enable a cycle approximate memory model;
- Accurate, to reflect the different ISAs' memory models and their interactions (e.g., for atomic instructions).

5 Fused-kernel Operating Systems Design

We introduce a "fused approach" to multiple-kernel operating system design, pulling together – fusing – OS services running on different kernel instances. This means that (some of) the data structures of one kernel instance can be accessed by the other kernel instance(s) directly via shared memory. Hence, enabling multiple kernel instances to work as one, reducing OS service latency and improving execution speed.

Design and Methodology. We introduce a new OS design, the fused-kernel OS, rightmost design in Figure 2. Differently from previous multiple-kernel OSes, which are based on the shared nothing principle, fused-kernel OSes are based on the **principle of shared-mostly** – i.e., different kernel instances do communicate using shared memory or share a single state in shared memory. Thus, avoiding prior work's communication overheads including serialization and deserialization of data structures, coordination protocols, message copying, etc. Such design enables tight coordination between kernels to share hardware and software resources.

OS services in a fused-kernel OS are either: **local** or **global**. Local OS services do not require any communication between kernel instances, while global do. Global OS services are mainly built upon shared memory, but when that is too complicated, inter-kernel message-passing is used, like in a multiple-kernel. As shown in previous literature, message-passing requires an OS service to be rewritten as a distributed service, i.e., using coordination protocols. When using shared memory, we introduce two OS service architectures:

- one in which an OS service is rewritten to adopt a **common data format** in shared memory amongst kernels;
- one in which each kernel instance keeps **its own data format**, but the others use *accessor functions* to read/write the original data, including locks.

The latter is necessary when handling architecture-dependent data, like page tables, while the first fits services that are architecture independent. A collection of accessor functions targeting a specific ISA makes up a remote CPU driver.

Based on our design principles and the way OS services communicate, our fused-kernel design further introduces the following architectural choices described below.

Minimal Resource Provisioning. To quickly provide global resources, each kernel instance fully utilizes its own private hardware resources (e.g., memory) when available, and acquires any other shared resource only when needed, returning resources to global allocators when no longer needed.

This is in stark contrast to traditional OSes, which discover and initialize all resources available on a machine at boot, ready for later allocation. In our design, while all resources are discovered and initialized at boot, shared resources are maintained in a global pool before being assigned to a kernel instance – such as a CXL shared memory pool. Thus, at boot time a kernel instance is given a minimal amount of resources.

Interrupts and Inter-kernel Notification. In most cases all hardware resources are globally accessible, and each kernel instance knows about those – including memory and devices, but based on the previous architectural choice kernels are only provided with needed resources. Despite that, each kernel always maps all interrupts, especially IPIs, for notification.

Single virtual address space. To simplify the development of *accessor functions*, and reduce their pointer arithmetic overheads, we introduce a single kernel-level virtual address space among kernel instances – which still allows for part of the kernel-level address space to be private.

Minimal/secure kernel-level data sharing. As heterogeneous SoCs are gaining traction, the number of security issues has been rising steadily [14, 55, 59]; e.g., a simple defect in a wifi chip could enable remote code execution on Android [30]. Thus, although not the primary focus of this paper, a fused-kernel OS needs to consider the security aspects when running on shared memory on different heterogeneous-ISA cores.

Specifically, we postulate that kernel instances should share only required data structures. Everything else should be in private memory or protected by hardware enforcement, like MPU, MMU, IOMMU, and hardware capabilities. To make hardware protection effective, we also propose to pack data structures' data in contiguous physical memory – so it is simple to categorize and share between kernels.

Applications' Compiler and Linker. Applications must be compiled in a way that makes them amenable to migration, such that they can continue executing on another ISA-CPU carrying over the existing application state minus the CPU-state that is converted. Inter-kernel thread migration is offered as an OS service and includes functionalities to show the same application state on different kernel instances. Inter-kernel process migration is simpler because there is no kernel state to be kept consistent after migration. In this work, we reuse the open-source Popcorn-Linux Compiler Toolchain [49] to compile applications to run on Stramash OS. We direct interested readers to such project literature [41, 42].

6 Stramash-Linux Implementation

We implemented a prototype of our fused-kernel OS, called Stramash-Linux, based on the Linux kernel 5.2.12. To avoid reinventing the wheel, we adopted OS components from the open-source Popcorn-Linux project, including process/thread migration and the messaging layer. To build Stramash-Linux we contributed around 9200 LoC atop Popcorn-Linux. We chose the Popcorn-Linux project because it is the only one providing an open-source, fully-functioning OS and compiler toolchain, which actually produces executable binaries that can migrate across heterogeneous-ISA CPUs.

Prototype Limitations. Because we based our work on the Popcorn project, which fully supports only the x86 and Arm ISAs, our Stramash prototype inherits the same limitation. However, we believe that our design applies to other ISA mixtures as well, including different bit-widths and endianesses. x86-64 and AArch64 are widely adopted, specifically on emerging platforms targeted by this paper. Hence, other ISA mixtures are out of the scope of this paper.

While we did implement support for data packing in contiguous physical memory – including moving pages to reorganize data, we did not find an efficient method to limit the kernel-space remotely accessible memory between ISAs. Capabilities being a potential candidate, but it is future work.

6.1 Booting Kernels

Within the hardware model proposed, heterogeneous ISA cores would have access to the entire shared memory, devices, etc. Therefore, like a traditional OS, Stramash-Linux will discover all memory and devices, but initialize only a minimal set of those to enable a working system; the rest of the resources are managed by global pools. At the time of writing, we limit the area usable by each kernel instance using BIOS tables/device trees. The OS reads the memory map tables provided by the firmware and adjusts its boundaries based on that. Thus, kernel instances' memory areas do not overlap (see Figure 4). Once the boot is complete, kernel instances establish a communication channel to coordinate and bring up all OS services to share resources in a fused manner.

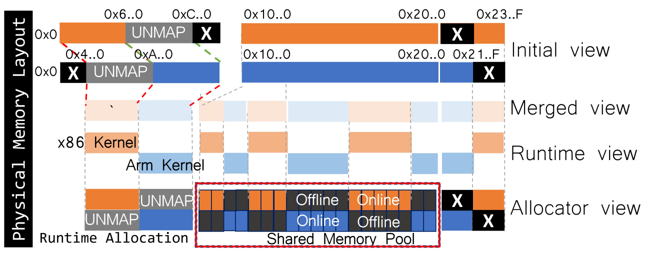

Figure 4. Physical memory layout. x86 instance starts at `0x0`; Arm instance starts at `0xA0000000`; Shared Memory Pool from `0x100000000` to `0x200000000`.

6.2 Message-passing Communication

Stramash-Linux uses a messaging layer to communicate between kernel instances. This is based on one or more pairs of shared memory ring buffers per kernel pair, to exploit shared memory to minimize latency and payload cost. After a message has been enqueued on a ring buffer, the messaging layer sends a cross-ISA IPI to the receiver core group. We also support polling in place of interrupt dispatching.

6.3 Global Memory Allocator

Stramash-Linux implements a global memory allocator that manages physical memory among kernel instances. The current memory allocator exploits and extends the memory Hot-plug subsystem, with several modifications. Different from Hot-plug, Stramash-Linux does not require the memory block to be unplugged. Instead, for hot removal, it first evacuates the memory block and then isolates the pages. We picked this solution after testing different options, including the continuous memory allocator (CMA) as in K2 [38], all required major modifications to the Linux source code.

The Memory Allocator. Figure 4 shows an example of memory layout in Stramash on an Arm+x86 configuration with 8GB of RAM. We have implemented a fixed-size global memory allocator with a configurable block size (from 32MB to 4GB). We opted for a minimum size of 32MB to reduce the overhead associated with frequent memory assignments. Each kernel starts with a fixed amount of blocks. When the total memory pressure on a certain kernel instance passes 70%, that kernel requests an additional block from the global memory allocator. If a block is free, it is directly assigned. If there are no free blocks, the allocator will try to evict a block from the other kernels until it also reaches the same memory pressure.

6.4 Fused Virtual Address Space

Stramash-Linux supports user-space process/thread migration across ISAs. Unlike Popcorn-Linux, which uses software DSM to provide a single application virtual address space among kernels – passing memory pages as messages, Stramash-Linux leverages cache-coherent shared memory to maintain a page table per kernel instance due to the differing page table format in Arm and x86. Both page tables refer to the same physical memory pages for the same application.

Fused Kernel Virtual Address Space. Stramash-Linux aligns kernel virtual addresses across different kernel instances, enabling full addressability of another kernel's memory. By adjusting the vmalloc ranges of x86 to align with the direct map range of the Arm instance, the Arm's virtual address space becomes fully addressable to the x86 kernel instance, and vice versa. We refer to this configuration as the *Fused Kernel Virtual Address Space.* This distinctive capability distinguishes Stramash from other multi-kernel operating systems by allowing it to share kernel virtual addresses and data structures seamlessly, enhancing interactions without redundancy.

It is worth noting that for some data structures, today we need to disable the randomized layout to enable direct remote access. If the layout of shared data structures varies, some (simple) handling is needed. However, this is typically not a concern, as few data structures vary across ISAs.

Software Remote Page Table Walker. Stramash-Linux allows one kernel to access the page table of the other kernel through a cross-ISA software page walk to reduce long-latency round-trips message passing overhead. Currently, both x86 and Arm in Stramash-Linux are using 5-level page tables. To acquire the page table entry (PTE) in the origin kernel, the remote kernel has to walk through those tables with proper page masks. Each level page mask is re-defined if it is different between x86 and Arm.

Software Remote VMA Walker. Unlike Popcorn-Linux, where a VMA fault triggers a message exchange to the original kernel, in Stramash-Linux, each kernel can access the other kernel's VMA lists, with appropriate VMA locks acquired. Note that in our Linux kernel, the VMA lists are still maintained using the RB-tree structure not a Maple-tree.

Stramash Page Fault Handler. In the current version of Popcorn-Linux, anonymous pages are allocated in the origin kernel – where the application starts, which introduces at least 2 rounds of message passing, i.e., the request/response of the page allocation and replication. In Stramash-Linux, the remote kernel allocates anonymous pages without needing to notify the origin kernel. The remote kernel first allocates a page, inserts it into its own page table, and adds it to the origin kernel's page table with the remote node ISA format. Once the process migrates back to the origin kernel, the origin kernel can simply reconfigure the PTE to its own format and access the page via cache-coherent shared memory. When a process is terminated, the origin kernel only invalidates the PTE and does not attempt to release the page, as it was allocated by the remote kernel. The remote kernel, on the other hand, takes responsibility for invalidating its own PTE and releasing the page, thus finalizing the memory recycling. To manage simultaneous access, we have implemented a cross-ISA page table lock (`Stramash-PTL`), ensuring that only one instance can modify the page table at a time. This method effectively bypasses the reliance on the Copy-On-Write (COW) policy heavily utilized in Popcorn-Linux. In contrast to Popcorn, where the COW policy minimizes messaging during frequent page updates, Stramash-Linux allows direct page access without such overhead.

6.5 Cross-ISA locking

Atomicity. Stramash-Linux's AArch64 kernel includes support for Large System Extensions (LSE) [5], which provides a non-interruptible read-modify-write sequence in a single instruction, Compare-and-Swap (CAS). These atomic instructions can replace Load-Link/Store-Conditional (LL/SC) operations. Stramash ensures that all kernel spinlock-related instructions use the CAS instruction, providing a more efficient and robust mechanism for handling cross-ISA locks. Additionally, with the integration of Stramash-QEMU, detailed in Section 7.1, operations involving atomic instructions maintain their integrity cross-ISAs.

Futex. Popcorn-Linux manages Futex (fast userspace mutex) operations by relying on the origin kernel to create and control all Futex instances. When a lock is requested, the remote kernel must message the origin kernel to engage the lock, and all subsequent locking actions are maintained by the origin kernel. In contrast, Stramash-Linux allows the remote kernel to directly access the Futex locking list. This reduces dependency on the origin kernel for locking operations. Upon unlocking, if the thread is currently waiting in the origin kernel, the remote kernel sends a cross-ISA IPI to the origin kernel to wake up the thread.

6.6 Fused Namespace

For applications that migrate inter-ISA, Stramash-Linux enables the same mount, PID, net, UTS, user, and cgroup namespaces. These provide the same environment when an application migrates. Also, the same list of CPUs including topological information is available on every kernel instance.

7 Stramash Hardware Simulator

A hardware simulator is necessary to thoroughly evaluate our Fused-kernel OS against the state-of-the-art. Inspired by the Fused-kernel OS, our fused-simulator connects multiple traditional single-ISA simulators as one and presents cache coherent shared memory to every simulated core. Stramash-QEMU also supports mechanisms for communication across ISAs other than shared memory, including IPIs, memory remapping, private memory, device sharing, parallel bootup, and cache simulator. Stramash-QEMU targets emerging platforms with cache-coherent shared memory, focusing primarily on memory system modeling. It is worth noting that Stramash-QEMU is generic enough to be reconfigured for different hardware models. We based our simulator on QEMU 8.0.0 to execute software on heterogeneous CPU cores, specifically AArch64 and x86-64. We have extended the current QEMU Cache plugin [51] to support a 3-level cache and CXL. Our contributions amount to about 7100 LoC to QEMU.

We employ system-level simulation to model all OS and application details. As the application runs, it accesses a main memory that is coherent across all simulator instances. With Cache simulation enabled, all memory accesses are forwarded to our Cache plugin, which provides detailed memory access overhead and feedback on latency to our timing model.

7.1 Pervasive Cache-coherent Shared Memory

In Stramash-QEMU, guest memory is allocated on a per-host basis. Any memory operation from a single guest will be reflected in others, respecting the rules of cache coherence. By running both Stramash-QEMU instances on the x86 host, the actual memory operations will follow the host's x86 TSO memory consistency model. Given that this is a stricter model, this setup ensures that both instances are protected from memory consistency issues. However, there is a potential complication where the x86 host translates the Arm guest's LL/SC instructions into the host's CAS instructions [24, 40]. Stramash-QEMU uses the Cortex-A76 as the Arm CPU core, which supports LSE and thus CAS instructions. We have carefully configured the QEMU tiny code generator (TCG) to ensure relevant Arm instructions are correctly translated.

7.2 Inter-ISA Interrupts

Interrupts are crucial for interactions between CPU cores and between cores and external devices. In a heterogeneous-ISA setting, different ISAs have their approaches to interrupt management. To take a fused approach to interrupt management and thus facilitate interrupt sharing across architectures, we have prototyped cross-ISA IPIs that enable native IPI communication between CPUs of different ISAs. We extended the AArch64 SGI and x86 APIC by adding extra logic to route the native IPI to the peripheral device, and then notify the other ISA to generate a native IPI. We assign an unused IRQ number in Linux and set up respective handlers for each ISA's kernel to handle the IPI.

7.3 Stramash Timebase

Stramash-QEMU introduces a cross-ISA timing model to coordinate simulated time. Our design resembles manycore simulators such as PriME [28], where the memory system becomes the primary factor, and core performance is, by default, modeled with a fixed non-memory IPC [29]. This allows us to implement our custom Stramash time base, which can quickly simulate cycle-accurate timing information.

Time in QEMU. QEMU is a functional simulator rather than a cycle-accurate simulator. While QEMU can provide the guest with emulated time, this measurement is rather basic and does not offer much insight into the performance of the emulated hardware. Additionally, QEMU's TCG backend supports instruction counting (icount), enabling the counting of executed instructions. Icount is a simple yet effective way to profile software. It is widely supported in tools such as perf, Valgrind, and Intel Pin [39]. We have configured QEMU to utilize an instruction count-based timing model, with time progressing according to instruction counting. This ensures alignment in speed between I/O and instruction emulation, preventing an emulated hardware device from running unrealistically fast. We have disabled the warp time feature of QEMU to eliminate any influence from the host's real-time adjustments.

Instruction Counting. Measuring programs' execution time in a heterogeneous-ISA platform is not as straightforward as in homogeneous-ISA platforms because the application can migrate between CPUs of diverse ISA at runtime. We have integrated our icount approach with Linux Perf to get an accurate measurement of the time that the application has actually executed. We use this approach in Stramash-QEMU validation, comparing our instruction count to the native instruction count on the physical machine, described in Section 9.1.2.

Timing Modelling. Although the alignment of the instruction count-based timing model ensures a minimum latency for each emulated instruction, it does not offer any performance modeling metrics. To address this limitation and enable system-level profiling, we have integrated a modified version of QEMU's Cache plugin. We have added a 3-level cache feature to model the performance of memory systems, which has been validated in comparison with the GEM5 MESI three-level cache memory model [33] in Section 9.1.3. Each memory instruction executed by QEMU passes to our Cache plugin which analyzes the cache behavior, whether it is a hit or miss, and accordingly adds memory access overhead to the icount. This information is then sent back to QEMU as feedback, enhancing the accuracy of the performance modeling metrics.

CXL Access Overhead Feedback. Stramash-QEMU Cache plugin simulates the latency of data transmission on the CXL bus. We model the overhead introduced by CXL to maintain coherence among replicas in the caches of various processors in heterogeneous systems. We consider the additional delays brought on by SNOOP messages and Responses, which play a crucial role in the invalidation and update processes, including Back-Invalidate Snoop, Snoop Invalidate, and Snoop Data [23]. When one processor attempts to write to the same memory location accessed by another, it issues a "Snoop Invalidate" request. This operation mandates that all other processors invalidate the corresponding cache lines they hold, ensuring coherence and preventing outdated data access. Conversely, if a processor intends to read from the same memory location already accessed by another, it triggers a "Snoop Data" request. This command converts the cache line's state from "Exclusive" to "Shared" in other processors.

7.4 IO Devices

We have enhanced QEMU such that when an instance lacks a particular device, it creates a memory mapping for that device. Consequently, all memory accesses are redirected to the QEMU instance containing the respective device.

8 Experimental Methodology

We evaluated Stramash on a Supermicro X11DPi-N(T) with dual Intel Xeon Gold 6230R CPU and 768GB of RAM. We set up Stramash-QEMU with the different hardware models described in Section 3 and compared Stramash-Linux to Popcorn-Linux. To the best of our knowledge, there are no other open-source projects that enable applications to run across different ISAs aside from Popcorn-Linux.

8.1 Stramash-QEMU Setup

Stramash-QEMU has been setup to simulate the three models in Figure 3, which we configure with different physical memory layouts for each QEMU instance. See Figure 4 as a reference. In the **Separated** model, the x86 instance has local memory ranges from 0x0 to 1.5GB, 4GB to 6GB, and the Arm instance has local memory ranges from 1.5GB to 3GB and 6GB to 8GB. In the **Shared** model, the memory range of 4GB to 8GB is considered remote memory for both x86 and Arm instances, and all other local memory ranges remain the same as in the **Separated** model. In the **Fully Shared** model, all memory ranges are considered local memory.

In the experiments, we use the memory latencies of the Xeon Gold and ThunderX2 pairs as shown in Table 2. When Stramash-QEMU encounters a memory operation to an address, it first checks if the address is in the cache and then it adds the corresponding cache latency. If the cache line is invalidated or missed, it reloads it from memory back into the cache. Based on different hardware models, the address could be in local memory or remote memory, and the corresponding latency is added accordingly. Therefore, in the **Shared** model, if an address is found in another cache, depending on whether it is a read or write operation, the Cache-Plugin will follow the appropriate MESI transition and add the simulated CXL snooping overhead. For example, if one node writes to a cache line, the other node will invalidate that cache line if it is in shared state through a simulated invalidation – a snoop invalidation overhead is added. The **Separated** model could be configured as NUMA or CXL; currently, we use the CXL snooping overhead to simulate the cost of cache coherence, but it can be set with the cost of Intel QPI[34] or AMD Infinity Fabric[3], etc. **Fully Shared** includes just one shared cache and memory. After such operations have been simulated, we add the corresponding overheads to the QEMU icount. We also actively maintain the same icount speed on both QEMU instances to ensure that both QEMU icounts increase and proceed at a similar rate.

8.2 OSes Setup

Local and remote physical memory ranges are fixed for both the x86 and Arm instances of each hardware model simulated by Stramash-QEMU. If the x86 instance acquires a physical memory range that belongs to the Arm's local memory, any memory operations to this physical memory from the x86 will incur additional remote memory access overhead, simulated by our Cache-Plugin.

For both Stramash-Linux and Popcorn-Linux, we provide a 128 MB shared memory area to serve as the message layer, which is configured as a ring buffer. Because Stramash-Linux can directly access each other's memory, depending on the memory model, the location of the message layer's memory area varies, and memory access latency is added accordingly.

Setup for Popcorn-Linux. Unlike Stramash-Linux, Popcorn-Linux kernel instances do not directly access each other's memory. We run two versions of Popcorn-Linux. The first, Popcorn-Linux Messaging over Shared Memory (**SHM**) employs a shared memory–based messaging layer. When either side accesses this shared memory, based on the corresponding hardware model, the Cache Plugin will add corresponding memory access latency. Meanwhile, all other physical memory is used exclusively by one kernel or the other. We set up **Popcorn SHM** on all 3 hardware models, **Separated-SHM**: the messaging layer is mapped at x86's local and Arm's remote memory; **Shared-SHM**: the messaging layer is mapped at both kernels' remote memory; **Fully Shared-SHM**: the messaging layer is mapped at both kernels' local memory. For Stramash-Linux and Popcorn-Linux messaging over shared memory, the IPI overhead is $2\mu s$, detailed in Section 9.1.1.

The second, Popcorn Linux Messaging over Network (**TCP**) where each kernel only accesses its local memory and communicates with other kernels via `TCP/IP`. Because this doesn't exploit shared memory, it performs the same independently of the hardware model. We add approximately $75\mu s$ delay for each message round trip to simulate network latency, which is the average latency of a 64KB (Default Popcorn-Linux message payload size [50]) packet round-trip time measured software-to-software on SmartNIC hardware [18].

8.3 Benchmarks

Amongst others, we extensively run workloads from the NAS Parallel Benchmark (NPB) [46] collecting execution time, instructions, and cycles number on bare-metal and on Stramash-QEMU – using our Linux perf. We selected NPB because it has different memory access patterns, including read and write intensive workloads.

Table 1. Machines for Popcorn-Linux baseline data collection

Name	Core	Hz	RAM
Small_Arm	Broadcom Armv8 A72 8 cores	3.0GHz	8GB
Big_Arm	Dual Cavium ThunderX2 CN9980 v2.2 (32 cores/128 threads)	2.0GHz	256GB
Small_x86	Xeon E5-2620 v4 (8 cores/16 threads)	2.1GHz	16GB
Big_x86	Dual Xeon Gold 6230R (26 cores/52 threads)	2.1GHz	768GB

Table 2. Stramash-QEMU Cache plugin Memory Operation Latency, CXL latency for remote memory[54]

Core	Operation	Latency(cycles)
Cortex-A72[27]	L1/L2/L3/mem/remote-mem	4/9/*/300/780
ThunderX2[32]	L1/L2/L3/mem/remote-mem	4/9/30/300/620
E5-2620[19]	L1/L2/L3/mem/remote-mem	4/12/38/300/640
Xeon Gold[61]	L1/L2/L3/mem/remote-mem	4/14/50/300/640

Table 3. Message Count During Migration and Replicate Page Count During Runtime Migration

	Message Count			Replicated Pages		
	Popcorn	**Stramash**	**Reduced Rate**	**Popcorn**	**Stramash**	**Reduced Rate**
IS	207124	22	99.98%	16918	7	99.96%
CG	16074	34	99.78%	5603	7	99.88%
MG	287672	6	99.99%	110275	9	99.99%
FT	164702	326	99.80%	98787	16459	83.34%

Table 4. Memory allocator overhead of performing offline and online operations with different memory slice sizes

	Qemu-x86		Qemu-Arm	
Num of Pages	**Offline**	**Online**	**Offline**	**Online**
2^{15}	12.5ms	5.8ms	4.8ms	5.8ms
2^{16}	27.7ms	10.9ms	16.1ms	12.3ms
2^{17}	38.6ms	14.3ms	14.3ms	16.7ms
2^{18}	66.6ms	22.6ms	21.1ms	28.8ms
2^{19}	129.4ms	36.5ms	36.4ms	42.6ms
2^{20}	246.3ms	68.1ms	64.4ms	80.9ms

9 Evaluation

9.1 Stramash-QEMU Validation

We validated the accuracy of our hardware simulator, Stramash QEMU, with bare-metal measurements on two reference platforms. Those are two pairs of Arm and x86 machines, small_Arm and small_x86, big_Arm and big_x86. The small_Arm, a smartNIC, and the small_x86, a low-spec server, are interconnected via PCIe bus where the smartNIC is plugged in. To run Popcorn-Linux on this setup we ported it to the Linux kernel 4.14.79 needed by the smartNIC. The big_Arm and big_x86, two high-spec servers, are running Popcorn-Linux kernel version 5.2.12. The servers are connected through 100Gbps Ethernet. In both cases, Popcorn-Linux exploits the `TCP/IP` messaging layer. Their technical details are shown in Table 1. We also compare Stramash-QEMU Cache-Plugin with Gem5 to assess the rigorousness of our cache model.

9.1.1 IPI Cost Characterisation. Stramash-QEMU integrates two or more QEMU instances, each emulating a different ISA processor complex. To the best of our knowledge, we are the first to enable QEMUs to exchange cross-ISA IPIs. In the absence of real hardware to measure our proposed cross-ISA IPI, the exact latency remains unknown. Therefore, we used the latency of cross-NUMA IPI as a placeholder for cross-ISA latency — a configurable parameter in Stramash. To obtain a realistic IPI overhead, we measured IPI latency on real machines.

We implemented a kernel module to measure IPI latency between all core pairs on both Arm and x86 machine sets with minimal overhead. The RDTSC instruction was used to capture timestamps, MONITOR/MWAIT to minimize measurement overheads. Figure 5 and 6 show the measurements. The average IPI latency is about $2\mu s$ in large machine pairs, and we have used this value as our simulated cross-ISA IPI cost.

9.1.2 Instruction Count. We use NPB benchmarks for the icount validation. We use the Stramash-QEMU perf+icount tool introduced in Section 7.3 to record the icount data from both kernel instances during the migration. By comparing the instruction-per-cycle (IPC) results from the (native) perf tool on a real machine, we estimate the IPC for Stramash. Since the

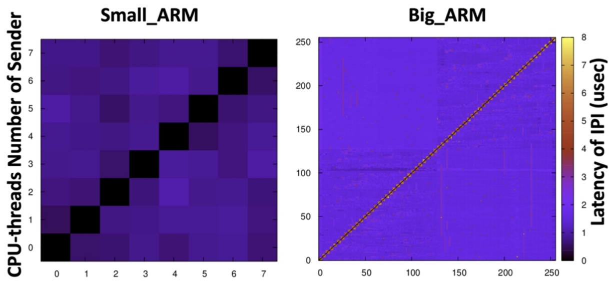

Figure 5. IPI latency results Arm

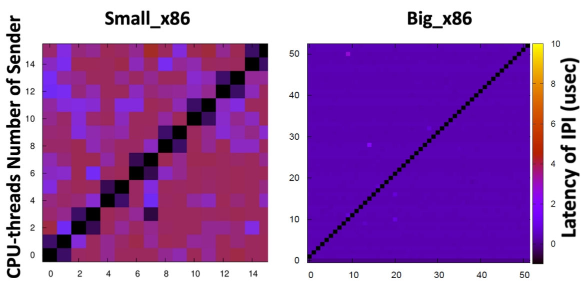

Figure 6. IPI latency results x86

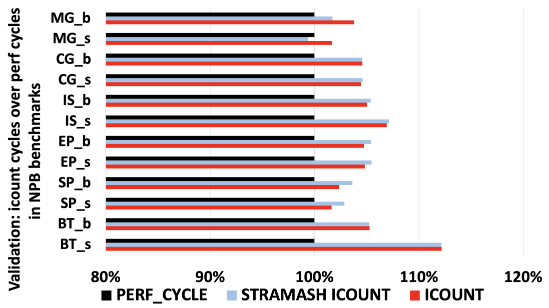

Figure 7. icount validation of small_x86 and small_Arm (*$\_s$*), and also big_x86 and big_Arm (*$\_b$*). Note relative error x axes, always < 13%.

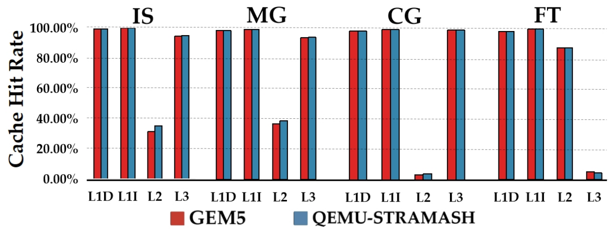

Figure 8. Cache Simulation Validation

Stramash-Linux target heterogeneous platform is not available, we use Popcorn-Linux on two real machines and native perf to profile the application pre- and post-migration, collecting instruction and cycle counts. We then align these native perf results with the Stramash icount data to approximate the cycles. Then we compare this approximation with the actual cycle counts from native perf on the two machines.

The results are shown in Figure 7, the suffixes **b** and **s** represent the native perf on the big machine set and small machine set shown in Table 1. The `ICOUNT` bars are the Popcorn messaging with `Shared Memory` approximated cycle result, the `STRAMASH ICOUNT` bars are the result with fused virtual address enabled, while the `PERF_CYCLE` bars are the native perf result. The overheads of icount are always less than 13%, and about 4% on average – hence, the accuracy of the tool.

9.1.3 Cache plugin.

Stramash-QEMU, with its Cache plugin, targets precise timing verification as mentioned before. We compare our Cache plugin results with the Gem5 MESI Three-Level cache model, a Ruby modular framework for building cache coherence protocols in simulation environments. The same three-level cache structure and size are constructed in our Cache plugin to ensure consistency in our comparisons. We evaluated the NPB benchmarks—CG, IS, MG, and FT—and analyzed their cache behavior, results in Figure 8. It shows the cache hit rates for different cache levels, including the L1 instruction/data cache, L2, and L3 caches. Across the four benchmarks, the performance of our cache simulator closely aligns with that of Gem5 Ruby, with discrepancies in L1, L2, and L3 caches being less than 5%. This demonstrates the accuracy of our cache simulator.

9.2 Stramash-Linux Evaluation

We conducted several experiments to demonstrate the difference between Stramash-Linux and Popcorn-Linux to answer the question: Is the Fused-kernel OS design better than Multi-kernel OS, or is there a trade-off? In the following experiments, we set up a pair of AArch64 and x86-64 kernel instances in Stramash-QEMU, with 8GB of memory in total. We ran single-thread NPB applications that migrate between ISA-different CPUs; the migration points chosen are the same as Popcorn Linux [11] – there is a migration and back-migration for each processing procedure (similarly to offloading).

9.2.1 Benchmarking Cross-ISA Migration.

The results are shown in Figure 9, with the y-axis representing execution time (lower is better) normalised to the Vanilla case (the application runs locally without migration involved). Popcorn-Linux Messaging over Network (**TCP**) and Popcorn-Linux Messaging over Shared Memory (**SHM** with 3 different memory models) serve as our baseline. **Fully Shared**, **Shared** and **Separated** show the results of Stramash-Linux with the specific memory models used in Stramash-QEMU, cf. Figure 3. We observed that Stramash Linux results provide up to a 2× speedup compared to Popcorn-Linux **SHM** with the same memory model setup and 2.6× to **TCP** in the IS benchmark. However, some benchmarks show a less significant speedup.

Performance improvement breakdown. From our observation, performance varies between different NPB benchmarks, indicating that performance improvement is application-specific. In all cases, Stramash Linux with (**Fully Shared**) memory model demonstrates the best performance, closely matching that of the Vanilla case, as it effectively eliminates remote

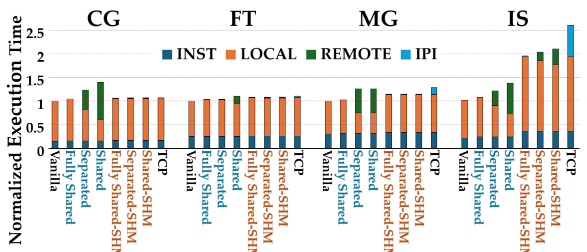

Figure 9. NPB benchmark results

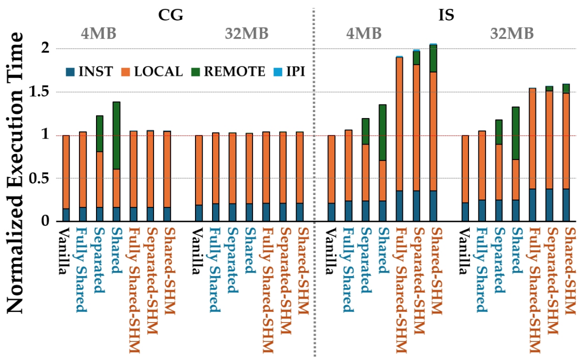

Figure 10. IS vs CG, different Cache size result

memory access and messaging overheads. In order to explore the reason why the performance varies between different applications, we break down the overhead of each NPB benchmark, focusing on three main components: the messaging cost (`MSG`), the memory access overhead (`Local` or `Remote`), and the instructions execution time (`INST`).

Message Passing Overheads. Based on the Popcorn Linux results, the major difference between the two models (**SHM** and **TCP**) arises from message passing overhead, as we added extra network latency to simulate the `TCP/IP` networking. Intuitively, we think messaging overhead would affect the performance significantly. Several benchmarks generate a huge amount of inter-kernel messages as shown in Table 3, for example, MG and IS. However, from the experimental results, the cost of messaging is not a significant factor. Especially from the experimental results, which demonstrate that message passing overhead can be dramatically reduced by utilizing shared memory that has faster interconnection speed (**SHM**). We also notice that the results of **SHM** running on 3 different memory models have similar results because **SHM** always replicates the page; the remote memory access overhead is minimal. Since Stramash-Linux also benefits from the faster inter-connection speed, therefore, we use (**SHM**) as our only baseline in the following experiments.

Read vs Write Intensive Behaviour. An interesting case is observed where Stramash Linux with **Shared** and **Separated** memory models exhibits weak performance, especially in the CG benchmark, whereas in the IS benchmark, it performs well in all cases. We found that the CG (Conjugate Gradient) benchmark is primarily read-intensive. This involves numerous sparse matrix-vector multiplications; 98.34% of memory instructions are load instructions [1]. Instead, the IS (Integer Sort) benchmark is more write-intensive. It tests the performance of integer sorting algorithms, which would modify the sequence of keys during the procedure stage [6].

9.2.2 Cross-ISA migration: Cache Size Sensibility.

In previous experiments, each QEMU instance has 4MB of L3 cache. To further compare Stramash Linux and Popcorn Linux, we increased each QEMU instance's total L3 cache size to 32 MB, similar to recently released multi-core processors [61]. The results for CG and IS are shown in Figure 10. For CG, which is predominantly read-intensive with fewer writes (and thus fewer invalidations), a larger L3 cache reduces the cache miss rate and overall memory accesses, significantly reducing execution time for Stramash Linux with **Shared** and **Separated** memory models. With a smaller cache, when a read operation loads a new cache line into a full cache, an eviction occurs even without invalidation. In Stramash-Linux, this could involve loading data from remote memory. In contrast, Popcorn Linux (**SHM**) always replicates the page, and the load is always from local memory, so its performance is less affected by changes in cache size. As a result, the performance of Popcorn-Linux (**SHM**) in the CG benchmark remains stable regardless of cache size. With a larger cache, reducing remote memory accesses, Stramash-Linux experiences a performance improvement. Its slowdown compared to **SHM** decreases from 34% with a 4MB L3 cache to below 1% with a 32MB L3 cache.

For the IS workload, which is write-intensive, the L3 cache faces a high invalidation rate even with an increased cache size, keeping the high cache miss rate. However, in Popcorn-Linux (**SHM**), a larger cache size reduces cache evictions due to the LRU policy. This leads to fewer write-backs as multiple writes can accumulate in the cache before being written back to memory. Each write-back to replicated pages can trigger the DSM consistency policy if a remote node is reading those pages, adding overhead. Therefore, reducing the replication of DSM pages and reducing write-backs improves the performance of Popcorn-Linux (**SHM**) on the IS benchmark. Meanwhile, because the IS workload's cache miss rate remains high, Stramash-Linux continues to access remote memory frequently, resulting in relatively stable performance despite the increased cache size. Thus, the Stramash-Linux improvement over **SHM** decreases from 2.1× to 1.6× with the larger L3.

9.2.3 Cross-ISA migration: Page replication.

Popcorn-Linux uses DSM to maintain a consistent address space amongst kernels. Replicating pages necessitates messaging to inform the shared page owner of any updates; thus, the software-based memory consistency policy requires extra memory capacity. Although the copy-on-write (COW) policy may mitigate some messaging, significant overheads persist. Stramash-Linux eliminates the need for page replication and ensures that updates are immediately visible to both kernels. Table 3 shows the count of replicated pages during runtime thread migration. In the current version of Stramash, replicated pages still exist because it only allows remote kernel allocation at the PTE level. If the upper layer of the page table is missing, the original kernel will handle the fault to reduce complexity.

9.2.4 Microbenchmarks: Memory Access Cost. We developed a memory-bound microbenchmark to investigate the memory access overhead when the DSM protocol is enabled to maintain page coherence. The results are shown in Figure 11 (lower is better). In this study, we highlight the performance impact of cross-ISA memory access. We allocate 10 MB of data in either the local or remote kernel instance and conduct sequential memory access operations on the allocated memory. We measured five different memory access activities: The origin kernel accesses the origin kernel's memory, serving as our baseline (Vanilla); the remote kernel accesses the origin kernel's memory (Remote access Origin); remote accesses origin, but the remote kernel has previously accessed the memory and thus has the latest version of the memory content (Remote access Origin No Cold). Origin access Remote case and Origin access Remote No Cold case are similar to the previous cases except with opposite access directions.

Results. Popcorn-Linux Messaging over Shared Memory (**SHM**) has less than 1% of REMOTE overhead in the breakdown, and the performance is the same when running on **Separated** and **Fully Shared**. Therefore, Stramash-Linux with the same **Shared** memory model outperforms **SHM** by up to 2.5×, and up to 4.5× in the **Fully Shared** memory model. The most interesting part is the No Cold case. In Popcorn-Linux, the first memory access requires the DSM protocol to replicate the page and keep the local page content up-to-date. However, during continued warm access (read or write), the local pages are already updated, resulting in no DSM overhead, and performance is close to the vanilla case since all memory access is local. In contrast, Stramash-Linux with the **Shared** or **Separated** memory model performs weaker because of remote memory access overheads; since there is no page replication, it needs to load the data again if the previously accessed data has been evicted from the cache, and in the worst case, the data are loaded from the remote memory with extra overhead.

Takeaway. In future architectures, where shared data reside in a remote memory pool, replicating data into local memory can potentially outperform direct remote access. However, any write operation to shared data invalidates all local replicas, thus introducing a trade-off between replication-based and direct-access approaches.

9.2.5 Microbenchmarks: Software vs Hardware Consistency. Software Distributed Shared Memory (DSM) is notorious for its overheads, especially as systems scale [16]. Overheads stem from the expensive maintenance of coherence when data, often in page size, is replicated across multiple nodes, which requires inter-node communication that is expensive. However, none have studied its performance when nodes are tightly connected like in **Shared** model. In contrast, Hardware-based Cache Coherence, such as that supported by CXL 3.0, enables data transfers at cacheline granularity. To quantify the performance differences between software and hardware coherence, we conducted experiments by accessing data ranging from a single cacheline (64 bytes) up to 64 cachelines (4096 bytes) – one page.

Results. As shown in Figure 12, when accessing just one cacheline, DSM incurs an overhead exceeding 300× compared to hardware coherence approaches, due to the unnecessary replication of entire pages. Even when accessing all data within a page, hardware coherence can still achieve approximately 2× faster performance than DSM.

Takeaway. These results underscore the inefficiencies of software DSM, particularly in scenarios where data access patterns are fine-grained and dispersed. Nevertheless, software-based consistency may still offer advantages in cases where sequential access dominates.

9.2.6 Microbenchmarks: Futex Lock. In Popcorn-Linux, all Futex instances are created and managed by the origin kernel, while Stramash-Linux allows the remote kernel to handle the Futex operation itself. We set up an experiment to demonstrate the performance improvement of Stramash-Linux in

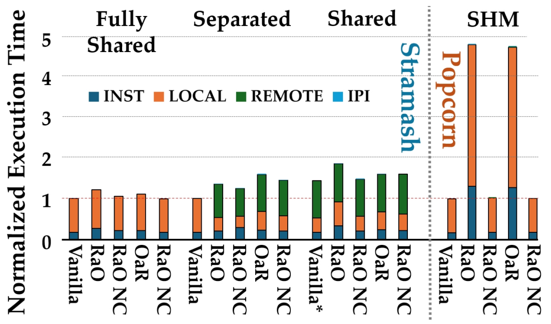

Figure 11. Memory Access Analysis (RaO: Remote Access Origin, OaR: Origin Access Remote, NC: No Cold access **Vanilla***: Vanilla experiment runs on `Shared` memory model, **SHM** performs same on all three memory model

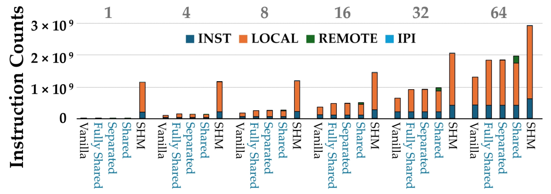

Figure 12. Page Access with cacheline granularity, from 1 cacheline size (64 Bytes) to 64 Cacheline size (4096 Byte), **SHM** performs same on all three memory model

handling Futex operations, with and without Futex optimizations, to eliminate the impact of all other factors. The Futex microbenchmark: The origin kernel continuously locks the Futex, while the remote kernel continuously unlocks the same Futex, performing a simple addition in each loop. A higher loop count indicates more Futex operations. Figure 13 shows the execution time results of the experiments, comparing the Stramash Futex-optimized case to the regular case.

In the Futex-Optimization case, which proves to be the best, only one cross-ISA IPI is needed to wake up the waiting thread, whereas the original solution requires a full Futex management protocol, including multiple requests and response messages to handle each Futex operation.

9.2.7 Global Memory Allocator Overheads. We present the overheads introduced by our allocator, including the time required for offlining and onlining memory slices. We set up Stramash-QEMU with 4GB of dynamically shared memory between two kernels; each slice is 256MB for a total of 16 slices. We recorded the average time to offline or online a slice on both the Arm and x86. Table 4 illustrates the overheads introduced by our allocator, measured in milliseconds, mainly due to the page isolation process.

9.2.8 Network-serving Application. In this experiment, we use the Redis-server [52] as an example of a network-serving application. We modified the Redis-server to migrate between heterogeneous-ISA CPUs. Because we cannot migrate a process/thread that reads/writes to a socket – a Popcorn-Linux limitation, our modified Redis-server migrates to the remote kernel during the processing of the `time_event`. We **do not** enable the Stramash-QEMU Cache plugin because: (a) the simulation has a different time model compared with real-time; (b) the simulation is slow enough to make connections time out. Thus, results demonstrate the functional validation of Stramash-Linux when applied to real-world applications.

On the host machine, the `TCP/IP` based messaging layer is connected through a Linux network bridge, which connects the QEMU tap device. The Redis benchmark runs with 10K requests with fixed payload sizes of 1024 bytes. Due to the different timebase of client and server, we measure processing time for each round of requests inside our modified Redis-server. The normalised results are shown in Figure 14, with the `TCP/IP`-based message layer (`POPCORN-TCP`) set as the baseline; higher is better. Experiment show that the `SHM`-based message layer (`POPCORN-SHM`) can gain around 4−10× speedup compared to the baseline. With Stramash-Linux enabled, the speedup can be even greater, up to 12×. Again, these results are indicative and for functional validation only.

10 Conclusion

With research in academia showing the potential benefits of heterogeneous-ISA platforms – including cache coherent ones, and emerging cache coherent interconnects over peripheral buses to connect heterogeneous-ISA processors on

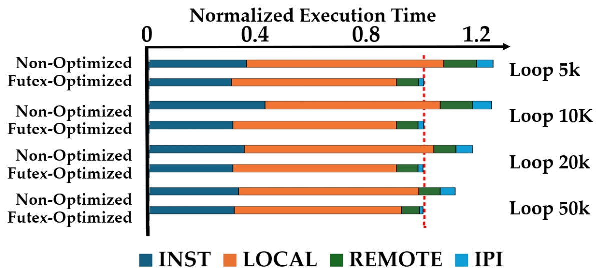

Figure 13. Futex experiment results

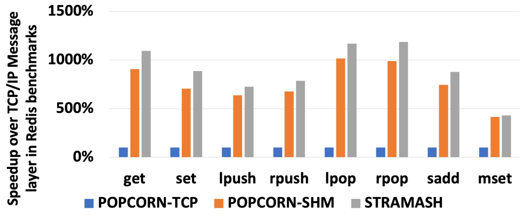

Figure 14. Redis speedup

the same platform, it is fundamental to investigate what operating system software on such platforms should look like. In this paper, we propose a new Operating System design targeting such emerging platforms, the fused-kernel OS, which is a multiple-kernel OS that exploits cache-coherent shared memory for inter-kernel coordination. We implemented a prototype fused-kernel OS based on Linux supporting Arm and x86 ISAs, named Stramash-Linux.

Because the real hardware of the platforms we targeted is not commercially available yet, we built a hardware simulator based on QEMU and Cache-plugin for memory-accurate simulation. We called this Stramash-QEMU. We used Stramash-QEMU to compare Stramash-Linux versus the state-of-the-art multiple-kernel OS for heterogeneous hardware, Popcorn Linux, on 3 different hardware models. We discovered that Stramash-Linux enables the best application performance in most cases, but performance improvements depend on an application's access pattern and the hardware model. Amongst other results, our simulated CXL 3.0 hardware model shows that using DSM with Popcorn-Linux may result in better application performance than directly accessing remote memory with Stramash-Linux, for at least NPB CG.

Acknowledgments

We thank the anonymous reviewers and our shepherd, Gerd Zellweger, for their suggestions and feedback that helped improve this paper. This work is supported in part by the US Office of Naval Research (ONR) under grants N00014-19-1-2493, N00014-22-1-2672, and N00014-21-1-2523, the US Naval Surface Warfare Center Dahlgren Division under grant N00174-20-1-0009, and the UKRI EPSRC under grant EP/V028154/1. Any opinions, findings, and conclusions expressed in this material are those of the authors and do not necessarily reflect the views of these agencies.

References

[1] Gheith A Abandah. 1997. Characterizing Shared-memory Applications: A Case Study of NAS Parallel Benchmarks. Citeseer.

[2] Gustavo Alonso, Timothy Roscoe, David Cock, Mohsen Ewaida, Kaan Kara, Dario Korolija, David Sidler, and Zeke Wang. 2020. Tackling Hardware/Software Co-design from a Database Perspective. In Conference on Innovative Data Systems Research (CIDR 2020).

[3] AMD. 2024. AMD Infinity Fabric™ Link. https://www.amd.com/content/dam/amd/en/documents/instinct-tech-docs/other/56978.pdf.

[4] AMD. 2024. AMD Unveils Their Embedded+ Architecture, Ryzen Embedded with Versal Together. https://www.anandtech.com/show/21254/amd-unveils-their-embedded-architecture-ryzen-embedded-with-versal-together.

[5] ARM. 2024. Introduction to Large System Extensions. https://learn.arm.com/learning-paths/servers-and-cloud-computing/lse/intro/.

[6] D. Bailey. 2024. NAS Parallel Benchmarks, RNR-94-007 (PDF-425KB) for IS, EP, CG, MG, FT, BT, SP, LU. https://www.nas.nasa.gov/assets/nas/pdf/techreports/1994/rnr-94-007.pdf.

[7] Jonathan Balkind, Katie Lim, Michael Schaffner, Fei Gao, Grigory Chirkov, Ang Li, Alexey Lavrov, Tri M. Nguyen, Yaosheng Fu, Florian Zaruba, and et al. 2020. BYOC: A "Bring Your Own Core" Framework for Heterogeneous-ISA Research. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (Lausanne, Switzerland) (ASPLOS '20). Association for Computing Machinery, New York, NY, USA, 699–714. https://doi.org/10.1145/3373376.3378479

[8] Jonathan Balkind, Michael McKeown, Yaosheng Fu, Tri Nguyen, Yanqi Zhou, Alexey Lavrov, Mohammad Shahrad, Adi Fuchs, Samuel Payne, Xiaohua Liang, Matthew Matl, and David Wentzlaff. 2016. OpenPiton: An Open Source Manycore Research Framework. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (Atlanta, Georgia, USA) (ASPLOS '16). Association for Computing Machinery, New York, NY, USA, 217–232. https://doi.org/10.1145/2872362.2872414

[9] Antonio Barbalace, Anthony Iliopoulos, Holm Rauchfuss, and Goetz Brasche. 2017. It's Time to Think About an Operating System for Near Data Processing Architectures. In Proceedings of the 16th Workshop on Hot Topics in Operating Systems. ACM, 56–61.

[10] Antonio Barbalace, Mohamed L. Karaoui, Wei Wang, Tong Xing, Pierre Olivier, and Binoy Ravindran. 2020. Edge Computing: The Case for Heterogeneous-ISA Container Migration (VEE '20).

[11] Antonio Barbalace, Robert Lyerly, Christopher Jelesnianski, Anthony Carno, Ho-Ren Chuang, Vincent Legout, and Binoy Ravindran. 2017. Breaking the Boundaries in Heterogeneous-ISA Datacenters. In Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems (Xi'an, China) (ASPLOS '17). ACM, New York, NY, USA, 645–659. https://doi.org/10.1145/3037697.3037738

[12] Antonio Barbalace, Pierre Olivier, and Binoy Ravindran. 2019. Rethinking Communication in Multiple-kernel OSes for New Shared Memory Interconnects. In Proceedings of the 10th Workshop on Programming Languages and Operating Systems (Huntsville, ON, Canada) (PLOS '19). Association for Computing Machinery, New York, NY, USA, 45–52. https://doi.org/10.1145/3365137.3365399

[13] Antonio Barbalace, Marina Sadini, Saif Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray, and Binoy Ravindran. 2015. Popcorn: Bridging the Programmability Gap in heterogeneous-ISA Platforms. In Proceedings of the Tenth European Conference on Computer Systems (EuroSys '15). 29:1–29:16.

[14] Elena Gabriela Barrantes, David H. Ackley, Stephanie Forrest, Trek S. Palmer, Darko Stefanovic, and Dino Dai Zovi. 2003. Randomized Instruction Set Emulation to Disrupt Binary Code Injection Attacks. In Proceedings of the 10th ACM Conference on Computer and Communications Security (Washington D.C., USA) (CCS '03). ACM, New York, NY, USA, 281–289. https://doi.org/10.1145/948109.948147

[15] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 2009. The Multikernel: A New OS Architecture for Scalable Multicore Systems. In Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (Big Sky, Montana, USA) (SOSP '09). Association for Computing Machinery, New York, NY, USA, 29–44. https://doi.org/10.1145/1629575.1629579

[16] Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. 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). 29–44.

[17] Sharath K. Bhat, Ajithchandra Saya, Hemedra K. Rawat, Antonio Barbalace, and Binoy Ravindran. 2015. Harnessing Energy Efficiency of heterogeneous-ISA Platforms. In Proceedings of the Workshop on Power-Aware Computing and Systems (Monterey, California) (HotPower '15). ACM, New York, NY, USA, 6–10. https://doi.org/10.1145/2818613.2818747

[18] Broadcom. 2018 (accessed June 30, 2020). Stingray PS225.

[19] Broadwell. 2024. Intel Broadwell. https://www.7-cpu.com/cpu/Broadwell.html.

[20] CCIX Consortium. 2017. Cache Coherent Interconnect for Accelerators (CCIX). http://www.ccixconsortium.com/.

[21] Shenghsun Cho, Han Chen, Sergey Madaminov, Michael Ferdman, and Peter Milder. 2020. Flick: Fast and Lightweight ISA-Crossing Call for Heterogeneous-ISA Environments. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). 187–198. https://doi.org/10.1109/ISCA45697.2020.00026

[22] Ho-Ren Chuang, Karim Manaouil, Tong Xing, Antonio Barbalace, Pierre Olivier, Balvansh Heerekar, and Binoy Ravindran. 2023. Aggregate VM: Why Reduce or Evict VM's Resources When You Can Borrow Them From Other Nodes?. In Proceedings of the Eighteenth European Conference on Computer Systems (Rome, Italy) (EuroSys '23). Association for Computing Machinery, New York, NY, USA, 469–487. https://doi.org/10.1145/3552326.3587452

[23] Compute Express Link Consortium, Inc. 2022. Compute Express Link (CXL) Specification (3.0 ed.). https://www.computeexpresslink.org/download-the-specification Available: Compute Express Link Consortium, https://www.computeexpresslink.org/download-the-specification.

[24] Emilio G. Cota, Paolo Bonzini, Alex Bennée, and Luca P. Carloni. 2017. Cross-ISA machine emulation for multicores. In Proceedings of the 2017 International Symposium on Code Generation and Optimization (Austin, USA) (CGO '17). IEEE Press, 210–220.

[25] CXL Consortium. 2022. CXL Specification. https://www.computeexpresslink.org/download-the-specification.

[26] Matthew DeVuyst, Ashish Venkat, and Dean M. Tullsen. 2012. Execution Migration in a heterogeneous-ISA Chip Multiprocessor. In Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (London, England, UK) (ASPLOS XVII). ACM, New York, NY, USA, 261–272. https://doi.org/10.1145/2150976.2151004

[27] Paul J. Drongowski. 2024. ARM Cortex-A72 execution and load/store. http://sandsoftwaresound.net/arm-cortex-a72-execution-and-load-store/.

[28] Yaosheng Fu and David Wentzlaff. 2014. PriME: A parallel and distributed simulator for thousand-core chips. In 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 116–125. https://doi.org/10.1109/ISPASS.2014.6844467

[29] Yaosheng Fu and David Wentzlaff. 2014. PriME: A parallel and distributed simulator for thousand-core chips. In 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). 116–125. https://doi.org/10.1109/ISPASS.2014.6844467

[30] Gal Beniamini, Project Zero. 2024. Over The Air: Exploiting Broadcom's Wi-Fi Stack (Part 1). https://googleprojectzero.blogspot.com/2017/04/over-air-exploiting-broadcoms-wi-fi_4.html.

[31] Joachim Gehweiler and Michael Thies. 2010. Thread migration and checkpointing in java. Heinz Nixdorf Institute, Tech. Rep. tr-ri-10 315 (2010).

[32] Johan De Gelas. 2024. Assessing Cavium's ThunderX2: The Arm Server Dream Realized At Last. https://www.anandtech.com/show/12694/assessing-cavium-thunderx2-arm-server-reality.

[33] GEM5. 2024. Ruby:MESI Three Level. https://www.gem5.org/documentation/general_docs/ruby/.

[34] Intel. 2009. An Introduction to the Intel QuickPath Interconnect.

[35] D. Katz, A. Barbalace, S. Ansary, A. Ravichandran, and B. Ravindran. 2015. Thread Migration in a Replicated-Kernel OS. In 2015 IEEE 35th International Conference on Distributed Computing Systems. 278–287.

[36] Katie Lim, Jonathan Balkind, and David Wentzlaff. 2019. JuxtaPiton: Enabling Heterogeneous-ISA Research with RISC-V and SPARC FPGA Soft-cores. In Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (Seaside, CA, USA) (FPGA '19). ACM, New York, NY, USA, 184–184. https://doi.org/10.1145/3289602.3293958

[37] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. 2014. K2: A Mobile Operating System for Heterogeneous Coherence Domains. In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '14). 285–300.

[38] Felix Xiaozhu Lin, Zhen Wang, and Lin Zhong. 2015. K2: A Mobile Operating System for Heterogeneous Coherence Domains. ACM Trans. Comput. Syst. 33, 2, Article 4 (June 2015), 27 pages. https://doi.org/10.1145/2699676

[39] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood. 2005. Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation. In Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation (Chicago, IL, USA) (PLDI '05). Association for Computing Machinery, New York, NY, USA, 190–200. https://doi.org/10.1145/1065010.1065034

[40] Daniel Lustig, Caroline Trippel, Michael Pellauer, and Margaret Martonosi. 2015. ArMOR: Defending against memory consistency model mismatches in heterogeneous architectures. In 2015 ACM/IEEE 42nd Annual International Symposium on Computer Architecture (ISCA). 388–400. https://doi.org/10.1145/2749469.2750378

[41] Rob Lyerly. 2024. Compiler Support for Application Migration in Heterogeneous-ISA Platforms. https://eurosys2015.labri.fr/posters/p46.pdf.

[42] Rob Lyerly. 2024. Popcorn Linux: A Compiler and Runtime for State Transformation Between Heterogeneous-ISA Architectures. https://www.ssrg.ece.vt.edu/theses/PhdProposal_Lyerly.pdf.

[43] Robert Lyerly, Antonio Barbalace, Christopher Jelesnianski, Vincent Legout, Anthony Carno, and Binoy Ravindran. [n. d.]. Operating System Process and Thread Migration in Heterogeneous Platforms.

[44] Nikolaos Mavrogeorgis, Christos Vasiladiotis, Pei Mu, Amir Khordadi, Björn Franke, and Antonio Barbalace. 2024. UNIFICO: Thread Migration in Heterogeneous-ISA CPUs without State Transformation. In Proceedings of the 33rd ACM SIGPLAN International Conference on Compiler Construction (Edinburgh, United Kingdom) (CC 2024). Association for Computing Machinery, New York, NY, USA, 86–99. https://doi.org/10.1145/3640537.3641565

[45] Mellanox Technologies. 2017. BlueField Multicore System on Chip. http://www.mellanox.com/related-docs/npu-multicore-processors/PB_Bluefield_SoC.pdf. Online, accessed 01/05/2019.

[46] NASA Advanced Supercomputing Division. 2024. NAS Parallel Benchmarks. https://tinyurl.com/y47k95cc.

[47] Edmund B Nightingale, Orion Hodson, Ross McIlroy, Chris Hawblitzel, and Galen Hunt. 2009. Helios: heterogeneous multiprocessing with satellite kernels. In Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 221–234.

[48] OpenCAPI Consortium. 2017. Welcome to OpenCAPI Consortium. http://opencapi.org/.

[49] Popcorn Project. 2024. Popcorn-compiler. https://github.com/ssrg-vt/popcorn-compiler.

[50] Popcorn Project. 2024. ssrg-vt/popcorn-kernel. https://github.com/ssrg-vt/popcorn-kernel/blob/main/include/popcorn/pcn_kmsg.h.

[51] QEMU. 2024. Cache Modelling TCG Plugin. https://www.qemu.org/2021/08/19/tcg-cache-modelling-plugin/.

[52] redislab. 2017. redis – open source data object store. http://redis.io.

[53] Marina Sadini, Antonio Barbalace, Binoy Ravindran, and Francesco Quaglia. [n. d.]. A Page Coherency Protocol for Popcorn Replicated-kernel Operating System. ([n. d.]).

[54] D. Sharma. 2023. Compute Express Link (CXL): Enabling Heterogeneous Data-Centric Computing With Heterogeneous Memory Hierarchy. IEEE Micro 43, 02 (mar 2023), 99–109. https://doi.org/10.1109/MM.2022.3228561

[55] K. Sinha, V. P. Kemerlis, and S. Sethumadhavan. 2017. Reviving instruction set randomization. In 2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). 21–28. https://doi.org/10.1109/HST.2017.7951732

[56] Sophgo. 2024. Dual-Core artificial intelligent processor SG200X. https://en.sophgo.com/sophon-u/product/introduce/sg200x.html.

[57] Texas Instruments. 2014. OMAP4430 Multimedia Device Silicon Revision 2.x Version AP Technical Reference Manual. https://www.ti.com/lit/pdf/swpu231?keyMatch=OMAP4430.

[58] Sudha Udanapalli Thiagarajan, Charles Congdon, Sumedh Naik, and Loc Q Nguyen. 2013. Intel Xeon Phi Coprocessor DEVELOPER'S QUICK START GUIDE, Version 1.5. https://www.intel.com/content/dam/develop/external/us/en/documents/intel-xeon-phi-coprocessor-quick-start-developers-guide.pdf, Online, accessed 01/01/2025.

[59] Ashish Venkat, Sriskanda Shamasunder, Hovav Shacham, and Dean M. Tullsen. 2016. HIPStR: Heterogeneous-ISA Program State Relocation. In Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems (Atlanta, Georgia, USA) (ASPLOS '16). ACM, New York, NY, USA, 727–741. https://doi.org/10.1145/2872362.2872408

[60] Ashish Venkat and Dean M. Tullsen. 2014. Harnessing ISA Diversity: Design of a heterogeneous-ISA Chip Multiprocessor. In Proceeding of the 41st Annual International Symposium on Computer Architecture (Minneapolis, Minnesota, USA) (ISCA '14). IEEE Press, Piscataway, NJ, USA, 121–132. http://dl.acm.org/citation.cfm?id=2665671.2665692

[61] Wikichip. 2024. Cascade Lake - Microarchitectures - Intel. https://en.wikichip.org/wiki/intel/microarchitectures/cascade_lake.

[62] Tong Xing, Antonio Barbalace, Pierre Olivier, Mohamed L. Karaoui, Wei Wang, and Binoy Ravindran. 2022. H-Container: Enabling Heterogeneous-ISA Container Migration in Edge Computing. 39, 1–4, Article 5 (July 2022), 36 pages. https://doi.org/10.1145/3524452

[63] Tong Xing, Hesam Tajbakhsh, Israat Haque, Michio Honda, and Antonio Barbalace. 2022. Towards portable end-to-end network performance characterization of SmartNICs. In Proceedings of the 13th ACM SIGOPS Asia-Pacific Workshop on Systems (Virtual Event, Singapore) (APSys '22). Association for Computing Machinery, New York, NY, USA, 46–52. https://doi.org/10.1145/3546591.3547528

A Artifact Appendix

A.1 Abstract

The provided artifact contains:

- Source code for Stramash Linux kernels (and a Modified Popcorn Linux kernel).
- Stramash QEMU (based on QEMU 8.0).
- A collection of helper scripts to automate building and running experiments.

The Stramash simulator can run on any server with sufficient resources. The NPB benchmarks must be compiled with the Popcorn compiler, though we also provide pre-compiled binaries and kernel images used in our experiments. A Docker environment is included to ease the build process.

A.2 Artifact Check-List (Meta-Information)

- **Program:** QEMU, Linux Kernel, Docker
- **Compilation:** GCC, Popcorn Compiler, Cross-compile toolchain
- **Binary:** Modified NPB benchmark suite
- **Run-Time Environment:** Linux
- **Disk Space Required:** ~10 GB
- **Publicly Available:** YES
- **Code Licenses (if publicly available):** MIT
- **Workflow Automation:** Shell scripts

A.3 Description

Stramash consists of three main software components:

1. **Stramash QEMU simulator**
2. **Stramash Linux kernel** (plus a modified Popcorn Linux kernel),
3. **Helper scripts** to automate building and running experiments.

A.3.1 How to Access

The artifact is publicly available at:

- https://github.com/systems-nuts/Stramash-AE
- https://doi.org/10.5281/zenodo.14847090

A.4 Installation

1. Set the root directory of Stramash

- `cd Stramash-AE`
- `STRAMASH_ROOT=$(pwd)`

2. Build the Docker image for compilation (Optional if you can natively build on Ubuntu 22.06)

- `cd $STRAMASH_ROOT/docker`
- `sudo docker build -t stramash_env .`
- `sudo docker run -dit -privileged -name stramash_container -mount type=bind,source="$(STRAMASH_ROOT)", target="$(STRAMASH_ROOT)" stramash_env`
- `sudo docker exec -it -w "$(STRAMASH_ROOT)" stramash_container /bin/bash`

3. Build the kernel (optional), file system, and QEMU

- (Inside the Docker container)
- `./build_fs.sh`
- `./build_kernel.sh` (Optional. We provide a pre-built kernel image and module.)
- `./build_qemu.sh`
- **exit** the Docker container once done.

4. Set up the kernel and file system

- If you compiled the kernel, run:
  - `sudo ./set_up.sh`
- Otherwise, if you are using the pre-compiled kernel image, run:
  - `chmod +x setup_no_kernel_compile.sh`
  - `sudo ./setup_no_kernel_compile.sh`

5. Start Stramash

- `sudo ./start.sh` (This will start three pairs of Stramash machines.)

6. Run NPB Benchmarks (Now we do everything in QEMU)

- After `start.sh`, three pairs of QEMU instances (each pair in a separate `tmux` window) are launched. Each pair corresponds to a specific memory model:
  1. **Stramash Shard model**
  2. **Stramash Separated model**
  3. **SHM model**
- In each `tmux` window, the left console is x86 and the right console is ARM (you can verify by running `uname -a`).
- To use Stramash, both kernels need the corresponding module inserted. For x86, insert the module first (it may be slower to load); then wait a few seconds and insert on ARM:
  - **First two Stramash QEMU pairs (Shard/Separated):**
    - `insmod stramash_msg_shm.ko`
  - **SHM QEMU pair:**
    - `insmod shm_msg_shm.ko`
- **Running NPB: (inside each pairs)**
  - For each benchmark, where `$BIN` is one of {`cg`, `is`, `ft`, `mg`}.
  - `./NPB_AE/$BIN; cat /proc/cache_sync_switch; cat /proc/popcorn_icount_switch`
  - Note that each benchmark may take multiple hours on a single core, depending on your hardware. We modifed the counter for help turn off the Plugin Model.

A.5 Evaluate (Figure 9: NPB Benchmark Results)

We evaluate the following runtime formula in our experiments:

- **Final Runtime** = (x86 Runtime) + (ARM Runtime).

STRAMASH results. For **Fully Shared** memory, there is no "remote" access. We can approximate it by subtracting remote accesses computed in the feedback from the **Separated** (or **Shard**) model, however, for the experiments with large cache need, because **Fully Shared** we consider shared L3 cache, it can be configed at the stramash-qemu/contrib/plugins/cache-sim-feedback.c at L2328 and L2350, and recompile the QEMU. – https://github.com/systems-nuts/Stramash-AE/blob/main/stramash-qemu/contrib/plugins/cache-sim-feedback.c

```
Fully Shared Runtime = Final Runtime -
Remote Memory Hits x 0.455

Here, the factor 0.455 is the ratio of
remote vs. local memory overhead derived from:
```

Link: https://github.com/systems-nuts/Stramash-AE/blob/main/stramash-qemu/contrib/plugins/cache-sim-feedback.c#L215

```
#define Local_mem_overhead   360
#define Remote_mem_overhead  660
660 / 360 = 1.83 -> remote is 1.83× the cost of local;
the difference ratio (remote-local)/remote = 0.455
```

POPCORN-SHM results. For the **SHM** model, only accesses to the message ring buffer are considered remote. By default, the launched SHM machine is set to the Shared model. For **SHM Fully Shared**, similarly subtract remote memory hits multiplied by 0.455. For **SHM Separated**, you only subtract the remote hits on arm side – Because in the **Separated** model, the x86 access to the shared memory ring is local, while it is exposed to the arm through simulated CXL, so we consider arm access to be remote access.

Example output

```
x86:
L1 Cache Hit Rate: 93.64%
L2 Cache Hit Rate: 56.06%
L3 Cache Hit Rate: 79.82%
L1 Cache Hits: 5127379213
L2 Cache Hits: 175113042
L3 Cache Hits: 261230203
L1 Cache Accesses: 5475752586
L2 Cache Accesses: 348373373
L3 Cache Accesses: 327260331
IPI: 17
Local Memory Hits: 59366
>>> Remote Memory Hits: 65970762 <<<
Remote Shared Memory Hits: 321312
Number of Instructions: 8601072931
Number of mem_access: 5471616305
>>> Runtime: 91254395261 <<<
Arm:
L1 Cache Hit Rate: 93.77%
L2 Cache Hit Rate: 49.65%
L3 Cache Hit Rate: 78.28%
L1 Cache Hits: 5819027543
L2 Cache Hits: 187330951
L3 Cache Hits: 273466584
L1 Cache Accesses: 6205724879
L2 Cache Accesses: 386697336
L3 Cache Accesses: 349366385
IPI: 10
Local Memory Hits: 37555
>>> Remote Memory Hits: 75862246 <<<
Remote Shared Memory Hits: 342424
Number of Instructions: 10133480114
Number of mem_access: 6201636747
>>> Runtime: 97501520205 <<<
```