Compute performance metric for idle-bound services

(October 2020)

Noor Mubeen, *Sr. Architect,* Intel India; Nagabhushan Reddy, *Principal Engineer,* Intel India.

Abstract

For Power and Energy assessment of non-benchmark real-life usage scenarios, the performance-per-watt metric plays a critical role. Determining the numerator (perf) by employing synthetic benchmarks, introduces two concerns; (a) The perf score is specific to its own process/threads and not system wide (b) The synthetic Utilization levels are often latched to very high levels (~100%) throughout the run. The requirement to cover system-wide perf metric (all process and threads), may be addressed by sampling the Instructions Per (*busy*) Cycles – the IPC or its reciprocal (CPI). Here, *busy-cycles* is function of Utilization levels. The requirement to have less than 100 percent and varying utilization level, implies IPC needs to be normalized for Utilization; that is, Instruction per Second (IPS). However, even IPS metrics between different platforms may not readily be comparable due to the dependency on a given platform's base clock speed. Essentially this calls for normalizations against base clock speed. Similarly, with the reciprocal CPI metric, a breakdown into its *stack* components helps to assess the lost cycles (*stalls*) owing to L1/L2/TLB Instruction/Data cache misses, branch mispredictions etc. However, *stalls* are a function of run-time frequency scaling change of a Dynamic Voltage Frequency Scaling (DVFS) systems. Here we propose a simplified metric that accounts the frequency scaling impact on *stalls* and *Utilization* based on normalizations at each sampled time-window. The metric can be derived from generic Perf Monitoring Counters or PMCs available on most Architectures. Important applications are also discussed. This includes a generic means for perf-per-watt evaluation without the need for benchmark specific synthetic perf scores. This is aptly suited to low-power idle-bound scenarios as well, since the time-window normalization of the metric imply overhead free logger implementation using deferrable-wake timers. We also empirically validate the correctness of the metric on Intel® systems using with different well-known workloads.

Most performance benchmarks spawn high Utilization levels.

***Index Terms*— DVFS, Energy efficiency, performance metric, productive performance, connected modern standby.**

SECTION I. INTRODUCTION

Workloads including benchmarks involve some degree of dependencies across multiple subsystem's such as memory, network, graphics etc. While it is ultimately important to consider the overall system performance including such dependencies, those complexities often become deterrents if the goal is to identify a fundamental and unique performance metric of only the CPU as one compute DVFS block. In this regard, IPC has been a popular CPU metric whenever using external perf scores is not an option. While benchmarks that evaluate maximum possible performance (example SPEC [1]) have near 100% utilization even with multi-thread, there are other benchmark with goal of scenario specific perf evaluation, for example PCMark® for productivity applications. Even if the latter workloads are semi-active utilization levels, it still represents a score specific to the application threads. The work completed by background activity threads and services cannot be neglected as we assess idle-bound semi-active utilization cases. In such cases, the IPC or similar metrics are indispensable. One concern consuming raw IPC values for comparisons is the variability in utilization across different scenarios. To account these variations in Utilization one could use (IPC x Utilization) - some refer this as weighted-IPC [2]. Given that IPC is defined as Instructions completed per busy cycles and utilization as busy cycles per total reference cycles, the weighted-IPC is same as Instructions executed Per Second or IPS as shown in equation 2. Other approach [3] also employ a reference IPS_{ref} as fraction of maximum IPS_{max}, along with frequency and scaling-factor considerations. Apart from Utilization, the IPC/CPI metric is a function of frequency with respect to the extent of memory dependent stall cycles. This is clearly observable from Fig 1 plots of CPI and stall cycles per Instruction as a function of frequency. There are many ways to handle these inter-dependencies; for example, building and using online learning relation [4] based on memory-dependency and p-state while observing the CPI and related PMC counters is one approach. Clearly it is also important to further comprehend stall cycles, such as, cache misses and other Instruction Architecture inherent reasons. This is well represented as CPI stack related analysis [5] and can be reasonably measured with generic ([6], [7], [8]) Performance Management Counters or PMC.

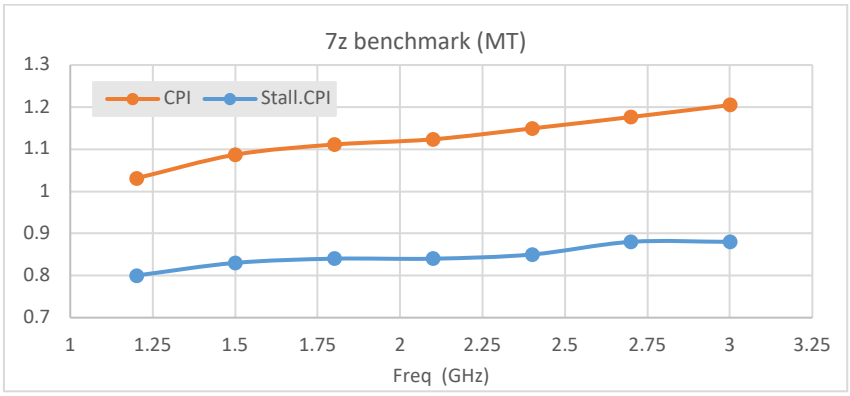

Fig. 1 Influence of stall on CPI. The workload used is a multi-threaded 7z compression as evaluated on a Core i5-3320M @2.6GHz processor running Linux.

SECTION II. BACKGROUND

Consider a workload sampled over time window (T) as shown in Figure 1. Let ΔTSC be the total counts elapsed from a reference clock running at ticks $1/F_{tsc}$. In the same window, let ΔM be the counter for busy cycles counted at the same rate (F_{tsc}) such that $T = \Delta M/F_{tsc}$.. The Utilization (l) in this window is defined as:

$$l = \Delta M/\Delta TSC \tag{1}$$

Additionally, if ΔA is the count of active Performance counter out of which only ΔP counts are productive cycles. So, the proposition of using IPS as (IPC * Utilization) is same as (Instructions retired/ΔA)* (ΔA/ΔTSC), i.e.,

$$IPS = IPC * l * F_{tsc} \tag{2}$$

F_{tsc} is fixed for given platform. In the subsequent time window, as the DVFS frequency f changes, it influences utilization l. The stall percentage (arising from cache missies or other sources inherent to the instruction) is also a function of frequency thus influencing IPC. Thus, in general:

$$IPS(f) = IPC(f) * l(f) * F_{tsc} \tag{3}$$

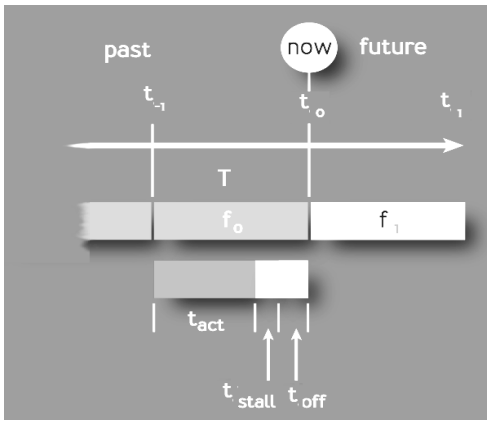

Fig. 2 Time-domain parameter representation

While IPS is an improvised metric compared to IPC, a performance metric needs additional considerations over equation (3). Firstly, it needs independence from F_{tsc} since platforms could differ in their TSC clock rates and any perf comparisons would not be normalized for TSC. Secondly, it is important to be able to estimate the perf change in subsequent time window (t_1) for proposed frequency (f_1) changes by DVFS governor.

On a DVFS system, it is possible to treat frequency change causalities as a ripple back into local Utilization (l) and scaling-factor (s) as related to stall percentage. The scaling factor can be defined as $s = \Delta P/\Delta A$. Considering a DVFS system with an initial scale-factor s_0 at time t_0, the estimated scaling of load ($l_0 \Rightarrow l_1$) as an artefact of frequency-causality from f_0 to f_1 can be derived and proved [9] as:

$$l_1 = l_0 \left\{ s_0 . \frac{f_0}{f_1} + (1 - s_0) \right\} \tag{4}$$

Similarly, the estimated scaling of scale-factor (s_0 => s_1) as an artefact of frequency causality from f_0 to f_1 can be derived [9] as:

$$s_1 = 1/\left\{1 + \frac{f_1}{f_0}\left(\frac{1}{s_0} - 1\right)\right\} \tag{5}$$

SECTION III. NORMALIZED PERF - Q

Starting with the Time-domain parameters in Figure 1, we derive an equivalent perf representation that address the above challenges. The frequency governance has run this window at actual frequency f_a . Among ΔA - a count of active Performance counter out, only ΔP counts are productive cycles where instructions are run to completion. That is, (ΔA - ΔP) is the count corresponding to time spent with *stall*s. Note that both counter tick at variable frequency f_a in that window. Note that ΔM count correspond to a value over same time that it took doing ΔA, but counted on the fixed reference clock F_{tsc}. In cumulative time-domain terms, these relations can be summarized using Figure 1 as follows:

$$\begin{aligned} t_a &= \frac{1}{f}\,\Delta P \\ t_a + t_s &= \frac{1}{f}\,\Delta A \\ t_a + t_s + t_{off} &= \frac{1}{F_{tsc}}\Delta\mathrm{TSC} = T \end{aligned} \tag{6}$$

Utilization or load (l) is:

$$l = \frac{(t_a + t_s)}{(t_a + t_s + t_{off})} \tag{7}$$

From above equations we have:

$$\Delta\mathrm{A} = f.l.\Delta\mathrm{TSC}/F_{tsc} \tag{8}$$

Scaling-factor s is defined as, $s = \frac{\Delta P}{\Delta\mathrm{A}}$

ΔP is the only productive output (count) in this window of time for this DVFS component. From the above equations, we get:

$$\Delta P_t = s_t\, f_t\, l_t\, T_t \tag{9}$$

At this juncture, it is important to establish that the theory in equation (9), is also empirically true using real platform data. On an Intel® Core™ i7-7560U Processor based platform the values of s, l, f and related ΔP perf counters [8] using PMC are captured for general idle-bound scenario. The samples are logged every fixed time window (few ms). Each independent point representing pair values of ΔP and ($s.l.f$) is plotted in Figure 5. The cumulative result fits a constant slope trendline demonstrating the proportionality in equation (9).

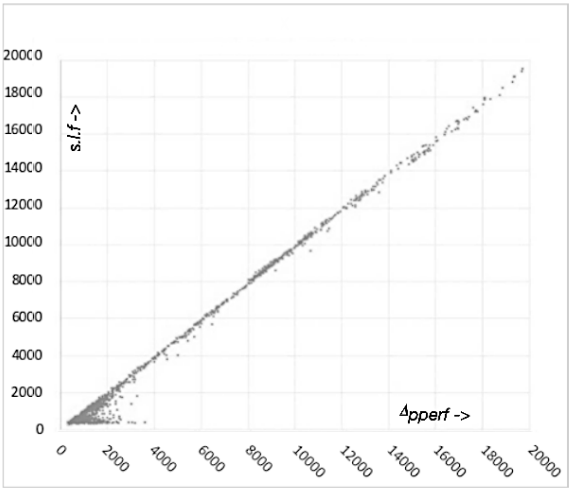

Fig. 3. Perf counter and Time association

The equation (9), does not fully separate the compute performance with the two attributes: utilization scaling as well as the duration of sampled time window. To normalize into a perf metric in every sample we define it as 'ΔP_t *per-unit-load per-unit-time*' or Q_t in every window *t*.

$$Q_t = \frac{\Delta P_t}{l_t T_t} = S_t f_t \tag{10}$$

For the entire duration spanning *n* samples, cumulative CPU perf 'Q' is given by the linear sum: $Q = \sum_{t=1}^{n} Q_t$

That is,

$$Q = \sum_{t=1}^{n} S_t f_t \tag{11}$$

Or alternately,

$$Q = \sum_{t=1}^{n} \frac{\Delta P_t}{l_t T_t} \tag{12}$$

The unit of Q would thus be 'productive performance reference count per unit time per unit load'. This represents an average unique performance for entire duration of the scenario spanning *n* samples. Practically, it may be convenient to report a value of Q in 'per milli-second'.

SECTION IV. VALIDATING THE Q PERF METRIC

The above equation (12), being a function of l_t, signifies a key difference between tight-loop benchmarks and idle-bound activity. For tight-loop benchmarks l_t is almost always close to 100% in every sample *t*. Further, f_t may often be bumped up to a high value (turbo frequency) to meet the demand as represented by the high utilization. This explains why tight-benchmarks demonstrate a performance that monotonically as well as proportional to scales with max-frequency of a platform. However, for low-power CPU workloads l_t gets scaled differently in *every* window *t* based on f_t. This is coupled with a similar scaling effect is applicable to scaling-factor s_t. Thus, it is important to validate the Q metric on both ends of utilization spectrum.

Specifically, certain native score and Q perf scores are evaluated on following workloads on Intel® Core™ i7-7560U platform running Windows® 10 OS release:

1. High-utilization turbo-bound - SPEC-2006-perlbench and Cinebench®. Native benchmark score.
2. Low-utilization idle-bound - Video play back (VPB). Native score: compute score based on induced frame dropped.

In the case of SPEC-2006 perlbench, we employ different power limit configurations along x-axis. Both, spec perlbench score and unique Q perf score, are captured simultaneously in each run. As shown in Figure 7. the unique Q metric follows the spec score, indicating again, that the latter is as reliable as the former metric for high-load (tight-loop) benchmarks case.

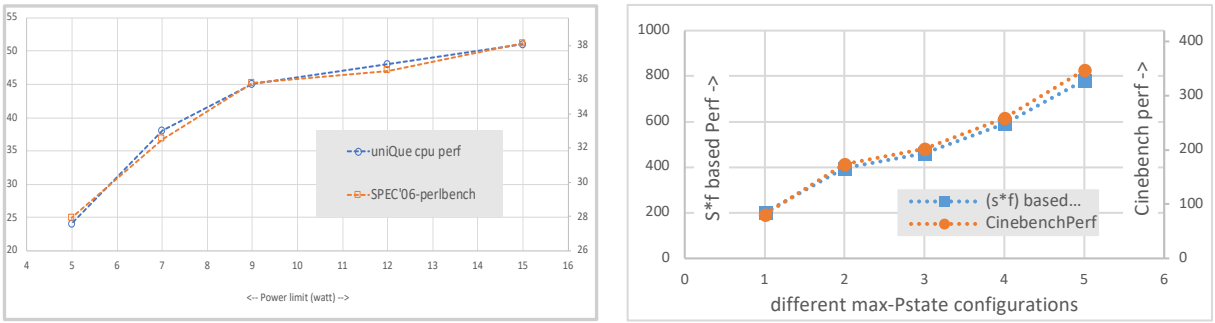

Fig. 4. SPEC2006 vs Unique perf Q

As shown in Figure 6, the x-axis (MHz) for VPB case is as configured to run at different max-frequencies. Both, the Q perf score and playback frame rate are captured simultaneously in each run. Note that frame drop is intentionally induced in the scenario to start with very low playback rate (about 20% playback with 80% frame drop at 500MHz).

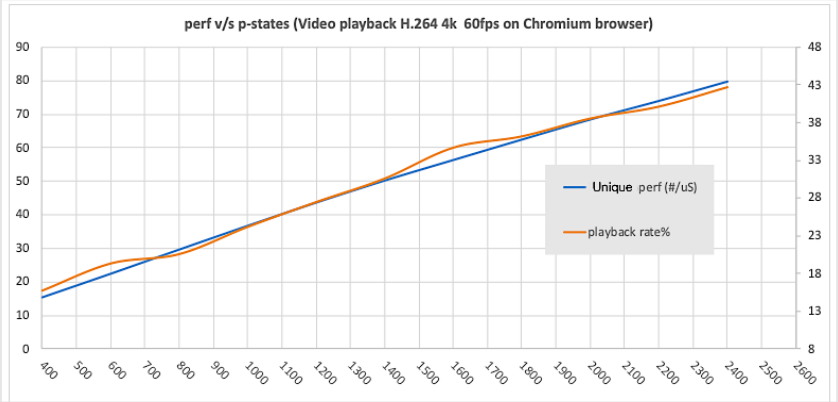

Fig. 5. Video Playback frame-rate compared with Unique perf Q

We derive two observations regarding Q perf metric from results in Figure 5:

- It scales monotonic with increasing max-frequencies despite utilization (CPU C0-state) being < 10%.
- It linearly follows the frame rate metric, indicating that the former is just as reliable as the latter metric in this case.

SECTION V. APPLICATIONS OF PROPOSED PERF METRIC

Few of the immediate significances and applicability of Unique perf metric Q [equation (11) and (12)] could be as follows:

a) QoS metric for background services during Idle and Connected Standby scenarios

Unlike servers, PC and mobile devices have significant duration of idle-bound scenarios such as Backgrounded services. In fact, OS vendors are increasingly focused optimizing the energy cost of background work [10]. The energy reduction goal is closely associated with performance. Consider a single fixed background task, triggered by an interrupt. The faster it is run-to-completion, the longer idle duration is lent to the system. However too high a performance would lead to power overhead. Hence there is always an *optimal* performance-per-watt that the DVFS governors can seek. In the present example of fixed background task, the performance can be considered reciprocal of time-taken to complete. Thus, energy reduction goal is same as perf-per-watt maximization goal.

$$\frac{task\_perf}{watt} = \frac{\left({}^{1}/_{task\_time}\right)}{watt} = \frac{1}{task\_Energy} \tag{13}$$

In the proposed metric, the reference to "unique" is the ability to measure consistent metric independent of task or scenario. The normalization in equation (12) generalizes it for all the background tasks on CPU per unit-time. This becomes increasingly relevant for baseline-power in cases of frequently interrupted idle cases such as *wake-on-Voice* or *wake-on-network* etc. during connected standby where battery-life considerations are important.

Further, by using known techniques of power models, a relative power for DVFS component can also be determined. The unique perf count Q is total count across all the CPU sharing voltage domain. The perf-per-watt (PPW) has units of $Watt^{-1}$ $second^{-1}$ or $Joule^{-1}$. Even though software-based measurements on power are likely to be less accurate (compared to analogue sense-resistor based rail power measurements), it is important to note that the application here is one of relative comparisons. Hence, absolute value accuracy is less significant.

$$PPW_{ref} = \frac{Q_{total\_cpu}}{Power_{cpu}} \tag{14}$$

Further, it is possible to define a relative runtime CPU efficiency. If PPW_{max} is the maximum recorded PPW on a system, and the present runtime is PPW_{ref}, then the runtime efficiency (η) can be determined.

$$\eta_{ref} = \frac{PPW_{ref}}{PPW_{max}} \tag{15}$$

b) Proactive DVFS governance with performance lookahead

As far as utilization and stall scaling is considered, most prominent p-state governors on Linux and windows systems reactively select next p-state based on past time windows. The proposed scheme allows for *apriori* quantification of scaling impact to performance, arising from utilization and scaling-factor before dispensing next p-state. One immediate implication of reactive schemes is limiting of frequency p-states based on power limit breach due to excessive performance disposition. This is the case for both simple power-capping technique as well as other advanced techniques [11]. On the other hand, conservative performance disposition is also obviously inefficient. Frequency governors could potentially take cognisance of

both perf and power impact in every cycle before an actual frequency change decision. While it may eventually be unavoidable to limit p-states, the proactive scheme could be potentially avoid, or at least defer the power-limit breach itself. In that context, using stall factor scaling equation (5) allows estimation of perf aspect for the next cycle. Thus, if Q_0 was the perf is present cycle with frequency f_0, the estimated perf in proposed next p-sate (f_1) would be:

$$Q_1 = Q_0 / \left\{ s_0 . \frac{f_0}{f_1} + (1 - s_0) \right\} \tag{16}$$

SECTION VI. LIMITATION

The unique perf metric demonstrated here has been based on CPU as compute resource. Similar counters can be created and tracked for other DVFS blocks. However, such PMC metrics may not be readily available on all DVFS blocks or significant Hardware/firmware support may be required to enable the same. The Q metric is not readily intended to track or substitute perf of complex scenarios involving multiple blocks, for example, a gaming score that depends on CPU, GPU and Memory. In fact, without a fully evolved DVFS mesh orchestrations, partial application of the approach may regress any benchmarking scores which may already be achieved by classic schemes built largely guided by such scores. This *limitation* has to be fully comprehended before applying or comparing against other schemes which have larger application specific scope.

SECTION VII. CONCLUSION

Using generic PMC counters, we derived a method to normalize the productive performance metric against memory-stall scaling and Utilization scaling with frequency change. These are not factored in raw IPC. Additionally, we avoid platform specific dependency of TSC value unlike IPS. Further normalization with sample time window duration, lets us to have practical implementation (deferrable wake timers) that do not have to poll at fixed period T. This is critical for the proposed application during idle and modern connected standby, where wake sources influence the baseline power. Using actual platform data, we validate the results that the unique perf metric Q extended to both high-utilization scenarios as well as low-CPU usages We presented the imminent application on perf-per-watt for energy efficiency assessment goal. Similarly, the ability to derive runtime efficiency relative to a max value on same platform was defined. A means to *estimate* the Q value in proposed next-frequency windows was also presented. Thus, unique perf Q helps to generically expose and improve localized DVFS efficiencies. As future study, the approach holds potential to apply on a composite scenario that involve assessment of multiple interacting components among the DVFS mesh.

BIBLIOGRAPHY

[1] "Standard Performance Evaluation Corporation," 2006. [Online]. Available: https://www.spec.org/benchmarks.html#cpu.

[2] G. Dhiman, G. Marchetti and T. Rosing, "vGreen: A System for Energy Efficient Computing in Virtualized Environments," in *ISLPED '09: Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, 2009.

[3] R. Ayob, U. Ogras, E. Gorbator, Y. Jin, T. Kam, T. Rosing and P. Diefenbaugh, "OS-level Power Minimization Under Tight Performance Constraints in General Purpose Systems," in *IEEE/ACM International Symposium on Low Power Electronics and Design*, Fukuoka, Japan, 2011.

[4] G. Dhiman and T. S. Rosing, "Dynamic Voltage Frequecy Scaling for Multi-tasking Systems Using Online Learning," in *ISLPED '09: Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, 2009.

[5] S. Eyerman, L. Eeckhout, T. Karkhanis and J. E. Smith, "A performance counter architecture for computing accurate CPI components," *ACM,* vol. 41, 2006.

[6] AMD, "Processor Programming Reference (PPR) for AMD Family.," [Online]. Available: https://www.amd.com/en/support/tech-docs.

[7] ARM, "Cortex-A9 Technical Reference Manual.," [Online].

[8] "Intel® 64 and IA-32 Architectures Software Developer's Manual," vol. 3B: System Programming Guide, pp. Part 2. Chapter 18, 19..

[9] N. Mubeen, "Workload Frequency Scaling Law - Derivation and Verification," *Queue,* April 2008.

[10] Microsoft, "Optimize background activity," 2017. [Online]. Available: https://docs.microsoft.com/en-us/windows/uwp/debug-test-perf/optimize-background-activity.

[11] S. Reda, R. Cochran and A. K. Coskun, "Adaptive Power Capping for Servers with Multithreaded Workloads".