Energy Model of devices

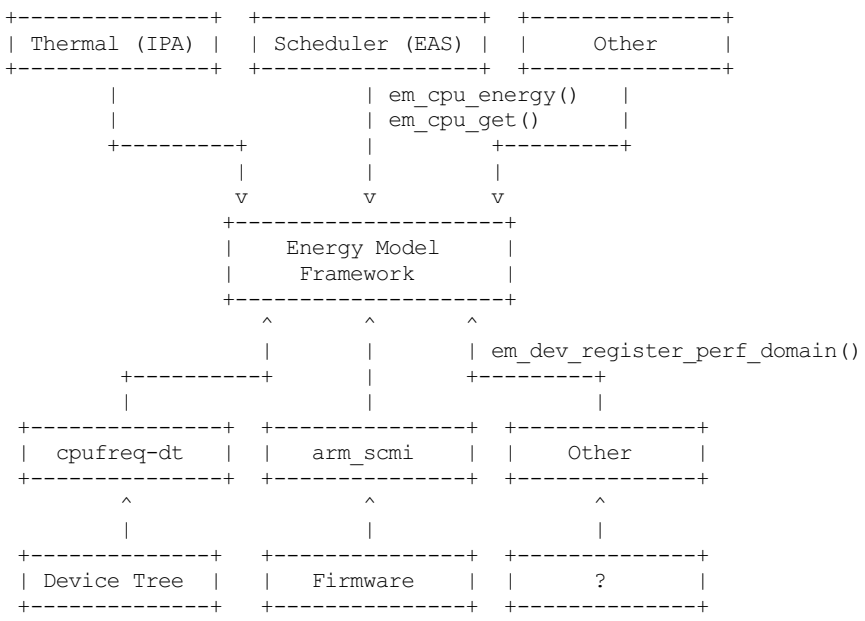
1. Overview

The Energy Model (EM) framework serves as an interface between drivers knowing the power consumed by devices at various performance levels, and the kernel subsystems willing to use that information to make energy-aware decisions.

The source of the information about the power consumed by devices can vary greatly from one platform to another. These power costs can be estimated using devicetree data in some cases. In others, the firmware will know better. Alternatively, userspace might be best positioned. And so on. In order to avoid each and every client subsystem to re-implement support for each and every possible source of information on its own, the EM framework intervenes as an abstraction layer which standardizes the format of power cost tables in the kernel, hence enabling to avoid redundant work.

The power values might be expressed in milli-Watts or in an 'abstract scale'. Multiple subsystems might use the EM and it is up to the system integrator to check that the requirements for the power value scale types are met. An example can be found in the Energy-Aware Scheduler documentation Documentation/scheduler/sched-energy.rst. For some subsystems like thermal or powercap power values expressed in an 'abstract scale' might cause issues. These subsystems are more interested in estimation of power used in the past, thus the real milli-Watts might be needed. An example of these requirements can be found in the Intelligent Power Allocation in Documentation/driver-api/thermal/power_allocator.rst. Kernel subsystems might implement automatic detection to check whether EM registered devices have inconsistent scale (based on EM internal flag). Important thing to keep in mind is that when the power values are expressed in an 'abstract scale' deriving real energy in milli-Joules would not be possible.

The figure below depicts an example of drivers (Arm-specific here, but the approach is applicable to any architecture) providing power costs to the EM framework, and interested clients reading the data from it:



In case of CPU devices the EM framework manages power cost tables per 'performance domain' in the system. A performance domain is a group of CPUs whose performance is scaled together. Performance domains generally have a 1-to-1 mapping with CPUFreq policies. All CPUs in a performance domain are required to have the same micro-architecture. CPUs in different performance domains can have different micro-architectures.

2. Core APIs

2.1 Config options

CONFIG_ENERGY_MODEL must be enabled to use the EM framework.

2.2 Registration of performance domains

Registration of 'advanced' EM

The 'advanced' EM gets it's name due to the fact that the driver is allowed to provide more precised power model. It's not limited to some implemented math formula in the framework (like it's in 'simple' EM case). It can better reflect the real power measurements performed for each performance state. Thus, this registration method should be preferred in case considering EM static power (leakage) is important.

Drivers are expected to register performance domains into the EM framework by calling the following API:

```
int em_dev_register_perf_domain(struct device *dev, unsigned int nr_states,
                struct em_data_callback *cb, cpumask_t *cpus, bool milliwatts);
```

Drivers must provide a callback function returning <frequency, power> tuples for each performance state. The callback function provided by the driver is free to fetch data from any relevant location (DT, firmware, ...), and by any mean deemed necessary. Only for CPU devices, drivers must specify the CPUs of the performance domains using cpumask. For other devices than CPUs the last argument must be set to NULL. The last argument 'milliwatts' is important to set with correct value. Kernel subsystems which use EM might rely on this flag to check if all EM devices use the same scale. If there are different scales, these subsystems might decide to: return warning/error, stop working or panic. See Section 3. for an example of driver implementing this callback, or Section 2.4 for further documentation on this API

Registration of EM using DT

The EM can also be registered using OPP framework and information in DT "operating-points-v2". Each OPP entry in DT can be extended with a property "opp-microwatt" containing micro-Watts power value. This OPP DT property allows a platform to register EM power values which are reflecting total power (static + dynamic). These power values might be coming directly from experiments and measurements.

Registration of 'simple' EM

The 'simple' EM is registered using the framework helper function cpufreq_register_em_with_opp(). It implements a power model which is tight to math formula:

```
Power = C * V^2 * f
```

The EM which is registered using this method might not reflect correctly the physics of a real device, e.g. when static power (leakage) is important.

2.3 Accessing performance domains

There are two API functions which provide the access to the energy model: em_cpu_get() which takes CPU id as an argument and em_pd_get() with device pointer as an argument. It depends on the subsystem which interface it is going to use, but in case of CPU devices both functions return the same performance domain.

Subsystems interested in the energy model of a CPU can retrieve it using the em_cpu_get() API. The energy model tables are allocated once upon creation of the performance domains, and kept in memory untouched.

The energy consumed by a performance domain can be estimated using the em_cpu_energy() API. The estimation is performed assuming that the schedutil CPUfreq governor is in use in case of CPU device. Currently this calculation is not provided for other type of devices.

More details about the above APIs can be found in `<linux/energy_model.h>` or in Section 2.4

2.4 Description details of this API

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\power\(linux-master)(Documentation)(power)energy-model.rst, line 163)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: include/linux/energy_model.h
   :internal:
```

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\power\(linux-master)(Documentation)(power)energy-model.rst, line 166)

Unknown directive type "kernel-doc".

```
.. kernel-doc:: kernel/power/energy_model.c
   :export:
```

3. Example driver

The CPUFreq framework supports dedicated callback for registering the EM for a given CPU(s) 'policy' object: cpufreq_driver::register_em(). That callback has to be implemented properly for a given driver, because the framework would call it at the right time during setup. This section provides a simple example of a CPUFreq driver registering a performance domain in the Energy Model framework using the (fake) 'foo' protocol. The driver implements an est_power() function to be provided to the EM framework:

-> drivers/cpufreq/foo_cpufreq.c

```
01    static int est_power(unsigned long *mW, unsigned long *KHz,
02                         struct device *dev)
03    {
04            long freq, power;
05
06            /* Use the 'foo' protocol to ceil the frequency */
07            freq = foo_get_freq_ceil(dev, *KHz);
08            if (freq < 0);
09                    return freq;
10
11            /* Estimate the power cost for the dev at the relevant freq. */
12            power = foo_estimate_power(dev, freq);
13            if (power < 0);
14                    return power;
15
16            /* Return the values to the EM framework */
17            *mW = power;
18            *KHz = freq;
19
20            return 0;
21    }
22
23    static void foo_cpufreq_register_em(struct cpufreq_policy *policy)
24    {
25            struct em_data_callback em_cb = EM_DATA_CB(est_power);
26            struct device *cpu_dev;
27            int nr_opp;
28
29            cpu_dev = get_cpu_device(cpumask_first(policy->cpus));
30
31            /* Find the number of OPPs for this policy */
32            nr_opp = foo_get_nr_opp(policy);
33
34            /* And register the new performance domain */
35            em_dev_register_perf_domain(cpu_dev, nr_opp, &em_cb, policy->cpus,
36                                        true);
37    }
38
39    static struct cpufreq_driver foo_cpufreq_driver = {
40            .register_em = foo_cpufreq_register_em,
41    };
```