# Performance-efficient CPU resource management algorithm on Heterogeneous multi-processor

Jonglae Park, Bumgyu Park, Youngtae Lee, Chulmin Jo, and Seogjun Lee
S.LSI, Samsung Electronics, Hwaseong-si, Republic of Korea
Email: {jonglae.park, bumgyu.park, yt0729.lee, cm.jo, seogjun.lee}@samsung.com

***Abstract*-- The biggest difference of mobile devices over AC powered such as servers and desktops is that their power budget is very limited. In other words, the CPU resource management solution for mobile devices makes a greater contribution to product quality. Their main algorithm was to choose the minimum energy or the best-fit performance to accommodate a given job. However, in real world, performance and energy are essential to each other and cannot be treated in these independent and consecutive referencing methods. In this document, we would like to give you new ideas for how to consider performance efficiency in CPU resource management solution.**

## I. Introduction

OS (*Operating System*) Scheduler, CPU DVFS (*Dynamic Voltage and Frequency Scaling*), and CPU Idle are commonly used to operate CPU resources efficiently. The underlying theory of these three solutions is to assign a given job to the appropriate HW resource, or to provide the appropriate HW resource for that job. State-of-the-art scheduling and CPU DVFS technology for mobile devices that best meets the needs of the times is EAS (*Energy Aware Scheduling*) [2], [3]. Its main algorithm is to choose the best core or frequency that uses the least amount of energy to process the amount of job given. CPU Idle also has a same purpose in terms of choosing the idle states with minimum energy for a given idle time calculated by the scheduler. Although all of the above methods are expected to reduce the power consumption without performance degradation, in the real world, this method behaves differently than expected due to the characteristics of silicon, cache utilization, discrete range of frequency, and intentional frequency boosting.

The following chapters explain in detail how the above factors make the system behave differently than expected and how they have been overcome.

## II. Considering the Performance Efficiency

### *A. PASE (Performance Aware Scheduling within a given Energy budget)*

This paper is based on big.LITTLE HMP architecture (Exynos9820, CortexA55 × 4 for Little core, CortexA73 × 2 for middle, and M4 × 2 for big) [1], [4], [8] of three processor sets with different performance efficiency. To support various topologies as a common solution, EAS uses an algorithm to "select the core with the minimum energy that meets the performance requirements of a given job".

In Fig.1, big core with 'min-power @ BIG' will be chosen with EAS algorithm for 'Demanding Perf (*Performance*)'. In reality, however, such a decision is not reasonable. This is because the DVFS frequency level is not continuous, or the performance of the target core has been already determined externally.

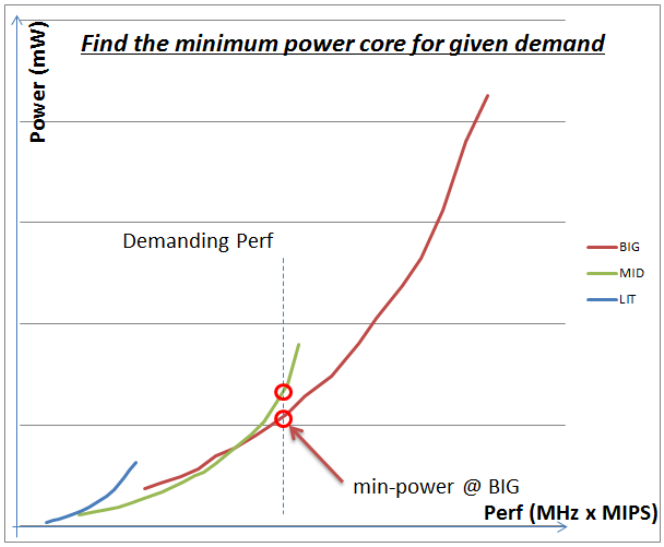


Fig. 1. For given demanding performance, big core with minimum power will be selected instead of MID core.

As shown in Fig.2, the big core using minimum power (C. Est. <Perf, Power> of BIG @ EAS) for required performance (blue solid vertical line, "Required Perf"), was selected in the conventional method of EAS. Due to the discrete DVFS level, the actual performance of the big core is different from the estimation (C). So the actual performance and power for required performance is to be (B). That is, scheduler must choose the best among (A) and (B), not (A) and (C).

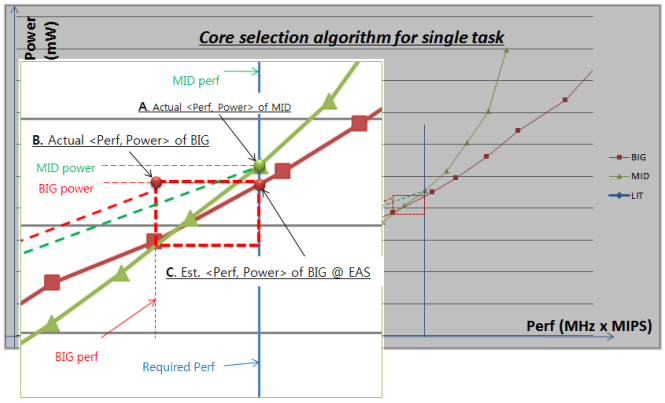


Fig. 2. The actual power of Big cores (B) is the same as estimated (C), but their actual performance (B) is overestimated. EAS choose the best among (A) and (C).



978-1-7281-5186-1/20/$31.00 ©2020 IEEE

In order to compare the superiority between (A) and (B), we will use the performance per energy. Considering 'the performance return on energy investment', the middle core should be chosen (A. Actual <Perf, Power> of MID).

Furthermore, this kind of wrong decision would happen more often and more fatally when the performance of candidate cores has already been determined due to various external factors (such as intentional frequency boosting, or load unbalance situation). Let's suppose that the performances of big cores and middle cores are determined externally as shown in Fig. 3. And both the big core and the middle core are capable of "Demanding Perf", however, the power of the big and the middle core are the same. In this situation, the big core must be chosen for "Demanding Perf", but middle core is chosen by the conventional EAS. EAS does not consider these exceptions.

So we propose that best target core must be decided not by the minimum energy, but by the 'performance per energy' for given job.

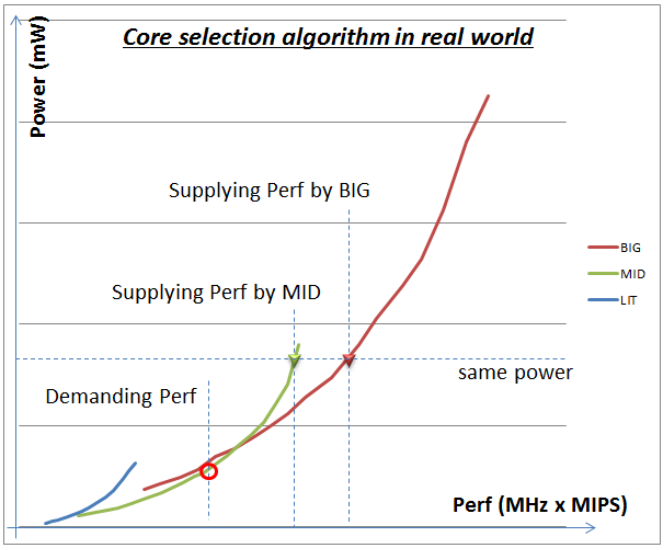


Fig. 3. Big and Mid core have same power, but Big core has superior performance.

The proposed core selection algorithm for given job '*j*' satisfies the following equation (1), and will choose the '*target*' core.

$$\frac{perf(target, j)}{energy(target, j)} = MAX(\frac{perf(cpu, j)}{energy(cpu, j)}), \quad (1)$$

where $\forall cpu \in$ available cores accommodating given 'j'

, energy (cpu, j) is consumed energy for 'j' on cpu

, perf (cpu, j) is estimated latency for 'j' on cpu

Previously, the '*target*' is the core in which only the energy is minimum for given '*j*'.

With using the maximum 'real performance per energy' method, instead of a minimum energy, we achieved overall 12% performance improvement especially in UX (*User eXperience*) benchmark, but the DoU (*Day of Use*, battery life time) dropped only 1.3%.

### *B. Power step-wise frequency scaling (PSF)*

In this chapter, we introduce 'power step', a new method of frequency selection in DVFS. The key concept of PSF is to consider the energy cost when choosing the next OPP (*Operating Performance Point*, same as frequency). In the point of considering energy cost, H-EARtH [9] and energy aware Schedutil [10] have similar intentions in common, but PSF differs in that it refers to the energy cost when predicting the appropriate OPP in the future. As shown in Fig. 4, the existing DVFS algorithm selects the next OPP proportional to the current OPP and utilization on it. But the PSF uses the energy cost as an input parameter in addition.

Formally speaking, the problem in DVFS is that there are initial OPP given and job to be done, but invisible for its quantity. And the goal of problem is to find the optimal path of series of OPP selection minimizing the energy consumption and latency. PSF is explained as following equation (2).

$$f'_{n+1} = PowerToFreq((FreqToPower(f'_n) + \Delta p) \times U_n), \quad (2)$$

where $f_n$ , $f_{n+1}$ is current and next OPP respectively

, PowerToFreq(P) returns OPP for power P (mW)

, FreqToPower(F) returns power (mW) for OPP F

, $U_n$ is the utilization of CPU so far

, $\Delta p$ : CPU power budget / N.

Since most of job is not measurable of its size except deadline one, 'energy cost' is substituted for 'power' in (2).

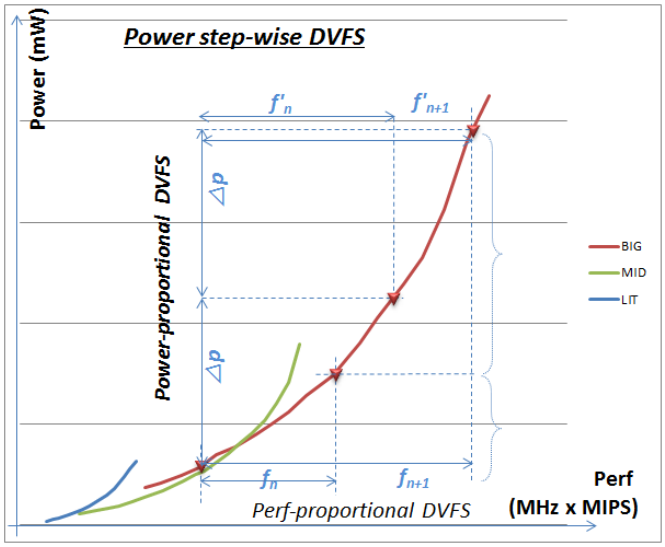


Fig. 4. Concept of power-proportional DVFS (PSF) and performance-proportional DVFS.

As shown in Fig.4, PSF will ramp up the OPP quickly in lower performance range ($f'_n > f_n$), but slowly in higher OPP ($f'_{n+1} < f_{n+1}$). In the previous DVFS solution, these behaviors

are done by the pre-defined frequency-variant tuning parameters through the lots of experiment. These behaviors can increase the consumer satisfaction with both responsiveness and power consumption of the product. But PSF can provide with the ideal ramp-up shape of OPP selection in DVFS by considering the energy cost without heuristic tuning parameter. In Fig.5, each plotting graph shows the shape of OPP selection for each $N$ value of (2). The smaller the $N$ is, the faster the OPP goes up or down.

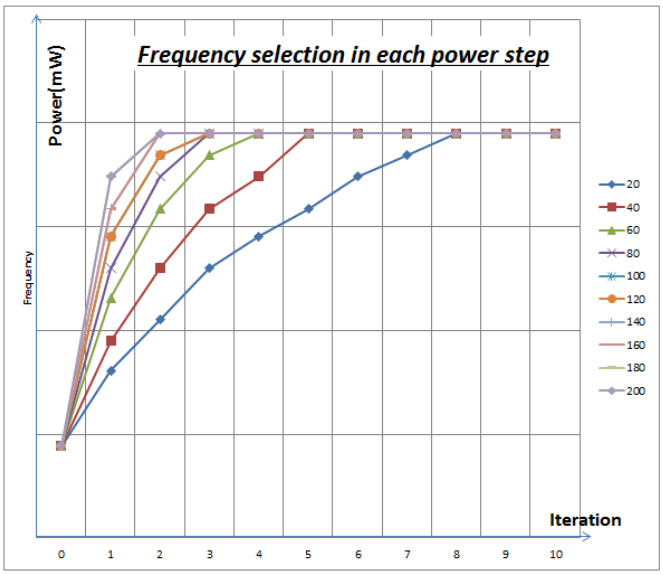


Fig. 5. PSF derives the next frequency based on power budget as well as the required performance - utilization.

We compared the 'performance return on consumed energy' for various DVFS algorithm including PSF. Previously, $f_{n+1}$ is calculated by multiplying the product of the utilization and the fn by a constant margin. 'x1.25' is the previous DVFS with 25% margin. 'PSF-/8' means that the N value in (2) is 8. 'max' is a DVFS algorithm that provides a max OPP when utilization is above a certain threshold. Finally, PSF-BS means that power step is proportional to 'power of max OPP – power of current OPP'.

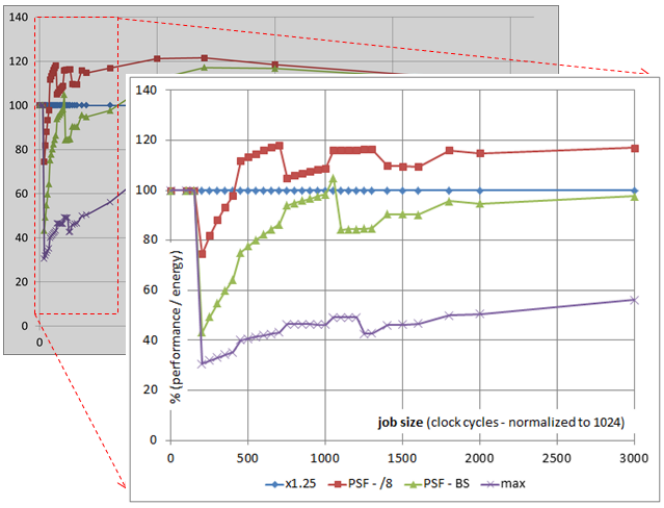


Fig. 6. Performance/energy ratio of each DVFS algorithm versus the performance efficiency of x1.25.

In Fig.6 and Table I, it can be said that the PSF algorithm is generally superior to other algorithms in the view of performance efficiency.

TABLE I
PERFORMANCE / ENERGY FOR EACH DVFS

| | Constant margin | | | PSF | | | | max |
|---|---|---|---|---|---|---|---|---|
| | x1.5 | x2.5 | x3.5 | N=4 | N=8 | N=16 | BS | max |
| P/E (%) | 100 | 91.25 | 94.52 | 100.33 | 108.71 | 110.87 | 88.24 | 53.11 |

The performance is a reciprocal of total execution time. And the energy is the accumulation of product of power and time for each OPP.

In recent study, Linaro (Collaborative engineering organization consolidating and optimizing open source software and tools for ARM) introduced the new DVFS algorithm called EA-SU (*Energy Aware SchedUtil*) mentioned above. This algorithm uses the variable margin of OPP selection error converted from the energy differentials through the series of workload pattern. Using this algorithm, OPP reaches the target frequency very quickly, and reduces the redundant margin when repeated workload pattern. This algorithm can be said to be the smart way to consider energy costs, but performance efficiency is not good as shown in the Table II, if the simulation is based on the real silicon power data.

With this method, when measuring the score on the PCMark benchmark, the overall score is increased by 15% (about 26% for some sub tests, such as photo editing). At the same time the power consumption is just increased by only 6%. In the typical user scenario, the power consumption is just increased by 4%.

TABLE II
PERFORMANCE / ENERGY OF EA-SU

| P/E (%) | Random load – PCMark | Load Rush – GeekBench | Load Rush – Dhrystone |
|---|---|---|---|
| x1.25 | 100 | 100 | 100 |
| EA-SU | 94.23 | 100.15 | 100.46 |
| PSF | 126.46 | 103.66 | 100.60 |

Refer the Fig.7, 8 and 9 for each load pattern.

### C. *Performance aware idle depth selection (PAI)*

The CPU idle framework predicts a state that consumes the least energy among the available idle states for the estimated idle time [6]. In recent technology, the private cache can be turned off in the C2 state or deeper state [7]. In this case, CPU idle framework is overlooking something related with follow-up performance after idle, by considering only minimum energy. If most of instruction and data is hit in cache, and the estimated idle time is long, cache will be flushed to the memory by idle framework. In this case, if the flushed instruction and data are to be reused, performance degradation may become severe after idle exit. Another thing that is overlooked with cache flush is that performance impact due to cache miss has a very important relation with memory clock in addition to CPU clock. But, the Linux idle framework does not consider the target idle state according to the any clock [6]. So we propose a new idle state prediction method based on cache

miss rate and relate frequency. That is, not only the energy consumed, but also the performance impact will be considered for selecting the idle state. The following equation (3) describes the PAI algorithm.

*Let $C_n$'s target residency and transition time, $R_n$, $T_n$*
*For given estimated idle time I,*
*Find the min energy idle state, $C_t$ in the existing method*

$$C_t = \min(Energy(R_k, T_k, I)),$$

*Where k = 1, .., n (available idle state $C_n$)*

*In addition to the above equation,*
*Let $D_n$ for performance impact by cache dirtiness:*

$$D_n = {}^{1}\!/_{cache\_dirty\_rate},$$

*$F_n$ for performance impact by frequencies, such as CPU, MIF (Memory I/F), ....:*

$$F_n = \tau/(w_{cpu} \cdot F_{CPU} + w_{MIF} \cdot F_{MIF} + \ldots),$$

*Where $\tau$ = constant for performance impact of frequency*

*Estimated Negative Performance Impact:*

$$PD_n = \gamma \cdot F_n \cdot D_n,$$

*Where $\gamma$ = constant for performance impact*

*Find the min energy and min performance impact in proposed method:*

$$C_t = \min(Energy(R_k, T_k, I) \cdot PD_k), \quad (3)$$

*Where k = 1... n (available idle state $C_n$)*

To see the quick proof of concept, we implemented the frequency variant criteria for target residency and exit latency and applied some frequency-variant parameters. And we were able to increase the original 47 frames to 51 frames on the commercial benchmark called Manhattan off-screen. Considering multiple frequencies or referring to cache usage is currently under review and we are in the process of identifying the practical feasibility of the proposed method.

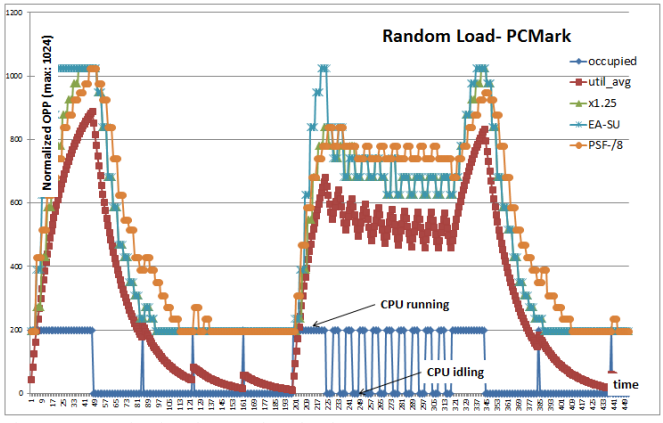

Fig. 7. OPP Selection for ramdom load.

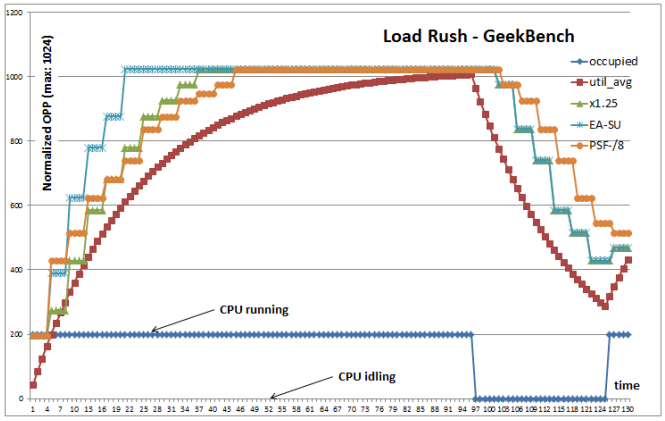

Fig. 8. OPP selection for load rush of GeekBench.

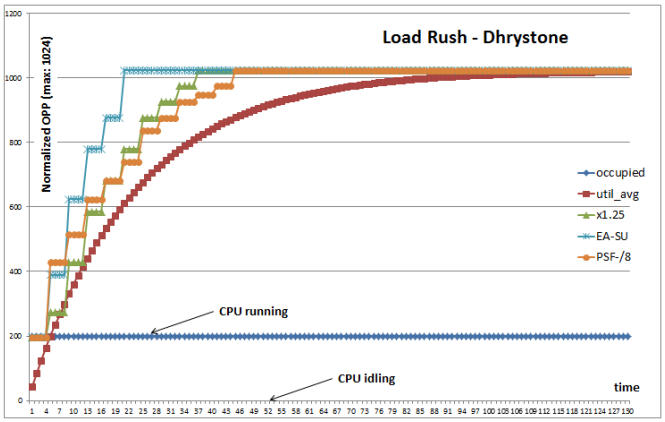

Fig. 9. OPP selection for load rush of Dhrystone.

## III. Conclusion

### A. Experiments

The following table shows the measured data for the performance and power consumption of the proposed algorithms. The values obtained in these experiments are the percentage of improvement compared to the scores of the shipped product. The percentage of improvement mentioned in each chapter is the value when the each algorithm is applied alone in the controlled environment.

TABLE III
PERFORMANCE AND POWER IMPROVEMENT

| | CPU/System Performance | | | UX Performance | | | |
|---|---|---|---|---|---|---|---|
| % | Antutu v7 | GeekBench Single – v4 | GeekBench Multi – v4 | PCMark 2.0 | BBench | App Launch | DoU |
| PASE | 0.25 | -2.13 | -0.92 | 11.00 | 37.50 | -4.34 | -1.30 |
| PSF | | 0.20 | 0.50 | 5.30 | | 7.00 | 1.04 |

The performance is a reciprocal of total execution time. And the energy is the accumulation of product of power and time for each OPP.

### B. Summary and future work

As mentioned in abstract, the energy given and the performance that can be achieved by using it are closely

related. Thus, independent and consecutive referencing methods have verified that there are various errors. This is a problem that can only be confirmed in the real world that cannot be found in the controlled environment. And these are should be considered in the development process for mass product.

Unfortunately, the PAI has no additional benefit beyond the above-mentioned experiments in the above experiment. Currently, we are working on additional implementation and experiments for PAI optimization, and if new HW is available to obtain the cache utilization information, such as AMU (*Activity Monitor Unit*) [11], we will also implement the ultimate implementation of PAI idea.

## REFERENCES


[1] Geenhalgh , P., "Big.LITTLE Processing with ARM® Cortex™-A15 & Cortex-A7," ARM® White paper, Sep. 2011.

[2] Rickards, I. (ARM®) and Kucheria, A. (Linaro), "Energy Aware Scheduling (EAS) progress update," Linaro article, available at: https://www.linaro.org/blog/energy-aware-scheduling-eas-progress-update/, Sep. 2015.

[3] Wysoski, R.J., "CPUFreq and The Scheduler, Revolution in CPU Power management," in *LinuxCon + ContainerCon North America 2016*, available at: https://events.static.linuxfound.org/sites/events/files/slides/cpufreq_and_scheduler_0.pdf, Aug. 2016.

[4] "Exynos 9820 Processors: Specs, Features | Samsung Exynos," available at: https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-9820/

[5] Chan, M., "cpufreq: interactive: New 'interactive' governor," LWN article, available at: https://lwn.net/Articles/662209/, Oct. 2015.

[6] Ven , A.V.D., "Some improvements to the cpuidle menu governor," LWN article, available at: https://lwn.net/Articles/386990/, May. 2010.

[7] Hansson, U., "Cluster idle - Now and next," in *OSPM Summit 2019*, available at: http://retis.sssup.it/ospm-summit/, May. 2019

[8] Craeynest , K.V., Jaleel , A., Eeckhout, L., Narvaez , P., Emer , J., "Scheduling heterogeneous multi-cores through performance impact estimation (PIE)," in *39th Annual International Symposium on Computer Architecture*, Jun. 2012.

[9] Rotem, E., Weiser, U.C., Mendelson, A., Ginosar, R., Weissmann, E., Aizik, Y., "H-EARtH: Heterogeneous Multicore Platform Energy Management," in vol 49, Issue 10, *IEEE Computer, Oct. 2016*.

[10] Raillard, D., "sched/cpufreq: Make schedutil energy aware," LWN article, available at: https://lwn.net/Articles/792252/, (Jun. 2019).

[11] "ARM® CortexTM-A75 Core Revision: r2p0," in chapter C3.3 Technical Reference Manual.

[12] Corbet, J., "Per-entity load tracking," LWN article, available at: https://lwn.net/Articles/531853/

[13] Mulukutla, V., "sched: Introduce Window Assisted Load Tracking," LWN article, available at: https://lwn.net/Articles/704903/

[14] Bellasi, P., "SchedTune: central, scheduler-driven, power-performance control," LWN article, available at: https://lwn.net/Articles/704859/

[15] "Power management," Google documents, available at: https://source.android.com/devices/tech/power/performance