

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

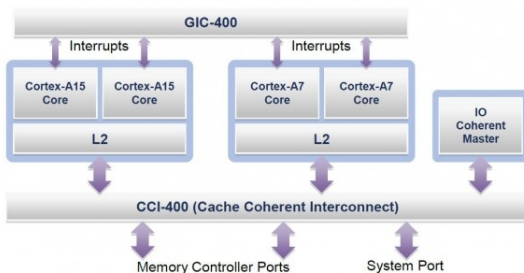
1 CPU idle for Clusters of CPUs

- 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 ARM's (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 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 `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



CPU idle cluster management back-end requirements

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
 - Idrex/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



CPU idle cluster management back-end requirements

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;
}

```

« v1 » « v2 » « v3 » « v4 » « v5 » « v6 » « v7 » « v8 » « v9 » « v10 » « v11 » « v12 » « v13 » « v14 » « v15 » « v16 » « v17 » « v18 » « v19 » « v20 » « v21 » « v22 » « v23 » « v24 » « v25 » « v26 » « v27 » « v28 » « v29 » « v30 » « v31 » « v32 » « v33 » « v34 » « v35 » « v36 » « v37 » « v38 » « v39 » « v40 » « v41 » « v42 » « v43 » « v44 » « v45 » « v46 » « v47 » « v48 » « v49 » « v50 » « v51 » « v52 » « v53 » « v54 » « v55 » « v56 » « v57 » « v58 » « v59 » « v60 » « v61 » « v62 » « v63 » « v64 » « v65 » « v66 » « v67 » « v68 » « v69 » « v70 » « v71 » « v72 » « v73 » « v74 » « v75 » « v76 » « v77 » « v78 » « v79 » « v80 » « v81 » « v82 » « v83 » « v84 » « v85 » « v86 » « v87 » « v88 » « v89 » « v90 » « v91 » « v92 » « v93 » « v94 » « v95 » « v96 » « v97 » « v98 » « v99 » « v100 » « v101 » « v102 » « v103 » « v104 » « v105 » « v106 » « v107 » « v108 » « v109 » « v110 » « v111 » « v112 » « v113 » « v114 » « v115 » « v116 » « v117 » « v118 » « v119 » « v120 » « v121 » « v122 » « v123 » « v124 » « v125 » « v126 » « v127 » « v128 » « v129 » « v130 » « v131 » « v132 » « v133 » « v134 » « v135 » « v136 » « v137 » « v138 » « v139 » « v140 » « v141 » « v142 » « v143 » « v144 » « v145 » « v146 » « v147 » « v148 » « v149 » « v150 » « v151 » « v152 » « v153 » « v154 » « v155 » « v156 » « v157 » « v158 » « v159 » « v160 » « v161 » « v162 » « v163 » « v164 » « v165 » « v166 » « v167 » « v168 » « v169 » « v170 » « v171 » « v172 » « v173 » « v174 » « v175 » « v176 » « v177 » « v178 » « v179 » « v180 » « v181 » « v182 » « v183 » « v184 » « v185 » « v186 » « v187 » « v188 » « v189 » « v190 » « v191 » « v192 » « v193 » « v194 » « v195 » « v196 » « v197 » « v198 » « v199 » « v200 » « v201 » « v202 » « v203 » « v204 » « v205 » « v206 » « v207 » « v208 » « v209 » « v210 » « v211 » « v212 » « v213 » « v214 » « v215 » « v216 » « v217 » « v218 » « v219 » « v220 » « v221 » « v222 » « v223 » « v224 » « v225 » « v226 » « v227 » « v228 » « v229 » « v230 » « v231 » « v232 » « v233 » « v234 » « v235 » « v236 » « v237 » « v238 » « v239 » « v240 » « v241 » « v242 » « v243 » « v244 » « v245 » « v246 » « v247 » « v248 » « v249 » « v250 » « v251 » « v252 » « v253 » « v254 » « v255 » « v256 » « v257 » « v258 » « v259 » « v260 » « v261 » « v262 » « v263 » « v264 » « v265 » « v266 » « v267 » « v268 » « v269 » « v270 » « v271 » « v272 » « v273 » « v274 » « v275 » « v276 » « v277 » « v278 » « v279 » « v280 » « v281 » « v282 » « v283 » « v284 » « v285 » « v286 » « v287 » « v288 » « v289 » « v290 » « v291 » « v292 » « v293 » « v294 » « v295 » « v296 » « v297 » « v298 » « v299 » « v300 » « v301 » « v302 » « v303 » « v304 » « v305 » « v306 » « v307 » « v308 » « v309 » « v310 » « v311 » « v312 » « v313 » « v314 » « v315 » « v316 » « v317 » « v318 » « v319 » « v320 » « v321 » « v322 » « v323 » « v324 » « v325 » « v326 » « v327 » « v328 » « v329 » « v330 » « v331 » « v332 » « v333 » « v334 » « v335 » « v336 » « v337 » « v338 » « v339 » « v340 » « v341 » « v342 » « v343 » « v344 » « v345 » « v346 » « v347 » « v348 » « v349 » « v350 » « v351 » « v352 » « v353 » « v354 » « v355 » « v356 » « v357 » « v358 » « v359 » « v360 » « v361 » « v362 » « v363 » « v364 » « v365 » « v366 » « v367 » « v368 » « v369 » « v370 » « v371 » « v372 » « v373 » « v374 » « v375 » « v376 » « v377 » « v378 » « v379 » « v380 » « v381 » « v382 » « v383 » « v384 » « v385 » « v386 » « v387 » « v388 » « v389 » « v390 » « v391 » « v392 » « v393 » « v394 » « v395 » « v396 » « v397 » « v398 » « v399 » « v400 » « v401 » « v402 » « v403 » « v404 » « v405 » « v406 » « v407 » « v408 » « v409 » « v410 » « v411 » « v412 » « v413 » « v414 » « v415 » « v416 » « v417 » « v418 » « v419 » « v420 » « v421 » « v422 » « v423 » « v424 » « v425 » « v426 » « v427 » « v428 » « v429 » « v430 » « v431 » « v432 » « v433 » « v434 » « v435 » « v436 » « v437 » « v438 » « v439 » « v440 » « v441 » « v442 » « v443 » « v444 » « v445 » « v446 » « v447 » « v448 » « v449 » « v450 » « v451 » « v452 » « v453 » « v454 » « v455 » « v456 » « v457 » « v458 » « v459 » « v460 » « v461 » « v462 » « v463 » « v464 » « v465 » « v466 » « v467 » « v468 » « v469 » « v470 » « v471 » « v472 » « v473 » « v474 » « v475 » « v476 » « v477 » « v478 » « v479 » « v480 » « v481 » « v482 » « v483 » « v484 » « v485 » « v486 » « v487 » « v488 » « v489 » « v490 » « v491 » « v492 » « v493 » « v494 » « v495 » « v496 » « v497 » « v498 » « v499 » « v500 » « v501 » « v502 » « v503 » « v504 » « v505 » « v506 » « v507 » « v508 » « v509 » « v510 » « v511 » « v512 » « v513 » « v514 » « v515 » « v516 » « v517 » « v518 » « v519 » « v520 » « v521 » « v522 » « v523 » « v524 » « v525 » « v526 » « v527 » « v528 » « v529 » « v530 » « v531 » « v532 » « v533 » « v534 » « v535 » « v536 » « v537 » « v538 » « v539 » « v540 » « v541 » « v542 » « v543 » « v544 » « v545 » « v546 » « v547 » « v548 » « v549 » « v550 » « v551 » « v552 » « v553 » « v554 » « v555 » « v556 » « v557 » « v558 » « v559 » « v560 » « v561 » « v562 » « v563 » « v564 » « v565 » « v566 » « v567 » « v568 » « v569 » « v570 » « v571 » « v572 » « v573 » « v574 » « v575 » « v576 » « v577 » « v578 » « v579 » « v580 » « v581 » « v582 » « v583 » « v584 » « v585 » « v586 » « v587 » « v588 » « v589 » « v590 » « v591 » « v592 » « v593 » « v594 » « v595 » « v596 » « v597 » « v598 » « v599 » « v600 » « v601 » « v602 » « v603 » « v604 » « v605 » « v606 » « v607 » « v608 » « v609 » « v610 » « v611 » « v612 » « v613 » « v614 » « v615 » « v616 » « v617 » « v618 » « v619 » « v620 » « v621 » « v622 » « v623 » « v624 » « v625 » « v626 » « v627 » « v628 » « v629 » « v630 » « v631 » « v632 » « v633 » « v634 » « v635 » « v636 » « v637 » « v638 » « v639 » « v640 » « v641 » « v642 » « v643 » « v644 » « v645 » « v646 » « v647 » « v648 » « v649 » « v650 » « v651 » « v652 » « v653 » « v654 » « v655 » « v656 » « v657 » « v658 » « v659 » « v660 » « v661 » « v662 » « v663 » « v664 » « v665 » « v666 » « v667 » « v668 » « v669 » « v670 » « v671 » « v672 » « v673 » « v674 » « v675 » « v676 » « v677 » « v678 » « v679 » « v680 » « v681 » « v682 » « v683 » « v684 » « v685 » « v686 » « v687 » « v688 » « v689 » « v690 » « v691 » « v692 » « v693 » « v694 » « v695 » « v696 » « v697 » « v698 » « v699 » « v700 » « v701 » « v702 » « v703 » « v704 » « v705 » « v706 » « v707 » « v708 » « v709 » « v710 » « v711 » « v712 » « v713 » « v714 » « v715 » « v716 » « v717 » « v718 » « v719 » « v720 » « v721 » « v722 » « v723 » « v724 » « v725 » « v726 » « v727 » « v728 » « v729 » « v730 » « v731 » « v732 » « v733 » « v734 » « v735 » « v736 » « v737 » « v738 » « v739 » « v740 » « v741 » « v742 » « v743 » « v744 » « v745 » « v746 » « v747 » « v748 » « v749 » « v750 » « v751 » « v752 » « v753 » « v754 » « v755 » « v756 » « v757 » « v758 » « v759 » « v760 » « v761 » « v762 » « v763 » « v764 » « v765 » « v766 » « v767 » « v768 » « v769 » « v770 » « v771 » « v772 » « v773 » « v774 » « v775 » « v776 » « v777 » « v778 » « v779 » « v780 » « v781 » « v782 » « v783 » « v784 » « v785 » « v786 » « v787 » « v788 » « v789 » « v790 » « v791 » « v792 » « v793 » « v794 » « v795 » « v796 » « v797 » « v798 » « v799 » « v800 » « v801 » « v802 » « v803 » « v804 » « v805 » « v806 » « v807 » « v808 » « v809 » « v810 » « v811 » « v812 » « v813 » « v814 » « v815 » « v816 » « v817 » « v818 » « v819 » « v820 » « v821 » « v822 » « v823 » « v824 » « v825 » « v826 » « v827 » « v828 » « v829 » « v830 » « v831 » « v832 » « v833 » « v834 » « v835 » « v836 » « v837 » « v838 » « v839 » « v840 » « v841 » « v842 » « v843 » « v844 » « v845 » « v846 » « v847 » « v848 » « v849 » « v850 » « v851 » « v852 » « v853 » « v854 » « v855 » « v856 » « v857 » « v858 » « v859 » « v860 » « v861 » « v862 » « v863 » « v864 » « v865 » « v866 » « v867 » « v868 » « v869 » « v870 » « v871 » « v872 » « v873 » « v874 » « v875 » « v876 » « v877 » « v878 » « v879 » « v880 » « v881 » « v882 » « v883 » « v884 » « v885 » « v886 » « v887 » « v888 » « v889 » « v890 » « v891 » « v892 » « v893 » « v894 » « v895 » « v896 » « v897 » « v898 » « v899 » « v900 » « v901 » « v902 » « v903 » « v904 » « v905 » « v906 » « v907 » « v908 » « v909 » « v910 » « v911 » « v912 » « v913 » « v914 » « v915 » « v916 » « v917 » « v918 » « v919 » « v920 » « v921 » « v922 » « v923 » « v924 » « v925 » « v926 » « v927 » « v928 » « v929 » « v930 » « v931 » « v932 » « v933 » « v934 » « v935 » « v936 » « v937 » « v938 » « v939 » « v940 » « v941 » « v942 » « v943 » « v944 » « v945 » « v946 » « v947 » « v948 » « v949 » « v950 » « v951 » « v952 » « v953 » « v954 » « v955 » « v956 » « v957 » « v958 » « v959 » « v960 » « v961 » « v962 » « v963 » « v964 » « v965 » « v966 » « v967 » « v968 » « v969 » « v970 » « v971 » « v972 » « v973 » « v974 » « v975 » « v976 » « v977 » « v978 » « v979 » « v980 » « v981 » « v982 » « v983 » « v984 » « v985 » « v986 » « v987 » « v988 » « v989 » « v990 » « v991 » « v992 » « v993 » « v994 » « v995 » « v996 » « v997 » « v998 » « v999 » « v1000 »





CPU idle cluster management back-end requirements

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 different 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 `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_pgdt)
        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);

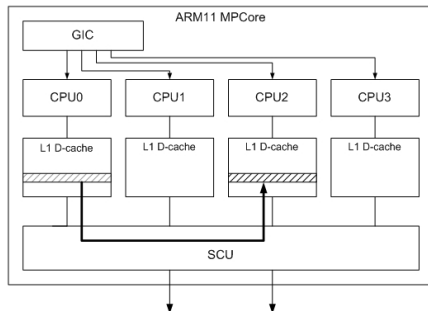
    flush_cache_all();
    outer_clean_range(*save_ptr, *save_ptr + ptrsz);
    outer_clean_range(virt_to_phys(save_ptr),
                      virt_to_phys(save_ptr) + sizeof(*save_ptr));
}
```



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));
[...]
```

