CPU Idle Cooling

Situation:

Under certain circumstances a SoC can reach a critical temperature limit and is unable to stabilize the temperature around a temperature control. When the SoC has to stabilize the temperature, the kernel can act on a cooling device to mitigate the dissipated power. When the critical temperature is reached, a decision must be taken to reduce the temperature, that, in turn impacts performance.

Another situation is when the silicon temperature continues to increase even after the dynamic leakage is reduced to its minimum by clock gating the component. This runaway phenomenon can continue due to the static leakage. The only solution is to power down the component, thus dropping the dynamic and static leakage that will allow the component to cool down.

Last but not least, the system can ask for a specific power budget but because of the OPP density, we can only choose an OPP with a power budget lower than the requested one and under-utilize the CPU, thus losing performance. In other words, one OPP under-utilizes the CPU with a power less than the requested power budget and the next OPP exceeds the power budget. An intermediate OPP could have been used if it were present.

Solutions:

If we can remove the static and the dynamic leakage for a specific duration in a controlled period, the SoC temperature will decrease. Acting on the idle state duration or the idle cycle injection period, we can mitigate the temperature by modulating the power budget.

The Operating Performance Point (OPP) density has a great influence on the control precision of cpufreq, however different vendors have a plethora of OPP density, and some have large power gap between OPPs, that will result in loss of performance during thermal control and loss of power in other scenarios.

At a specific OPP, we can assume that injecting idle cycle on all CPUs belong to the same cluster, with a duration greater than the cluster idle state target residency, we lead to dropping the static and the dynamic leakage for this period (modulo the energy needed to enter this state). So the sustainable power with idle cycles has a linear relation with the OPPâ€™s sustainable power and can be computed with a coefficient similar to:

```
Power(IdleCycle) = Coef x Power(OPP)
```

Idle Injection:

The base concept of the idle injection is to force the CPU to go to an idle state for a specified time each control cycle, it provides another way to control CPU power and heat in addition to cpufreq. Ideally, if all CPUs belonging to the same cluster, inject their idle cycles synchronously, the cluster can reach its power down state with a minimum power consumption and reduce the static leakage to almost zero. However, these idle cycles injection will add extra latencies as the CPUs will have to wakeup from a deep sleep state.

We use a fixed duration of idle injection that gives an acceptable performance penalty and a fixed latency. Mitigation can be increased or decreased by modulating the duty cycle of the idle injection.

```
^
|
|
|-------                         -------
|_______|_______________________|_______|___________

<------>
  idle  <---------------------->
              running

 <----------------------------->
         duty cycle 25%
```

The implementation of the cooling device bases the number of states on the duty cycle percentage. When no mitigation is happening the cooling device state is zero, meaning the duty cycle is 0%.

When the mitigation begins, depending on the governor's policy, a starting state is selected. With a fixed idle duration and the duty cycle (aka the cooling device state), the running duration can be computed.

The governor will change the cooling device state thus the duty cycle and this variation will modulate the cooling effect.

```
^
|
|
|-------                 -------
|_______|_______________|_______|___________

<------>
```

```
     idle  <-------------->
                running

    <--------------------->
        duty cycle 33%


   ^
   |
   |
   |-------         -------
   |_______|_______|_______|___________

   <------>
     idle  <------>
               running

    <------------->
     duty cycle 50%
```

The idle injection duration value must comply with the constraints:

- It is less than or equal to the latency we tolerate when the mitigation begins. It is platform dependent and will depend on the user experience, reactivity vs performance trade off we want. This value should be specified.
- It is greater than the idle stateâ€™s target residency we want to go for thermal mitigation, otherwise we end up consuming more energy.

Power considerations

When we reach the thermal trip point, we have to sustain a specified power for a specific temperature but at this time we consume:

```
   Power = Capacitance x Voltage^2 x Frequency x Utilisation
```

... which is more than the sustainable power (or there is something wrong in the system setup). The â€˜Capacitanceâ€™ and â€˜Utilisationâ€™ are a fixed value, â€˜Voltageâ€™ and the â€˜Frequencyâ€™ are fixed artificially because we donâ€™t want to change the OPP. We can group the â€˜Capacitanceâ€™ and the â€˜Utilisationâ€™ into a single term which is the â€˜Dynamic Power Coefficient (Cdyn)â€™ Simplifying the above, we have:

```
   Pdyn = Cdyn x Voltage^2 x Frequency
```

The power allocator governor will ask us somehow to reduce our power in order to target the sustainable power defined in the device tree. So with the idle injection mechanism, we want an average power (Ptarget) resulting in an amount of time running at full power on a specific OPP and idle another amount of time. That could be put in a equation:

```
   P(opp)target = ((Trunning x (P(opp)running) + (Tidle x P(opp)idle)) /
                           (Trunning + Tidle)

    ...

   Tidle = Trunning x ((P(opp)running / P(opp)target) - 1)
```

At this point if we know the running period for the CPU, that gives us the idle injection we need. Alternatively if we have the idle injection duration, we can compute the running duration with:

```
   Trunning = Tidle / ((P(opp)running / P(opp)target) - 1)
```

Practically, if the running power is less than the targeted power, we end up with a negative time value, so obviously the equation usage is bound to a power reduction, hence a higher OPP is needed to have the running power greater than the targeted power.

However, in this demonstration we ignore three aspects:

- The static leakage is not defined here, we can introduce it in the equation but assuming it will be zero most of the time as it is difficult to get the values from the SoC vendors
- The idle state wake up latency (or entry + exit latency) is not taken into account, it must be added in the equation in order to rigorously compute the idle injection
- The injected idle duration must be greater than the idle state target residency, otherwise we end up consuming more energy and potentially invert the mitigation effect

So the final equation is:

```
   Trunning = (Tidle - Twakeup ) x
                  (((P(opp)dyn + P(opp)static ) - P(opp)target) / P(opp)target )
```