CPU cooling APIs How To

Written by Amit Daniel Kachhap <amit.kachhap@linaro.org>

Updated: 6 Jan 2015

0. Introduction

The generic cpu cooling(freq clipping) provides registration/unregistration APIs to the caller. The binding of the cooling devices to the trip point is left for the user. The registration APIs returns the cooling device pointer.

1. cpu cooling APIs

1.1 cpufreq registration/unregistration APIs

```
struct thermal_cooling_device
*cpufreq_cooling_register(struct cpumask *clip_cpus)
```

This interface function registers the cpufreq cooling device with the name "thermal-cpufreq-%x". This api can support multiple instances of cpufreq cooling devices.

clip_cpus:

cpumask of cpus where the frequency constraints will happen.

```
struct thermal_cooling_device
*of_cpufreq_cooling_register(struct cpufreq_policy *policy)
```

This interface function registers the cpufreq cooling device with the name "thermal-cpufreq-%x" linking it with a device tree node, in order to bind it via the thermal DT code. This api can support multiple instances of cpufreq cooling devices.

policy:

CPUFreq policy.

```
void cpufreq_cooling_unregister(struct thermal_cooling_device *cdev)
```

This interface function unregisters the "thermal-cpufreq-%x" cooling device.

cdev: Cooling device pointer which has to be unregistered.

2. Power models

The power API registration functions provide a simple power model for CPUs. The current power is calculated as dynamic power (static power isn't supported currently). This power model requires that the operating-points of the CPUs are registered using the kernel's opp library and the *cpufreq_frequency_table* is assigned to the *struct device* of the cpu. If you are using CONFIG_CPUFREQ_DT then the *cpufreq_frequency_table* should already be assigned to the cpu device.

The dynamic power consumption of a processor depends on many factors. For a given processor implementation the primary factors are:

- The time the processor spends running, consuming dynamic power, as compared to the time in idle states where dynamic consumption is negligible. Herein we refer to this as 'utilisation'.
- The voltage and frequency levels as a result of DVFS. The DVFS level is a dominant factor governing power consumption.
- In running time the 'execution' behaviour (instruction types, memory access patterns and so forth) causes, in most cases, a second order variation. In pathological cases this variation can be significant, but typically it is of a much lesser impact than the factors above.

A high level dynamic power consumption model may then be represented as:

```
Pdyn = f(run) * Voltage^2 * Frequency * Utilisation
```

f(run) here represents the described execution behaviour and its result has a units of Watts/Hz/Volt^2 (this often expressed in mW/MHz/uVolt^2)

The detailed behaviour for f(run) could be modelled on-line. However, in practice, such an on-line model has dependencies on a number of implementation specific processor support and characterisation factors. Therefore, in initial implementation that contribution is represented as a constant coefficient. This is a simplification consistent with the relative contribution to overall power variation.

In this simplified representation our model becomes:

```
Pdyn = Capacitance * Voltage^2 * Frequency * Utilisation
```

Where *capacitance* is a constant that represents an indicative running time dynamic power coefficient in fundamental units of mW/MHz/uVolt^2. Typical values for mobile CPUs might lie in range from 100 to 500. For reference, the approximate values for the SoC in ARM's Juno Development Platform are 530 for the Cortex-A57 cluster and 140 for the Cortex-A53 cluster.