CPU idle: Introducing Cluster Management

L.Pieralisi, D.Lezcano, A.Kucheria

Linux Plumbers 2012

Outline

1 CPU idle for Clusters of CPUs
- Towards multi-cluster ARM systems
- CPU idle current status
- CPU idle cluster requirements

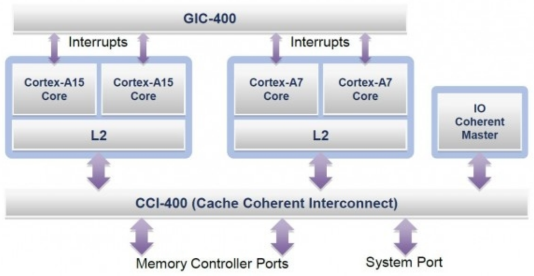
2 ARM Kernel CPU idle Plumbing
- Introduction
- CPU idle cluster management back-end requirements

Outline

1 CPU idle for Clusters of CPUs
- Towards multi-cluster ARM systems
- CPU idle current status
- CPU idle cluster requirements

2 ARM Kernel CPU idle Plumbing
- Introduction
- CPU idle cluster management back-end requirements

ARM big.LITTLE Systems



- Heterogeneous systems
- Coherent CCI interconnect
- Per-Cluster unified L2
- Shared GIC 400

Outline

1 CPU idle for Clusters of CPUs
- Towards multi-cluster ARM systems
- CPU idle current status
- CPU idle cluster requirements

2 ARM Kernel CPU idle Plumbing
- Introduction
- CPU idle cluster management back-end requirements

The Case for Idling ARMs (1/2)

- CPU idle framework written for Intel platforms (ACPI driven)
  - per-CPU idle devices
  - ..and cluster (package) states managed under the hood by HW
  - cache layout completely transparent to the kernel

CPU idle cluster management

- Cluster shutdown iff all cores in a cluster are idle
- but...CPU idle framework has no notion of "cluster" of cpus
- every ARM MP platform idle driver coordinates CPUs in a platform specific way

The Case for Idling ARMs (2/2)

- CPU idle cluster management for ARM: from hotplug to coupled C-states
- coupled C-states are a necessity (for some platforms) not a holistic solution
- parked cores and CPUs coordination: we all do it, in a platform specific way
- CPU PM notifiers updates
- cache levels link to power domains

Introducing CPU idle cluster management

If it idles as a cluster, it is powered as a cluster,
it must be a cluster

Cluster Concept in the Kernel (1/2)

- On ARM platforms it is defined by the MPIDR register
  - **MPIDR[23:16]**: affinity level 2
  - **MPIDR[15:8]**: affinity level 1
  - **MPIDR[7:0]**: affinity level 0
- No relation to power domains

Affinity levels do not always define power domains boundaries

- Every CPU in a cluster can have its own power rail
- Cache topology and power domains are not related to affinity levels

Cluster Concept in the Kernel (2/2)

Topology used by scheduler domains

{core, thread}_cpumask

```
struct cputopo_arm {
        int thread_id;
        int core_id;
        int socket_id;
        cpumask_t thread_sibling;
        cpumask_t core_sibling;
};

extern struct cputopo_arm cpu_topology[NR_CPUS];

#define topology_physical_package_id(cpu)       (cpu_topology[cpu].socket_id)
#define topology_core_id(cpu)           (cpu_topology[cpu].core_id)
#define topology_core_cpumask(cpu)      (&cpu_topology[cpu].core_sibling)
#define topology_thread_cpumask(cpu)    (&cpu_topology[cpu].thread_sibling)
```

CPU idle: coupled C-states (1/3)

- C.Cross code consolidating existing OMAP4 and Tegra code shipped with current devices
- Cores go idle at unrelated times that depend on the scheduler and next event
- Cluster states (whether SoC support per-CPU power rail or not) can be attained iff all cores are idle
- Hotplugging CPUs to force idle is not a solution

CPU idle barrier

- All CPUs request power down at almost the same time
- but...if power down fails, they all have to abort together

CPU idle: coupled C-states (2/3)

- CPUs put into a safe state and woken up (IPI) to enter the idle barrier

```
int cpuidle_enter_state_coupled(struct cpuidle_device *dev,
                struct cpuidle_driver *drv, int next_state)
{
[...]
retry:
        /*
         * Wait for all coupled cpus to be idle, using the deepest state
         * allowed for a single cpu.
         */
        while (!cpuidle_coupled_cpus_waiting(coupled)) {
                if (cpuidle_coupled_clear_pokes(dev->cpu)) {
                        cpuidle_coupled_set_not_waiting(dev->cpu, coupled);
                        goto out;
                }

                if (coupled->prevent) {
                        cpuidle_coupled_set_not_waiting(dev->cpu, coupled);
                        goto out;
                }

                entered_state = cpuidle_enter_state(dev, drv,
                        dev->safe_state_index);
        }
[...]
}
```

CPU idle: coupled C-states (2/3)

- CPU coordinated but not by the governor

```
static int cpuidle_coupled_clear_pokes(int cpu)
{
        local_irq_enable();
        while (cpumask_test_cpu(cpu, &cpuidle_coupled_poked_mask))
                cpu_relax();
        local_irq_disable();

        return need_resched() ? -EINTR : 0;
}
```

CPU idle: coupled C-states (3/3)

"You may delay, but time will not."
B.Franklin

CPU idle: coupled C-states (3/3)

```
int arch_cpuidle_enter(struct cpuidle_device *dev, ...)
{
        if (arch_turn_off_irq_controller()) {
                /* returns an error if an irq is pending and would be lost
                   if idle continued and turned off power */
                abort_flag = true;
        }

        cpuidle_coupled_parallel_barrier(dev, &abort_barrier);

        if (abort_flag) {
                /* One of the cpus didn't turn off it's irq controller */
                arch_turn_on_irq_controller();
                return -EINTR;
        }

        /* continue with idle */
        ...
}
```

- Disable or move IRQs to one CPU
- CPUs coordinated but wake-up events do not disappear

Outline

1 CPU idle for Clusters of CPUs
- Towards multi-cluster ARM systems
- CPU idle current status
- CPU idle cluster requirements

2 ARM Kernel CPU idle Plumbing
- Introduction
- CPU idle cluster management back-end requirements

CPU idle: governors and next event (1/4)

- Current governors make decisions on a per-CPU basis
- Different CPUs in a cluster enter idle at arbitrary times
- Need to peek at next event to avoid initiating cluster shutdown

CPU idle: governors and next event (2/4)

- next_event must become a cluster concept
- if broadcast emulation is used, peek at next_event related to cluster mask
- if per-cpu (always-on) timers are used, all per-CPU timers in a cluster mask should be checked

CPU idle: governors and next event (3/4)

```
/*
 * @index: target C-state index
 *
 * Return:
 * 1: pending events within target residency time window
 * 0: no cluster pending events within target residency
 */
static inline int event_shutdown(unsigned int index)
{
        struct tick_device *td = tick_get_broadcast_device();
        s64 delta;
        delta = ktime_to_us(ktime_sub(td->evtdev->next_event, ktime_get()));
        if (delta <= 0 || delta < c[index].target_residency)
                return 1;
        return 0;
}
```

Broadcast emulation

- if broadcast emulation is used, peek at $\mathrm{next\_event}$ related to cluster mask
- broadcast device IRQ affinity must be controlled so that useless cluster wake-ups are prevented

CPU idle: governors and next event (4/4)

```
/*
 * Return:
 * 1: pending events within target residency time window
 * 0: no cluster pending events within target residency
 */
static inline int event_shutdown(void)
{
        s64 delta;
        unsigned int idx, cluster = (read_cpuid_mpidr() >> 8) & 0xf;

        for_each_cpu(idx, &cluster_mask[cluster]) {
                delta = ktime_to_us(ktime_sub(per_cpu(next_event, idx), ktime_get()));
                if ((delta <= 0) || (per_cpu(cur_residency, idx) > delta))
                        return 1;
        }

        return 0;
}
```

Always on per-cpu timers

if per-cpu (always-on) timers are used through power down, all per-CPU timers in a cluster mask should be checked

CPU idle: asymmetric state tables (1/2)

- Current kernel code exports a single latency table to all CPUs
- b.L clusters sport asymmetric latency tables and must be treated as such
- In the switching case, idle tables must be switched at run-time (upon notification)
- D.Lezcano created per-CPU idle states (through a pointer in struct cpuidle_device)

CPU idle: asymmetric state tables (2/2)

```
struct cpuidle_device {
        [...]
        unsigned int            cpu;

        int                     last_residency;
        int                     state_count;
+       struct cpuidle_state    *states;
        struct cpuidle_state_usage      states_usage[CPUIDLE_STATE_MAX];
        struct cpuidle_state_kobj *kobjs[CPUIDLE_STATE_MAX];

        struct list_head        device_list;
        struct kobject          kobj;
        struct completion       kobj_unregister;
};

int cpuidle_register_states(struct cpuidle_device *dev,
                            struct cpuidle_state *states,
                            int state_count)
{
        [...]
        dev->states = states;
        dev->state_count = state_count;

        return 0;
}
```

Outline

1 CPU idle for Clusters of CPUs
- Towards multi-cluster ARM systems
- CPU idle current status
- CPU idle cluster requirements

2 ARM Kernel CPU idle Plumbing
- Introduction
- CPU idle cluster management back-end requirements

Idling ARM CPUs

- CPU idle deep C-states require context save/restore
- CPU architectural state (inclusive of VFP and PMU)
- Peripheral state (GIC, CCI)
- Cache management (clear/invalidate, pipelining)
- Multi-cluster systems require some code tweaks

Our Goal

Implement a full-blown CPU idle framework that supports ARM multi-cluster platforms

Focus on CPU/Cluster Power Management (PM)

Unified code in the kernel to support saving and restoring of CPU and Cluster state

ARM Common PM Code Components

- CPU PM notifiers
- Local timers save/restore
- CPU suspend/resume
- L2 suspend/resume

Outline

1 CPU idle for Clusters of CPUs
- Towards multi-cluster ARM systems
- CPU idle current status
- CPU idle cluster requirements

2 ARM Kernel CPU idle Plumbing
- Introduction
- CPU idle cluster management back-end requirements

A15/A7 and big.LITTLE PM Requirements

- Integrated L2 caches management
- Inter-Cluster snoops (pipeline)
- MPIDR affinity levels (boot, suspend/resume)
- CCI management
- Need to peek at $\mathrm{next\_event}$ on a per-cluster basis

v7 Cache Levels (1/2)

```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);

        flush_cache_all(); <- !! This call cleans all cache levels up to LoC to main memory !!
        outer_clean_range(*save_ptr, *save_ptr + ptrsz);
        outer_clean_range(virt_to_phys(save_ptr),
                          virt_to_phys(save_ptr) + sizeof(*save_ptr));
}
```

- A15/A7 flush_cache_all() also cleans L2 !
- single CPU shutdown must not clean unified L2, not required

v7 Cache Levels (2/2)

```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);
        u32 *ctx = ptr;

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);

        flush_dcache_level(flush_cache_level_cpu());

        [...]

        __cpuc_flush_dcache_area(ctx, ptrsz);
        __cpuc_flush_dcache_area(save_ptr, sizeof(*save_ptr));
        outer_clean_range(*save_ptr, *save_ptr + ptrsz);
        outer_clean_range(virt_to_phys(save_ptr),
                          virt_to_phys(save_ptr) + sizeof(*save_ptr));
}
```

- Introduce cache level patches into the kernel
- Flush all caches to the LoU-IS (but should be linked to power domains)

MPIDR - suspend/resume (1/2)

MPIDR[23:16]: affinity level 2
MPIDR[15:8]: affinity level 1
MPIDR[7:0]: affinity level 0

- MPIDRs of existing cores cannot be generated at boot unless we probe them (pen release)
- MPIDR must NOT be considered a linear index

MPIDR - suspend/resume (2/2)

```
ENTRY(cpu_resume)
#ifdef CONFIG_SMP
        adr     r0, sleep_save_sp
        ALT_SMP(mrc p15, 0, r1, c0, c0, 5)
        ALT_UP(mov r1, #0)
        and     r1, r1, #15
        ldr     r0, [r0, r1, lsl #2]    @ stack phys addr
#else
        ldr     r0, sleep_save_sp       @ stack phys addr
#endif
        setmode PSR_I_BIT | PSR_F_BIT | SVC_MODE, r1  @ set SVC, irqs off
        @ load phys pgd, stack, resume fn
        ARM(  ldmia   r0!, {r1, sp, pc}       )
        [...]
ENDPROC(cpu_resume)

sleep_save_sp:
        .rept   CONFIG_NR_CPUS
        .long   0                               @ preserve stack phys ptr here
        .endr
```

Create a map of MPIDR to logical indexes at boot to fetch the proper context address

CCI coherency management

Inter-Cluster Coherency enablement

- In CPU idle every CPU in a cluster can become primary upon wake-up from deep C-states
- Enabling coherency before the MMU can be turned on
- Multiple CPUs can be reset at once and enter the kernel
  - Power controller HW/policy dependent
  - ldrex/strex might not be available (MMU off - memory controller support required)
  - Strongly ordered memory locking algorithm

Misc tweaks

- CPU PM notifiers modifications (GIC, CCI)
- Timer broadcast mask, IRQ affinity and next event peek function

Security Management

- Most of the operations should be carried out in secure world
- non-secure cache-line clean/invalidate can be deferred
- Policy decisions made in Linux
- ARM SMC protocol proposal implementation in the making

Putting Everything Together (1/2)

CPU idle skeleton state enter function

```
struct pm_pms {
        unsigned int cluster_state;
        unsigned int cpu_state;
};

void enter_idle(unsigned int cluster_state, unsigned int cpu_state)
{
        int cpu = smp_processor_id();
        struct pm_pms pms;
        struct cpumask tmp;
        [...]

        cpu_set(cpu, cpuidle_mask);
        cpumask_and(&tmp, &cpuidle_mask, topology_core_cpumask(cpu));

        pms.cluster_state = cluster_state;
        pms.cpu_state = cpu_state;

        if (!cpumask_equal(&tmp, topology_core_cpumask(cpu)))
                pms.cluster_state = 0;

        cpu_pm_enter();
        if (pms.cluster_state >= SHUTDOWN)
                cpu_cluster_pm_enter();

        cpu_suspend(&pms, suspend_finisher);

        cpu_pm_exit();

        if (pms.power_state >= SHUTDOWN)
                cpu_cluster_pm_exit();
        cpu_clear(cpu, cpu_idle_mask);
        return 0;
}
```

v7 shutdown next

Putting Everything Together (2/2)

```
int suspend_finisher(unsigned long arg)
{
        struct pm_pms *pp = (struct pm_pms *) arg;
        [...]
        smc_down(...);
        return 1;
}

smc_down:
        ldr r0, =#SMC_NUMBER
        smc #0
        /*
         * Pseudo code describing what secure world
         * should do
         */
        {
        disable_clean_inv_dcache_all();
        if (cluster->cluster_down && cluster->power_state == SHUTDOWN) {
                flush_cache_all();
                outer_flush_all();
        }
        normal_uncached_memory_lock();
        disable_cci_snoops();
        normal_uncached_memory_unlock();
        power_down_command();
        cpu_do_idle();
        }
```

Conclusion

- CPU idle core skewed towards per-CPU idle management
- Multi-cluster ARM systems require changes to core code and CPU idle core drivers
  - Synchronization algorithm
  - Next event management
  - Cache hierarchy and power domain linkage
- Effort will gain momentum as soon as big.LITTLE platforms start getting merged in the mainline
- Outlook
  - Consolidate next event management
  - Define Cluster states
  - Integrate ARM SMC proposal

THANKS !!!

BACK-UP

Gearing Idleness Towards Cluster States

- Race-to-idle
- Cluster states residency maximization
  - Concepts discussed at length but never implemented
- Improve governors patterns prediction to maximise cluster states residency [1]
- Leakage power on the rise, maximise idle time if deep C-states are enabled [2]
- Scheduler can definitely play a role
  - Scheduler knowledge of idle states
  - https://lkml.org/lkml/2012/8/13/139

[1] "Prediction of CPU idle-busy activity patterns", Q.Diao, J.Song

[2] "Processor Power Management features and Process Scheduler: Do we need to tie them together ?" V.Pallipadi, S.Siddha

CPU PM notifiers (1/3)

- Introduced by C.Cross to overcome code duplication in idle and suspend code path
- CPU events and CLUSTER events
- GIC, VFP, PMU

CPU PM notifiers (2/3)

```
static int cpu_pm_notify(enum cpu_pm_event event, int nr_to_call, int *nr_calls)
{
        int ret;

        ret = __raw_notifier_call_chain(&cpu_pm_notifier_chain, event, NULL,
                                nr_to_call, nr_calls);

        return notifier_to_errno(ret);
}

int cpu_pm_enter(void)
{
        [...]

        ret = cpu_pm_notify(CPU_PM_ENTER, -1, &nr_calls);
        if (ret)
                cpu_pm_notify(CPU_PM_ENTER_FAILED, nr_calls - 1, NULL);

        [...]

        return ret;
}

//CPU shutdown
cpu_pm_{enter,exit}();
//Cluster shutdown
cpu_cluster_pm_{enter,exit}();
```

CPU PM notifiers (3/3)

```
static int gic_notifier(struct notifier_block *self, unsigned long cmd, void *v)
{
        int i;

        [...]
        switch (cmd) {
        case CPU_PM_ENTER:
                gic_cpu_save(i);
                break;
        case CPU_PM_ENTER_FAILED:
        case CPU_PM_EXIT:
                gic_cpu_restore(i);
                break;
        case CPU_CLUSTER_PM_ENTER:
                gic_dist_save(i);
                break;
        case CPU_CLUSTER_PM_ENTER_FAILED:
        case CPU_CLUSTER_PM_EXIT:
                gic_dist_restore(i);
                break;
        }

        return NOTIFY_OK;
}

static struct notifier_block gic_notifier_block = {
        .notifier_call = gic_notifier,
};
```

v7 shutdown

Local timers save/restore

```
void enter_idle(...)
{
        [...]
        clockevents_notify(CLOCK_EVT_NOTIFY_BROADCAST_ENTER, &cpu);
        [...]
        cpu_do_idle();
        [...]
        clockevents_notify(CLOCK_EVT_NOTIFY_BROADCAST_EXIT, &cpu);
        [...]
}

void enter_idle(...)
{
        struct tick_device *tdev = tick_get_device(cpu);
        [...]
        cpu_do_idle();
        [...]
        /* Restore the per-cpu timer event */
        clockevents_program_event(tdev->evtdev, tdev->evtdev->next_event, 1);
}
```

- Enter broadcast mode if a global timer is available
- Rely on always-on firmware timer and restore timer through clock events programming API

ARM v7 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. call wfi (wait for interrupt)

ARM v7 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. call wfi (wait for interrupt)

ARM v7 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. call wfi (wait for interrupt)

ARM v7 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. call wfi (wait for interrupt)

ARM v7 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. call wfi (wait for interrupt)

ARM v7 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. call wfi (wait for interrupt)

This is the standard procedure that must be adopted by all platforms, for cpu switching, cpu hotplug (cache cleaning and wfi), suspend and idle

ARM v7 SMP CPU Shutdown Procedure

1. save per CPU peripherals (IC, VFP, PMU)
2. save CPU registers
3. clean L1 D$
4. clean state from L2
5. disable L1 D$ allocation
6. clean L1 D$
7. exit coherency
8. call wfi (wait for interrupt)

This is the standard procedure that must be adopted by all platforms, for cpu switching, cpu hotplug (cache cleaning and wfi), suspend and idle

idle

notifiers

CPU suspend (1/3)

- Introduced by R.King to consolidate existing (and duplicated) code across diffent ARM platforms
- save/restore core registers, clean L1 and some bits of L2
- L2 RAM retention handling poses further challenges

CPU suspend (2/3)

- 1:1 mapping page tables cloned from $\mathrm{init\_mm}$
- C API, generic for all ARM architectures

```
int cpu_suspend(unsigned long arg, int (*fn)(unsigned long))
{
        struct mm_struct *mm = current->active_mm;
        int ret;

        if (!suspend_pgd)
                return -EINVAL;

        [...]

        ret = __cpu_suspend(arg, fn);
        if (ret == 0) {
                cpu_switch_mm(mm->pgd, mm);
                local_flush_tlb_all();
        }

        return ret;
}
```

CPU suspend (3/3)

- registers saved on the stack

```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);
}
```

CPU suspend (3/3)

- registers saved on the stack
- L1 complete cleaning

```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);

        flush_cache_all();
}
```

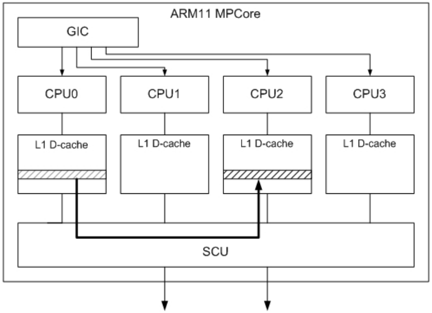
CPU suspend (3/3)

- registers saved on the stack
- L1 complete cleaning
- L2 partial cleaning

```
void __cpu_suspend_save(u32 *ptr, u32 ptrsz, u32 sp, u32 *save_ptr)
{
        *save_ptr = virt_to_phys(ptr);

        /* This must correspond to the LDM in cpu_resume() assembly */
        *ptr++ = virt_to_phys(suspend_pgd);
        *ptr++ = sp;
        *ptr++ = virt_to_phys(cpu_do_resume);

        cpu_do_suspend(ptr);

        flush_cache_all();
        outer_clean_range(*save_ptr, *save_ptr + ptrsz);
        outer_clean_range(virt_to_phys(save_ptr),
                          virt_to_phys(save_ptr) + sizeof(*save_ptr));
}
```

We Are Not Done, Yet: Cache-to-Cache migration

We Are Not Done, Yet: Cache-to-Cache migration



- SCU keeps a copy of D$ cache TAG RAMs
- To avoid data traffic ARM MPCore systems move dirty lines across cores
- Lower L1 bus traffic
- Dirty data might be fetched from another core during power-down sequence

We Are Not Done, Yet: Cache-to-Cache migration

- When the suspend finisher is called L1 is still allocating
- accessing current implies accessing the sp
- Snooping Direct Data Intervention (DDI), CPU might pull dirty line in

```
ENTRY(disable_clean_inv_dcache_v7_all)
        stmfd   sp!, {r4-r5, r7, r9-r11, lr}
        mrc     p15, 0, r3, c1, c0, 0
        bic     r3, #4                      @ clear C bit
        mcr     p15, 0, r3, c1, c0, 0
        isb

        bl      v7_flush_dcache_all
        mrc     p15, 0, r0, c1, c0, 1
        bic     r0, r0, #0x40               @ exit SMP
        mcr     p15, 0, r0, c1, c0, 1
        ldmfd   sp!, {r4-r5, r7, r9-r11, pc}
ENDPROC(disable_clean_inv_dcache_v7_all)
```

We Are Not Done, Yet: Cache-to-Cache migration

- When the suspend finisher is called L1 is still allocating
- accessing current implies accessing the sp
- Snooping Direct Data Intervention (DDI), CPU might pull dirty line in

```
ENTRY(disable_clean_inv_dcache_v7_all)
        stmfd   sp!, {r4-r5, r7, r9-r11, lr}
        mrc     p15, 0, r3, c1, c0, 0
        bic     r3, #4                          @ clear C bit
        mcr     p15, 0, r3, c1, c0, 0
        isb

        bl      v7_flush_dcache_all
        mrc     p15, 0, r0, c1, c0, 1
        bic     r0, r0, #0x40                   @ exit SMP
        mcr     p15, 0, r0, c1, c0, 1
        ldmfd   sp!, {r4-r5, r7, r9-r11, pc}
ENDPROC(disable_clean_inv_dcache_v7_all)
```

We Are Not Done, Yet: Cache-to-Cache migration

- When the suspend finisher is called L1 is still allocating
- accessing current implies accessing the sp
- Snooping Direct Data Intervention (DDI), CPU might pull dirty line in

```
ENTRY(disable_clean_inv_dcache_v7_all)
        stmfd   sp!, {r4-r5, r7, r9-r11, lr}
        mrc     p15, 0, r3, c1, c0, 0
        bic     r3, #4                          @ clear C bit
        mcr     p15, 0, r3, c1, c0, 0
        isb

        bl      v7_flush_dcache_all
        mrc     p15, 0, r0, c1, c0, 1
        bic     r0, r0, #0x40                   @ exit SMP
        mcr     p15, 0, r0, c1, c0, 1
        ldmfd   sp!, {r4-r5, r7, r9-r11, pc}
ENDPROC(disable_clean_inv_dcache_v7_all)
```

Outer Cache Management: The Odd One Out (1/2)

- L310 memory mapped device (aka outer cache)
- Clearing C bit does NOT prevent allocation
- L2 RAM retention, data sitting in L2, not accessible if MMU is off
- If not invalidated, L2 might contain stale data if resume code runs with L2 off before enabling it
- We could clean some specific bits: which ones ?
- If retained, L2 must be resumed before turning MMU on

Outer Cache Management: The Odd One Out (1/2)

- L310 memory mapped device (aka outer cache)
- Clearing C bit does NOT prevent allocation
- L2 RAM retention, data sitting in L2, not accessible if MMU is off
- If not invalidated, L2 might contain stale data if resume code runs with L2 off before enabling it
- We could clean some specific bits: which ones ?
- If retained, L2 must be resumed before turning MMU on

Outer Cache Management: The Odd One Out (2/2)

- if L2 content is lost, it must be cleaned on shutdown but can be resumed in C
- if L2 is retained, it must be resumed in assembly before calling cpu_resume

```
static void __init pl310_save(void)
{
        u32 l2x0_revision = readl_relaxed(l2x0_base + L2X0_CACHE_ID) &
                L2X0_CACHE_ID_RTL_MASK;

        l2x0_saved_regs.tag_latency = readl_relaxed(l2x0_base +
                L2X0_TAG_LATENCY_CTRL);
        l2x0_saved_regs.data_latency = readl_relaxed(l2x0_base +
                L2X0_DATA_LATENCY_CTRL);
        [...]
}

//asm-offsets.c
DEFINE(L2X0_R_PHY_BASE, offsetof(struct l2x0_regs, phy_base));
DEFINE(L2X0_R_AUX_CTRL, offsetof(struct l2x0_regs, aux_ctrl));
[...]
```