

Multikernel on ARM64

Cong Wang

Founder and CEO, Multikernel Technologies, Inc.

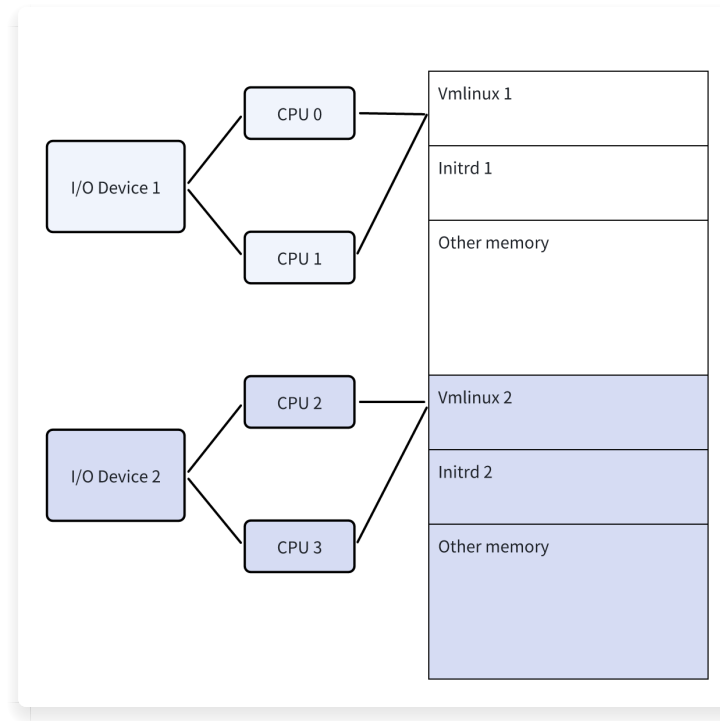
What is Multikernel?

Core Concept: Run multiple Linux kernels simultaneously without virtualization

- **Isolation over replication:** Each kernel is independent, not a replica
- **Dedicated resources:** Each kernel owns exclusive hardware partitions
- **Per-application kernels:** Tailored Linux kernel for each workload
- **Linux Compatible:** Full Linux compatibility

Web Server	Database	ML Training
Linux Kernel (Web-tuned)	Linux Kernel (I/O-optimized)	Linux Kernel (GPU-optimized)
CPU + NIC	CPU + NVMe	CPU + GPU

Architecture Overview



Why ARM64?

- **ARM servers are mainstream:** Graviton, Ampere, Grace, Cobalt, Axion
- **High core counts:** 128 to 192 cores per socket, begging to be partitioned
- **Firmware does the hard part:** PSCI already owns the CPU lifecycle
- **Same user interface:** kernfs, device tree overlays and `kerf` are unchanged

Only what sits below the interface differs

x86 vs ARM64 at a Glance

	x86-64	ARM64
Idle pool CPU	Park loop in host memory	PSCI <code>OFF</code> , held by firmware
Start a kernel	Trampoline + identity page tables	<code>CPU_ON</code> at Image entry
Kernel image	<code>vmlinux</code>	Image via <code>kexec_file</code>
Boot info	<code>boot_params</code> + manifest	One device tree in <code>x0</code>
Doorbell	<code>MULTIKERNEL_VECTOR</code>	SGI 7
Device MSIs	Address names the LAPIC	Host maps via its ITS
Force halt	NMI per APIC ID	Stop IPI as pseudo-NMI
DMA containment	None yet	SMMU domain per instance

CPU: Firmware Holds the Pool

- **Pool CPU = PSCI OFF**: executes nothing, no park loop, no mailbox
- **Into the pool**: stock CPU hotplug, `cpu_psci_cpu_die()` and `cpu_psci_cpu_kill()`
- **Spawn**: `CPU_ON` at the Image entry point, device tree address as context ID
- **New PSCI helper**: `psci_cpu_on_context()` lets the caller choose `x0`
- **Secondaries**: the spawn brings them up with the stock PSCI enable method

No trampoline, no identity page tables, no purgatory

CPUs: Going Down Safely

Problem: stock arm64 halt parks CPUs in `cpu_park_loop()` inside the kernel image, which the host is about to overwrite

Our Solution:

- A spawn returns **every** CPU to firmware on halt, reboot and panic
- Never reaches PSCI `SYSTEM_OFF` or `SYSTEM_RESET`, which would kill the host
- Host confirms with `AFFINITY_INFO` before reusing memory
- Refuses to load a new image over a CPU that is still on

Force Halt Without an NMI

PSCI has no call to turn *another* CPU off

- Host sends `IPI_CPU_STOP_NMI` to the target MPIDR
- Spawn takes it as a **pseudo-NMI**, even with interrupts masked
- Foreign stop IPIs are obeyed only when the force-halt marker is armed
- Spawn requires `CONFIG_ARM64_PSEUDO_NMI` and `irqchip.gicv3_pseudo_nmi=1`; `kerfload` adds it

Next step: SDEI, which reaches a CPU regardless of how its kernel booted

One Boot Tree

On arm64 the device tree in `x0` is the **only** boot channel, so the instance tree becomes the boot tree, rebuilt on every start:

```
/memory@<base>    one node per memory range granted  
/cpus             own CPUs first (boot CPU leads), then all others  
/psci, /timer, GIC copied from host DT, or written from FADT/GTDT/MADT  
/chosen          bootargs, initrd, linux, pci-probe-only  
/resources        the multikernel instance description
```

- Works whether the host booted from **DT or UEFI/ACPI**
- The spawn is always a DT platform
- No UART node: use `earlycon` to poll the host's UART

Sharing One GIC

Problem: one distributor. A new kernel's `gic_dist_init()` resets it and steals every SPI from the host

Our Solution: GIC tenant mode (`multikernel,tenant`)

- Distributor left alone, only checked for affinity routing
- Each kernel owns the redistributors and CPU interfaces of its CPUs
- Only SPIs granted in `multikernel,spis` can be mapped, others fail with `-EPERM`
- SGIs and PPIs are per CPU and just work

Inter-kernel Doorbell: SGI 7

- `ICC_SGI1R` addresses a CPU by **affinity**, not by logical CPU number
- New `gic_v3_send_sgi_to_mpidr()` reaches a CPU running another kernel
- SGI number is ABI between kernels
- All 8 non-secure SGIs are taken: the doorbell replaces the KGDB roundup IPI
- KGDB falls back to its generic roundup

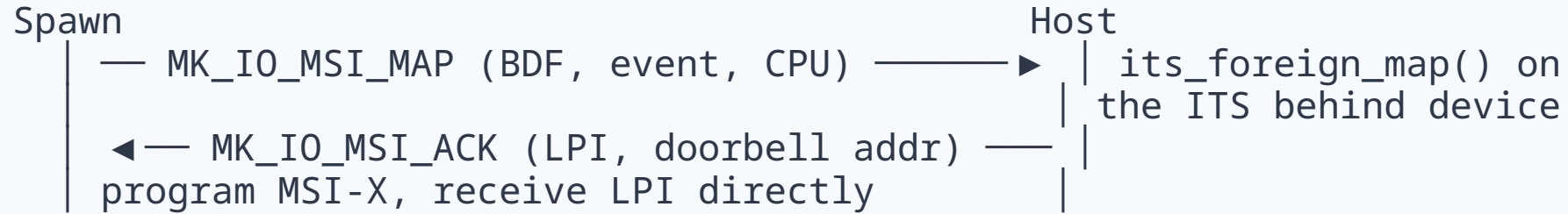
The message ring itself needed no change

PCI Devices

- Root buses appear as `pci-host-ecam-generic` bridges, granted devices as the only children
- Stock driver binds, BARs and windows stay as the host assigned them
- **Bridges above are foreign:** no driver binds, config writes are dropped
- Root port config space is byte-for-byte unchanged across spawn and halt

MSIs Through the Host's ITS

Problem: an ITS has one command queue, owned by the host



- Works with **any number of ITSs**, found via the device's MSI domain
- `MK_IO_MSI_MOVE` is synchronous, even from `stop_machine()` during CPU offline
- Host drops all mappings when the instance halts, however it halts

SMMU DMA Containment

Problem: the spawn hands physical addresses to its devices. Behind a bypassing SMMU, a buggy driver can DMA anywhere in the host

Our Solution:

- SMMU stays with the host, never described to the spawn
- Each instance device attached to a paging domain mapping **only** its memory (1:1) plus the ITS doorbell
- Follows memory hotplug of a running instance
- Generic IOMMU API, the way VFIO does it

Platform Devices

- Granted by name, e.g. `a003e00.virtio_mmio`
- Host copies the device node into the boot tree and grants its SPIs
- Host unbinds its driver first, rebinds after the instance halts
- **Limitation**: self-contained devices only, no clocks, resets, power domains or pinctrl
- **Limitation**: needs a DT host, an ACPI host has no node to copy

Platform Requirements

- **PSCI 0.2+:** `CPU_ON`, `CPU_OFF`, `AFFINITY_INFO` (no spin-table)
- **GICv3**, with an ITS behind PCI devices that need MSIs
- **Kernel:** `CONFIG_MULTIKERNEL`, `CONFIG_KEXEC_FILE`, `CONFIG_HOTPLUG_CPU`
- **Spawn:** `CONFIG_ARM64_PSEUDO_NMI`
- **Host:** device tree or UEFI/ACPI. Any SBSA server qualifies

Status

Tested on QEMU `virt`, booted from DT, under TF-A at EL2, and from UEFI/ACPI:

- Spawn boots, halts, and re-execs on the same CPUs
- virtio-mmio disk on a granted SPI, host keeps its UART
- virtio-net PCI with MSI-X as LPIs, also behind an SMMUv3
- NVMe in a spawn, CPU offline under live interrupts
- Force halt of a CPU stuck in hard lockup

Next: bring-up on real ARM servers, multi-ITS hardware, SDEI, GICv5

Try It Under QEMU

1. Build the kernel:

```
git clone -b multikernel-arm https://github.com/multikernel/linux.git
cd linux
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- defconfig
# Enable CONFIG_MULTIKERNEL and CONFIG_ARM64_PSEUDO_NMI
make ARCH=arm64 CROSS_COMPILE=aarch64-linux-gnu- -j$(nproc) Image
```

2. Boot the host:

```
qemu-system-aarch64 -M virt,gic-version=3,iommu=smmu-v3 \
    -cpu max -smp 4 -m 4G -kernel arch/arm64/boot/Image ...
```

Manage Instances with kerf

Same tool, same commands as on x86:

```
kerf init --cpus=2-3
kerf create web-server --cpus=2-3 --memory=1GB
kerf load web-server --kernel=Image --initrd=initramfs.cpio.gz \
    --cmdline="earlycon=pl011,mmio32,0x90000000 keep_bootcon"
kerf exec web-server
```

`kerf load` appends `irqchip.gicv3_pseudo_nmi=1` automatically

Use Cases on ARM Servers

- **Cloud**: per-tenant kernels on Graviton-class instances without nested virtualization
- **AI agent sandbox**: native performance, SR-IOV devices passed through
- **Live kernel update**: boot the new kernel alongside, migrate, retire the old one
- **Kernel auto-healing**: backup kernel takes over in sub-second time
- **Edge and automotive**: isolate real-time and general-purpose kernels on one SoC

Call For Collaboration

Looking for:

- ARM server hardware for bring-up and validation
- Firmware partners for SDEI and TF-A testing
- Early adopters on ARM cloud and edge

What you get:

- Direct input on design decisions
- Free consulting during deployment
- Linux kernel community recognition

Questions and Feedback

Contact: cwang@multikernel.io

Open Source Projects: github.com/multikernel

Video Demo: youtube.com/@multikernel-tech

Discord: <https://discord.gg/KsGgrZMS>

Join the mailing list: multikernel@lists.linux.dev