Intel Powerclamp Driver

By:

- Arjan van de Ven <arjan@linux.intel.com>
- Jacob Pan <jacob.jun.pan@linux.intel.com>

INTRODUCTION

Consider the situation where a systemâ€™s power consumption must be reduced at runtime, due to power budget, thermal constraint, or noise level, and where active cooling is not preferred. Software managed passive power reduction must be performed to prevent the hardware actions that are designed for catastrophic scenarios.

Currently, P-states, T-states (clock modulation), and CPU offlining are used for CPU throttling.

On Intel CPUs, C-states provide effective power reduction, but so far theyâ€™re only used opportunistically, based on workload. With the development of intel_powerclamp driver, the method of synchronizing idle injection across all online CPU threads was introduced. The goal is to achieve forced and controllable C-state residency.

Test/Analysis has been made in the areas of power, performance, scalability, and user experience. In many cases, clear advantage is shown over taking the CPU offline or modulating the CPU clock.

THEORY OF OPERATION

Idle Injection

On modern Intel processors (Nehalem or later), package level C-state residency is available in MSRs, thus also available to the kernel.

These MSRs are:

```
#define MSR_PKG_C2_RESIDENCY      0x60D
#define MSR_PKG_C3_RESIDENCY      0x3F8
#define MSR_PKG_C6_RESIDENCY      0x3F9
#define MSR_PKG_C7_RESIDENCY      0x3FA
```

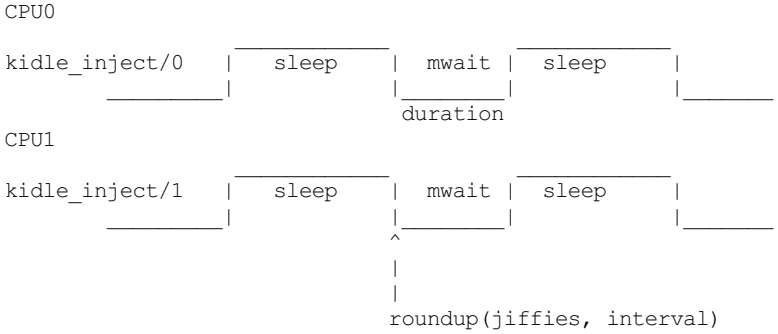
If the kernel can also inject idle time to the system, then a closed-loop control system can be established that manages package level C-state. The intel_powerclamp driver is conceived as such a control system, where the target set point is a user-selected idle ratio (based on power reduction), and the error is the difference between the actual package level C-state residency ratio and the target idle ratio.

Injection is controlled by high priority kernel threads, spawned for each online CPU.

These kernel threads, with SCHED_FIFO class, are created to perform clamping actions of controlled duty ratio and duration. Each per-CPU thread synchronizes its idle time and duration, based on the rounding of jiffies, so accumulated errors can be prevented to avoid a jittery effect. Threads are also bound to the CPU such that they cannot be migrated, unless the CPU is taken offline. In this case, threads belong to the offlined CPUs will be terminated immediately.

Running as SCHED_FIFO and relatively high priority, also allows such scheme to work for both preemptable and non-preemptable kernels. Alignment of idle time around jiffies ensures scalability for HZ values. This effect can be better visualized using a Perf timechart. The following diagram shows the behavior of kernel thread kidle_inject/cpu. During idle injection, it runs monitor/mwait idle for a given "duration", then relinquishes the CPU to other tasks, until the next time interval.

The NOHZ schedule tick is disabled during idle time, but interrupts are not masked. Tests show that the extra wakeups from scheduler tick have a dramatic impact on the effectiveness of the powerclamp driver on large scale systems (Westmere system with 80 processors).



Only one CPU is allowed to collect statistics and update global control parameters. This CPU is referred to as the controlling CPU in this document. The controlling CPU is elected at runtime, with a policy that favors BSP, taking into account the possibility of a CPU

hot-plug.

In terms of dynamics of the idle control system, package level idle time is considered largely as a non-causal system where its behavior cannot be based on the past or current input. Therefore, the intel_powerclamp driver attempts to enforce the desired idle time instantly as given input (target idle ratio). After injection, powerclamp monitors the actual idle for a given time window and adjust the next injection accordingly to avoid over/under correction.

When used in a causal control system, such as a temperature control, it is up to the user of this driver to implement algorithms where past samples and outputs are included in the feedback. For example, a PID-based thermal controller can use the powerclamp driver to maintain a desired target temperature, based on integral and derivative gains of the past samples.

Calibration

During scalability testing, it is observed that synchronized actions among CPUs become challenging as the number of cores grows. This is also true for the ability of a system to enter package level C-states.

To make sure the intel_powerclamp driver scales well, online calibration is implemented. The goals for doing such a calibration are:

a. determine the effective range of idle injection ratio
b. determine the amount of compensation needed at each target ratio

Compensation to each target ratio consists of two parts:

a) steady state error compensation This is to offset the error occurring when the system can enter idle without extra wakeups (such as external interrupts).

b) dynamic error compensation When an excessive amount of wakeups occurs during idle, an additional idle ratio can be added to quiet interrupts, by slowing down CPU activities.

A debugfs file is provided for the user to examine compensation progress and results, such as on a Westmere system:

```
[jacob@nex01 ~]$ cat
/sys/kernel/debug/intel_powerclamp/powerclamp_calib
controlling cpu: 0
pct confidence steady dynamic (compensation)
0       0       0       0
1       1       0       0
2       1       1       0
3       3       1       0
4       3       1       0
5       3       1       0
6       3       1       0
7       3       1       0
8       3       1       0
...
30      3       2       0
31      3       2       0
32      3       1       0
33      3       2       0
34      3       1       0
35      3       2       0
36      3       1       0
37      3       2       0
38      3       1       0
39      3       2       0
40      3       3       0
41      3       1       0
42      3       2       0
43      3       1       0
44      3       1       0
45      3       2       0
46      3       3       0
47      3       0       0
48      3       2       0
49      3       3       0
```

Calibration occurs during runtime. No offline method is available. Steady state compensation is used only when confidence levels of all adjacent ratios have reached satisfactory level. A confidence level is accumulated based on clean data collected at runtime. Data collected during a period without extra interrupts is considered clean.

To compensate for excessive amounts of wakeup during idle, additional idle time is injected when such a condition is detected. Currently, we have a simple algorithm to double the injection ratio. A possible enhancement might be to throttle the offending IRQ, such as delaying EOI for level triggered interrupts. But it is a challenge to be non-intrusive to the scheduler or the IRQ core code.

CPU Online/Offline

Per-CPU kernel threads are started/stopped upon receiving notifications of CPU hotplug activities. The intel_powerclamp driver keeps track of clamping kernel threads, even after they are migrated to other CPUs, after a CPU offline event.

Performance Analysis

This section describes the general performance data collected on multiple systems, including Westmere (80P) and Ivy Bridge (4P, 8P).

Effectiveness and Limitations

The maximum range that idle injection is allowed is capped at 50 percent. As mentioned earlier, since interrupts are allowed during forced idle time, excessive interrupts could result in less effectiveness. The extreme case would be doing a ping -f to generated flooded network interrupts without much CPU acknowledgement. In this case, little can be done from the idle injection threads. In most normal cases, such as scp a large file, applications can be throttled by the powerclamp driver, since slowing down the CPU also slows down network protocol processing, which in turn reduces interrupts.

When control parameters change at runtime by the controlling CPU, it may take an additional period for the rest of the CPUs to catch up with the changes. During this time, idle injection is out of sync, thus not able to enter package C- states at the expected ratio. But this effect is minor, in that in most cases change to the target ratio is updated much less frequently than the idle injection frequency.

Scalability

Tests also show a minor, but measurable, difference between the 4P/8P Ivy Bridge system and the 80P Westmere server under 50% idle ratio. More compensation is needed on Westmere for the same amount of target idle ratio. The compensation also increases as the idle ratio gets larger. The above reason constitutes the need for the calibration code.

On the IVB 8P system, compared to an offline CPU, powerclamp can achieve up to 40% better performance per watt. (measured by a spin counter summed over per CPU counting threads spawned for all running CPUs).

Usage and Interfaces

The powerclamp driver is registered to the generic thermal layer as a cooling device. Currently, itâ€™s not bound to any thermal zones:

```
jacob@chromoly:/sys/class/thermal/cooling_device14$ grep . *
cur_state:0
max_state:50
type:intel_powerclamp
```

cur_state allows user to set the desired idle percentage. Writing 0 to cur_state will stop idle injection. Writing a value between 1 and max_state will start the idle injection. Reading cur_state returns the actual and current idle percentage. This may not be the same value set by the user in that current idle percentage depends on workload and includes natural idle. When idle injection is disabled, reading cur_state returns value -1 instead of 0 which is to avoid confusing 100% busy state with the disabled state.

Example usage: - To inject 25% idle time:

```
$ sudo sh -c "echo 25 > /sys/class/thermal/cooling_device80/cur_state
```

If the system is not busy and has more than 25% idle time already, then the powerclamp driver will not start idle injection. Using Top will not show idle injection kernel threads.

If the system is busy (spin test below) and has less than 25% natural idle time, powerclamp kernel threads will do idle injection. Forced idle time is accounted as normal idle in that common code path is taken as the idle task.

In this example, 24.1% idle is shown. This helps the system admin or user determine the cause of slowdown, when a powerclamp driver is in action:

```
Tasks: 197 total,   1 running, 196 sleeping,   0 stopped,   0 zombie
Cpu(s): 71.2%us,  4.7%sy,  0.0%ni, 24.1%id,  0.0%wa,  0.0%hi,  0.0%si,  0.0%st
Mem:   3943228k total,  1689632k used,  2253596k free,    74960k buffers
Swap:  4087804k total,        0k used,  4087804k free,   945336k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
 3352 jacob     20   0  262m  644  428 S  286  0.0   0:17.16 spin
 3341 root     -51   0     0    0    0 D   25  0.0   0:01.62 kidle_inject/0
 3344 root     -51   0     0    0    0 D   25  0.0   0:01.60 kidle_inject/3
 3342 root     -51   0     0    0    0 D   25  0.0   0:01.61 kidle_inject/1
 3343 root     -51   0     0    0    0 D   25  0.0   0:01.60 kidle_inject/2
 2935 jacob     20   0  696m 125m  35m S    5  3.3   0:31.11 firefox
 1546 root      20   0  158m  20m 6640 S    3  0.5   0:26.97 Xorg
 2100 jacob     20   0 1223m  88m  30m S    3  2.3   0:23.68 compiz
```

Tests have shown that by using the powerclamp driver as a cooling device, a PID based userspace thermal controller can manage to control CPU temperature effectively, when no other thermal influence is added. For example, a UltraBook user can compile the kernel under certain temperature (below most active trip points).