DOI:10.1145/3190560

Article development led by acmqueue
queue.acm.org

Workload scalability has a cascade relation via the scale factor.

BY NOOR MUBEEN

Workload Frequency Scaling Law—Derivation and Verification

MANY PROCESSORS EXPOSE performance-monitoring counters that help measure 'productive performance' associated with workloads. Productive performance is typically represented by *scale factor*, a term that refers to the extent of *stalls* compared with *stall-free* cycles within a time window. The scale factor of workload is also influenced by clock frequency as selected by frequency-selection governors. Hence, in a dynamic voltage/frequency scaling or DVFS system (such as Intel Speed Shift[1]), the utilization, power, and performance outputs are also functions of the scale factor and its variations. Some governance algorithms do treat the scale factor in ways that are native to their governance philosophy.

This article presents equations that relate to workload utilization scaling at a per-DVFS subsystem level. A relation between frequency, utilization, and scale factor (which itself varies with frequency) is established. The verification of these equations turns out to be tricky, since inherent to workload, the utilization also varies seemingly in an unspecified manner at the granularity of governance samples. Thus, a novel approach called *histogram ridge trace* is applied. Quantifying the scaling impact is critical when treating DVFS as a building block. Typical application includes DVFS governors and/or other layers that influence utilization, power, and performance of the system. The scope here though, is limited to demonstrating well-quantified and verified scaling equations.

Workload Scaling

Intel has three architecture-independent registers that are relevant to this topic:

Figure 1. Scalability-related definitions.

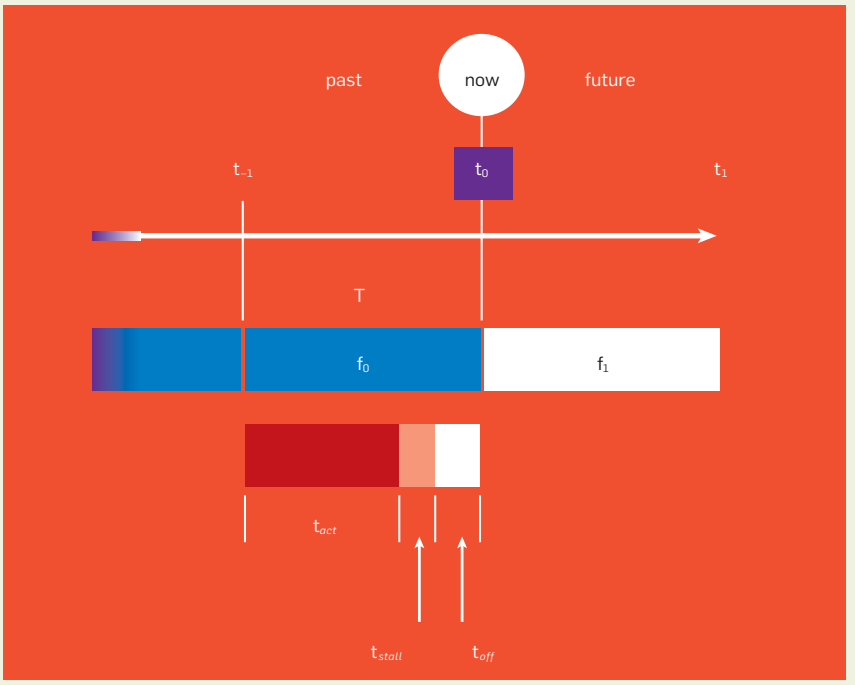


Figure 2. Perf counter delta relating to reference time window.

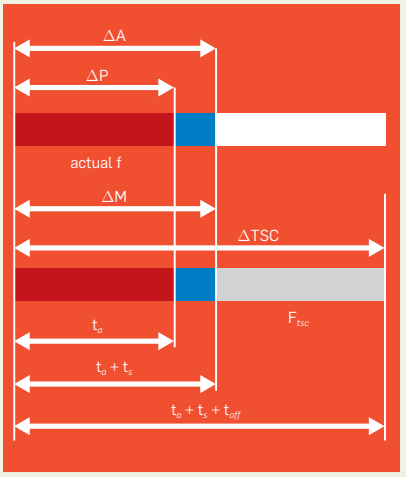
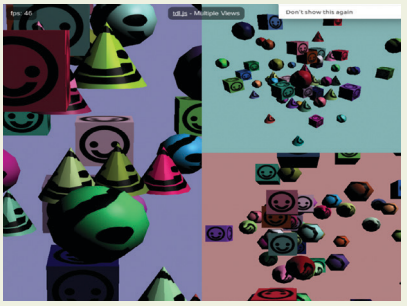


Figure 3. A browser-based workload from WebGLSamples.org



▸ A_{perf}. A running counter that counts at actual clock rate of execution at that instant. This actual clock frequency may vary over time based on governance and/or other algorithms. This register counts only during active state (C0).

▸ M_{perf}. A running counter for activity that counts at a fixed TSC (time stamp counter) clock rate. It counts only during active state (C0).

▸ P_{perf}. This counter is similar to A_{perf}, except that it does not count when the activity stalls as a result of some dependency, likely gated on another IP's clock domain (for example, memory).

The delta of these counters in a given time window commonly interpret the following:

▸ Utilization. $U = M_{perf}/TSC$

▸ Scale factor. $S = \Delta P_{perf}/\Delta A_{perf}$

Ideally, if an activity is free of stalls, then $P_{perf} = A_{perf}$ (that is, the scale factor S will be 1). In such a situation, the time taken to complete an activity in a given time window would simply be the inverse of the actual frequency in that window. In most real workloads the scale factor varies and is often less than 1. Establishing accurate equations relating utilization, scale factor, and frequency allows DVFS governors to dispense well-informed frequency-change decisions.

At the outset, it may seem the workload-scaling problem is similar to the well-known Amdahl's Law, which deals with workload speedup as associated with parallel-compute resources. However, Amdahl's Law cannot be applied here since the "parallelizability factor" analogy to "scale factor" does not hold up—the former is independent of the resource being scaled (that is, parallel processors), while the latter is a function of the resource being scaled (frequency). Utilization scalability over frequency has a cascade relation via the scale factor.

Terminology and Assumptions

This article makes use of following terminology:

▸ *Utilization or C0 activity percentage* refers to the active (non-idle) clocked state[3] within a time window. Load and utilization, used interchangeably here, refer to the same attribute.

▸ *Scalable activity* refers to the part (or percentage) of the operation whose execution-time scales inversely with frequency.

▸ *Stall in activity* refers to the part (or percentage) of the operation that involves stall during its completion. While a longer delay actually causes C-state demotion, the instruction stalls referred to here are much shorter and occur with the CPU in C0 active state.

Consider a frequency governor on a DVFS system, updating frequency at every periodic time window T. Consider the present instant at time t_0 ("NOW"), as depicted in Figure 1. Let the just-completed window T of workload have t_{act} (or t_a) as part of its scalable activity (that is, t_{act} represents the cumulative portion of activity that keeps the processor busy in execution without the need for any dependent delay or stalls). Let t_{stall} (or t_s) be the sub-duration depicting the effective stalls experienced (inter-subsystem dependency stalls). Also let t_{off} (or t_o) be the cumulative duration where the DVFS subsystem is in a deeper C-state, which is a significantly lower power than C0 state. The fundamental difference between t_{off} and t_{stall} is that, in the latter case the subsystem is still active C0 while experiencing very short dependency stalls; whereas in the former case the delays are large enough and put the subsystem into a momentary deeper C-state.

In the given window, the load execution active-time is the inverse of the frequency f.

Or, in general, in any window:

$$t_{act} \propto \frac{1}{f}$$

Intel productive performance[2] register counts the productive (non-stall) counts proportional to t_{act}. Defining scalability S as a ratio of delta in productive-count (P_{perf}) to active-count (A_{perf}), results in the following general equations:

$$s = \frac{delta\ P_{perf}}{delta\ A_{perf}} = \frac{t_{act}}{(t_{act} + t_{stall})}$$

Also, the load l (C0 percentage) in this window can be defined as:

$$l = \frac{(t_{act} + t_{stall})}{(t_{act} + t_{stall} + t_{off})}$$

Causality. Consider one specific window T between t_{-1} and t_0. Here, the governor has dispensed a frequency f_0. *If* the governor had dispensed a different frequency f'_0 in the *same* window, the scalable part of the work (that is, t_a) would have been, say t'_a and thereby t_0 to t'_0. For practical purposes, the stall cycles can be assumed to be unrelated to the selected frequency, since the stalls are not explicitly dependent on a local subsystem's clock frequency. Hence t_s remain independent as far as their causality with frequency is considered. Similarly, within the same window the number of productive (stall-free) instructions to be executed remain fixed. Specifically, we can generalize and imply that:

Stall time and productive cycle count are invariant of frequency causality.

Note also that frequency and load are *causality* considerations *within* a time window, not across. In the rest of the derivation, a reference to change in frequency is not along the time flow, but another possible causality if the frequency were some other.

Utilization (Load) to Frequency Change Estimation

An important aspect in scalability of workload relates to the extent of change in the load caused by a change in frequency.

As shown in Figure 2, the sampled delta values of P_{perf} and A_{perf} as ΔP_0, ΔA_0 within windows ΔTSC can be associated with active, stall, and off time within T.

$$t_a = \frac{1}{f_0} \Delta P_0$$

$$t_a + t_s = \frac{1}{f_0} \Delta A_0$$

$$t_a + t_s + t_{off} = \frac{1}{F_{tsc}} \Delta TSC$$

Now, assume the DVFS governor *had* chosen some target f'_0 instead of initial actual frequency f_0. The requirement is to find a relation that accurately represents the changed final target load l'_0 as a result of this causality. Hence derive estimated load change l_1 at time t_1.

Let the A_{perf} count in the causality window with frequency f_0 hypothetically transition from the present ΔA_0 to $\Delta A'_0$

$$\Delta A'_0 = f'_0 (t'_a + t'_s)$$

The stall time is not an explicit function of a local subsystem clock, the time spent in stall remains the same. In other words, *stall time is invariant of frequency causality*.

$$t'_s = t_s$$

$$\Delta A'_0 = f'_0 \left\{ \frac{\Delta P'_0}{f'_0} + \frac{\Delta A_0 - \Delta P_0}{f_0} \right\}$$

When we solve this with the following two equations for load and scale-factor in general, at time t_n:

$$\Delta A = f_n\, l_n\, \Delta TSC / F_{tsc}$$

and the scale factor as

$$s_n = \frac{\Delta P_n}{\Delta A_n}$$

Figure 4. Utilization variation in the time domain.

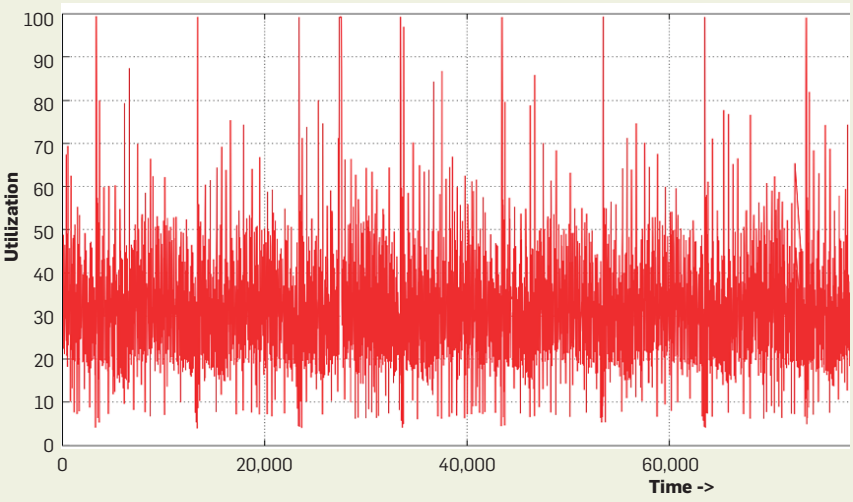


Figure 5. Utilization histogram plot at fixed frequency 1GHz.

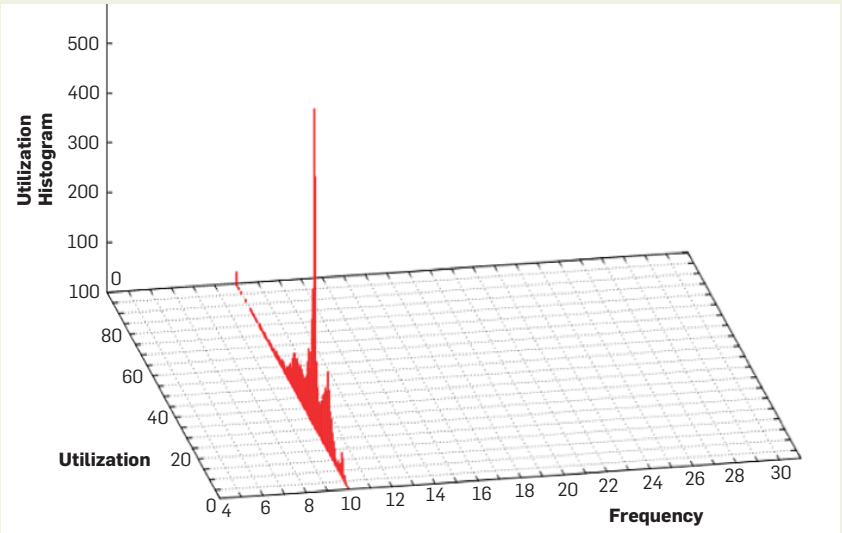


Figure 6. Collection of utilization histograms for each frequency.

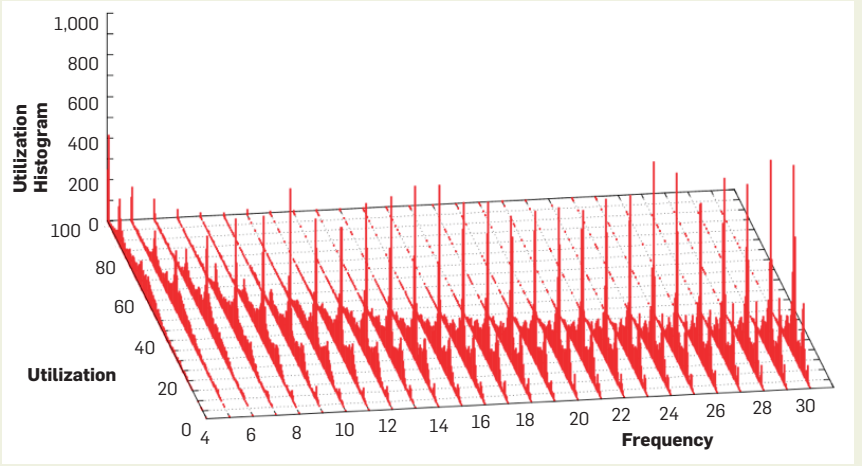
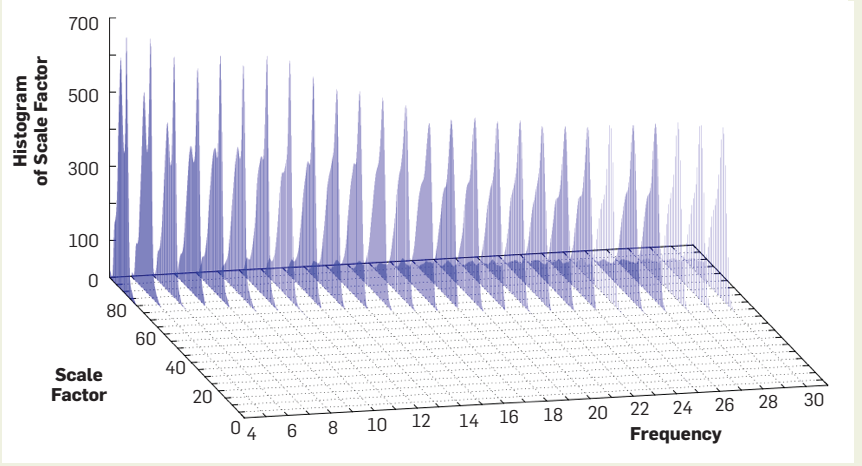


Figure 7. Scale-factor histograms for each frequency.



solving yields

$$l'_0 = s'_0\, l'_0 + l_0 - s_0 l$$

or

$$l'_0 = l_0 \frac{1 - s'_0}{1 - s_0}$$

Now,

$$s'_0 = \frac{t'_a}{(t'_a + t'_s)}$$

Again, due to stall time invariance,

$$\frac{(\frac{1}{s'_0} - 1)\,\Delta P'_0}{f'_0} = \frac{(\frac{1}{s_0} - 1)\,\Delta P_0}{f_0}$$

Since this is not an iteration to the next time window, the productive cycles to be executed remain the same. In other words,

The productive cycle count is invariant of frequency causality.

$$\Delta P'_0 = \Delta P_0$$

This results in the *scaling of scale factor* equation

$$s'_0 = \frac{1}{1 + \frac{f'_0}{f_0}(\frac{1}{s_0} - 1)}$$

This intermediate equation defines the changes (scaling) of the scale factor itself over frequency. Solving further, we get the *scaling of load* equation:

$$l'_0 = l_0 (s_0 \cdot \frac{f_0}{f'_0} + (1 - s_0))$$

In the causality-based derivation, there was no workload inherent change in that window. In other words, if the causality from f_0 to f'_0 were actual change f_1 in adjacent future time window t_1, and the equation were used to estimate such a target load l_1, then any divergence ($l_{1actual} - l'_0$) from expected target load is clearly workload inherent change culminated over the time window t_0 to t_1. Since $l_{1actual}$ can be known, any divergence from the estimated load can be calculated. The estimated scaled load is this given by:

$$l_1 = l_0 (s_0 \cdot \frac{f_0}{f_0} + (1 - s_0))$$

Similarly, the estimated scaled scale-factor is approximately given by:

$$s_1 = \frac{1}{1 + \frac{f_1}{f_0}(\frac{1}{s_0} - 1)}$$

Verification on an Actual Platform

The last two equations can be called the workload scaling equations. Verifying them empirically on an actual platform, however, isn't as straightforward. At the granularity of a single cycle, the three parameters on the right (f_0,f_1,s_0) may be determined; but as described, the math result cannot be compared directly to l after the end of that cycle; primarily because the incoming load itself could fluctuate as seen at a discrete sample basis. It is possible though, to statistically verify by applying a series of histograms at different frequencies and tracing its ridge. Using this *histogram ridge trace* method, the generality of the equation is verified.

To begin, a workload is run completely at a fixed frequency.

Workload #1 is a random browser-based graphics load (Figure 3; https://webglsamples.org/multiple-views/multiple-views.html)

While capturing the data, it is preferred to stop unwanted background services that are irrelevant to the workload under analysis. Also, a precise and lightweight data-logging tool is preferred. On Linux systems, the Intel PSST[4] can accomplish logging at fixed, low, and precise overhead. The data is captured for, say, a duration of 100 seconds and a sampling poll period of 20ms. Data captured with every sample includes:

- CPU load.
- scale factor.
- clamped frequency value.

The entire run is repeated at every frequency of CPU, while keeping other DVFS as fixed sources (for example, Gfx frequency at a moderate but fixed value).

The CPU scale-factor and utiliza-

tion logs are captured. In this example, time-domain analysis is not practical for gathering scalability and utilization. Instead, a histogram analysis identifies in which bucket the maximum number of load points lie for a given frequency of operation. The peak of the histogram represents the load value with the highest probability of occurrence for the given frequency. In the example data in Figure 5, about 500 samples (or 80% of total samples) occurred at a 33% load. Such inferences are obviously hard to derive from the time chart in Figure 4 even if the frequency is fixed.

The experiment is iterated through all other frequencies, one at a time, to get a range of peak histogram values, as shown in Figure 6.

With the same data from the set of experiments, a similar histogram ridge trace for scale-factor can be plotted as shown in Figure 7.

Scaling of scale factor. A plotting tool is used to create a map-view of peak histogram points (in scale factor) in Figure 7, which is then plotted with red dots in Figure 8. On the same map-view, the *scaling of scale factor* equation is plotted for different possible target frequencies based on any one initial *seed* value for utilization and scale factor. The correlation between the equation and the full data set is shown in Figure 8.

Scaling of load. The map-view of peak histogram points of utilization (Figure 6) is plotted with red dots in Figure 9. On the same map view, the *scaling of load* equation is plotted for different possible target frequencies based on one initial seed value for utilization and scale factor.

As summarized by the results shown in these figures, the independent estimates of the equation fit nearly precisely with the actual experiment's statistical results. This methodology is applied to multiple other workloads (with different scale factors and load levels) and verified as noted here.

Conclusion

This article elaborated on the causalities of utilization and scale factor in relation to possible frequency-selection choices within a time window. Based on this, generic workload scaling equations are derived, quantifying utilization impact. The equations are verified by applying the histogram ridge trace method at discrete DVFS block level. Though only the CPU core example was detailed, similar equation agreements were observed on other DVFS blocks (graphics sub blocks), other workloads and other operating systems (Windows/Linux). The equation-estimated curve correlates very accurately with the actual ridge trace curve. This implies the utilization impact can be "predicted" to be statistically accurate in every cycle and any divergence from it can be attributed to workload-inherent utilization change in that cycle and treated as appropriate to the solution using it.

Figure 8. Scale-factor ridge points contour compared with equation.

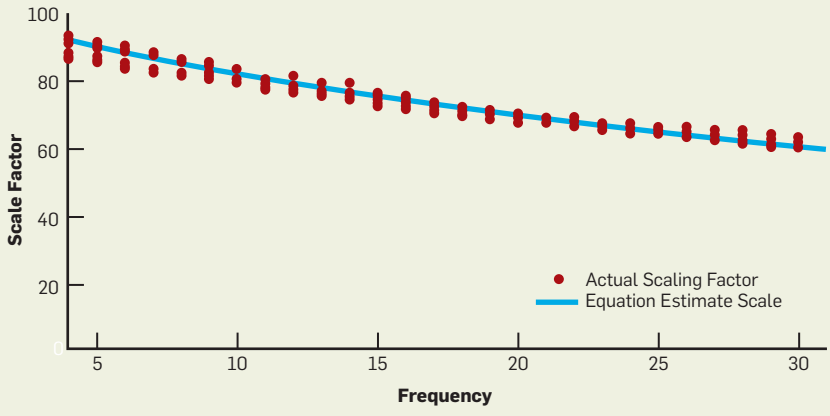
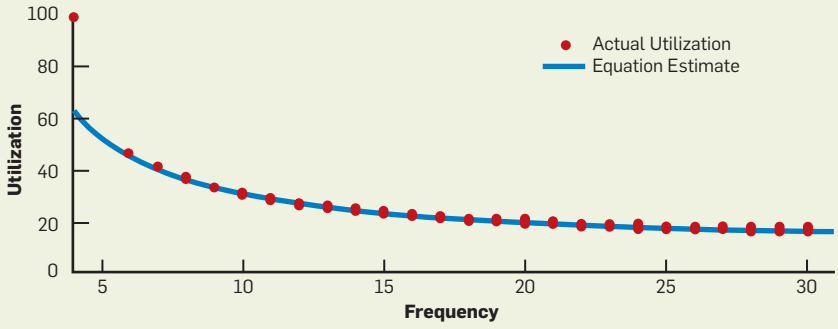


Figure 9. Utilization ridge point contour compared with equation.



Acknowledgments

The author is thankful for all the help from colleagues at Intel, specifically, valuable input from principal engineers Harinarayanan Seshadri and Rajeev Muralidhar and software engineer B. M. Shravan K.

Related articles on queue.acm.org

Power-Efficient Software
Eric Saxe
https://queue.acm.org/detail.cfm?id=1698225

Maximizing Power Efficiency with Asymmetric Multicore Systems
Alexandra Fedorova et al.
https://queue.acm.org/detail.cfm?id=1658422

Energy Management on Handheld Devices
Marc A. Viredaz et al.
https://queue.acm.org/detail.cfm?id=957768

References

1. Howse, B. Examining Intel's new Speed Shift tech on Skylake: more responsive processors. AnandTech, 2015; https://bit.ly/2Jf3Gxq.
2. Intel. Intel 64 and IA-32 Architectures, Software Developer's Manual, Volume 3B: System Programming Guide, Part 2; https://intel.ly/2sKeSqA
3. Kidd, T. Power management states: P-states, C-states, and Package C-states. Intel Developer Zone, 2014; https://intel.ly/2xM094b
4. Power Shaping and Stress Tool (PSST) for Intel Platforms. https://github.com/intel/psst

Noor Mubeen is a software architect at Intel client R&D group, focusing on power, thermal, and energy management breakthroughs. He has over 17 years' experience spanning a breadth of domains including networking, storage file systems, and embedded devices.