ARM

DynamIQ power management support

Version 1.1

Revision Information

The following revisions have been made to this document.

Date	Version	Confidentiality	Change
31 March 2017	1.0	Non-confidential	Initial version of the document
30 October 2017	1.1	Non-confidential	DSU L3 partial power-down update DynamIQ energy model guidelines

Confidentiality Status

This document is Non-Confidential.

Product Status

The information in this document is final, that is for a developed product.

Web Address

http://www.arm.com

ARM

Contents

1 EAS support for DynamIQ

1.1 Introduction

1.1.1 Overview

DynamIQ flexible microarchitecture introduces the ability to have a cluster of up to 8 cores of different microarchitecture & implementation, each with optional Level-2 cache, sharing a common Level-3 cache. The L3 cache in the ARM DynamIQ Shared Unit (DSU) supports partial power down which is detailed in another chapter below - DynamIQ Shared Unit Level-3 cache partial power down.

EAS extends the Linux kernel scheduler to make it aware of the power/performance capabilities of the cores in the system, to optimize energy consumption for advanced multi-core SoCs including big.LITTLE. With EAS, the Linux kernel will use the Energy Model and the task utilization to control task placement to select the optimal (from the perspective of energy consumption) core to run on, when its actions do not violate performance requirements.

1.1.2 Challenges DynamIQ imposes on EAS

1.1.2.1 CPU Topology

A cluster of cores is determined by the physical boundary of the last level cache these cores are sharing. This is called the Cluster Physical Domain (PhyD). Each core by itself forms the Core PhyD.

If the system contains one cluster, the task Scheduler Domain (SD) hierarchy is modelled as one SD level (called Multi-Cluster level (MC), in which all cpus (as a destination for tasks) can perform load balance operations against every CPU (as the source of tasks) independently.

In today's multi-core systems, the Power Management (PM) domains, like Frequency Domain (FD), Voltage Domain (VD) or Power Domain (PwrD) are normally congruent to the PhyDs.

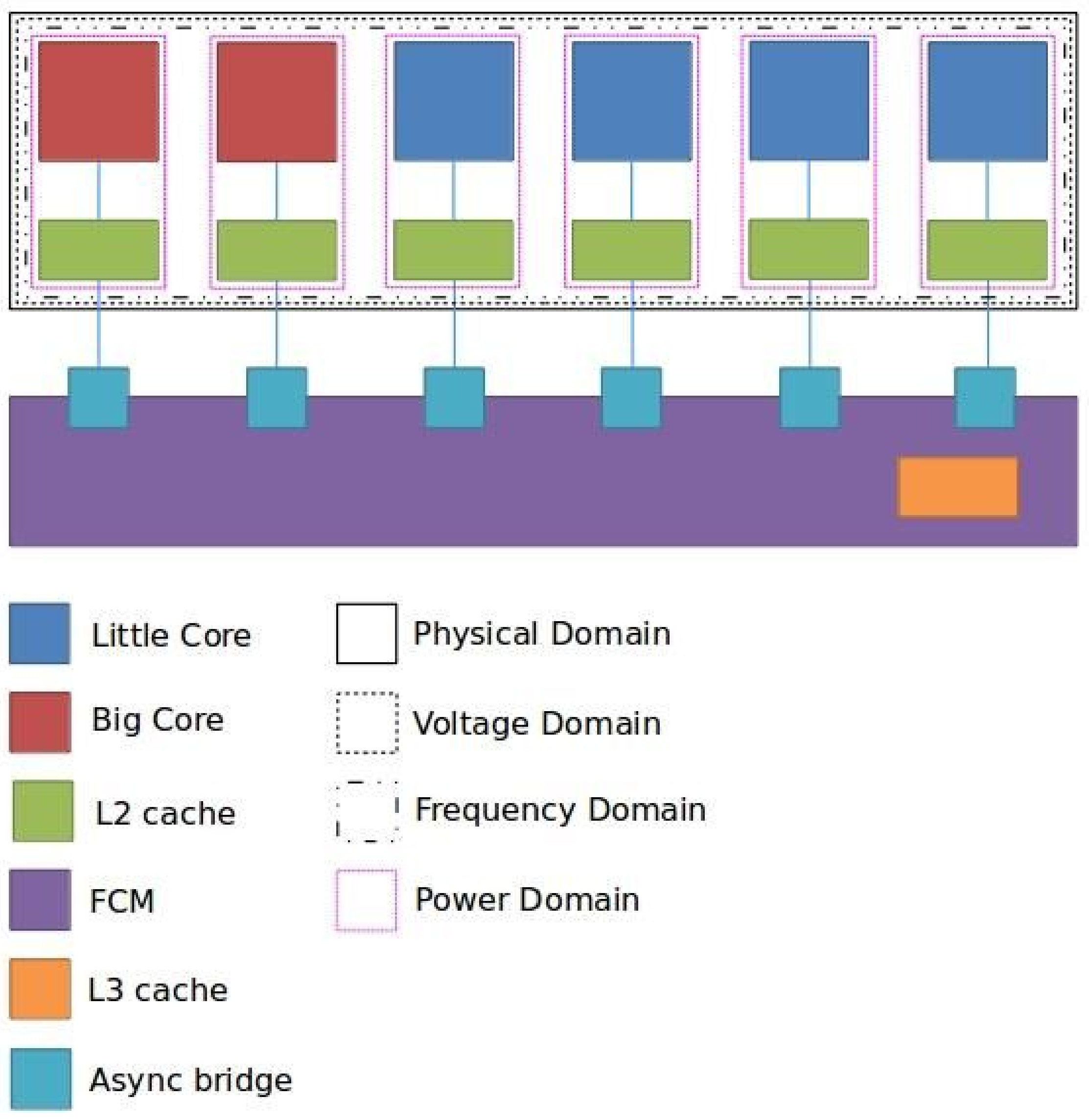


Figure 1 - 2 Big/4 Little DynamIQ system with congruent PhyD and PM domains

Figure 1 depicts a 2 big/4 little DynamIQ system where this is the case. The FD and the VD are congruent with the Cluster PhyD, so all cores are sharing the FD and the VD. The PwrD is congruent with Core PhyD. This is a rather theoretical DynamIQ system layout because it implies that Big and Little cores have to use the same Operating Performance Points (OPPs).

Its EAS Energy Model (EM) would only consist of core related EM data on a single SD level (so called MC level). Figure 2 shows the resulting SD hierarchy from the viewpoint of core 0 (Big core).

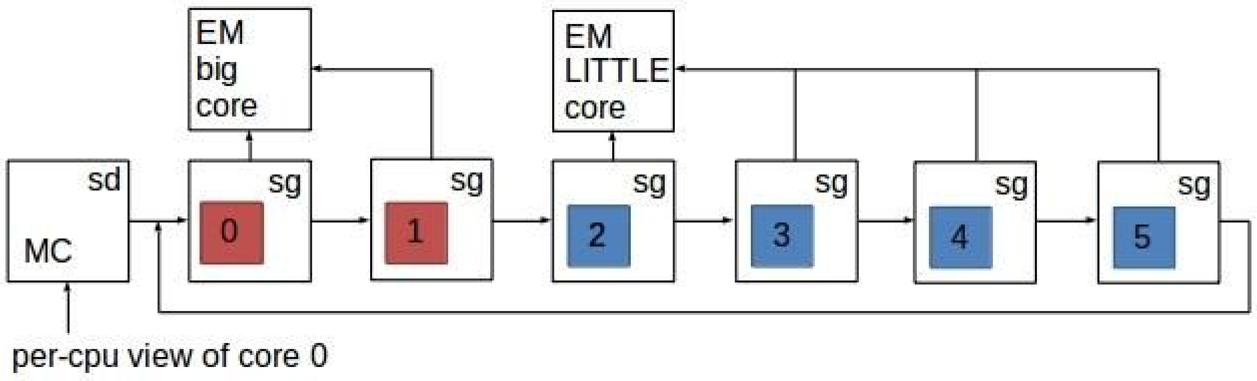


Figure 2 SD hierarchy of a 2 big/4 little DynamIQ system with congruent PhyD and PM domains

If the assumption that the PM domains are congruent with the PhyDs does not hold, the modelling of EAS EM data becomes more complicated.

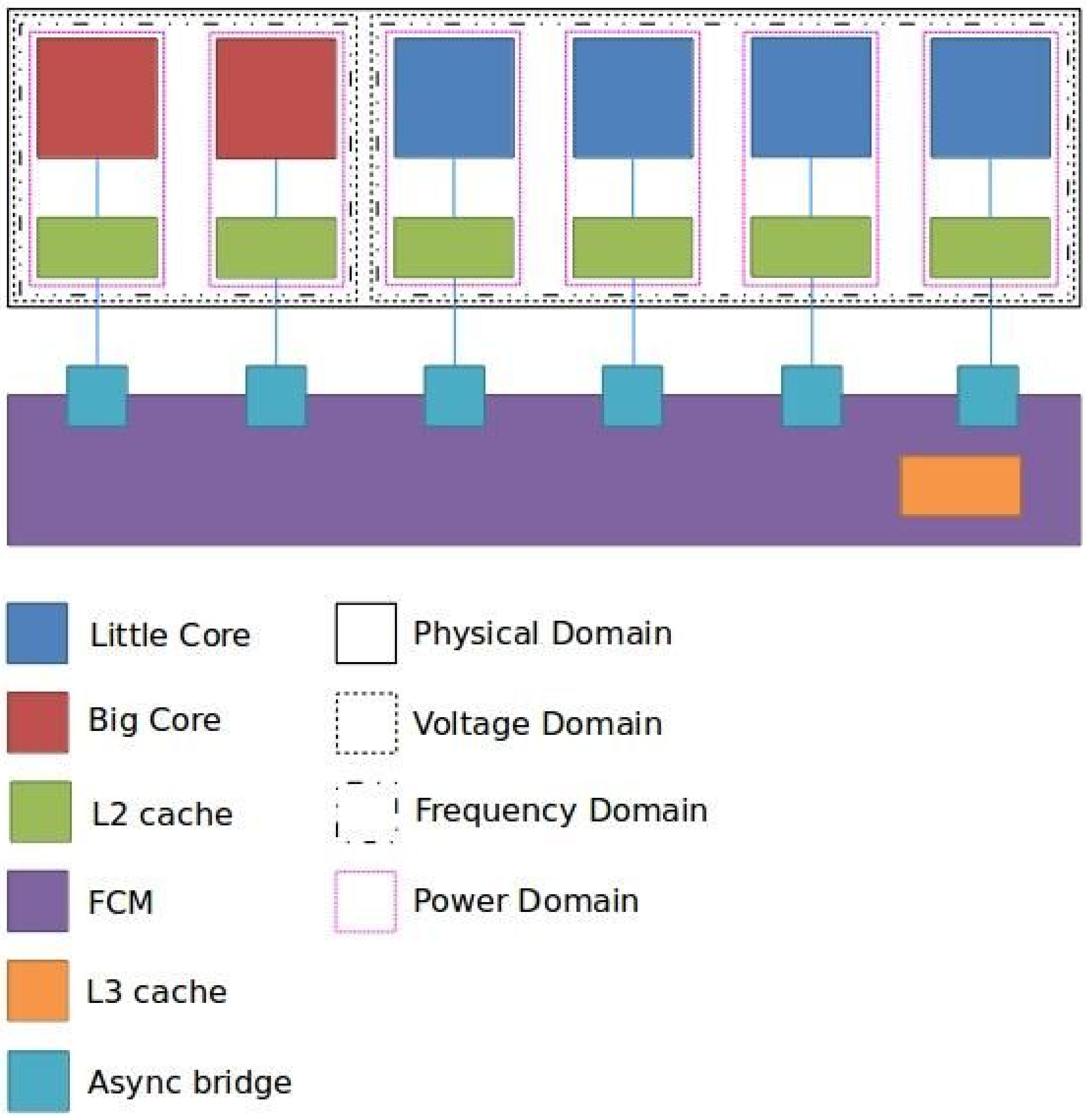


Figure 3 - 2 Big/4 Little DynamIQ system with non-congruent PhyD and PM domains

Figure 3 depicts a 2 big/4 little DynamIQ system where the Big cores share FD/VD and the Little cores share FD/VD. To model this within EAS another SD level (so called DIE level) has to be introduced in which the group of cores sharing FD/VD (0,1 and 2-5) are represented as Scheduler Group (SG). This is necessary because EAS has to attach EM data which relates to these groups of cores (clusters) to these SG's. Figure 4 shows the resulting SD hierarchy from the viewpoint of core 0 (Big core).

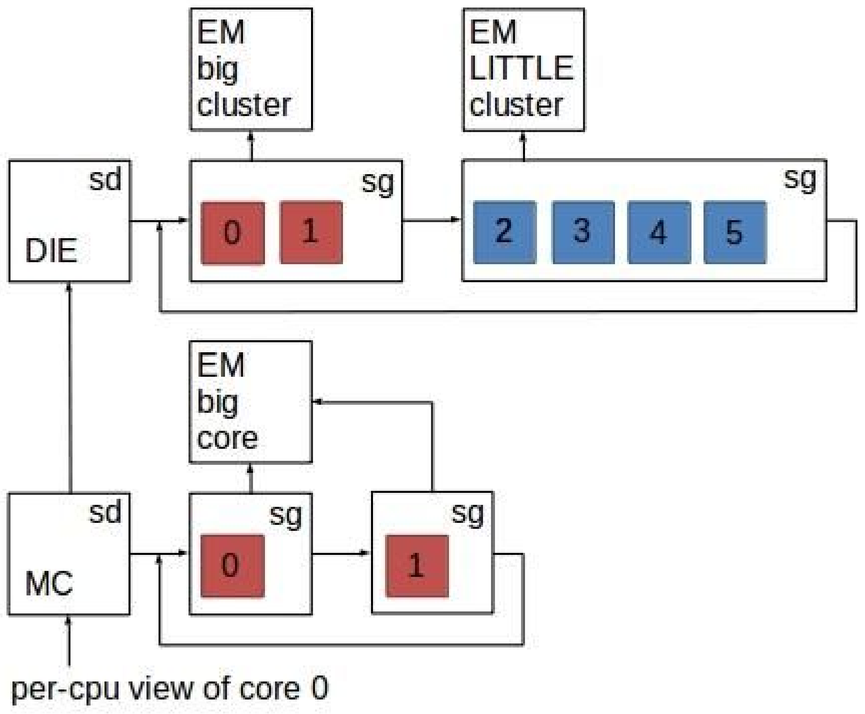


Figure 4 SD hierarchy of a 2 Big/4 Little DynamIQ system with non-congruent PhyD and PM domains

The challenge DynamIQ imposes on EAS is the fact that DynamIQ topology layouts in terms of PM Domain (PMD) related boundaries are not necessarily congruent with the PhyD boundaries that the task scheduler in the current Linux kernel models its SD hierarchy upon.

A solution has to be found to model non-congruent PMDs so that they become visible to the EAS task scheduler.

1.1.2.2 Last Level Cache

In case the SD DIE level is based on PMDs rather than PhyD boundaries, the semantics of the Last Level Cache (LLC) related data in the task scheduler might be wrong.

EAS must take into account the effect this discrepancy has on task scheduler operation.

1.1.2.3 EAS EM data versus SD hierarchy bring-up optimization

For EAS to work properly it relies on EM data available on all SD hierarchy levels which are affected by EAS.

Under certain circumstances the current Linux task scheduler would not create the full SD hierarchy because from a pure scheduling perspective not every level would be necessary. E.g. if there is only one core (due to hotplug or topology layout, such as one Big core only) in one cluster of a multi-cluster system, the scheduler won't create the MC SD level for this core because the fundamental scheduling decision (take this core or another core) can't be made on this SD level.

EAS must be able to operate on a system with only one incarnation of a core type (e.g. Big or Little).

1.1.2.4 PM Domain Sharing between Cluster & DSU

In case one of the clusters shares PMDs (VD/FD or only VD) with the DSU, i.e. the cluster runs in lock-step with the DSU, there are DVFS-related dependencies between the other cluster and the DSU. These dependencies have to be controlled; otherwise there could be scenarios where cores on the cluster which doesn't share PMDs with the DSU are throttled by a DSU running on a low OPP.

Software must manage this DVFS dependency to avoid such throttling scenarios.

1.2 Detailed Analysis

1.2.1 CPU Topology

CPU topology is defined as the hierarchy of (hardware) threads (logical cpus in scheduler parlance) which are grouped into cores, with cores being grouped into clusters (of cores). In the absence of threads (no SMT), a core is a logical CPU.

A cluster represents a child node of the cpu-map node which for ARM defines the CPU topology. The cpu-map node is a direct child of the CPU node. The cpu-map node's child nodes can be one or more cluster nodes.

It is not specified what determines the boundary of a cluster, although in existing systems the cluster boundary is assumed to be congruent with the shared cache layout (PhyD).

Other possible cluster boundaries from the perspective of PM are:

- Frequency Domain (FD)
- Voltage Domain (VD)
- Power Domain (PwrD)

Big.LITTLE creates another possible cluster boundary:

- Micro Architecture Domain (MAD)

On existing platforms, all of these domains are normally congruent to either the Core or Cluster PhyD. Moreover, since it makes sense to operate cores with different microarchitectural characteristics in different OPP ranges, it is very likely that MAD is congruent to the VD/FD.

EAS operates on the Energy Model (EM) data which consists of the following PM related information:

1. Frequency state (OPP) – power consumption
   For groups of cores, this information is related to FD and VD as well as MAD.

2. Power state – power consumption
   For groups of cores, this information is related to the PwrD.

The OPP is directly linked to the CPU capacity. On Big.LITTLE systems the CPU capacity is also related to the MAD.

1.2.1.1 Linkage between Energy Model and Scheduler Domain hierarchy

The EM data can be expressed per core and per cluster of cores. This data is then mapped to a SG in different Scheduling Domains (SDs). On a classical system, which only considers PhyDs, the core EM data is mapped to the SG on SD level MC and the cluster EM data is mapped to a SG on SD level DIE.

For a one cluster DynamIQ system there is no SD DIE level from the viewpoint of the PhyD. But in case there are n subgroups of cores in this DynamIQ cluster sharing e.g. FD/VD, those subgroups of cores can be represented as SGs on the SD DIE level. There are different ways to map these PMDs into CPU topology functionality. The CPU topology functionality determines which SD levels are created.

1.2.1.2 Ways to map PM Domains into CPU topology functionality

1. Phantom Domains

   This approach would use the existing cpu_map to specify PMDs as clusters. The word phantom here refers to the fact that those domains are not real cluster PhyDs.

   The advantage is that the existing cpu topology related data representations `struct cpu_topolgy` and DT parsing code `parse_dt_topology()` can be reused.

   The disadvantage is that only one of these other possible PMDs (congruent VP/FD and possibly MAD as well, as the presumably most common example) can be represented. Representing multiple of them would require using nested clusters in cpu_map which requires changing the parsing code (to not flatten out the cluster information) and to enhance the existing data representation.

2. Using PMD specific DT information

   There are DT bindings from which PMDs could be constructed.

● Frequency Domain	- operating-points-v2
● Voltage Domain	- operating-points-v2
● Power Domain	- power-domains (pm_domain) (not mainline yet)

   There is also DT binding which takes MAD via the CPU capacity specification into consideration:

● Cpu capacity Domain	- capacity-dmips-mhz

   This approach would require creating an enhanced dynamic data representation of CPU topology and more sophisticated DT parsing code for the ARM/ARM64 architecture.

   The actual mapping of non-congruent PMDs into SD levels has to be further analysed.

1.2.2 Last Level Cache

If subgroups of cores represent PMDs which are non-congruent to the (phantom) PhyDs the assumption that the highest SD level with the SD flag `SD_SHARE_PKG_RESOURCES` (0x200) set is the one in which cpus share their Last Level Cache (LLC) breaks. This is because the cores belonging to the PhyD (the DynamIQ cluster) would share the L3 cache as their LLC.

On a traditional two cluster big.LITTLE system `SD_SHARE_PKG_RESOURCES` is currently set on the MC SD level, indicating that cpus of a cluster (PhyD, as well as congruent VD/FD and MAD) share the L2 cache as their LLC.

There are three LLC per-cpu data elements in the Linux task scheduler code:

1. `DEFINE_PER_CPU(struct sched_domain *,` **`sd_llc`**`)`
   Pointer to the highest SD that has SD_SHARE_PKG_RESOURCE set
2. `DEFINE_PER_CPU(int,` **`sd_llc_size`**`)`
   Number of cpus in this SD
3. `DEFINE_PER_CPU(int,` **`sd_llc_id`**`)`
   Unique ID (first CPU number in the cpumask of the SD)

There is another LLC related per-cpu data element:

1. `DEFINE_PER_CPU(struct sched_domain *,` **`sd_busy`**`)`
   Pointer to the parent SD of `sd_llc`

The LLC information is used in the following functional areas:

1. wakeup path (`select_idle_sibling()`)
   This wakeup path is not active if EAS operates in non-over-utilized mode.
2. want_affine logic in wakeup path (`wake_wide()`)
3. test if (two) CPUs share the LLC (`cpus_share_cache()`)
4. nohz kick handling (`set_cpu_sd_state_busy()`, `set_cpu_sd_state_idle()`, `nohz_kick_needed()`)
   Nohz kick needed based on number of busy cores is disabled in case EAS is enabled.

So far the semantics of `SD_SHARE_PKG_RESOURCES` constrain all cores of the SD to share the LLC as the common Package (refers to Cluster/Domain) Resource. Since on traditional two cluster big.LITTLE systems, the Cluster PhyD is congruent to the FD/VD and MAD, these cores also share the same maximum (or original) CPU Capacity.

To strictly comply with the semantics, the highest SD for `SD_SHARE_PKG_RESOURCE` would have to be the DIE SD level for a one cluster DynamIQ system. This would require some minor changes in `arch/$ARCH/kernel/topology.c` related to how `SD_SHARE_PKG_RESOURCES` is set via the `sched_domain_flags_f` function pointer in in the $ARCH_topology[] table.

The semantics of `SD_SHARE_PKG_RESOURCES` make the cores share the same maximum CPU Capacity together with the LLC on a system with congruent FD/VDs and MADS, and this is what shapes the functional behaviour. Since maximum frequency and microarchitecture influence the maximum CPU capacity, a congruent FD/VD and MAD can be defined as (maximum) CPU Capacity Domain (CCD).

1. wakeup path
   The meaning of sibling CPU (target CPU) would be sharing the maximum CPU capacity with the previous CPU. The LLC is still shared between target and previous CPU because CCD is a subset of the LLC Domain.

2. want_affine logic in wakeup path
   The decision to spread or consolidate tasks would change from inside the LLC domain to inside the CCD.

3. test if two CPUs share the LLC
   a. `select_idle_sibling()`
      The decision to return an idle previous CPU would change from doing so if it shares the LLC with the Current CPU (CPU the scheduler is currently running on) to if it shares the CCD with the current CPU.

   b. `ttwu_queue()`
      The decision to queue remote wakeups on the CPU and process them using the scheduler IPI would change from doing so if the CPU does not share the LLC with the Current CPU to if the CPU does not share the CCD with the current CPU. Since the scheduler feature `TTWU_QUEUE` is enabled by default, this might slightly increase rq->lock contention/bounces.

   c. Block IO scheduler (softirq and mutliqueue)
      The decision to run the softirq/completion on the Current CPU would change from doing so if the CPU shares the LLC with the Current CPU to if the CPU shares the CCD with the current CPU.

4. nohz kick handling
   The number of busy cores is maintained per SG on the SD sd_busy. By leaving the SD flag `SD_SHARE_PKG_RESOURCE` on MC SD level, `nohz_kick_needed()` would kick the idle load balancer in the presence of an idle core in the system if there is more than one busy core among those cores being represented by the SG on SD `sd_busy`. If the SD flag `SD_SHARE_PKG_RESOURCE` would be set on SD DIE level instead, there isn't any nohz kick handling based on the number of busy CPUs since `sd_busy` would be NULL (there is no parent SD for SD DIE level).

1.2.3 EAS EM data versus SD hierarchy bring-up optimization

For EAS to work properly, even in the case that there is only one core per cluster or that cores are hot-plugged out, the EM data on all energy-aware SDs has to be present for all online cpus. Current mainline Linux Task Scheduler code will remove SDs which are not useful for task scheduling e.g. in the following situations:

1. Only one core is/remains in one cluster of a multi cluster system.
   This remaining core only has DIE and no MC SD level.
2. A complete cluster in a two cluster system is hot-plugged out.
   The cores of the remaining cluster only have MC and no DIE SD level.

To make sure that all online cores keep all their energy-aware SDs the patch ‘sched: EAS & 'single cpu per cluster'/cpu hotplug interoperability’ changes the SD degenerate functionality to not free a SD if its first SG contains EM data in case:

1. There is only one core left in the SD.
2. There have to be at least two SGs if certain SD flags are set.

Further details about the functionality are available in the patch header. The patch is part of EASv1.2.

1.2.4 PM Domain Sharing between Cluster & DSU

DynamIQ supports system configurations in which a cluster shares PMDs with the DSU. The following configurations are possible:

1. Big core cluster shares FD/VD with DSU (Big Sync)
2. Little core cluster shares FD/VD with DSU (Little Sync)
3. Big core cluster shares VD with DSU (Voltage Big Sync)
4. Little core cluster shares VD with DSU (Voltage Little Sync)

1.2.4.1 Big Sync and Little Sync

The cluster which runs synchronously with the DSU is called the Sync-Cluster. The cluster which runs asynchronously with the DSU is called the Async-Cluster.

The EAS requirement which breaks on such a system is:

> The Frequency Invariance Engine (FIE) on each core needs to know the correct current frequency value to be able to provide a correct frequency scaling factor. This frequency scaling factor would be wrong if the DFVS system is not informed about the fact that the core is forced to run at a higher OPP from the outside.

The following use-case would be harmful for EAS functionality to work properly:

> If the Async-Cluster runs at an OPP_{ac} which would require that DSU, and therefore the Sync-Cluster, run at an OPP_{sc2} which is higher than the OPP_{sc1} the Sync-Cluster is currently running on.

The EAS requirement can still be met by a DFVS based notification mechanism from the Async-Cluster to the DFVS subsystem of the Sync-Cluster to inform about this higher OPP so that the FIE on the Sync-Cluster can deliver the correct frequency scaling factor.

It has to be further investigated if such an inter-cluster DFVS based notification mechanism has to be implemented or if the error introduced by not controlling this dependency is insignificant for the EAS functionality.

Possible issues with the accuracy of the PELT signals due to OPP changes imposed by the inter-cluster notification mechanism need to be investigated further.

1.2.4.2 Voltage Big Sync and Voltage Little Sync

The cluster which shares the VD with the DSU is called the Voltage-Shared-Cluster. The cluster which doesn't share the VD with the DSU is called the Voltage-Unshared-Cluster.

The same problems as for the Sync systems would occur for the Voltage Sync systems.

The solution to fix them would be more complicated because a DVFS based notification scheme wouldn't be sufficient any more. The EM would need 3 dimensional capacity state vectors which would allow EAS to figure out the correct power consumption at the Voltage-Shared-Cluster since a possible scenario would be that the Voltage-Shared-Cluster runs on OPP_{vsc1} but the DSU is forced by the Voltage-Unshared-Cluster to run at OPP_{dsu1} which uses V_{dsu1} which is greater than the Voltage V_{vsc1} of OPP_{vsc1}. The actual power consumption of the Voltage-Shared-Cluster would be higher than the value in the EM capacity state vector element for OPP_{vsc1}

It has to be further investigated if this 3 dimensional capacity state vector for the EM has to be provisioned or if the error introduced by using a classical EM is insignificant for the EAS functionality.

1.2.5 Per-core DVFS

In terms of per-core DVFS support for DynamIQ there are two different possible configurations to consider.

1.2.5.1 Single core in MAD with per-core DVFS topology

This depicts a special form of per-core DVFS since there is only one core in a domain. The MC SD for such a core has only one SG containing this core.

The EAS SD flag `SD_SHARE_CAP_STATES` indicates that the SGs (the cores) share the FD/VD. This SD flag is currently set statically in `arch/$ARCH/kernel/topology.c`.

In case the MC SD contains only one SG for this single core, the SD flag is still valid, since the single core shares the FD/VD with itself.

1.2.5.2 Two or more cores in a MAD with per-core DVFS topology

In this case the setup code in `arch/$ARCH/kernel/topology.c` would have to be extended to be able to distinguish during start-up (possibly from device tree) that a core has per-cluster DVFS (i.e. set `SD_SHARE_CAP_STATES` on MC SD) or per-core DVFS (i.e. do not set `SD_SHARE_CAP_STATES` on MC SD) capability.

There is a solution without the change in the setup code for this because these MADs are essentially Phantom Domains inside the DynamIQ cluster. By creating n single Phantom Domains (core MC SDs) for these n cores in the MAD the topology would be changed into a 'Single core in MAD with per-core DVFS' topology.

Example: 2 Big (with per-core DVFS) + 2 Little (with per-cluster DVFS)

```
Default:

cpu-map {
        cluster0 {
                core0 {
                        cpu = <&Big_0>;
                };
                core1 {
                        cpu = <&Big_1>;
                };
        };
        cluster1 {
                core0 {
                        cpu = <&Little_0>;
                };
                core1 {
                        cpu = <&Little_1>;
                };
        };
};

Solution:

cpu-map {
        cluster0 {
                core0 {
                        cpu = <&Big_0>;
                };
        };
        cluster1 {
                core0 {
                        cpu = <&Big_1>;
                };
        };
        cluster2 {
                core0 {
                        cpu = <&Little_0>;
                };
                core1 {
                        cpu = <&Little_1>;
                };
        };
};
```

1.3 DynamIQ Architectural Property vs Linux Task Scheduler/EAS Functionality

DynamIQ Architectural Property	Linux Task Scheduler/EAS Functionality	DynamIQ Support Plan	Reference to Detailed Analysis chapter
Sub-clustering of cores due to multiple MADs and or PMDs inside a DynamIQ cluster	Create Phantom domains	Basic Support	1.2.1 CPU Topology
No L2 cache for some of the cores	Scheduler does not model L2 if it is not LLC. Considered to be second-order effect	Basic Support	1.2.2 Last Level Cache
Different L2 cache size between cores in a PMD	Scheduler does not model L2 if it is not LLC	Basic Support	1.2.2 Last Level Cache
Single core in a PMD	SD of a single core is collapsed in existing Linux scheduler. Need to allow single core in a DynamIQ cluster as a Phantom domain and also 'hotplug' a cluster down to a single core scenario.	Basic Support	1.2.3 EAS EM data versus SD hierarchy bring-up optimization
PMD Sharing between Cluster & DSU	Functionality outside the Scheduler. Currently not supported in EAS.	Future Support	1.2.4 PM Domain Sharing between Cluster & DSU
Per-core DVFS	Supported for Single core in MAD Solution for [2..*] cores in MAD	Basic Support	1.2.5 Per-core DVFS
Multiple DynamIQ clusters (i.e. system with > 8 cores)	Would require an extra SD on top of SD DIE level	Future Support	Tbd.

Table 1 DynamIQ Architectural Property vs Linux Task Scheduler/EAS Functionality mapping

1.4 DynamIQ EAS Software Recommendations

1.4.1 Basic Support

1.4.1.1 CPU Topology

The device tree cpu-map node would have to be setup to allow the scheduler to create SGs on MC and DIE SD hierarchy level. Since the DIE level SGs base on PMDs and/or MAD's and not on PhyDs they are referred as Phantom domains.

- Example1: 2 Big+2 Little system (2 Big sharing FD/VD and 2 Little sharing FD/VD)

```
cpu-map {
        cluster0 {
                core0 {
                        cpu = <&Big_0>;
                };
                core1 {
                        cpu = <&Big_1>;
                };
        };
        cluster1 {
                core0 {
                        cpu = <&Little_0>;
                };
                core1 {
                        cpu = <&Little_1>;
                };
        };
};
```

- Example2: 1 Big + 2 Little + 2 Little (smaller L2 $) system (1 Big sharing FD/VD, 2 Little sharing FD/VD, 2 Little (smaller L2 $) sharing FD/VD)

```
cpu-map {
        cluster0 {
                core0 {
                        cpu = <&Big_0>;
                };
        };
        cluster1 {
                core0 {
                        cpu = <&Little_0>;
                };
                core1 {
                        cpu = <&Little_1>;
                };
        };
        Cluster2 {
                core0 {
                        cpu = <&Little_smallerL2$_0>;
                };
                core1 {
                        cpu = <& Little_smallerL2$_1>;
                };
        };

};
```

1.4.1.2 Last Level Cache

The SD flag SD_SHARE_PKG_RESOURCES stays on the SD MC level.

1.4.1.3 EAS EM data versus SD hierarchy bring-up optimization

Single core in a PMD is supported by EASv1.2 patch `“sched: EAS & 'single cpu per cluster'/cpu hotplug interoperability”`.

1.4.1.4 PM Domain Sharing between Cluster & DSU

Basic Support will neither provide an inter-cluster DFVS based notification mechanism nor an EM with 3 dimensional capacity state vectors. Basic support assumes that the error introduced is insignificant.

ARM

2 DynamIQ Shared Unit Level-3 cache partial power down

2.1 Introduction

The Level-3 cache in the ARM DynamIQ Shared Unit (DSU) supports partial power down by splitting the cache into an implementation-specific number of portions.

A “portion” is defined as the smallest single unit of the cache that can be powered up. A portion is always an integral number of cache ways.

The DSU R0 register interface supports powering up the cache in increments of 25% of the capacity available. Therefore, at any point in time we can have powered up 25% (1 portion), 50% (2 portions), 75% (3 portions) or 100% (4 portions) of the cache.
Later variants also support the cache to be fully powered off - 0 portions.

2.2 DSU partial power down software support

2.2.1 Overview

Support for the DSU Level-3 partial power down capability is provided in software in the form of a reference DevFreq driver. The DSU portion control DevFreq driver provides an energy-cost justified demand-driven policy for controlling the number of portions enabled.

It should be noted that the DevFreq infrastructure is normally used to control frequency on non-CPU devices, while this driver is controlling the number of portions enabled. Therefore, the implemented policy maps the number of portions enabled to frequency, thus considering 1Hz as representative for 1 portion, 2Hz as representative for 2 portions, etc.

The limitations of the current implementation are:

- Support for a single DynamIQ Shared Unit.
- This version of the code is not suitable for use with the simple ondemand governor.

The following sections will describe the policy used for controlling the number of powered up portions, the device tree information that has to be provided, the trace events used for testing, test procedure and test results.

2.2.2 DSU partial power down policy

Describing it simply, the portion control DevFreq driver will allow a dynamic number of portions in the DSU to be enabled depending on the DSU Level-3 cache miss rate. The implementation is timer based, looking at windows of time configured in the device tree node, measuring miss and hit bandwidths to determine if the number of active portions needs to be adjusted.

Starting with the full cache powered up, resizing is done by an increase or decrease of one portion at a time, as follows:

1. Upsizing

   Upsizing will have higher priority compared to downsize for performance considerations. For the same reason, we want to upsize more aggressively and therefore we check for the upsize condition every polling interval and check for the downsize condition less often.

   To check if we want to enable one more portion, we need to weigh the additional cost in energy (due to cache static leakage) when we enable an additional portion, against the potential savings in energy we can achieve by decreasing the dynamic cost of accessing DRAM, due to the decrease in miss-rate we expect to obtain. Miss bandwidth (MBW) is used as an indicator that performance and energy might be impacted.

   We introduce the following notations:

   - MBW - miss bandwidth, expressed in MiB/sec, obtained using the miss counter, the size of the cache line and the duration of the evaluation.
   - L - static cache leakage for a single portion, expressed in uW or uJ/sec
   - ED - amount of energy used by DRAM per MiB of transferred data (on average), expressed in uJ/MiB
   - CB = L / ED - cost bandwidth, expressed in MiB/sec, is the expenditure of the additional static power to enable another portion balanced out with the savings of dynamic energy obtained by converting the the cache misses to cache hits and reducing the the number of accesses to DRAM.
   - Tu - upsizing threshold, expressed in fraction value in the range 0.0 to 1.0, represents the amount of static cache leakage we do not need to justify when deciding to up-size.

   Therefore, the condition for upsizing by one portion is:

   $$MBW > ((1.0 - Tu) * CB)$$

   A value for Tu of 0.9 (90%) means we only need to justify the energy leaked by 10% of the portion before we enable an additional portion.

   This is done as to favor upsize and benefit performance.

2. Downsizing

   Downsizing will have lower priority compared to upsize for performance considerations. Therefore downsize will only be done if the upsize condition is not accomplished and we will downsize less aggressively by only checking for the downsize condition once in a few polling time intervals (defined by POLLING_DOWN_INTERVAL).

From an energy trade-off perspective, to justify a cache portion to be powered on requires a hit bandwidth that pays for its leakage. When this condition is not accomplished we consider downsizing by one portion.

We introduce the following notations:

- HBW - hit bandwidth, expressed in MiB/sec, obtained using the hit counter, the size of the cache line and the duration of the evaluation.
- N - the current number of portions enabled.
- L - static cache leakage for a single portion, expressed in uW or uJ/sec
- ED - amount of energy used by DRAM per MiB of transferred data (on average), expressed in uJ/MiB
- CB = L / ED - cost bandwidth, expressed in MiB/sec, is the expenditure of the additional static power to have one portion enabled balanced out with the savings of dynamic energy obtained by converting the the cache misses to cache hits and reducing the the number of accesses to DRAM.
- Td - downsizing threshold, expressed in fraction value in the range 0.0 to 1.0, represents the amount of static cache leakage for one portion that we ignore when we evaluate the downsize condition.

Therefore, the condition for downsizing by one portion is:

$$HBW < ((N - Td) * CB)$$

A value for Td of 0.9 (90%) means we ignore 90% of the energy leaked by one portion when evaluating if the hit bandwidth is still able to justify the current static cache leakage.

2.2.3 DSU variant configuration

The variant is the major revision number *x* in the *rx* part of the *rxpy* description of the product revision status.

When initialised, the DevFreq device will read variant information from the DSU cluster ID register [1] and based on the read variant it will set values for the minimum and maximum number of portions supported:

- Variant 0: A minimum of 1 portion and a maximum of 2 portions
  The first released variant of the DSU (also referred to as r0 in the TRM reference below [1]) does not support the cache to be fully powered off.
- Variant x, x > 0: A minimum of 0 portions and a maximum of 2 portions
  Later variants will support 0 portions enabled, this being the equivalent of having the cache fully powered off.

Although the smallest unit that can be powered on/off within the DSU L3 cache belongs to the tag RAMs - portions of 25%, the data RAM can only pe powered on/off in portions of 50%. If we choose the tag RAM as the unit for portioning, this will result in the vast majority of the leakage being no different at 1 and 2 portions, and likewise at 3 and 4 portions, due to the fact that the most significant part of the cache - the data RAM, will stay powered on. Therefore, if we select 1 or 3 portions, we're losing a quarter of the cache for almost no leakage reduction.

For this reason, a better choice in terms of portioning is at data RAM granularity, resulting in splitting the cache in 2 logical portions (which groups together 2 physical portions) rather than using the 4 existing physical portions. This results in having all variants either being:

- Fully off - 0 portions enabled (rx, x > 0)
- Half on - 1 portion enabled
- Fully on - 2 portions enabled

[1] ARM® DynamIQ™ Shared Unit Technical Reference Manual Revision r0p2

2.2.4 Trace

The provided code also adds a trace event for the ARM DSU cache partial power down device status. This reports the counters for cache misses and cache hits together with the current number of active portions (reported as current frequency). The trace event also reports the total and busy time such that busy_time = (total_time * hit_count) / access_count.

These values can be used to simulate the policy outside of the system and to evaluate if the decisions to upsize or downsize were correct.

2.2.5 Device tree

A device tree node is used to configure the driver to use proper thresholds when evaluating resizing conditions. The only compatible string supported is `"arm,dsu_pctrl_r0".` This will result in using an interval of 1 to 4 as the possible numbers of active portions. Other properties include the size of the cache, the cache line size, the static cache leakage specific to the hardware implementation, the DRAM dynamic energy for accessed MiB and the polling interval.

- Example:

```
dsu_pctrl: dsu_pctrl {

        compatible = "arm,dsu_pctrl_r0";

        size = <1024>; /* size in KB */

        line-size = <64>;

        static-leakage-per-mb = <10000>; /*uW/MiB */

        dram-energy-per-mb = <130>; /* uJ/MiB */

        polling = <10000>; /* milliseconds */

};
```

2.3 DSU partial power down evaluation

The evaluation of this driver was done only on virtual platforms as silicon is not yet available. At the moment, only functional evaluation is possible as the virtual platforms available do not support accurate measurement of energy consumption or performance metrics.

These results were obtained on a r0 variant of the DSU being configured to support between a minimum of 1 and a maximum of 4 portions, therefore using the physical split in 4 portions and not the logical split in 2 portions as described in DSU variant configuration.

Therefore, a functional implementation and integration should have the behaviour detailed below. The behaviour can be observed by using the trace event added to analyse transitions from one number of active portions to another.

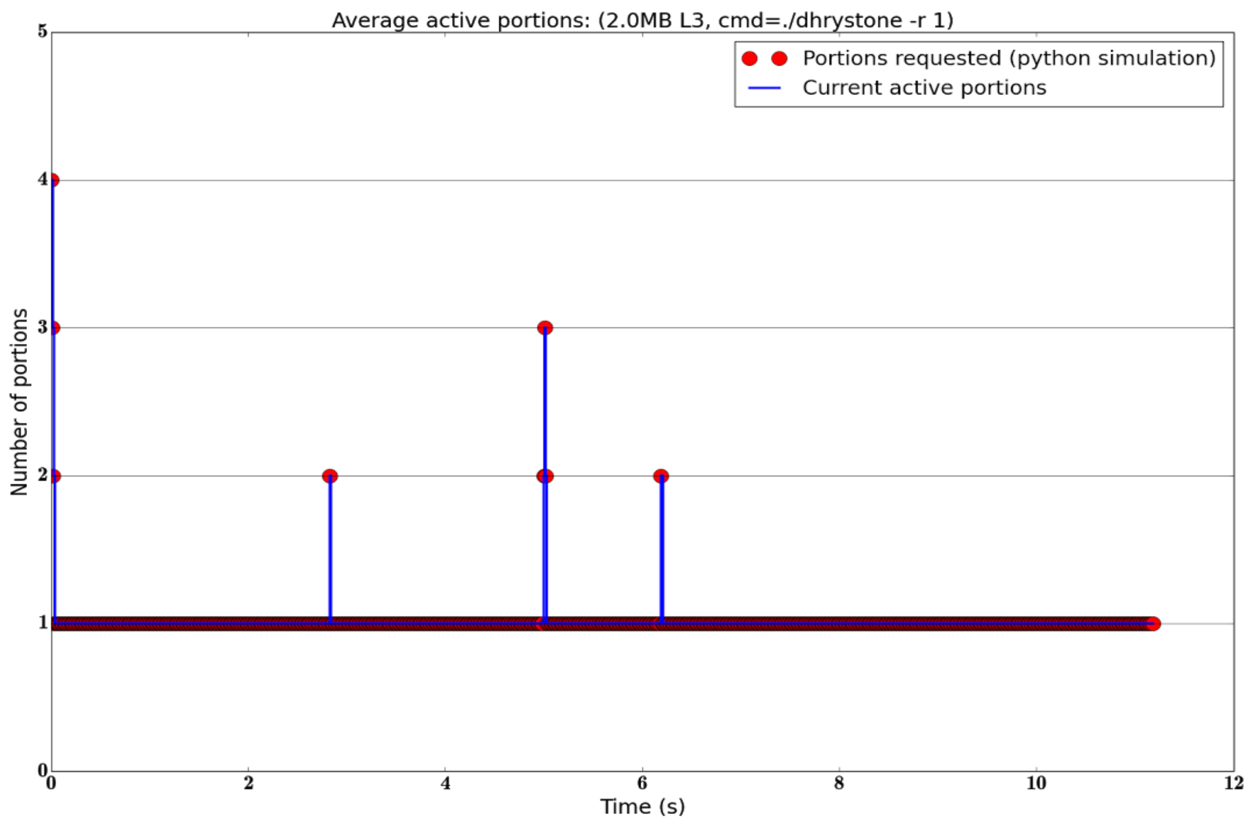
- CPU intensive loads should not have an effect on the number of active portions.

  This behaviour was evaluated using dhrystone as test workload and one of the results can be observed in Figure 5.

- I/O intensive loads should raise portions when the cache is well-used.

  This behaviour was evaluated using memcpy as test workload and one of the results can be observed in Figure 6.

Observation: The spikes observed in the figures (from 1 active portion to either 2 or 3 active portions) are results of either noise in the system or show activity on the L3 cache resulting from loading the workload binary or the shell binary, when returning from the workload.



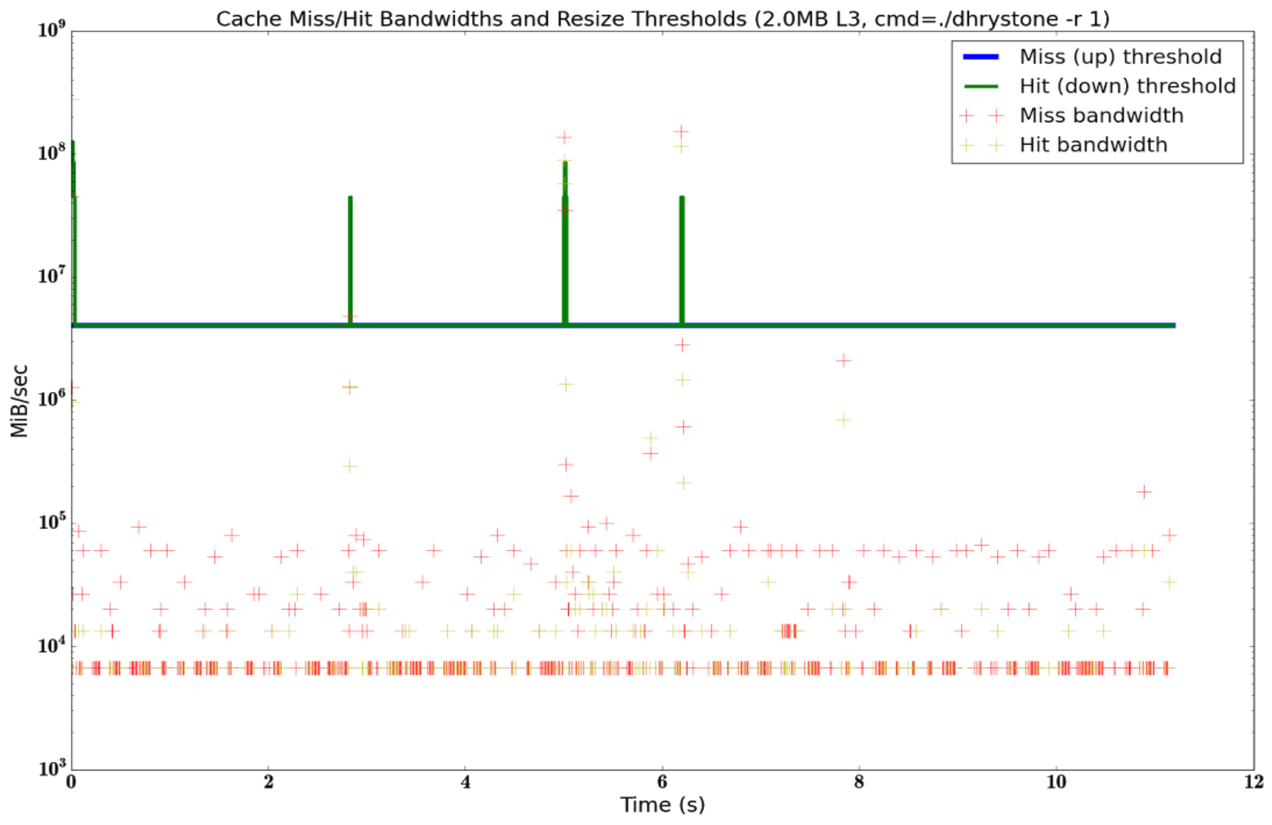
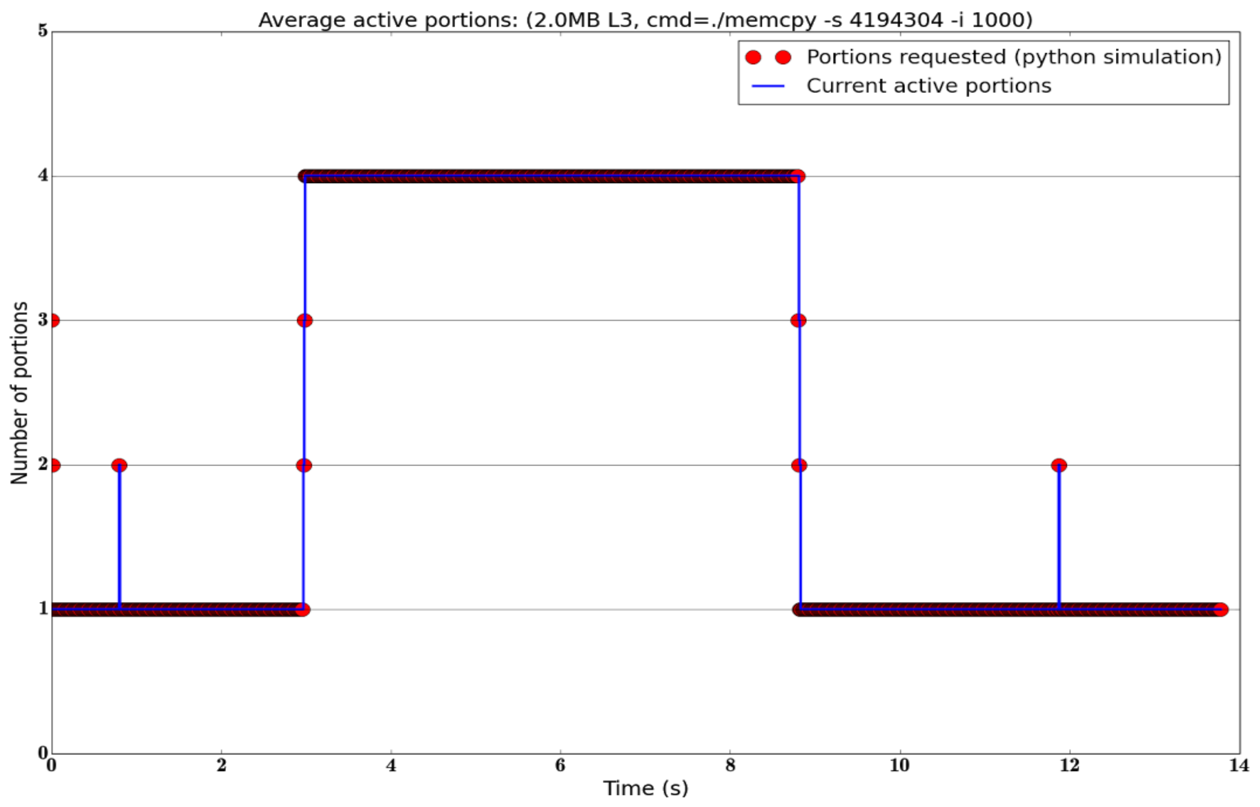


Figure 5: Analysis on 2.0MiB L3, cmd=./dhrystone -r 1



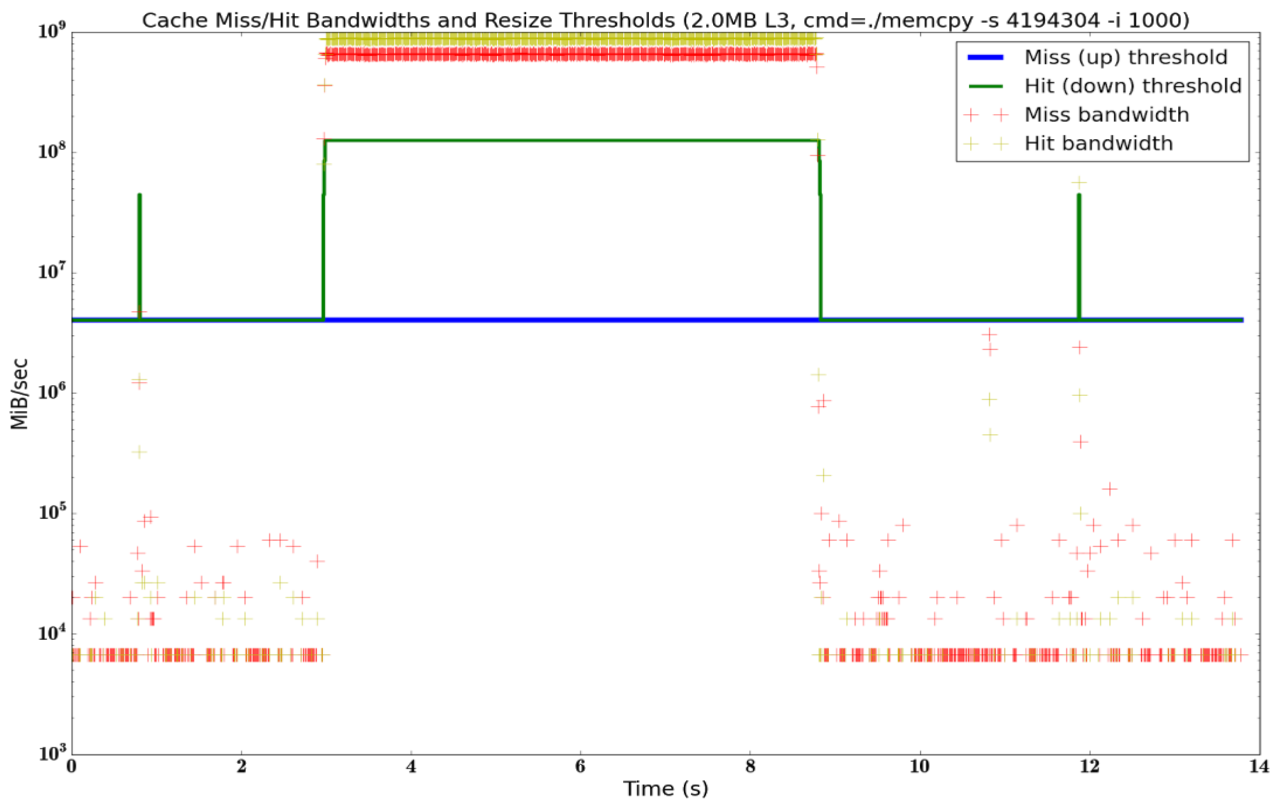


Figure 6: Analysis on 2.0MiB L3, cmd=./memcpy -s 4194304 -i 1000

Experiment	Parameters	L3$ (MiB)
Dhrystone	1 second	0.5
memcpy-L1	1 KiB	0.5
memcpy-L2	64 KiB	0.5
memcpy-L3	512 KiB	0.5
memcpy-dram	4 MiB	0.5
Dhrystone	1 second	1
memcpy-L1	1 KiB	1
memcpy-L2	64 KiB	1
memcpy-L3	512 KiB	1
memcpy-dram	4 MiB	1
Dhrystone	1 second	2
memcpy-L1	1 KiB	2
memcpy-L2	64 KiB	2
memcpy-L3	512 KiB	2
memcpy-dram	4 MiB	2
Dhrystone	1 second	4
memcpy-L1	1 KiB	4
memcpy-L2	64 KiB	4
memcpy-L3	512 KiB	4
memcpy-dram	4 MiB	4
Dhrystone	1 second	8
memcpy-L1	1 KiB	8
memcpy-L2	64 KiB	8
memcpy-L3	512 KiB	8
memcpy-dram	4 MiB	8

Table 2: DSU Level-3 cache partial power down test configurations

The examples provided above in Figure 5 and Figure 6 are given for workloads run on a system having a 2MiB Level-3 cache, but tests have been run for multiple configurations as Table 2 describes. The overall results can be observed in Figure 7.

Observation: For CPU-intensive workloads and for memcpy experiments with buffers that fit in L1 or L2 cache, the number of active portions will remain at 1. In the chart below that presents the tests results, those lines overlap.

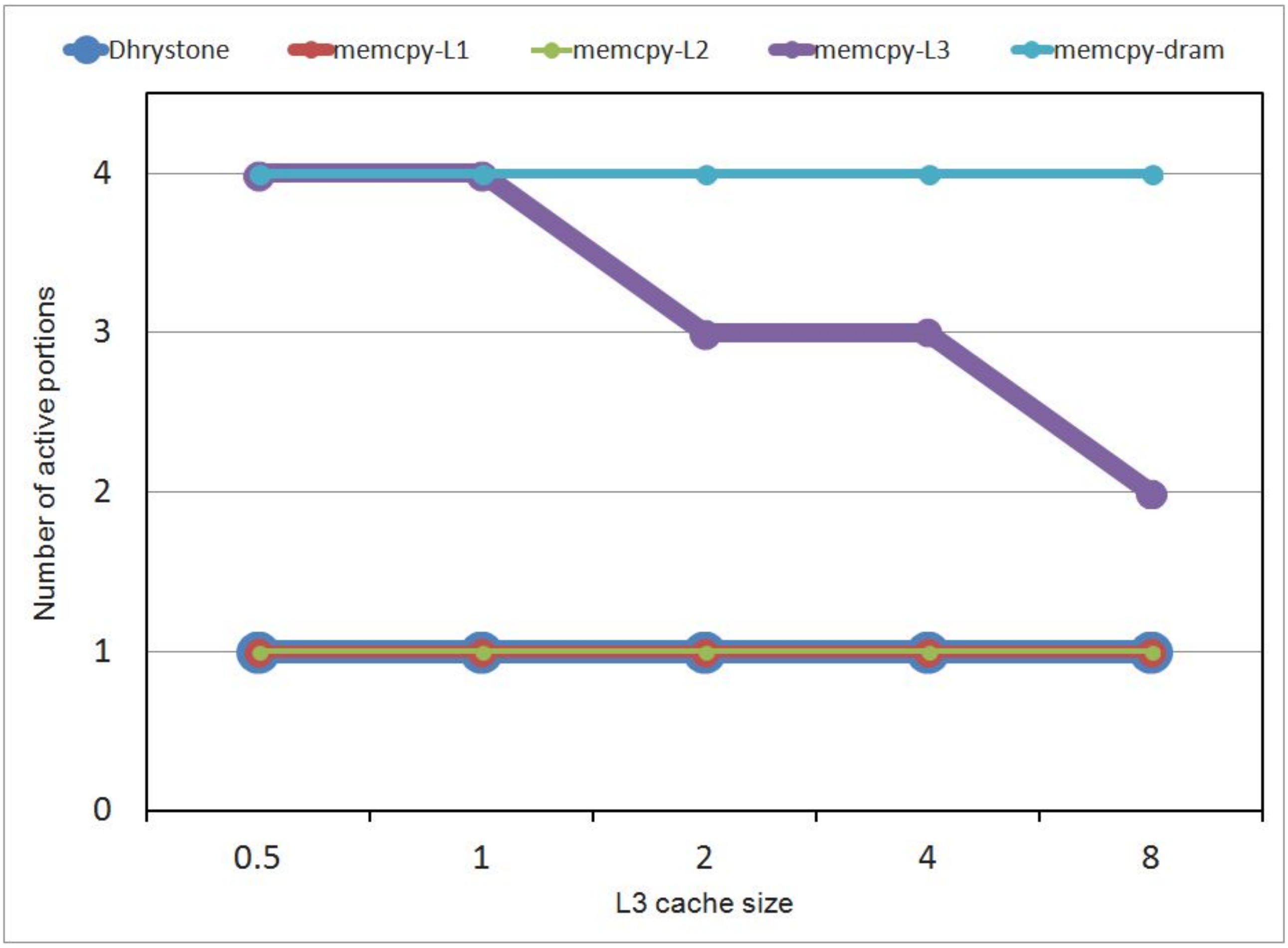


Figure 7: DSU Level-3 cache partial power down test results

ARM

3 DynamIQ energy model

3.1 Introduction

The energy model building processes provided by LISA [1] and described in the EAS overview and integration guide [2] apply to DynamIQ systems as well, but the energy data might have to be interpreted differently for these systems, depending on the CPU topology, when considering current limitations of the scheduler domain hierarchy and the use of phantom domains, as described in Linkage between Energy Model and Scheduler Domain hierarchy.

A precise and generic energy building process cannot be presented as it is dependent on the availability of energy measuring points on the hardware platform - core level, cluster level, system level, etc., and it might not apply to every platform, given the great diversity of CPU configurations that are possible with DynamIQ.

In the chapters below you'll find details on requirements when it comes to platform specific support that will enable you to run an energy model building flow as well as some guidelines on how to interpret data in the case of DynamIQ systems.

3.2 Energy model building process - requirements

Obtaining the actual model data for a particular platform requires some way of measuring power/energy. LISA currently provides integration for some types of energy probes [3] that can be used to instrument your platform. Consider the information that can be obtained when using different measurement points and how information at different levels fits into the energy model.

Before building your energy model, make sure your platform boots to userspace and has validated power management support that includes:

- Hotplug
- CpuFreq
- CpuIdle

Also ensure that you are able to run a series of workloads (e.g. dhrystone, sysbench, etc.) that will provide performance information.

This will enable you to use the examples in LISA [1] to create a flow that fits your platform. Before designing this flow, read and assimilate the guidelines below.

3.3 Energy model building process - guidelines

Generic information and guidelines are provided in the Android common kernel documentation [4].

Due to the hierarchical nature of power domains, energy costs are modeled by associating energy values with groups of CPU: energy costs of shared resources are associated with the group of CPUs that share the resource; only the cost of powering the CPU itself and any private resource (e.g. private L1 caches) is associated with per-cpu groups.

For DynamIQ systems additional elements need to be considered:

- Due to current limitations of the scheduler domain hierarchy, it's only possible to model two levels: per-cpu energy costs and per domain (phantom or physical) energy costs.
  - For CPUs with private L2 caches, the L2 static and dynamic energy should be considered as part of the CPU energy, as the power state of the cache will follow the power state of the CPU. Therefore, the L2 cache also becomes a private resource of the CPU, together with the L1 caches.
  - Given that for DynamIQ systems, PMDs, MADs and PhyDs are no longer congruent, the decision to define the cluster of CPUs at PMD level, MAD level or PhyD level through the use of cpu_map (as described, for example, for phantom domains in CPU Topology), will decide the energy information that will be filled in as cluster level data in the energy model, as it will be determined by the shared resources that are associated with the group of CPUs for those clusters.
- If DSU energy or PhyD level energy costs are to be provided, a mix of workloads should be used that will stress all logic inside the DSU, including the L3 cache. CPU intensive workloads might not be sufficient to characterize the capacity states. You must determine whether the DSU level (or DynamIQ physical cluster level) energy costs have to be provided or if the error introduced by not modeling this level is insignificant for EAS behaviour, based on your CPU topology and PM domain sharing specifications (shared frequency or voltage domains between the DSU and groups of cores).

[1] LISA: ipynb/energy

[2] EAS overview and integration guide: EAS1.3

[3] LISA: Energy-Meters-Requirements

[4] Android common kernel: android-4.4/Documentation/scheduler/sched-energy.txt

4 Glossary

CPU Capacity Domain (CCD)

A (maximum) CPU Capacity Domain represents a set of cores sharing microarchitecture and maximum frequency (which can also be defined as a congruent FD/VD and MAD).

Dynamic Voltage and Frequency Scaling (DVFS)

Adjustment of voltage and frequency settings on processors and peripheral devices to maximize power savings in case these resources are not used

Energy Model (EM)

Contains energy consumption values related to Operating Performance Points and Idle

States for a group of cores. This information allows the Energy-Aware Scheduler to evaluate the implications of its decisions from a standpoint of energy usage.

Frequency Domain (FD)

A set of entities (cores, DSU) which operate at the same frequency. A Frequency Domain is identified by a Clock Generator (normally a PLL).

Frequency Invariance

Frequency invariance makes the load and utilization signal of Per-Entity Load-Tracking (PELT) aware of CPU frequency scaling.

Without frequency invariance a task with 25% load on a CPU operating at 100% of its maximum frequency would change its load to 50% in case the frequency decreases to 50% of the maximum frequency. With frequency invariance the load of the task remains 25% regardless of the CPU frequency. The same is true for the utilization signal.

Frequency Invariance Engine (FIE)

Software Entity implementing Frequency Invariance.

Last Level Cache (LLC)

The last level of cache shared by a group of CPUs.

Micro Architecture Domain (MAD)

A set of cores sharing the same microarchitecture within a heterogeneous processing architecture.

Operating Performance Point (OPP)

A set of discrete tuples consisting of frequency and voltage pairs that the entity inside a Frequency Domain/Voltage Domain supports is called Operating Performance Points or OPPs.

Per-Entity Load Tracking (PELT)

Load and utilization signals of a task or CPU which are used for task placement and load balancing inside the Completely Fair Scheduler (CFS) scheduling class of the Linux task scheduler.

PM Domain (PMD)

Generic term for a Power Management related domain of cores. This could be a Frequency Domain and/or Voltage Domain as well as a Power Domain.

Power Domain (PD)

A set of cores which can be powered-up and powered-down independently from the rest of the processor.

Cluster Physical Domain (PhyD)

A Cluster Physical Domain represents a cluster of cores determined by the physical boundary of the last level cache these cores are sharing.

Scheduling Domain (SD)

A scheduling domain (`struct sched_domain`) is a set of CPUs which share properties and scheduling policies and which can be balanced against each other. Scheduling domains are hierarchical. A multi-level system will have multiple levels of domains.

E.g. the multi-cluster level (`MC` level) contains all the cpus belonging to a certain cluster whereas the physical processor level (`DIE` level) spans all the cpus of the processor.

Scheduling Group (SG)

Each scheduling domain contains two or more scheduling groups (`struct sched_group`) which are treated as a single unit by the scheduling domain. When the scheduler tries to balance the load within a scheduling domain, it tries to even out the load carried by each scheduling group without worrying directly about what is happening within the scheduling group.

Voltage Domain (VD)

A set of entities (cores, DSU) which operate at the same voltage. A Voltage Domain is identified by a Voltage Regulator.