TOKYO, JAPAN / DECEMBER 11-13, 2025

Generic Page Table & IOMMUFD Discussion

Jason Gunthorpe

東京 2025
LINUX
PLUMBERS CONFERENCE
TOKYO, JAPAN / DEC. 11-13, 2025

IOMMUFD Features

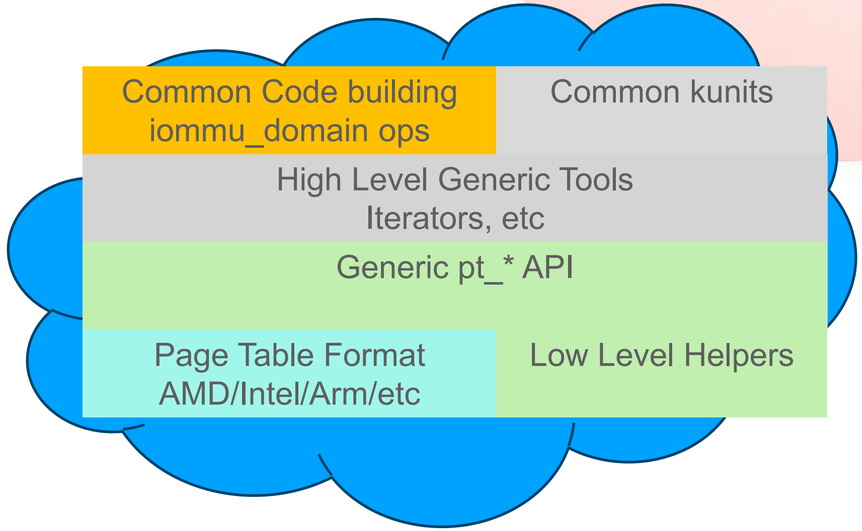
Feature	Version
IOMMU_IOAS_MAP_FILE For memfd	v6.13
IOMMU_IOAS_CHANGE_PROCESS	v6.13
vIOMMU, vDEVICE, and SMMUv3 nesting	v6.13
ARM ITS management with nested	v6.15
vEVENTQ	v6.15
PASID Support	v6.15
IOMMU_HW_QUEUE_ALLOC	v6.15
VFIO DMABUF for IOAS_MAP_FILE	v6.19
Consolidated Page Table	v6.19

Feature	Version
AMD VIOMMU and Nested	v5
Kernel Live Update	RFC
No IOMMU support	RFC
ARM ITS Direct Routing	v2
Confidential Compute	N/A
Page Table Optimizations and Features	N/A

Generic Page Table

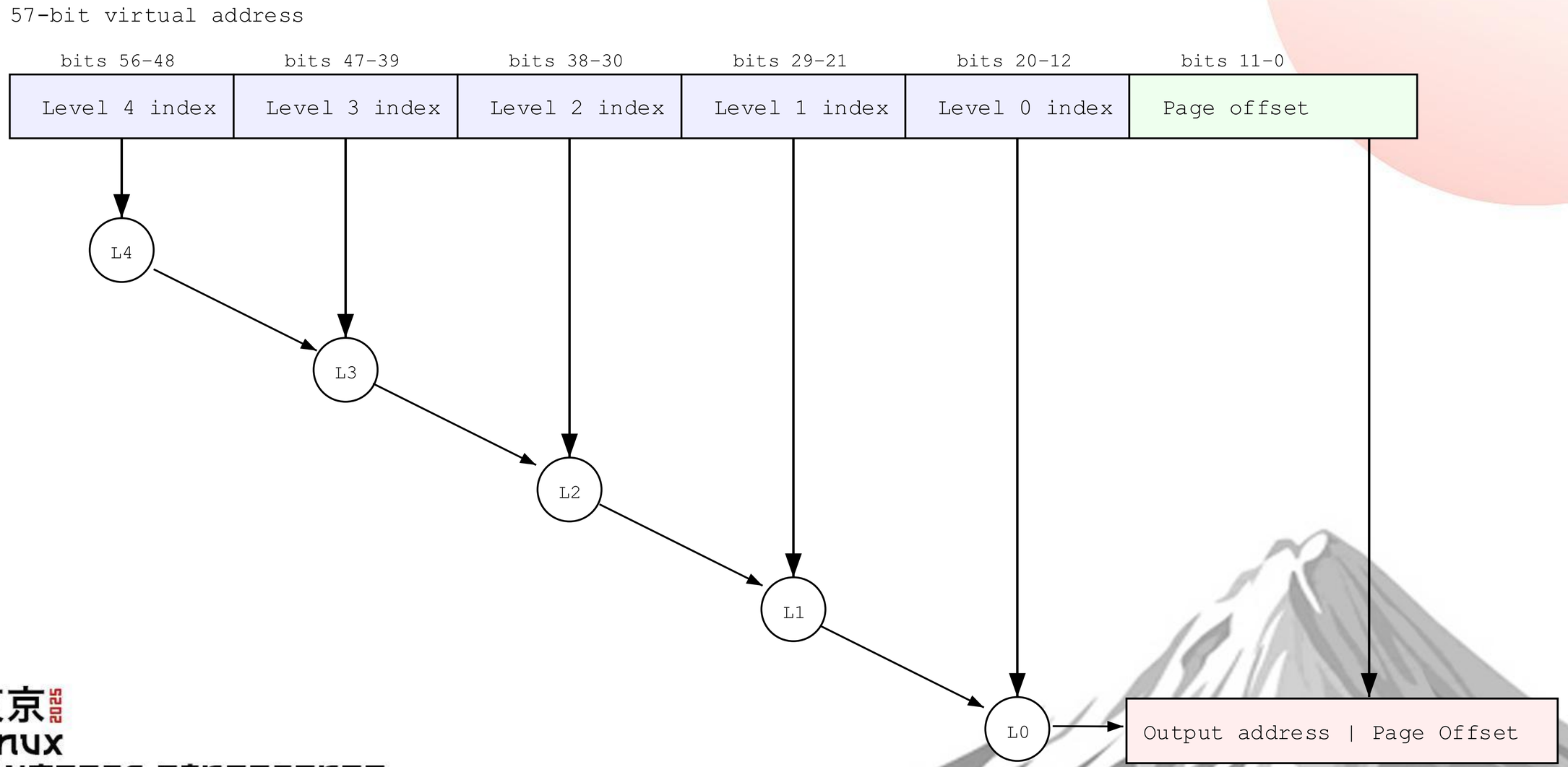
- Single C code providing iommu_domain ops
  One source for all page table
  map/unmap/iova_to_phys
- Multi compilation approach to optimize for each format
- Starting point merged v6.19 for
  AMD and VT-d

- Generic Layer re-usable out side of iommu_domain use cases (mm/kvm/etc)



Compiled together
iommu_pt_amdv1.ko

Radix Page Table



Key Terms

- VA/OA: Virtual (Input) and Output (Physical) address
- Level: Table hops from the VA's LSB. 0 is all leaves
- Top Level: A two level table has a top of 1
- Item: A single index in a level's table
- Entry: A group of Items representing a contiguous logical entry
- Leaf: An entry pointing to an output address
- Table Item: An item pointing to a lower level table

Driver Use 1

- Include the page table structs in the driver's iommu_domain struct
- Multi-inheritance like approach, pt_iommu_* has **struct iommu_domain** as well.

```
struct protection_domain {
        union {
                struct iommu_domain domain;
                struct pt_iommu_amdv1 amdv1;
                struct pt_iommu_x86_64 amdv2;
        };
        /* .. */
};
PT_IOMMU_CHECK_DOMAIN(struct protection_domain, amdv1.iommu, domain);
PT_IOMMU_CHECK_DOMAIN(struct protection_domain, amdv2.iommu, domain);
```

Driver Use 2

- Directly connect the ops, no function pointer trampolines

```
static const struct pt_iommu_driver_ops amd_hw_driver_ops_v1 = {
        .get_top_lock = amd_iommu_get_top_lock,
        .change_top = amd_iommu_change_top,
};

static const struct iommu_domain_ops amdv1_ops = {
        IOMMU_PT_DOMAIN_OPS(amdv1),
        .iotlb_sync_map = amd_iommu_iotlb_sync_map,
        .flush_iotlb_all = amd_iommu_flush_iotlb_all,
        .iotlb_sync = amd_iommu_iotlb_sync,
        .attach_dev = amd_iommu_attach_device,
        .free = amd_iommu_domain_free,
        .enforce_cache_coherency = amd_iommu_enforce_cache_coherency,
};

static const struct iommu_dirty_ops amdv1_dirty_ops = {
        IOMMU_PT_DIRTY_OPS(amdv1),
        .set_dirty_tracking = amd_iommu_set_dirty_tracking,
};
```

Driver Use 3

- Configure and initialize during domain allocation

```
cfg.common.features = BIT(PT_FEAT_DYNAMIC_TOP) |
                      BIT(PT_FEAT_AMDV1_ENCRYPT_TABLES) |
                      BIT(PT_FEAT_AMDV1_FORCE_COHERENCE);
if (amd_iommu_np_cache)
        cfg.common.features |= BIT(PT_FEAT_FLUSH_RANGE_NO_GAPS);
else
        cfg.common.features |= BIT(PT_FEAT_FLUSH_RANGE);

cfg.common.hw_max_vasz_lg2 =
        min(64, (amd_iommu_hpt_level - 1) * 9 + 21);
cfg.common.hw_max_oasz_lg2 = 52;
cfg.starting_level = 2;
domain->domain.ops = &amdv1_ops;

ret = pt_iommu_amdv1_init(&domain->amdv1, &cfg, GFP_KERNEL);
if (ret)
        /* ... */
```

Driver Use 4

- Program to hardware

```
struct pt_iommu_amdv1_hw_info pt_info;
struct dev_table_entry new = {};

make_clear_dte(dev_data, dte, &new);

if (domain->domain.type & __IOMMU_DOMAIN_PAGING) {
        pt_iommu_amdv1_hw_info(&domain->amdv1, &pt_info);

        new.data[0] |= __sme_set(pt_info.host_pt_root) |
                       (pt_info.mode & DEV_ENTRY_MODE_MASK)
                                << DEV_ENTRY_MODE_SHIFT;
}
```

General Driver Guidance

- Driver should be modernized to fully setup the page table during allocate
  - Use the allocating instance and device to set the geometry and parameters
  - Check during attach that the attaching instance and device are compatible
- Dis-entangle multi-format call chains to single format. Each format needs its own iommu_domain_ops
- Be careful about max IOVA and required # of levels.
  Think about sign extension
- Push hw_info close to the HW programming

Making a Format 1

```
#define PT_FMT vtdss
#define PT_SUPPORTED_FEATURES                                          \
        (BIT(PT_FEAT_FLUSH_RANGE) | BIT(PT_FEAT_VTDSS_FORCE_COHERENCE) | \
         BIT(PT_FEAT_VTDSS_FORCE_WRITEABLE) | BIT(PT_FEAT_DMA_INCOHERENT))

#include "iommu_template.h"
```

```
#include <linux/generic_pt/common.h>
#include "drivers/iommu/generic_pt/fmt/defs_vtdss.h"
#include "drivers/iommu/generic_pt/pt_defs.h"
#include "drivers/iommu/generic_pt/fmt/vtdss.h"
#include "drivers/iommu/generic_pt/pt_common.h"
```

```
static inline int vtdss_pt_iommu_fmt_init(struct pt_iommu_vtdss *iommu_table,
                                          const struct pt_iommu_vtdss_cfg *cfg)
{
        struct pt_vtdss *table = &iommu_table->vtdss_pt;

        if (cfg->top_level > 4 || cfg->top_level < 2)
                return -EOPNOTSUPP;

        pt_top_set_level(&table->common, cfg->top_level);
        return 0;
}
#define pt_iommu_fmt_init vtdss_pt_iommu_fmt_init
```

Making a Format 2

- Define required format functions. See `drivers/iommu/generic_pt/pt_common.h`
- Bare minimum:

pt_table_pa()	pt_entry_oa()	pt_can_have_leaf()	pt_num_items_lg2()
pt_load_entry_raw()	pt_install_leaf_entry()	pt_install_table()	pt_attr_from_entry()
pt_iommu_set_prot()	pt_iommu_fmt_init()	pt_iomu_fmt_hw_info()	kunit_fmt_cfgs[]

- Consider dirty tracking, contiguous pages, software bits

Making a Format 3

- Run the generic kunit for the format

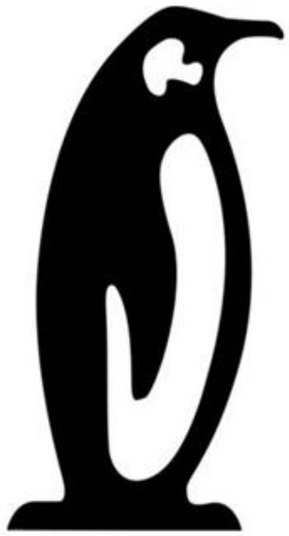
```
tools/testing/kunit/kunit.py run --build_dir build_kunit_x86_64 --arch x86_64 \
   --kunitconfig ./drivers/iommu/generic_pt/.kunitconfig XXXX_fmt_test.*
```

```
[11:34:23] ================== [PASSED] test_dirty ==================
[11:34:23] ==================== test_sw_bit_leaf  ====================
[11:34:23] [PASSED] vtdss_cfg_0
[11:34:23] [PASSED] vtdss_cfg_1
[11:34:23] [PASSED] vtdss_cfg_2
[11:34:23] ================ [PASSED] test_sw_bit_leaf =================
[11:34:23] ==================== test_sw_bit_table  ====================
[11:34:23] [PASSED] vtdss_cfg_0
[11:34:23] [PASSED] vtdss_cfg_1
[11:34:23] [PASSED] vtdss_cfg_2
[11:34:23] ================ [PASSED] test_sw_bit_table ================
[11:34:23] ================= [PASSED] vtdss_fmt_test ==================
[11:34:23] ============================================================
[11:34:23] Testing complete. Ran 36 tests: passed: 36
[11:34:23] Elapsed time: 3.519s total, 0.001s configuring, 2.748s building, 0.347s running
```

- Run all the kunits

Short Term Next Steps

- Move page size fragmentation below the domain callback
  Don't rewalk when the page size changes, batch flushes internally
  ~10% benchmark improvement
- debugfs dumping and remove unique VT-d dumping
- More drivers - riscv (needs testing), SMMUv3 (needs more work)
- New ops: cut, increase page size, decrease page size
- Efficient ops for working with batches of address lists
  - Don't rewalk to change physical addresses
  - Optimize cache flushing
  - Avoid double walking iova_to_phys and then unmap



東京 2025
LINUX PLUMBERS CONFERENCE

TOKYO, JAPAN / DECEMBER 11-13, 2025