Design and Implementation of a New CPU Governor-Power Aware Governor

Arjun Haris, Johns Paul, Lijin C, Jayaraj P B
Department of Computer Science and Engineering
NIT Calicut,
Calicut, Kerala, India

arjunharis_bcs10@nitc.ac.in, johnspaul_bsc10@nitc.ac.in, lijin_bcs10@nitc.ac.in, jayarajpb@nitc.ac.in

Abstract

In this project, we propose a new CPU governor that provides good performance as compared to the other existing CPU governors. This CPU governor also optimizes the battery consumption. This is useful in devices like laptops, tablets and mobiles, i.e., any device that uses the Linux Kernel, especially when the battery charge is low. By lowering the CPU frequency and by reducing the time interval between the frequency switches, the battery charge is conserved. The battery charge remaining can be used as a factor which decides the frequency at which the CPU should run. Also, if more performance is required even at lower battery levels, we increase the frequency at regular intervals until an optimum performance is reached.

Keywords:CPU Governor, Power Aware Governor, battery conservation, dynamic frequency scaling.

Introduction

The power consumed by a processor is directly proportional to its frequency. The frequencies of the modern day processors can be easily adjusted by the Operating Systems. The power consumption in a modern day processor is given by

$$P = C * V^2 * f$$

Where P is the power, C is the capacitance being switched per clock cycle, V is the voltage, and f is the processor frequency (cycles per second).

The system does not require its process to run at maximum frequency at all times. For normal use, a CPU is needed to run at its maximum frequency for only 20-30% of the total time. So the modern operating systems use a feature called dynamic frequency scaling, where the frequency of the processor is changed dynamically to reduce the power consumption. And this frequency scaling is done by the use of CPU governors.
Studies show that with a decrease in the frequency of the CPU there is considerable decrease in the power consumed by it. Also lower the intervals at which frequency switches happen higher is the power consumption.

The following diagram implies that the CPU utilization is much below 40% most of the time for a normal user. The small peaks (in Fig1) show that a new application has been opened or there has been a switch between applications. The CPU utilization is maximum when we run heavy applications like video editing software or playing games. During this time, the CPU frequency is mostly switched between 1.3 GHz to 1.5 GHz and when there is a switch between applications it can go up to 1.9 GHz. The studies show that the use of low CPU frequency and a large interval between frequency switches is better in such a scenario (for saving power).

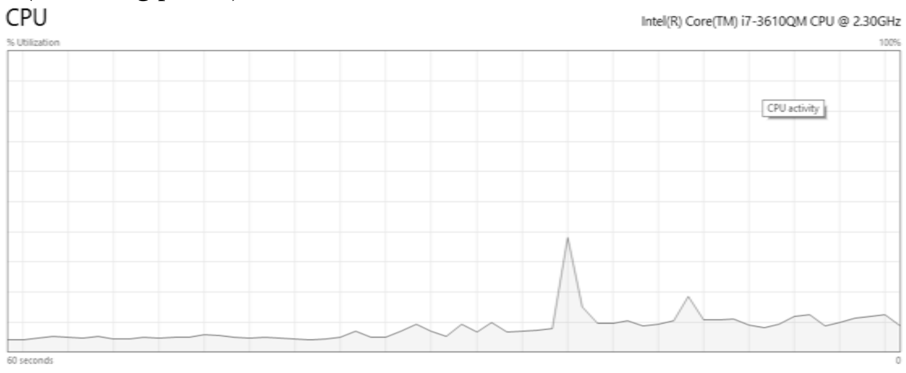
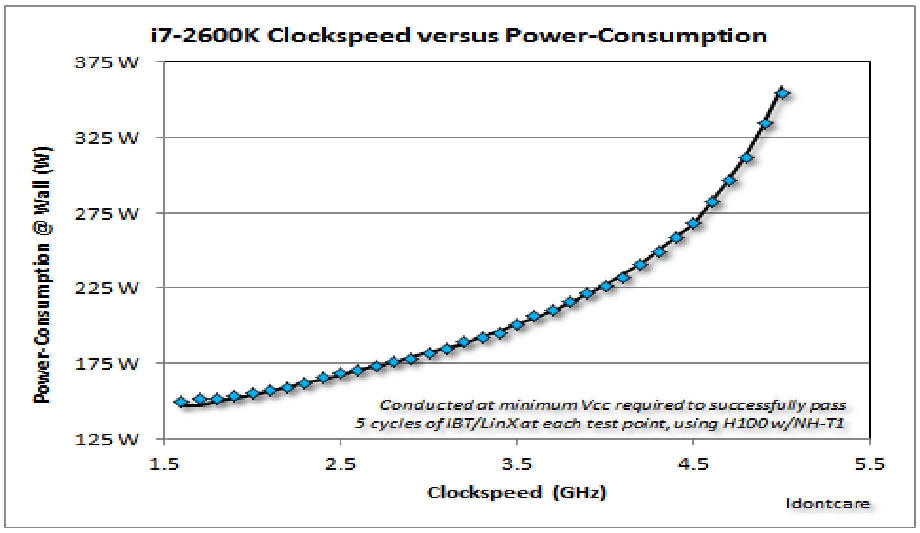


Figure 1. Graph showing the CPU percentage utilization of an i7 processor for a typical laptop user



Graph 1. Power Consumption vs Clockspeed for an i7 processor [8]

This paper is divided in to 6 sections. Section 2 describes Background information on existing CPU governors. Section 3 shows our proposed design. The implementation and result obtained is explained in section 4. Conclusion and future scope are described in section 5.

Background

Dynamic frequency scaling (also known as CPU throttling) is a technique in computer architecture whereby the frequency of a microprocessor can be automatically adjusted "on the fly," either to conserve power or to reduce the amount of heat generated by the chip.

Dynamic frequency scaling is commonly used in laptops and other mobile devices, where energy comes from a battery and thus is limited. It is also used in quiet computing settings and to decrease energy and cooling costs for lightly loaded machines. Less heat output, in turn, allows the system cooling fans to be throttled down or turned off, reducing noise levels and further decreasing power consumption. It is also used for reducing heat in insufficiently cooled systems when the temperature reaches a certain threshold, such as in poorly cooled overclocked systems.

CPU frequency governors decide which frequency among the CPU frequency policy should be used. In other words CPU governor controls how the CPU raises and lowers its frequency in response to the loads the user is placing on the CPU.

CPU Governors in Linux

In the Linux operating system there are 5 CPU governors by default:

1. The PowerSave CPU Governor
2. The Performance CPU Governor
3. The OnDemand CPU Governor
4. The Conservative CPU Governor

1. PowerSave CPU Governor [4]

Powersave, sometimes called "Always Min," forces the CPU to run at its minimum allowed frequency constantly. This reduces power consumption and considerably increases battery life. But since the frequency is always set to minimum, the governor can result in poor performance and the system will become slow.

2. Performance CPU Governor [3]

Performance, also referred to as "Always Max," forces the CPU to run at its maximum allowed frequency constantly. It is mainly used during benchmarking. Since the frequency is set to the maximum value, this governor gives very high performance and speed but it will also cause higher power consumption which translates to lower battery life. This will cause an overheating of the device. If used for extended periods of time, it can even lead to permanent physical damage.

3. OnDemand CPU Governor [4]

OnDemand is the default governor in almost all stock kernels. One main goal of the OnDemand governor is to switch to max frequency as soon as there is a CPU activity detected to ensure the responsiveness of the system. Effectively, it uses the CPU busy time as the answer to “how critical is performance right now” question. So OnDemand jumps to maximum frequency when CPU is busy and decreases the frequency gradually when CPU is less loaded. One potential reason for OnDemand governor being not very power efficient is that the governor decide the next target frequency by instant requirement during sampling interval. The instant requirement can respond quickly to workload change, but it does not usually reflect the real CPU usage requirement in a longer time interval and it possibly causes frequent changes between the highest and the lowest frequency. Also if the workload goes above a specified percentage this governor simply increases the frequency to the maximum possible value rather than increasing the frequency linearly.

4. Conservative CPU Governor [4]

Conservative governor scales up the CPU frequency slowly, to optimize battery life. The conservative governor is similar to the OnDemand governor. It functions like the OnDemand governor by dynamically adjusting frequencies based on processor utilization. However, the conservative governor increases and decreases CPU speed more gradually. Simply put, this governor increases the frequency step by step on CPU load and jumps to lowest frequency on CPU idle. Conservative governor aims to dynamically adjust the CPU frequency to current utilization, without jumping to max frequency. The main advantage of this governor is that it optimizes power saving than OnDemand governor as it won't fully ramp up the CPU until it's really needed.Also if the CPU utilization goes below a specified value for a small time interval then the frequency is decreased to a minimum value and after that it takes some time to reach the optimum frequency value.

Drawbacks of the Existing CPU Governors

All these existing CPU governors can be considered useful in many of the real life situations. But we don't have a single governor that can be used in all the situations. Consider a laptop which is working on battery. When the battery is full, the performance governor seems to be a very good option because at this point we are not worried about the battery charge remaining and we want to concentrate more on the performance. Now when the battery charge is in the 50%-60% range, we might want to change the strategy, because at this point the amount of charge left in the battery is an issue. So the OnDemand governor seems to be a better option at this stage (as it can ramp up the frequency much faster than our proposed CPU governor). Now as the charge gets lower the powersave governor seems to be a very good option. But it means that the CPU frequency will always be a minimum value. So if the user wants to run a process that requires large amount of calculations it will take longer time to complete. And running a processor at lower frequency for a longer time drains out the battery more quickly than the performance strategy. But the performance governor is a bad choice if this CPU intensive process is running only for a small time interval.

If we go for OnDemand and if a CPU intensive process is running at regular intervals, it can cause rapid switching between the min and max frequencies and this will drain the battery quickly. If we use the conservative CPU governor, the CPU utilization goes below a specified value for a small time.

Proposed Design

The new CPU governor proposed here is a more efficient algorithm and considers the optimization of battery life and the performance. The frequency has to increase when the CPU intensive process is running but it need not increase to the max value like in the case of the OnDemand governor. Then the range over which the frequency varies will be much less and this helps in saving power. This is a CPU governor that is intended to come into action when the battery charge is less than a critical value because conserving the battery charge becomes necessary at low battery level. When the battery level is above critical value the algorithm used by the OnDemand governor can be used, as performance is more important at this point.

A typical modern processor works in the frequency range of 500 MHz to 2300 MHz.The proposed governor adjusts the frequency according to the formula

$$\mathrm{CPUFreq} = \max(\min, \frac{(\max - \min) * B}{100})$$

Where min is the minimum CPU frequency, max is the maximum CPU frequency and B is the percentage of charge left in the battery. So at a very low battery (below 15) range, this works in a manner that is similar to a PowerSave governor. We keep changing the CPU Frequency for every 2.5% decrease in the battery level. Due to this, rapid CPU frequency changes do not occur as in the case of the OnDemand governor.

But this strategy causes issues when high performance is required, even at lower battery levels. So to solve this issue some modifications can be done based on CPU utilization. CPU utilization monitoring shows the workload of a given physical processor. A measure of CPU utilization is the CPU idle time. Higher the idle time lower is the CPU utilization.

So if the CPU utilization remains very high (above an allowed value) for a predefined time interval then it increases the CPU frequency from the current value by a constant. 1 second is used as the predefined time interval (We use such a large value to avoid rapid switching of frequencies). So once the CPU frequency is limited to a certain value the governor keep monitoring the CPU utilization for time intervals of 1 second. And in this interval if the idle time is less than 20% a counter is set as 1 and the frequency is increased by an amount

counter * 100 MHz

If in the next interval also the idle time remains lower than 20% then increase the counter by 1 and increase the CPU frequency again by the amount given by the new counter value. And this continues until the frequency reaches the maximum possible value. If at some point the idle time value comes down to the range 20%-40% then reset the counter value to 1. And if the idle time goes above 40% then the frequency is reset to the value given by equation (2).

Similarly, if the CPU utilization is too low at the frequency set by equation (2) then the frequency is decreased by a similar strategy. If the idle time is in the rage 60%-100% the counter is set to 1 and the CPU frequency is decreased by an amount

counter * 100 MHz

If in the next interval also the idle time remains in the same range then we increase the counter by 1 and decrease the CPU frequency again by the amount given by the new counter value. And this continues until the frequency reaches the minimum possible value. And now if the idle time goes below 60% then the CPU frequency is reset to the values given by the equation (2).

Optimizing the Sampling Rate

Sampling rate (measured in micro seconds) tell us how often the kernel looks at the CPU usage and to make decisions on what to do about the frequency. The lower the sampling higher is the power consumption. This is because energy is lost because of the rapid switching of frequency. So for conserving battery its better to keep the sampling rate high. But sampling rate increases the system will become less responsive. The system won’t be able to scale up its frequency in case a heavy workload comes in instantaneously. So an optimal value of sampling rate should be used.

So to conserve battery charge the sampling rate should be increased until a value which does not cause a considerable performance impact. Also it is possible to adjust the sampling rate dynamically. The sampling rate can easily be changed dynamically based on the past history to obtain an optimal sampling rate. If a system was working under a steady workload for some time then it means that there is a high probability that the workload will remain the same for the next few sampling intervals. So in that case we can increase the sampling rate and thus avoid rapid switching of frequency and conserve battery charge.

An array is used to store the past history of the system workload. This array stores the frequency at which the system was working over the last 10 sampling rates. Now if all the entries in the array are the same then it means that the system was working under a steady workload. Then the sampling rate is made 11 times the default value to avoid rapid switching between frequencies

But there is possibility that the value of the frequency was different in just a single interval and because of this the sampling rate may not change. To avoid this the standard deviation of the set of frequencies can be obtained and the sampling rate can be scaled up if the value of standard deviation is below a threshold. This method provides a much better idea about the past history. But the process of standard deviation involves complex computations like finding the square root. So repeating these computation at regular interval creates a large overhead on the system. This can cause a considerable performance decline in the system and even reduce the battery performance thus reducing the effect of the algorithm. So it is better to use operations like XOR while analyzing the past history

Implementation and Result

Implementation

Algorithm 1: Power Aware Governor

```
1.  $bat\_flag \leftarrow 0$
2.  $freq\_up\_counter \leftarrow 0$
3.  $freq\_down\_counter \leftarrow 0$
4.  $bat\_prev \leftarrow 100 + bat\_int$
5.  $freq\_history[] \leftarrow \{1,2,3,4,5,6,7,8,9,10\}$
6.  $hist\_counter \leftarrow 0$
7.  $hist\_flag \leftarrow 1$
8.  $hist\_loop\_cntr \leftarrow 0$
9.  $for\ each\ curr\_sampling\_rate\ do$
10. $if\ bat\_precent <$ def_battery_threshold AND charging is $False\ then$
11. $if\ bat\_flag == 0$
12. $bat\_flag = 1$
13. $old\_sampling\_rate \leftarrow curr\_sampling\_rate$
14. $end\ if$
```

```
15. $if\ bat\_prev - bat\_percent\ \geq bat\_int\ then$
16. $bat\_prev \leftarrow bat\_percent$
17. $target\_freq \leftarrow \max\left(policy \rightarrow min, \frac{(policy \rightarrow max - policy \rightarrow \min) * \text{bat\_precent}}{100}\right)$
18. $end\ if$
19. $if\ hist\_counter == 10$
20. $hist\_counter \leftarrow 0$
21. $endif$
22. $freq\_history[hist\_counter] \leftarrow policy \rightarrow cur$
23. $hist\_counter \leftarrow hist\_counter + 1$
24. $hist\_flag \leftarrow 1$
25. $hist\_flag \leftarrow freq\_history[0]\text{^}freq\_history[1]\text{^} \ldots\ldots \text{^}freq\_history[9]$
26. $if\ hist\_flag == 0\ AND\ od\_tuners \rightarrow sampling\_rate == bat\_sampling\_rate$
27. $od\_tuners \rightarrow sampling\_rate \leftarrow od\_tuners \rightarrow sampling\_rate$
28. $else\ if\ hist\_flag! = 0$
29. $if\ od\_tuners \rightarrow sampling\_rate! = bat\_sampling\_rate$
30. $od\_tuners \rightarrow sampling\_rate \leftarrow bat\_sampling\_rate$
31. $hist\_flag \leftarrow 1$
32. $endif$
33. $endif$
34. $if\ load\_freq\ > up\_threshold * policy \rightarrow curr\ then$
35. $freq\_down\_counter \leftarrow 0$
36. $target\_freq \leftarrow target\_freq + freq\_up\_counter * 10000$
37. $freq\_up\_counter \leftarrow freq\_up\_counter + 1$
38. $end\ if$
39. $if\ load\_freq\ \leq up\_threshold * policy \rightarrow curr\ then$
40. $freq\_up\_counter \leftarrow 0$
41. $target\_freq \leftarrow target\_freq - freq\_down\_counter * 10000$
42. $freq\_down\_counter \leftarrow freq\_down\_counter + 1$
43. $end\ if$
44. $if\ target\_freq \leq policy \rightarrow \min then$
45. $freq\_down\_counter \leftarrow 0$
46. $target\_freq \leftarrow policy \rightarrow curr$
47. $end\ if$
48. $if\ target\_freq \geq policy \rightarrow \max then$
49. $freq\_up\_counter \leftarrow 0$
50. $target\_freq \leftarrow policy \rightarrow curr$
51. $end\ if$
52. $end\ if$
53. $if\ bat\_flag == 1$
54. $bat\_flag \leftarrow 0$
55. $bat\_prev \leftarrow 100 + bat\_int$
56. $freq\_up\_counter \leftarrow 0$
57. $freq\_down\_counter \leftarrow 0$
58. $curr\_sampling\_rate \leftarrow old\_sampling\_rate$
59. $end\ if$
60. $end\ for$
```

The above algorithm uses equation (2) to calculate target frequency when battery charge goes below a threshold value. If load_freq is greater than up_threshold*current frequency then we increase target frequency in steps of 100, 200, 300, 400 MHz… else we decrease target frequency by 100, 200, 300, 400 MHz… It also increases the sampling rate so that frequency does not change too often thus helping in faster race-to-idle compared to Conservative Governor.

Result

A new CPU governor was developed (using Linux kernel 3.10) according to the proposed design and was tested on a system with i5 processor.

Each governor was separately tested on a core-i5 system with 4GB ram running 3 applications –A 1080p H264 video with bitrate 3905 kbps on VLC, Google chrome with 3 Tabs Open and running make on Linux kernel source.

When compared to the OnDemand governor the new governor provided 5 minutes more battery backup for every 20% decrease in the battery level. The performance provided by the OnDemand governor was good but the proposed governor provided much better performance than the conservative governor. Performance governor gave more battery life than OnDemand because in new CPUs , running a CPU at its maximum frequency at all times will allow a faster race-to-idle.

Governor	**Time Taken to Finish 20% of Battery(mins)**
Powersave	31
Performance	25
OnDemand	23
Conservative	26
Interactive	23
Power Aware linear	28
Power Aware Exponential	30

Table 2. Table shows time taken by each governor to finish 20% battery charge.

Race-to-idle is the process by which a CPU completes a given task, and returns the CPU to the extremely efficient low-power state. This still requires extensive testing, and a kernel that properly implements a given CPU's C-states (low power states) [3].

The battery conserving capability of the proposed governor was also tested using a benchmarking software (phoronix test suite) which produced the following results:

Normal kernel running the default OnDemand governor
Average: 14067.26 (mW)
Minimum: 9025.76 (mW)
Maximum: 16730.98 (mW)

Kernel with new CPU governor (sampling rate scaled based on equality of previous frequencies)
Average: 13718.11 (mW)

Minimum: 8768.68 (mW)
Maximum: 14775.98 (mW)

Kernel with new CPU governor (sampling rate scaled based on standard deviation of previous frequencies)
Average: 13905.46 (mW)
Minimum: 8939.79 (mW)
Maximum: 15915.12 (mW)

References

[1] AlexeyStarikovskiy and VenkateshPallipadi,TheOnDemand Governor 2006

[2] Adding features to the Linux kernel page

Available at:http://xda-university.com/as-a-developer/adding-features-to-your-kernel

[3] CPU governors

Available at:http://rootzwiki.com/topic/40336-cpu-governors-explained/

[4] CPU governors

Available at:http://www.cyann.mobi/article/show/governor

[5] How to compile the Linux kernel page

Available at:http://linuxtweaking.blogspot.in/2010/05/how-to-compile-kernel-on-ubuntu-1004.html

[6] Linux source code from git

Available at:git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git

[7] Linux source code in lxr free electron

Available at:http://lxr.free-electrons.com/

[8] Power-Consumption Scaling with Clockspeed and Vcc for the i7-2600K

Available at:http://forums.anandtech.com/showthread.php?t=2195927

[9] DominikBrodowski. Current trend inlinux kernel power management, linuxtag 2005.