intel_pstate CPU Performance Scaling Driver

Copyright: © 2017 Intel Corporation
Author: Rafael J. Wysocki <rafael.j.wysocki@intel.com>

General Information

intel_pstate is a part of the :doc:`CPU performance scaling subsystem <cpufreq>` in the Linux kernel (CPUFreq). It is a scaling driver for the Sandy Bridge and later generations of Intel processors. Note, however, that some of those processors may not be supported. [To understand intel_pstate it is necessary to know how CPUFreq works in general, so this is the time to read Documentation/admin-guide/pm/cpufreq.rst if you have not done that yet.]

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\[linux-master][Documentation][admin-guide][pm]intel_pstate.rst, line 16); *backlink***
>
> Unknown interpreted text role "doc".

For the processors supported by intel_pstate, the P-state concept is broader than just an operating frequency or an operating performance point (see the LinuxCon Europe 2015 presentation by Kristen Accardi [1] for more information about that). For this reason, the representation of P-states used by intel_pstate internally follows the hardware specification (for details refer to Intel Software Developerâ€™s Manual [2]). However, the CPUFreq core uses frequencies for identifying operating performance points of CPUs and frequencies are involved in the user space interface exposed by it, so intel_pstate maps its internal representation of P-states to frequencies too (fortunately, that mapping is unambiguous). At the same time, it would not be practical for intel_pstate to supply the CPUFreq core with a table of available frequencies due to the possible size of it, so the driver does not do that. Some functionality of the core is limited by that.

Since the hardware P-state selection interface used by intel_pstate is available at the logical CPU level, the driver always works with individual CPUs. Consequently, if intel_pstate is in use, every CPUFreq policy object corresponds to one logical CPU and CPUFreq policies are effectively equivalent to CPUs. In particular, this means that they become "inactive" every time the corresponding CPU is taken offline and need to be re-initialized when it goes back online.

intel_pstate is not modular, so it cannot be unloaded, which means that the only way to pass early-configuration-time parameters to it is via the kernel command line. However, its configuration can be adjusted via sysfs to a great extent. In some configurations it even is possible to unregister it via sysfs which allows another CPUFreq scaling driver to be loaded and registered (see below).

Operation Modes

intel_pstate can operate in two different modes, active or passive. In the active mode, it uses its own internal performance scaling governor algorithm or allows the hardware to do performance scaling by itself, while in the passive mode it responds to requests made by a generic CPUFreq governor implementing a certain performance scaling algorithm. Which of them will be in effect depends on what kernel command line options are used and on the capabilities of the processor.

Active Mode

This is the default operation mode of intel_pstate for processors with hardware-managed P-states (HWP) support. If it works in this mode, the scaling_driver policy attribute in sysfs for all CPUFreq policies contains the string "intel_pstate".

In this mode the driver bypasses the scaling governors layer of CPUFreq and provides its own scaling algorithms for P-state selection. Those algorithms can be applied to CPUFreq policies in the same way as generic scaling governors (that is, through the scaling_governor policy attribute in sysfs). [Note that different P-state selection algorithms may be chosen for different policies, but that is not recommended.]

They are not generic scaling governors, but their names are the same as the names of some of those governors. Moreover, confusingly enough, they generally do not work in the same way as the generic governors they share the names with. For example, the powersave P-state selection algorithm provided by intel_pstate is not a counterpart of the generic powersave governor (roughly, it corresponds to the schedutil and ondemand governors).

There are two P-state selection algorithms provided by intel_pstate in the active mode: powersave and performance. The way they both operate depends on whether or not the hardware-managed P-states (HWP) feature has been enabled in the processor and possibly on the processor model.

Which of the P-state selection algorithms is used by default depends on the :c:macro:`CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE` kernel configuration option. Namely, if that option is set, the performance algorithm will be used by default, and the other one will be used by default if it is not set.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\[linux-master][Documentation][admin-guide][pm]intel_pstate.rst, line 92); *backlink***
>
> Unknown interpreted text role "c:macro".

Active Mode With HWP

If the processor supports the HWP feature, it will be enabled during the processor initialization and cannot be disabled after that. It is possible to avoid enabling it by passing the intel_pstate=no_hwp argument to the kernel in the command line.

If the HWP feature has been enabled, intel_pstate relies on the processor to select P-states by itself, but still it can give hints to the processor's internal P-state selection logic. What those hints are depends on which P-state selection algorithm has been applied to the given policy (or to the CPU it corresponds to).

Even though the P-state selection is carried out by the processor automatically, intel_pstate registers utilization update callbacks with the CPU scheduler in this mode. However, they are not used for running a P-state selection algorithm, but for periodic updates of the current CPU frequency information to be made available from the scaling_cur_freq policy attribute in sysfs.

HWP + performance

In this configuration intel_pstate will write 0 to the processor's Energy-Performance Preference (EPP) knob (if supported) or its Energy-Performance Bias (EPB) knob (otherwise), which means that the processor's internal P-state selection logic is expected to focus entirely on performance.

This will override the EPP/EPB setting coming from the sysfs interface (see Energy vs Performance Hints below). Moreover, any attempts to change the EPP/EPB to a value different from 0 ("performance") via sysfs in this configuration will be rejected.

Also, in this configuration the range of P-states available to the processor's internal P-state selection logic is always restricted to the upper boundary (that is, the maximum P-state that the driver is allowed to use).

HWP + powersave

In this configuration intel_pstate will set the processor's Energy-Performance Preference (EPP) knob (if supported) or its Energy-Performance Bias (EPB) knob (otherwise) to whatever value it was previously set to via sysfs (or whatever default value it

was set to by the platform firmware). This usually causes the processor's internal P-state selection logic to be less performance-focused.

Active Mode Without HWP

This operation mode is optional for processors that do not support the HWP feature or when the `intel_pstate=no_hwp` argument is passed to the kernel in the command line. The active mode is used in those cases if the `intel_pstate=active` argument is passed to the kernel in the command line. In this mode `intel_pstate` may refuse to work with processors that are not recognized by it. [Note that `intel_pstate` will never refuse to work with any processor with the HWP feature enabled.]

In this mode `intel_pstate` registers utilization update callbacks with the CPU scheduler in order to run a P-state selection algorithm, either `powersave` or `performance`, depending on the `scaling_governor` policy setting in `sysfs`. The current CPU frequency information to be made available from the `scaling_cur_freq` policy attribute in `sysfs` is periodically updated by those utilization update callbacks too.

`performance`

Without HWP, this P-state selection algorithm is always the same regardless of the processor model and platform configuration.

It selects the maximum P-state it is allowed to use, subject to limits set via `sysfs`, every time the driver configuration for the given CPU is updated (e.g. via `sysfs`).

This is the default P-state selection algorithm if the :c:macro:`CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE` kernel configuration option is set.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\[linux-master][Documentation][admin-guide][pm]intel_pstate.rst`, line 172); *backlink*
>
> Unknown interpreted text role "c:macro".

`powersave`

Without HWP, this P-state selection algorithm is similar to the algorithm implemented by the generic `schedutil` scaling governor except that the utilization metric used by it is based on numbers coming from feedback registers of the CPU. It generally selects P-states proportional to the current CPU utilization.

This algorithm is run by the driver's utilization update callback for the given CPU when it is invoked by the CPU scheduler, but not more often than every 10 ms. Like in the `performance` case, the hardware configuration is not touched if the new P-state turns out to be the same as the current one.

This is the default P-state selection algorithm if the :c:macro:`CONFIG_CPU_FREQ_DEFAULT_GOV_PERFORMANCE` kernel configuration option is not set.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\[linux-master][Documentation][admin-guide][pm]intel_pstate.rst`, line 191); *backlink*
>
> Unknown interpreted text role "c:macro".

Passive Mode

This is the default operation mode of `intel_pstate` for processors without hardware-managed P-states (HWP) support. It is always used if the `intel_pstate=passive` argument is passed to the kernel in the command line regardless of whether or not the given processor supports HWP. [Note that the `intel_pstate=no_hwp` setting causes the driver to start in the passive mode if it is not combined with `intel_pstate=active`.] Like in the active mode without HWP support, in this mode `intel_pstate` may refuse to work with processors that are not recognized by it if HWP is prevented from being enabled through the kernel command line.

If the driver works in this mode, the `scaling_driver` policy attribute in `sysfs` for all `CPUFreq` policies contains the string "intel_cpufreq". Then, the driver behaves like a regular `CPUFreq` scaling driver. That is, it is invoked by generic scaling governors when necessary to talk to the hardware in order to change the P-state of a CPU (in particular, the `schedutil` governor can invoke it directly from scheduler context).

While in this mode, `intel_pstate` can be used with all of the (generic) scaling governors listed by the `scaling_available_governors` policy attribute in `sysfs` (and the P-state selection algorithms described above are not used). Then, it is responsible for the configuration of policy objects corresponding to CPUs and provides the `CPUFreq` core (and the scaling governors attached to the policy objects) with accurate information on the maximum and minimum operating frequencies supported by the hardware (including the so-called "turbo" frequency ranges). In other words, in the passive mode the entire range of available P-states is exposed by `intel_pstate` to the `CPUFreq` core. However, in this mode the driver does not register utilization update callbacks with the CPU scheduler and the `scaling_cur_freq` information comes from the `CPUFreq` core (and is the last frequency selected by the current scaling governor for the given policy).

Turbo P-states Support

In the majority of cases, the entire range of P-states available to `intel_pstate` can be divided into two sub-ranges that correspond to different types of processor behavior, above and below a boundary that will be referred to as the "turbo threshold" in what follows.

The P-states above the turbo threshold are referred to as "turbo P-states" and the whole sub-range of P-states they belong to is referred to as the "turbo range". These names are related to the Turbo Boost technology allowing a multicore processor to opportunistically increase the P-state of one or more cores if there is enough power to do that and if that is not going to cause the thermal envelope of the processor package to be exceeded.

Specifically, if software sets the P-state of a CPU core within the turbo range (that is, above the turbo threshold), the processor is permitted to take over performance scaling control for that core and put it into turbo P-states of its choice going forward. However, that permission is interpreted differently by different processor generations. Namely, the Sandy Bridge generation of processors will never use any P-states above the last one set by software for the given core, even if it is within the turbo range, whereas all of the later processor generations will take it as a license to use any P-states from the turbo range, even above the one set by software. In other words, on those processors setting any P-state from the turbo range will enable the processor to put the given core into all turbo P-states up to and including the maximum supported one as it sees fit.

One important property of turbo P-states is that they are not sustainable. More precisely, there is no guarantee that any CPUs will be able to stay in any of those states indefinitely, because the power distribution within the processor package may change over time or the thermal envelope it was designed for might be exceeded if a turbo P-state was used for too long.

In turn, the P-states below the turbo threshold generally are sustainable. In fact, if one of them is set by software, the processor is not expected to change it to a lower one unless in a thermal stress or a power limit violation situation (a higher P-state may still be used if it is set for another CPU in the same package at the same time, for example).

Some processors allow multiple cores to be in turbo P-states at the same time, but the maximum P-state that can be set for them generally depends on the number of cores running concurrently. The maximum turbo P-state that can be set for 3 cores at the same time usually is lower than the analogous maximum P-state for 2 cores, which in turn usually is lower than the maximum turbo P-state that can be set for 1 core. The one-core maximum turbo P-state is thus the maximum supported one overall.

The maximum supported turbo P-state, the turbo threshold (the maximum supported non-turbo P-state) and the minimum supported P-state are specific to the processor model and can be determined by reading the processor's model-specific registers (MSRs). Moreover, some processors support the Configurable TDP (Thermal Design Power) feature and, when that feature is enabled, the turbo threshold effectively becomes a configurable value that can be set by the platform firmware.

Unlike `_PSS` objects in the ACPI tables, `intel_pstate` always exposes the entire range of available P-states, including the whole turbo range, to the `CPUFreq` core and (in the passive mode) to generic scaling governors. This generally causes turbo P-states to be set more often when `intel_pstate` is used relative to ACPI-based CPU performance scaling (see below for more information).

Moreover, since `intel_pstate` always knows what the real turbo threshold is (even if the Configurable TDP feature is enabled in the processor), its `no_turbo` attribute in `sysfs` (described below) should work as expected in all cases (that is, if set to disable turbo P-states, it always should prevent `intel_pstate` from using them).

Processor Support

To handle a given processor `intel_pstate` requires a number of different pieces of information on it to be known, including:

- The minimum supported P-state.
- The maximum supported non-turbo P-state.
- Whether or not turbo P-states are supported at all.
- The maximum supported one-core turbo P-state (if turbo P-states are supported).
- The scaling formula to translate the driver's internal representation of P-states into frequencies and the other way around.

Generally, ways to obtain that information are specific to the processor model or family. Although it often is possible to obtain all of it from the processor itself (using model-specific registers), there are cases in which hardware manuals need to be consulted to get to it too.

For this reason, there is a list of supported processors in `intel_pstate` and the driver initialization will fail if the detected processor is not in that list, unless it supports the HWP feature. [The interface to obtain all of the information listed above is the same for all of the processors supporting the HWP feature, which is why `intel_pstate` works with all of them.]

User Space Interface in `sysfs`

Global Attributes

`intel_pstate` exposes several global attributes (files) in `sysfs` to control its functionality at the system level. They are located in the `/sys/devices/system/cpu/intel_pstate/` directory and affect all CPUs.

Some of them are not present if the `intel_pstate=per_cpu_perf_limits` argument is passed to the kernel in the command line.

`max_perf_pct`

Maximum P-state the driver is allowed to set in percent of the maximum supported performance level (the highest supported turbo P-state).

This attribute will not be exposed if the `intel_pstate=per_cpu_perf_limits` argument is present in the kernel command line.

`min_perf_pct`

Minimum P-state the driver is allowed to set in percent of the maximum supported performance level (the highest supported turbo P-state).

This attribute will not be exposed if the `intel_pstate=per_cpu_perf_limits` argument is present in the kernel command line.

`num_pstates`

Number of P-states supported by the processor (between 0 and 255 inclusive) including both turbo and non-turbo P-states (see Turbo P-states Support).

This attribute is present only if the value exposed by it is the same for all of the CPUs in the system.

The value of this attribute is not affected by the `no_turbo` setting described below.

This attribute is read-only.

`turbo_pct`

Ratio of the turbo range size to the size of the entire range of supported P-states, in percent.

This attribute is present only if the value exposed by it is the same for all of the CPUs in the system.

This attribute is read-only.

`no_turbo`

If set (equal to 1), the driver is not allowed to set any turbo P-states (see Turbo P-states Support). If unset (equal to 0, which is the default), turbo P-states can be set by the driver. [Note that `intel_pstate` does not support the general `boost` attribute (supported by some other scaling drivers) which is replaced by this one.]

This attribute does not affect the maximum supported frequency value supplied to the `CPUFreq` core and exposed via the policy interface, but it affects the maximum possible value of per-policy P-state limits (see Interpretation of Policy Attributes below for details).

`hwp_dynamic_boost`

This attribute is only present if `intel_pstate` works in the active mode with the HWP feature enabled in the processor. If set (equal to 1), it causes the minimum P-state limit to be increased dynamically for a short time whenever a task previously waiting on I/O is selected to run on a given logical CPU (the purpose of this mechanism is to improve performance).

This setting has no effect on logical CPUs whose minimum P-state limit is directly set to the highest non-turbo P-state or above it.

`status`

Operation mode of the driver: "active", "passive" or "off".

"active"
: The driver is functional and in the active mode.

"passive"
: The driver is functional and in the passive mode.

"off"
: The driver is not functional (it is not registered as a scaling driver with the `CPUFreq` core).

This attribute can be written to in order to change the driver's operation mode or to unregister it. The string written to it must be one of the possible values of it and, if successful, the write will cause the driver to switch over to the operation mode represented by that string - or to be unregistered in the "off" case. [Actually, switching over from the active mode to the passive mode or the other way around causes the driver to be unregistered and registered again with a different set of callbacks, so all of its settings (the global as well as the per-policy ones) are then reset to their default values, possibly depending on the target operation mode.]

`energy_efficiency`

This attribute is only present on platforms with CPUs matching the Kaby Lake or Coffee Lake desktop CPU model. By default, energy-efficiency optimizations are disabled on these CPU models if HWP is enabled. Enabling energy-efficiency optimizations may limit maximum operating frequency with or without the HWP feature. With HWP enabled, the optimizations are done only in the turbo frequency range. Without it, they are done in the entire available frequency range. Setting this attribute to "1" enables the energy-efficiency optimizations and setting to "0" disables them.

Interpretation of Policy Attributes

The interpretation of some `CPUFreq` policy attributes described in Documentation/admin-guide/pm/cpufreq.rst is special with `intel_pstate` as the current scaling driver and it generally depends on the driver's operation mode.

First of all, the values of the `cpuinfo_max_freq`, `cpuinfo_min_freq` and `scaling_cur_freq` attributes are produced by applying a processor-specific multiplier to the internal P-state representation used by `intel_pstate`. Also, the values of the `scaling_max_freq` and `scaling_min_freq` attributes are capped by the frequency corresponding to the maximum P-state that the driver is allowed to set.

If the `no_turbo` global attribute is set, the driver is not allowed to use turbo P-states, so the maximum value of `scaling_max_freq` and `scaling_min_freq` is limited to the maximum non-turbo P-state frequency. Accordingly, setting `no_turbo` causes `scaling_max_freq` and `scaling_min_freq` to go down to that value if they were above it before. However, the old values of `scaling_max_freq` and `scaling_min_freq` will be restored after unsetting `no_turbo`, unless these attributes have been written to after `no_turbo` was set.

If `no_turbo` is not set, the maximum possible value of `scaling_max_freq` and `scaling_min_freq` corresponds to the maximum supported turbo P-state, which also is the value of `cpuinfo_max_freq` in either case.

Next, the following policy attributes have special meaning if `intel_pstate` works in the active mode:

`scaling_available_governors`
List of P-state selection algorithms provided by `intel_pstate`.

`scaling_governor`
P-state selection algorithm provided by `intel_pstate` currently in use with the given policy.

`scaling_cur_freq`
Frequency of the average P-state of the CPU represented by the given policy for the time interval between the last two invocations of the driver's utilization update callback by the CPU scheduler for that CPU.

One more policy attribute is present if the HWP feature is enabled in the processor:

`base_frequency`
Shows the base frequency of the CPU. Any frequency above this will be in the turbo frequency range.

The meaning of these attributes in the passive mode is the same as for other scaling drivers.

Additionally, the value of the `scaling_driver` attribute for `intel_pstate` depends on the operation mode of the driver. Namely, it is either "intel_pstate" (in the active mode) or "intel_cpufreq" (in the passive mode).

Coordination of P-State Limits

`intel_pstate` allows P-state limits to be set in two ways: with the help of the `max_perf_pct` and `min_perf_pct` global attributes or via the `scaling_max_freq` and `scaling_min_freq` `CPUFreq` policy attributes. The coordination between those limits is based on the following rules, regardless of the current operation mode of the driver:

1. All CPUs are affected by the global limits (that is, none of them can be requested to run faster than the global maximum and none of them can be requested to run slower than the global minimum).
2. Each individual CPU is affected by its own per-policy limits (that is, it cannot be requested to run faster than its own per-policy maximum and it cannot be requested to run slower than its own per-policy minimum). The effective performance depends on whether the platform supports per core P-states, hyper-threading is enabled and on current performance requests from other CPUs. When platform doesn't support per core P-states, the effective performance can be more than the policy limits set on a CPU, if other CPUs are requesting higher performance at that moment. Even with per core P-states support, when hyper-threading is enabled, if the sibling CPU is requesting higher performance, the other siblings will get higher performance than their policy limits.
3. The global and per-policy limits can be set independently.

In the active mode with the HWP feature enabled, the resulting effective values are written into hardware registers whenever the limits change in order to request its internal P-state selection logic to always set P-states within these limits. Otherwise, the limits are taken into account by scaling governors (in the passive mode) and by the driver every time before setting a new P-state for a CPU.

Additionally, if the `intel_pstate=per_cpu_perf_limits` command line argument is passed to the kernel, `max_perf_pct` and `min_perf_pct` are not exposed at all and the only way to set the limits is by using the policy attributes.

Energy vs Performance Hints

If the hardware-managed P-states (HWP) is enabled in the processor, additional attributes, intended to allow user space to help `intel_pstate` to adjust the processor's internal P-state selection logic by focusing it on performance or on energy-efficiency, or somewhere between the two extremes, are present in every `CPUFreq` policy directory in `sysfs`. They are :

`energy_performance_preference`

Current value of the energy vs performance hint for the given policy (or the CPU represented by it).

The hint can be changed by writing to this attribute.

`energy_performance_available_preferences`

List of strings that can be written to the `energy_performance_preference` attribute.

They represent different energy vs performance hints and should be self-explanatory, except that `default` represents whatever hint value was set by the platform firmware.

Strings written to the `energy_performance_preference` attribute are internally translated to integer values written to the processor's Energy-Performance Preference (EPP) knob (if supported) or its Energy-Performance Bias (EPB) knob. It is also possible to write a positive integer value between 0 to 255, if the EPP feature is present. If the EPP feature is not present, writing integer value to this attribute is not supported. In this case, user can use the "/sys/devices/system/cpu/cpu*/power/energy_perf_bias" interface.

[Note that tasks may by migrated from one CPU to another by the scheduler's load-balancing algorithm and if different energy vs performance hints are set for those CPUs, that may lead to undesirable outcomes. To avoid such issues it is better to set the same energy vs performance hint for all CPUs or to pin every task potentially sensitive to them to a specific CPU.]

`intel_pstate` vs `acpi-cpufreq`

On the majority of systems supported by `intel_pstate`, the ACPI tables provided by the platform firmware contain `_PSS` objects returning information that can be used for CPU performance scaling (refer to the ACPI specification [3] for details on the `_PSS` objects and the format of the information returned by them).

The information returned by the ACPI `_PSS` objects is used by the `acpi-cpufreq` scaling driver. On systems supported by `intel_pstate` the `acpi-cpufreq` driver uses the same hardware CPU performance scaling interface, but the set of P-states it can

use is limited by the _PSS output.

On those systems each _PSS object returns a list of P-states supported by the corresponding CPU which basically is a subset of the P-states range that can be used by intel_pstate on the same system, with one exception: the whole turbo range is represented by one item in it (the topmost one). By convention, the frequency returned by _PSS for that item is greater by 1 MHz than the frequency of the highest non-turbo P-state listed by it, but the corresponding P-state representation (following the hardware specification) returned for it matches the maximum supported turbo P-state (or is the special value 255 meaning essentially "go as high as you can get").

The list of P-states returned by _PSS is reflected by the table of available frequencies supplied by acpi-cpufreq to the CPUFreq core and scaling governors and the minimum and maximum supported frequencies reported by it come from that list as well. In particular, given the special representation of the turbo range described above, this means that the maximum supported frequency reported by acpi-cpufreq is higher by 1 MHz than the frequency of the highest supported non-turbo P-state listed by _PSS which, of course, affects decisions made by the scaling governors, except for powersave and performance.

For example, if a given governor attempts to select a frequency proportional to estimated CPU load and maps the load of 100% to the maximum supported frequency (possibly multiplied by a constant), then it will tend to choose P-states below the turbo threshold if acpi-cpufreq is used as the scaling driver, because in that case the turbo range corresponds to a small fraction of the frequency band it can use (1 MHz vs 1 GHz or more). In consequence, it will only go to the turbo range for the highest loads and the other loads above 50% that might benefit from running at turbo frequencies will be given non-turbo P-states instead.

One more issue related to that may appear on systems supporting the Configurable TDP feature allowing the platform firmware to set the turbo threshold. Namely, if that is not coordinated with the lists of P-states returned by _PSS properly, there may be more than one item corresponding to a turbo P-state in those lists and there may be a problem with avoiding the turbo range (if desirable or necessary). Usually, to avoid using turbo P-states overall, acpi-cpufreq simply avoids using the topmost state listed by _PSS, but that is not sufficient when there are other turbo P-states in the list returned by it.

Apart from the above, acpi-cpufreq works like intel_pstate in the passive mode, except that the number of P-states it can set is limited to the ones listed by the ACPI _PSS objects.

Kernel Command Line Options for intel_pstate

Several kernel command line options can be used to pass early-configuration-time parameters to intel_pstate in order to enforce specific behavior of it. All of them have to be prepended with the intel_pstate= prefix.

disable

Do not register intel_pstate as the scaling driver even if the processor is supported by it.

active

Register intel_pstate in the active mode to start with.

passive

Register intel_pstate in the passive mode to start with.

force

Register intel_pstate as the scaling driver instead of acpi-cpufreq even if the latter is preferred on the given system.

This may prevent some platform features (such as thermal controls and power capping) that rely on the availability of ACPI P-states information from functioning as expected, so it should be used with caution.

This option does not work with processors that are not supported by intel_pstate and on platforms where the pcc-cpufreq scaling driver is used instead of acpi-cpufreq.

no_hwp

Do not enable the hardware-managed P-states (HWP) feature even if it is supported by the processor.

hwp_only

Register intel_pstate as the scaling driver only if the hardware-managed P-states (HWP) feature is supported by the processor.

support_acpi_ppc

Take ACPI _PPC performance limits into account.

If the preferred power management profile in the FADT (Fixed ACPI Description Table) is set to "Enterprise Server" or "Performance Server", the ACPI _PPC limits are taken into account by default and this option has no effect.

per_cpu_perf_limits

Use per-logical-CPU P-State limits (see Coordination of P-state Limits for details).

Diagnostics and Tuning

Trace Events

There are two static trace events that can be used for intel_pstate diagnostics. One of them is the cpu_frequency trace event generally used by CPUFreq, and the other one is the pstate_sample trace event specific to intel_pstate. Both of them are triggered by intel_pstate only if it works in the active mode.

The following sequence of shell commands can be used to enable them and see their output (if the kernel is generally configured to support event tracing):

```
# cd /sys/kernel/debug/tracing/
# echo 1 > events/power/pstate_sample/enable
# echo 1 > events/power/cpu_frequency/enable
# cat trace
gnome-terminal--4510  [001] ..s.  1177.680733: pstate_sample: core_busy=107 scaled=94 from=26 to=26 mperf=1143818 aperf=1230607
cat-5235  [002] ..s.  1177.681723: cpu_frequency: state=2900000 cpu_id=2
```

If intel_pstate works in the passive mode, the cpu_frequency trace event will be triggered either by the schedutil scaling governor (for the policies it is attached to), or by the CPUFreq core (for the policies with other scaling governors).

ftrace

The ftrace interface can be used for low-level diagnostics of intel_pstate. For example, to check how often the function to set a P-state is called, the ftrace filter can be set to :c:func:`intel_pstate_set_pstate`:

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\admin-guide\pm\[linux-master][Documentation][admin-guide][pm]intel_pstate.rst, line 730); backlink

Unknown interpreted text role "c:func".

```
# cd /sys/kernel/debug/tracing/
# cat available_filter_functions | grep -i pstate
intel_pstate_set_pstate
```

```
intel_pstate_cpu_init
...
# echo intel_pstate_set_pstate > set_ftrace_filter
# echo function > current_tracer
# cat trace | head -15
# tracer: function
#
# entries-in-buffer/entries-written: 80/80   #P:4
#
#                              _-----=> irqs-off
#                             / _----=> need-resched
#                            | / _---=> hardirq/softirq
#                            || / _--=> preempt-depth
#                            ||| /     delay
#           TASK-PID   CPU#  ||||    TIMESTAMP  FUNCTION
#              | |       |   ||||       |         |
              Xorg-3129  [000] ..s.  2537.644844: intel_pstate_set_pstate <-intel_pstate_timer_func
 gnome-terminal--4510  [002] ..s.  2537.649844: intel_pstate_set_pstate <-intel_pstate_timer_func
     gnome-shell-3409  [001] ..s.  2537.650850: intel_pstate_set_pstate <-intel_pstate_timer_func
          <idle>-0     [000] ..s.  2537.654843: intel_pstate_set_pstate <-intel_pstate_timer_func
```

References

[1] Kristen Accardi, *Balancing Power and Performance in the Linux Kernel*, https://events.static.linuxfound.org/sites/events/files/slides/LinuxConEurope_2015.pdf

[2] *IntelÂ® 64 and IA-32 Architectures Software Developerâ€™s Manual Volume 3: System Programming Guide*, https://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-software-developer-system-programming-manual-325384.html

[3] *Advanced Configuration and Power Interface Specification*, https://uefi.org/sites/default/files/resources/ACPI_6_3_final_Jan30.pdf