Agenda

❖ Architecture Overview
❖ NOVA Building Blocks
❖ Recent Innovations
❖ ARM/x86 Code Unification
❖ Advanced Security Features (x86)
❖ Performance
❖ Q&A
Architecture Overview

NOVA Microhypervisor

ARMv8-A or x86

UltraVisor™

UltraSecurity™

VMM

VMM

VMM

VMM

VMM

VMM

VM (Linux)

VM (Windows)

VM (Appliance)

VM (RTOS)

VM (Unikernel)

UART MUX (UMX)

VirtIO Socket MUX (vSMX)

Network MUX (vSwitch)

UART Driver

Storage Driver

Platform Manager

Network Driver

Host Apps

Master Controller

NOVA Microhypervisor

ARMv8-A or x86

UltraSecurity™

VMI

VAS

VAS

VAS

VAS

user

kernel

guest

host
NOVA Microhypervisor Building Blocks

Protection Domains, Execution+Scheduling Contexts, Portals, Semaphores

Hypercall interface uses capability-based access control for all operations

- Very fast synchronous IPC with time donation and priority inheritance
**NOVA Innovation Timeline**

<table>
<thead>
<tr>
<th>Release</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>21.08</td>
<td>CPU Freq. Enumeration, Unified Time Mgmt., CR0 / CR4 Intercepts</td>
</tr>
<tr>
<td>21.17</td>
<td>SMMU Stream Masking, CET Support</td>
</tr>
<tr>
<td>21.26</td>
<td>CAT / CDP Support, Extended State Mgmt., AVX512 Support</td>
</tr>
<tr>
<td>21.35</td>
<td>Enhanced SMMU Invalidation, TSC_AUX Support</td>
</tr>
<tr>
<td>21.43</td>
<td>TME / TME-MK Key Programming, DRNG Support</td>
</tr>
<tr>
<td>21.52</td>
<td>Unified Code Base, Generic Page Tables, ARM/x86 Feature Parity</td>
</tr>
<tr>
<td>22.08</td>
<td>ACPI S3 / S4 Suspend / Resume on x86</td>
</tr>
<tr>
<td>22.17</td>
<td>Enhanced ACPI S1 / S5 / Reset Platform Shutdown</td>
</tr>
<tr>
<td>22.26</td>
<td>ACPI S1 / S5 / Reset Platform Shutdown</td>
</tr>
<tr>
<td>22.35</td>
<td>ACPI S3 / S4 Suspend / Resume on x86</td>
</tr>
<tr>
<td>22.43</td>
<td>ACPI S1 / S5 / Reset Platform Shutdown</td>
</tr>
<tr>
<td>22.52</td>
<td>ACPI S1 / S5 / Reset Platform Shutdown</td>
</tr>
<tr>
<td>23.08</td>
<td>Extended ACPI S1 / S5 / Reset Platform Shutdown</td>
</tr>
</tbody>
</table>

**Approximately 2 months between releases**
NOVA Design Goals

❖ Provide the same (or similar) functionality across all architectures
❖ Generic API that抽象s from architectural differences as much as possible
❖ Simple Build Infrastructure

➢ make ARCH= BOARD=
❖ Standardized Boot Process and Resource Enumeration

➢ Use Multiboot v2/v1, UEFI, ACPI, PSCI
❖ Formal Verification of highly concurrent C++ code and weakly-ordered memory
❖ Modern, Small, Fast: Best-in-Class Security and Performance
Functions can be categorized as

- **Same API / Same Implementation**
  - can share source/header/spec file

- **Same API / Different Implementation**
  - can potentially share header/spec file

- **Different API and Implementation**
  - cannot share anything

Architecture-specific files will override generic files with the same name

---

**NOVA**

- **Makefile**
- **build-aarch64**
- **build-x86_64**
- **doc**
- **inc**
  - `aarch64`
  - `x86_64`
- **src**
  - `aarch64`
  - `x86_64`
Unified Code Base

<table>
<thead>
<tr>
<th>x86</th>
<th>11483 SLOC</th>
<th>37.4% generic</th>
<th>3.9% ASM</th>
</tr>
</thead>
<tbody>
<tr>
<td>x86_64</td>
<td>7183 SLOC</td>
<td>43.0% generic</td>
<td></td>
</tr>
<tr>
<td>generic</td>
<td>4300 SLOC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>aarch64</td>
<td>5683 SLOC</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARM</td>
<td>9983 SLOC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

NOVA x86 Binary (ACPI)
- 68808 Bytes Code
- 2464 Bytes Data

NOVA ARM Binary (ACPI)
- 66916 Bytes Code
- 300 Bytes Data

SLOC based on release-23.08.0, binary sizes based on gcc-12.2.0 build. Other versions will produce different numbers.
Protecting against Boot-Time DMA Attacks

- Firmware owns platform hardware prior to UEFI ExitBootServices
- ExitBootServices drops DMA protections for Legacy OS Support 😞
- Window of opportunity for DMA attack before NOVA can enable IOMMU

⇒ NOVA disables bus masters and manages ExitBootServices flow
Flexible Load Address

- No physical load-address range works across all platforms/architectures
  - Some platforms have RAM starting at 0, some have MMIO starting at 0
  - Bootloader may want to move image to end of memory

- Load address now flexible (can move NOVA up/down by multiples of 2 MiB)
  - No ELF relocation support needed
  - Init section mapped 1:1 and uses position-independent (mode-independent) code
  - Runtime section mapped V⇒P via paging and uses virtual memory

- Multiboot protocol deficiencies currently limit load address to < 4 GiB
  - Many multiboot structures defined as 32-bit and nothing defined for aarch64 😞
# Power Management: Overview of ACPI States

<table>
<thead>
<tr>
<th>Platform Global States</th>
<th>Platform Sleep States</th>
<th>Core Idle States</th>
<th>Core Performance States</th>
</tr>
</thead>
<tbody>
<tr>
<td>G0 (Working)</td>
<td>S0 (Working)</td>
<td>C0 (Active)</td>
<td>P0</td>
</tr>
<tr>
<td>G1 (Sleeping)</td>
<td>S1 (Stop Grant)</td>
<td>C1 (Halt)</td>
<td>P1</td>
</tr>
<tr>
<td></td>
<td>S2 (Power-On Suspend)</td>
<td>C2 (Stop Clock)</td>
<td>P2</td>
</tr>
<tr>
<td></td>
<td>S3 (Suspend-to-RAM)</td>
<td>C3 (Deep Sleep)</td>
<td>P3</td>
</tr>
<tr>
<td></td>
<td>S4 (Suspend-to-Disk)</td>
<td>C4 (Deeper Sleep)</td>
<td>P4</td>
</tr>
<tr>
<td></td>
<td>S5 (Soft Off)</td>
<td>C6 (Deep Power Down)</td>
<td>P5</td>
</tr>
</tbody>
</table>
Suspend/Resume: Possible Approaches

❖ Either save/restore entire register set of all devices
  ➢ Significant amount of state to manage
  ➢ Does not work for devices with hidden “internal state” or complex state machines

⇒ Possibly suitable for generic devices managed by some other component

❖ Or save high-level configuration and reinitialize devices based on that
  ➢ Less information to maintain ⇒ faster save/restore
  ➢ Resume similar to first initialization ⇒ using same code paths as initial boot

⇒ Approach for all devices managed by the NOVA microhypervisor
Performance: ACPI P-States on x86

- Max Turbo Frequency
- Guaranteed Frequency
- Maximum Efficiency

$P_{\text{dyn}} = CfV^2$

- Dynamic Voltage and Frequency Scaling
- Unused thermal and power headroom of idle/offline cores given to active cores

- HW-Controlled Turbo Bins
- HW-Controlled or OS-Controlled P-States

- Active Core 0
- Idle Core 1
- Idle Core 2
- Idle Core 3

- P0
- HFM
- LFM
- P1
- P2
- P3
- P4
- Pn

- Maximum Efficiency
- HW-Controlled
- Turbo Bins

- Guaranteed Frequency
- HW-Controlled or OS-Controlled P-States

- Max Turbo Frequency
- Guaranteed Frequency
- Maximum Efficiency
## Power Management: Feature Comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>x86_64</th>
<th>aarch64</th>
</tr>
</thead>
<tbody>
<tr>
<td>P-State Support</td>
<td>EIST (older) and HWP (SKL+)</td>
<td>Not Yet</td>
</tr>
<tr>
<td>S1</td>
<td>Supported</td>
<td>N/A</td>
</tr>
<tr>
<td>S2/S3</td>
<td>Supported</td>
<td>Supported (PSCI 1.0+ Optional)</td>
</tr>
<tr>
<td>S4/S5</td>
<td>Supported</td>
<td>Supported (PSCI 0.2+ Mandatory)</td>
</tr>
<tr>
<td>Platform Reset</td>
<td>Supported</td>
<td>Supported (PSCI 0.2+ Mandatory)</td>
</tr>
<tr>
<td>S0ix (Low-Power Idle)</td>
<td></td>
<td>Not Yet</td>
</tr>
<tr>
<td>Hypercall <code>ctrl_hw</code></td>
<td></td>
<td>Returns <code>BAD_FTR</code> for unsupported functionality</td>
</tr>
<tr>
<td>Hypercall <code>assign_dev</code></td>
<td></td>
<td>Device assignment preserved over Suspend/Resume</td>
</tr>
<tr>
<td>Hypercall <code>assign_int</code></td>
<td></td>
<td>Interrupt configuration preserved over Suspend/Resume</td>
</tr>
</tbody>
</table>
Past: Singleton Spaces built into Protection Domain

- **Host EC**: Thread
- **Guest EC**: Virtual CPU
- **Device**: BDF or SID

1. **create_pd**
2. **create_ec**
3. **assign_dev**

- **Object Space**: Object Capability Table
- **Host Space**: Stage-1 Page Table
- **Guest Space**: Stage-2 Page Table
- **DMA Space**: IOMMU Page Table
- **PIO Space**: I/O Permission Bitmap
- **MSR Space**: MSR Permission Bitmap

Protection Domain

- Suboptimal for Nested Virtualization, SMMU Virtualization, Advanced VMI
Multiple Spaces separated from Protection Domain

- **Host EC**
  - Thread

- **Object Space**
  - Object Capability Table

- **Host Space**
  - Stage-1 Page Table

- **Guest Space**
  - Stage-2 Page Table

- **DMA Space**
  - IOMMU Page Table

- **PIO Space**
  - I/O Permission Bitmap

- **MSR Space**
  - MSR Permission Bitmap

- **Protection Domain**
  - Slab Allocators

- **3. create_ec**
  - Permanent Binding

- **2. create_pd**
  - Flexible Assignment

- **1. create_pd**

6 new kernel object types
Multiple Spaces separated from Protection Domain

1. create_pd

2. create_pd

3. create_ec

4. ipc_reply (with MTD.SPACES set)

Flexible reassignment of guest spaces upon events

Host Space
- Stage-1 Page Table

Guest Space
- Stage-2 Page Table

DMA Space
- IOMMU Page Table

PIO Space
- I/O Permission Bitmap

MSR Space
- MSR Permission Bitmap

Protection Domain
- Slab Allocators

Object Space
- Object Capability Table

Guest EC
- Virtual CPU

SEL
- GST
- PIO
- MSR

x86 Only

- Permanent Binding
- Flexible Assignment
Multiple Spaces separated from Protection Domain

1. create_pd
2. create_pd
3. assign_dev

- Object Space: Object Capability Table
- Host Space: Stage-1 Page Table
- Guest Space: Stage-2 Page Table
- DMA Space: IOMMU Page Table
- PIO Space: I/O Permission Bitmap
- MSR Space: MSR Permission Bitmap

Flexible reassignment of DMA spaces anytime

Permanent Binding
Flexible Assignment

Device
BDF or SID
Generic Page Tables

- NOVA manages 3 Page-Table Types for each Architecture
  - Host (Stage-1), Guest (Stage-2), DMA (IOMMU/SMMU)
  - These correspond to the 3 memory spaces

- Architecture-Independent Template Base Class
  - Lock-less page-table traversal, (de)allocation and PTE updates
  - Substitution of large pages with page tables and vice versa

- Type-Specific Subclasses (CRTP)
  - Encode access permissions and memory attributes (cacheability, shareability, key index)
  - Ensure non-coherent agents observe updates in the correct order
Page Tables: 39-bit Address Space (3-Level)

Input Address
HVA or GPA

Level 2
9 Bits

Page Table
512 Entries

Level 1
9 Bits

Page Table
512 Entries

Level 0
9 Bits

Page Table
512 Entries

Offset
12 Bits

63:39 38:30 29:21 20:12 11:0

Root Ptr

Page Frame
4096 Bytes

1 GiB

2 MiB

4 KiB

 Applies to
➢ aarch64
➢ x86_64

NOVA Microhypervisor
Udo Steinberg - FOSDEM 2023
Page Tables: 40-bit Address Space (Concatenated)

- **Input Address**: HVA or GPA
- **Level 2**: 10 Bits
- **Level 1**: 9 Bits
- **Level 0**: 9 Bits
- **Offset**: 12 Bits
- **Page**: Frame 4096 Bytes
- **Page Table**: 512 Entries
- **Root Ptr**: L2 resolves 1 extra bit
- **Page Table**: 1024 Entries
- **Page Frame**: 4096 Bytes
- **L2 Page Table is 8 KiB**

- **Applies to**: aarch64 only
Page Tables: 41-bit Address Space (Concatenated)

Input Address
- HVA or GPA

Offset
- 12 Bits

Level 2
- 11 Bits

Level 1
- 9 Bits

Level 0
- 9 Bits

Page

Frame
- 4096 Bytes

Root Ptr

Page Table
- 2048 Entries

Page Table
- 512 Entries

Page Table
- 512 Entries

L2 resolves 2 extra bits
L2 Page Table is 16 KiB

Applies to
- aarch64 only

NOVA Microhypervisor
Udo Steinberg - FOSDEM 2023
Page Tables: 48-bit Address Space (4-Level)

### Input Address
- HVA or GPA

<table>
<thead>
<tr>
<th>Level</th>
<th>Offset</th>
<th>Address Space</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>11:0</td>
<td>1 GiB</td>
</tr>
<tr>
<td>1</td>
<td>20:12</td>
<td>2 MiB</td>
</tr>
<tr>
<td>2</td>
<td>29:21</td>
<td>4 KiB</td>
</tr>
<tr>
<td>3</td>
<td>38:30</td>
<td>512 GiB</td>
</tr>
<tr>
<td>4</td>
<td>47:39</td>
<td>512 MiB</td>
</tr>
<tr>
<td>5</td>
<td>56:48</td>
<td>512 GiB</td>
</tr>
</tbody>
</table>

- **Root Ptr**: 1 GiB
- **Page Table**: 512 Entries
- **Page Frame**: 4096 Bytes

 Applies to
- aarch64
- x86_64
Page Tables: Level-Indexing Configuration

### x86_64

<table>
<thead>
<tr>
<th>Bits</th>
<th>L4</th>
<th>L3</th>
<th>L2</th>
<th>L1</th>
<th>L0</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>57</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>48</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

### aarch64

<table>
<thead>
<tr>
<th>Bits</th>
<th>L3</th>
<th>L2</th>
<th>L1</th>
<th>L0</th>
<th>OFF</th>
</tr>
</thead>
<tbody>
<tr>
<td>52</td>
<td>4+9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>5</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>3+9</td>
<td>9</td>
<td>9</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>40</td>
<td>1+9</td>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>2+9</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **NOVA derives from input bits**
  - Uniform index bits per level
  - Non-uniform concatenation at top level
  - Number of required page-table levels
Multi-Key Total Memory Encryption (TME-MK)

- KeyID per page encoded in PTE
- Stealing upper physical bits

<table>
<thead>
<tr>
<th>Unused</th>
<th>KeyID</th>
<th>Physical Address</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unused</td>
<td>KeyID</td>
<td>Physical Address</td>
<td>Attributes</td>
</tr>
<tr>
<td>Unused</td>
<td>KeyID</td>
<td>Physical Address</td>
<td>Attributes</td>
</tr>
</tbody>
</table>

- Key Programming
  - random/tenant
  - DRNG entropy

Key0 - FW TME Key
Key1 - AES-XTS-128
Key2 - AES-XTS-256
Key3 - AES-XTS-256
Key4 - AES-XTS-128
Key5 - AES-XTS-128
Protecting against “Noisy Neighbor” Domains

[Diagram showing a microhypervisor architecture with VMs, cores, caches, and shared resources.]
Cache Allocation Technology (CAT)

- Limited Number of Classes of Service (COS)
  - Hypervisor assigns COS to all cache-sharing entities (VMs/Processes/Threads)

- Capacity Bitmask per COS
  - Specifies into which ways of the cache that COS can allocate
  - Can be defined per CPU and separately for L2/L3 caches
  - Bitmasks must be contiguous, but can overlap
  - Active COS per CPU set in IA32_PQR_ASSOC MSR

- Improves Predictability (WCET) and Cache Side-Channel Resistance
Cache Allocation Technology (CAT): Example

Full Isolation / No Capacity Sharing
Code and Data Prioritization (CDP)

- Capacity bitmasks redefined as pairs
  - Even bitmask number ⇒ data
  - Odd bitmask number ⇒ code
- More fine-grain control over how the cache is being used
- Total number of classes of service (COS) cut in half
  - 6 COS for CAT ⇒ 3 COS for CDP
- NOVA API enforces selection of CAT vs. CDP mode for L2/L3 cache
  - QoS configuration required before capacity bitmasks can be programmed
  - Mode cannot be changed subsequently
Code and Data Prioritization (CDP): Example

Competitive Capacity Sharing

Exclusive Use

- COS 0
  - D: 35%
  - C: 25%

- COS 1
  - D: 50%
  - C: 15%

- COS 2
  - D: 20%
  - C: 30%

- Exclusive Use
Class of Service (COS)

- Option 1: Assign COS to Protection Domain (PD)
  - For PDs that span multiple cores, COS settings must be consistent across cores

- Option 2: Assign COS to Execution Context (EC)
  - Expensive to set new COS during each context switch (WRMSR)

- In NOVA: COS is a Scheduling Context (SC) attribute
  - Only need to context-switch COS during scheduler invocations
  - Extends the call-chain time/priority donation model with COS donation
  - Servers use the cache capacity allocated to their respective clients
  - Additional benefit: COS settings can be configured differently on each core
Code Integrity Protection

- Long history of paging features raising the bar for code injection attacks
  - Non-writable code / Non-executable stack (W^X)
  - Supervisor Mode Execution Prevention (SMEP)
  - Supervisor Mode Access Prevention (SMAP)
  - Mode-Based Execution Control (MBEC) for Stage-2 with XU/XS permission bits

- Code snippets (gadgets) in existing code could still be chained together
  - Control-Flow Hijacking: COP / JOP / ROP attacks
  - Instruction length is fixed on ARM but varies on x86
Control-Flow Enforcement Technology (CET)

❖ Protects integrity of control-flow graph using x86 hardware features

❖ Indirect Branch Tracking (Forward-Edge)  
  ➢ Used with indirect JMP / CALL instructions
  ➢ Valid branch targets must be marked with ENDBR instruction
  ➢ Requires compiler support (available since gcc-8)

❖ Shadow Stacks (Backward-Edge)  
  ➢ Used with CALL / RET instructions
  ➢ Second stack used exclusively for return addresses
  ➢ Can only be written by control-transfer and shadow-stack-management instructions
CALL / JMP Instruction

- Next instruction must be ENDBR
- #CP exception otherwise
CALL instruction
- Pushes return address onto both stacks

RET instruction
- Pops return address from both stacks
- #CP exception if addresses not equal

Shadow Stack Management
- Busy bit in token prevents multi-activation
- NOVA must unwind supervisor shadow stack during context switches
Managing the ISA Evolution

- Modern CPU features require new instructions that fault on older CPUs
  - Example: Shadow Stack Management Instructions

- Newer CPUs provide instructions that do more/optimized work
  - Example: XSAVE / XSAVEC / XSAVEOPT / XSAVES (Extended State Management)

- Code using newer instructions will fault on older CPUs

- Possible options
  - Either compile different binaries with/without those instructions (compile-time)
  - Or select the most suitable instruction (each time) at run time (run-time)
  - Or install the most suitable instruction (once) at boot time (boot-time)
Boot-Time Code Patching

❖ Install the optimal instruction (sequence) once at boot time

➢ asm ("xsaves %0" : "=m" (fpu_state) ...); ⇒ asm ("xsave %0" : "=m" (fpu_state) ...);
➢ asm ("setssbsy"); ⇒ asm ("nop");

❖ Early feature checking to determine which patches to apply

➢ Patches only low-level assembler instructions (newer CPUs ⇒ less patching)
➢ No need for patching any high-level C++ code ⇒ compiles to innocuous instructions
➢ No need for repeated run-time feature tests ⇒ no extra overhead

❖ One generic binary that works across a wide range of hardware platforms

➢ Automatically adjusts to supported platform features
Round-Trip IPC Performance (with CET)

**NUC12WS**

Intel® Core™ i7-1270P
GLC / 2496 MHz TSC

- **Same PD**:
  - TSC Ticks: 217
  - B: 87ns +13%
  - R: 246 +59%
  - F: 346 +67%

- **Different PDs**:
  - TSC Ticks: 536
  - B: 215ns +5%
  - R: 563 +22%
  - F: 658 +26%

**NUC11TN**

Intel® Core™ i7-1185G7
WLC / 2995 MHz TSC

- **Same PD**:
  - TSC Ticks: 229
  - B: 77ns +15%
  - R: 265 +60%
  - F: 367 +73%

- **Different PDs**:
  - TSC Ticks: 584
  - B: 195ns +6%
  - R: 621 +23%
  - F: 721 +29%

---

**Release 23.08**

- CFP=none
- CFP=branch
- CFP=return
- CFP=full

**Bedrock Systems Inc.**

NOVA Microhypervisor
Udo Steinberg - FOSDEM 2023
Questions and Discussion

The NOVA microhypervisor is licensed under GPLv2

Releases: https://github.com/udosteinberg/NOVA/tags

More Information: bedrocksystems.com and hypervisor.org