Power consumption aware big.LITTLE scheduler for Linux operating system

Zsolt Bringye
John von Neumann Faculty of Informatics
Obudai Egyetem
Budapest, Hungary
bringye.zsolt@nik.uni-obuda.hu

Dezső Sima
John von Neumann Faculty of Informatics
Obudai Egyetem
Budapest, Hungary
sima@uni-obuda.hu

Miklós Kozlovszky
BioTech Research Center, EKIK
Obudai Egyetem
Budapest, Hungary
kozlovszky.miklos@nik.uni-obuda.hu

***Abstract*— The energy consumption of computer systems is generally an important design aspect. There are several well-known solutions for reducing the power consumption of processors, among others ARM big.LITTLE architecture. In Linux systems, CPUFreq and CPUIdle governors are traditionally responsible for managing CPU consumption and performance. Our primary goal has been to develop a user-mode CPUFreq governor alternative that provides a suitable framework for the development of governors utilizing the features of big.LITTLE architectures.**

Keywords*— *Linux, big.LITTLE, energy efficiency

I. INTRODUCTION

The energy consumption of computer systems is generally an important design aspect. When computer systems operate decoupled from the mains, the amount of energy used directly determines the operating time of the system. However, energy consumption is not a negligible issue even in case of mains-operated systems due to cost and environment reasons.

There are several well-known solutions for reducing the power consumption of processors, and with the development of the ARM big.LITTLE architecture the designers are given yet another tool. The Odroid-XU3 single-chip computer is an appropriate tool for experimenting, it is based on a Samsung Exynos 5422 big.LITTLE processor and is equipped with sensors to measure dynamic power consumption per channel [1].

In Linux systems, CPUFreq and CPUIdle governors are traditionally responsible for managing CPU consumption and performance. Our primary goal has been to create a user-mode CPUFreq governor alternative that provides a suitable framework for the development of governors that utilize the features of big.LITTLE architectures. Our purpose has been also to develop a testing framework for different load patterns while gaining as much data from the system as possible, and then to evaluate the test results.

Our paper is organized as follows: In the introduction section we provide a state-of-the-art overview and explain our motivations. In Section II. we discuss the performance characteristics of recent CPUs. Section III. Details our proposed solution for a big.LITTLE schedulers. Section IV. describes our testing environment and shows first results. Finally, we summarize our conclusions.

II. PERFORMANCE OF COMPUTER SYSTEMS

A. Elements of Power Consuption

The total power consumption of the system is the sum of the static and dynamic power consumption:

$$P_{total} = P_{static} + P_{dynamic} \quad (1)$$

For CMOS circuits, the use of dynamic energy is an essential part of energy consumption, which is associated with switching transistors [4] as expression (2) shows, where C is the parasitic capacity of the transistors:

$$E_{dynamic} \propto \frac{1}{2} \times C \times U^2 \quad (2)$$

Performance depends on the number of switches per unit of time, which is a function of the system clock frequency:

$$P_{dynamic} \propto \frac{1}{2} \times C \times U^2 \times f \quad (3)$$

While dynamic performance is associated with switching transistors, static power consumption occurs independently of system activity, as a result of leakage currents on semiconductors. Static power consumption is a function of supply voltage, the manufacturing technology used and ambient temperature

Dynamic power consumption is the power used during active device operation. it is not affected by the ambient temperature.

B. Common Optimization Options

The following optimization options are mentioned in the literature [4]:

- Clock gating: Recent processors suspend clocking of the inactive modules (e.g., the floating point unit), thereby saving dynamic power. This solution however, does not eliminate static power consumption. Clock gating was introduced in the second half on the 90's (e.g. in the FP module of DEC Alpha 21264 in 1996) and since then it became ubiquitous.
- Power gating: With "power gating" , the affected circuit unit (full unit, e.g., an idle core) is disconnected from

978-1-7281-0968-8/19/$31.00 ©2019 IEEE

the power, so both its dynamic and static consumption are eliminated. In the case of power gating, the status of the unit is lost, i.e. it has to be saved previously. The usage of power gating begun around 2010..

- Dynamic Voltage Frequency Scaling (DVFS) is a widely used option to reduce power. With DVFS the operating system determines the required performance per thread, selects the appropriate clock frequency and core voltage, communicates these requirements to the power control unit (PCU) that will set the requested clock rate and core voltage. DVFS was introduced in the early 2000s (AMD: 2000, Intel: 2003, IBM: 2004) but Samsung started its use only in 2012 (in its Exynos 4 CPUs).
- Adaptive Voltage and Frequency Scaling (AVFS): It is an improvement of DVFS, which dynamically determines the suitable voltage and frequency values in real time (based on a closed loop control) [5].

C. *ARM big.LITTLE Architectrue*

ARM's big.LITTLE solution is a power-consumption optimization technology for mobile processors that incorporate both high-performance and high-efficiency CPU core clusters. For low system load the high-efficiency core cluster whereas for high system load the high-performance core cluster is operated, thereby delivering good average consumption and high performance [6].

The figure below is based on measurements made on a Samsung Exynos 5422 processor and shows that at low performance requirements the power consumption of the "LITTLE" cores is much more favorable than that of the high-performance cores.

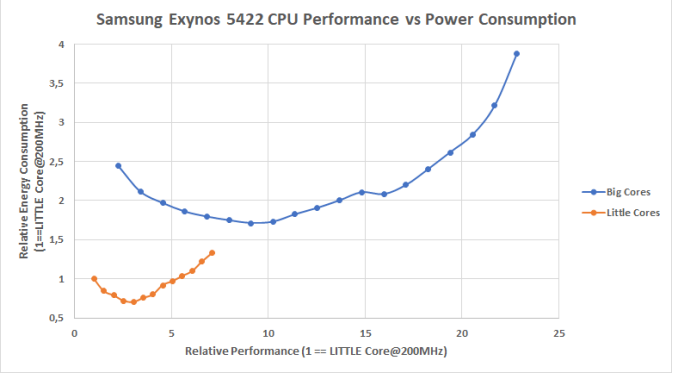

Fig. 1. Big.LITTLE CPU Performance vs Power Consumption

The Samsung Exynos 5422 CPU is built up of the ARM Cortex-A7 (LITTLE) and Cortex-A15 (big) cores. While the Cortex-A7 is an in-order, non-symmetric dual-issue processor with a pipeline length of between 8-stages and 10-stages, Cortex-A15 is an out-of-order triple-issue processor with a pipeline length of between 15-stages and 24-stages [7].

The key point of the solution is an algorithm that, based on the load of the cores, ensures that the tasks (processes, threads) are moved between the cores of different characteristics (but each core is built on the same ISA).

In the ARM's big.LITTLE solution, the high processing power and the low power cores are connected via a cache coherent switching network, so each core accesses the same shared memory. At application level, processor cores appear as a classical symmetric multiprocessor system, so there is no need to modify the application code running on them. Of course, optimization tasks already appear at the operating system level.

The processor used in our work consists of ARM Cortex A15 and Cortex A7 cores [8]. Both core types are ARM-v7A ISA compliant.

D. *Scheduling Schemes for Big.LITTLE Systems*

There are two aspects that determine the scheduling scheme, as follows:

- the task granularity of migration and
- the exclusive or inclusive use of big.LITTLE resources.

Granularity determines whether the task is distributed at cluster or individual core level. Exclusivity shows whether the combined use of big and LITTLE resources is allowed, or not). Based on the above, there are four possible models:

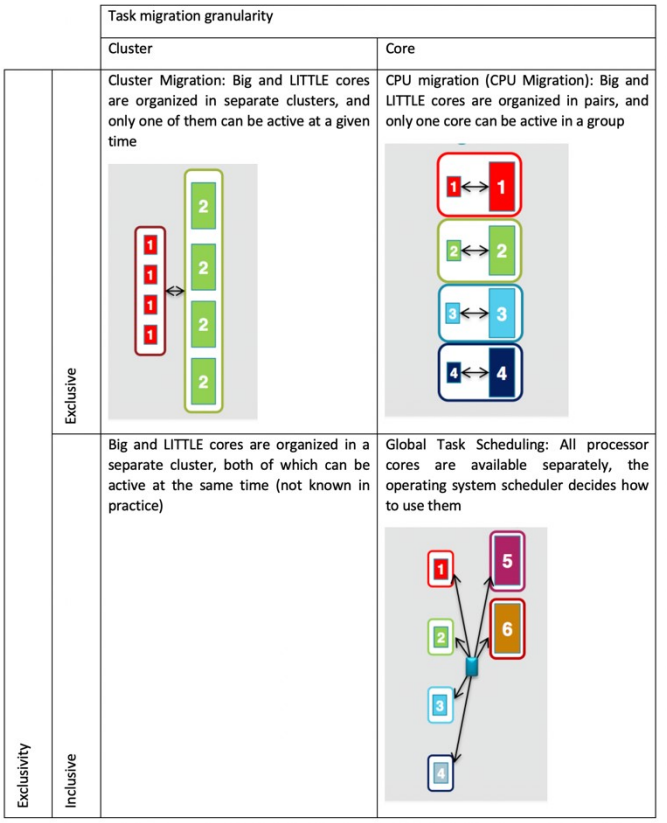

Fig. 2. Task Scheduling Possibilities in a big.LITTLE System [9]

The solution we are dealt with belongs to the Global Task Scheduling (GTS model). Currently, several companies have developed a scheduling solution for this purpose, such as Linaro or Qualcomm, but this has not yet been done for the Samsung Exynos 5422 processor.

E. Big.LITTLE Scheduling Implementations

ARM published their GTS solution in 2013. MediaTek introduced CorePilot 1.0 in 07/2013, Samsung its HMP in 09/2013, Qualcomm its EAS in 02/2014. The first production version of ARM/Linaro EAS solution was revealed in 10/2016 (on the Google Pixel Phone).

F. The Energy-Aware Scheduling Project

ARM and Linaro has started their Energy-Aware Scheduling (EAS) Project in 2013 [2].

The goal of the Energy-Aware Scheduling Project, is to overcome the duality found in Linux-based performance management subsystems (CPUFreq, CPUIdle subsystems), and to coordinate the scheduling and performance management.. In Linux, the CPUFreq and CPUIdle subsystems work independently of each other, and at times even against each other. Also, the scheduler does not consider any performance cost considerations when placing the tasks.

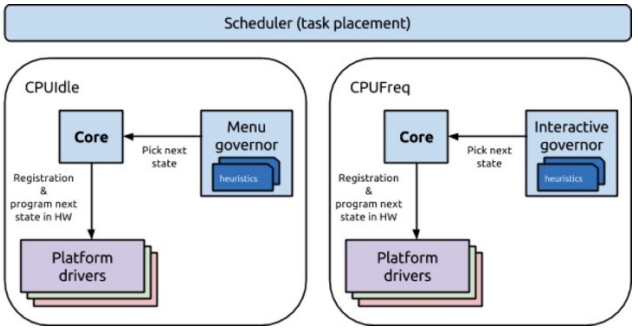

Fig. 3. Linux subsystems responsible for CPU management [3]

Although the consolidation of the scheduler and performance management subsystems seems to be a promising solution, there are several problems to be solved during its implementation. The key element of the problem is that the scheduler does not have any information about the processor architecture.

The purpose of the Linaro EAS project is to transform subsystems in such a way that - for energy use – subsystems provide a more efficient processor use [3].

G. Linux Governors

In Linux, Task Scheduling and Energy Management have been implemented in three different, loosely related subsystems:

- Scheduler
- Cpuidle that manages the inactive state of the CPUs
- Cpufreq that controls processor frequency.

The Scheduler is a Unix-like short-time scheduler. There are different algorithms for different task types, the most commonly used algorithm is the CFS.

The CPUIdle subsystem is responsible for managing inactive states. The governors belonging to this subsystem will become initialized by the idle cycle of the CPUs. The scheduler selects the idle cycle if the run queue for that processor is empty, i.e., when the processor is idle.

The CPUFreq subsystem implements the DVFS functionality. The subsystem consists of a combination of architecture-independent and architecture-dependent parts [10].

III. The Proposed Solution

A. Description of the CPUFreq Governor Solution

Since the development of the kernel code is difficult, we have decided to implement the solution in "mixed" mode: The execution part of the code is running in kernel mode, but the decisions are taken by the code running in user mode. Our findings show that the code of the execution part is relatively stable; the essential part of the algorithm is transferred to the user mode code. This kind of approach is not a novelty in operating systems, since the so-called microkernels have been based on this concept [11].

B. Kernel Mode Functions

In Kernel mode, codes are running that directly access kernel data and internal kernel functions. These are:

- Processor Frequency Query and Adjustment (per cluster)
- Processor Mode (Active / Passive) Query and Setup (per core)
- Processor Load value Query (per core)

Those kernel functions are accessible via the “sysfs” interface, which is a standardized interface on Linux systems between the kernel and the applications running in user mode.

C. Application in User Mode

The application running in user mode obtains the current load parameters at specified intervals through the sysfs interface and determines the target status that is set by kernel services calling via the sysfs interface.

The essence of this solution is that the program with help of a translation table determines the current computing power (relative to the computing power of a LITTLE core running at 200 MHz) for each run cycle based on the clock speed and load value of the processors, and then it determines the optimal setting of the CPU cores. The calculations are made by using tables prepared based on preliminary measurements.

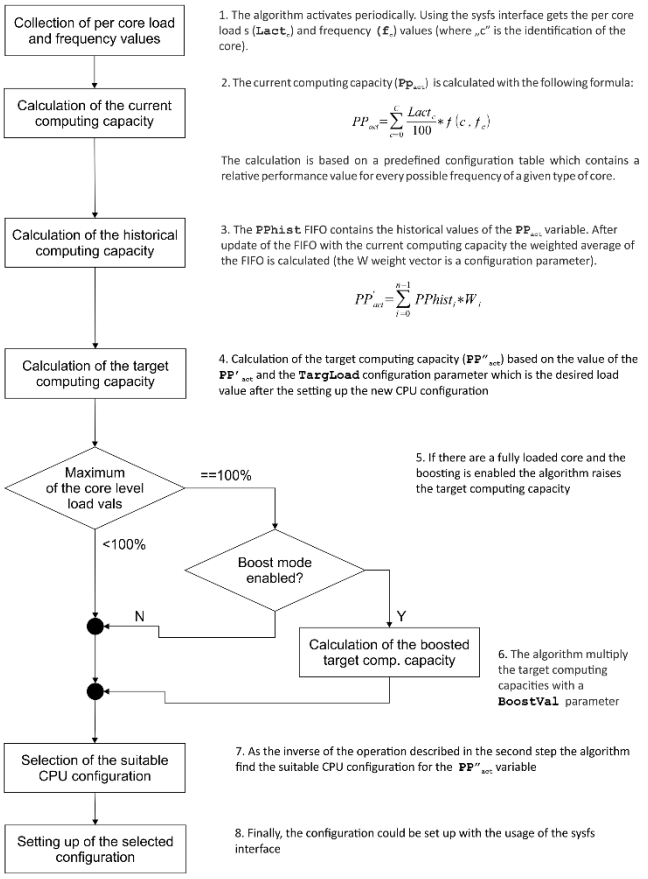

Fig. 4. The proposed governor algorithm

IV. The Proposed Testing Environment

Testing is currently based on running a simple prime search routine: the test procedure must look for the first "n" prime numbers and calculate their sum. The test runs with 32-bit fixed-point numbers. A framework designed to run a wide range of testing tasks is set up to run the prime search.

A. Test Environment

The test environment is modular. Parts of the test solution are the following:

- Test Generator: generates tasks according to a predefined scenario (test script).
- Worker Threads: the prime search code in a configurable number of instances.
- Message queues (Test tasks and Test responses): provides a connection between the "tester" and "worker" components, which also serves as a load balancer function.
- Monitor: during testing, the processor's performance characteristics (load values, CPU frequencies, core temperature, consumption) are collected by this component.
- Logger: a component that records the running characteristics of testing per task (task characteristics, timestamps, etc.).

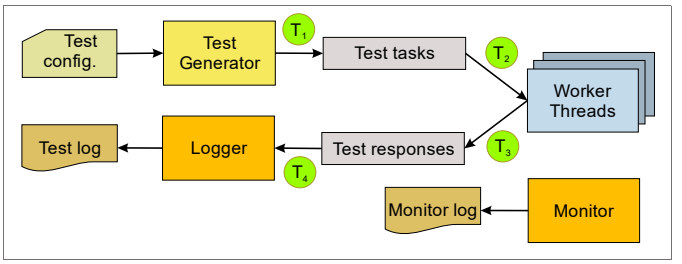

Fig. 5. The Testing Environment

B. Time Stamps

During the execution of the test task one packet travels through (a packet is a task), which contains the task characteristics and the related timestamps (in µs resolution):

- T_1 indicates when a task has been added to the task queue,
- T_2 indicates when the task processing has begun with a "worker" component,
- T_3 indicates when the "worker" component has processed the task and placed it back in the response queue,
- T_4 indicates when the Logger has removed the task from the response queue.

The tester configuration script consists of rows to be executed in succession, one line describing a test section. Features of a script are:

- The task to be performed (how many primes to search)
- Repetition time of the task (how often should the tester generate a new task)
- Burst: how many tasks the tester should give at a given time
- Cycle number: how many times the job is to be repeated

There is also a “wait” command as a special test line which specifies the time spent in inactivity.

C. Usage of Message Queues

POSIX Message Queues play a key role in the solution:

- They make operation asynchronous, e.g. the Logger cannot block Worker threads.
- They function as Load balancer between worker threads (if a thread is free, prompts next job from the line)
- They include an addressing mechanism to allow multiple test algorithms to run simultaneously (beyond prime search).

D. Delay Caused by Message Queues

The delay of the message queue-based solution has been investigated by a "NULL" test where the Worker thread immediately responds to the caller. The request has been repeated 10,000 times. Having performed the test with "Powersave" for minimum consumption and "Performance" governors with maximum computing power, the following turnaround times (TTR, calculated as T_4-T_1) have been added:

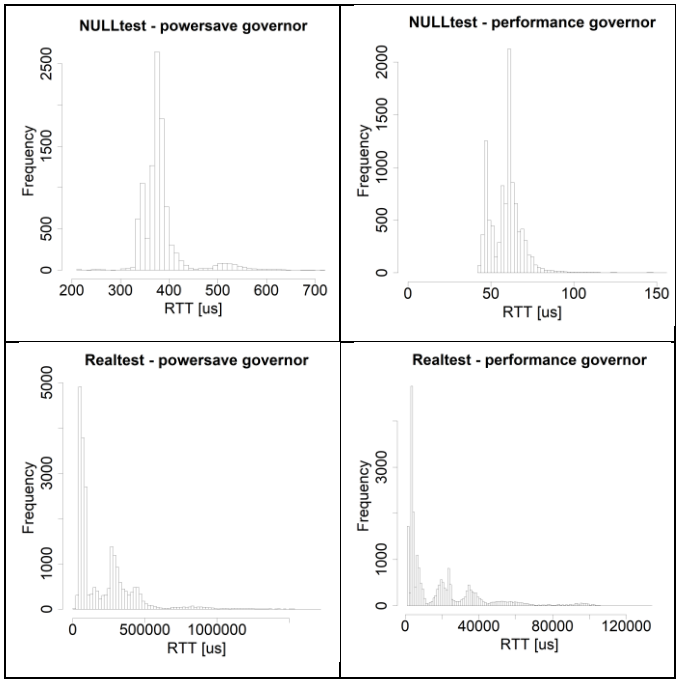

Fig. 6. Turnaround times

Based on the tests, it appears that the delay in handling message queues does not represent a significant amount of turnaround times in real tests.

E. Processing Test Data

Each run of the "Test Generator" program must be given a name (a string) that is also displayed in the "Logger" output. We have also generated a unique numerical project identifier with two elements:

- the timestamp of the start of the Test Generator with a second resolution,
- the number of the line in the Test Configuration file

In our work, we have count test cases in a counter type field. This counter restarts for each configuration file line.

The "Logger" and "Monitor" components write the measurement results into an easy-to-use "CSV" file. Between the rows of the two files, a µs resolution timestamp creates a connection. The output files are further processed by using R language functions.

The following diagrams show the primary processing of the execution results of a test run using the Interactive governor. (Fig. 7.) shows the characteristics of the test task:

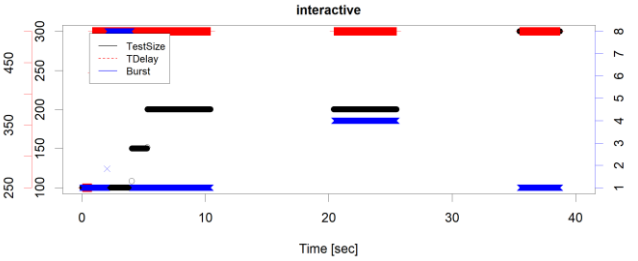

Fig. 7. Charasteristics of the Test task

(Fig. 8.) shows the size of the test queue (Red), the average waiting time for queues (Black), and the average execution time (T_3-T_2) for the tasks (Blue).

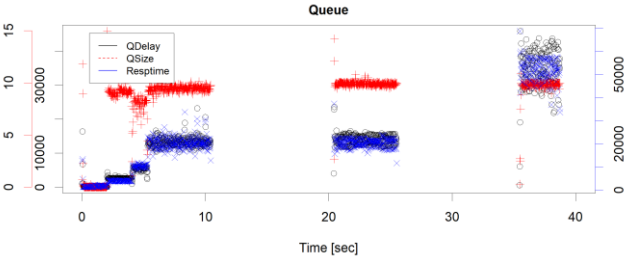

Fig. 8. Queue Load and execution time of tasks

Fig. 9 shows the frequency and the power consumption of the big and LITTLE cores:

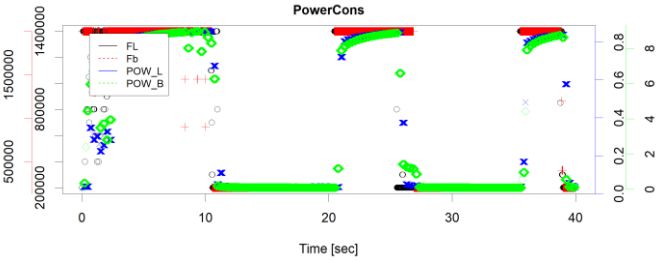

Fig. 9. CPU cluster frequencies and power consumption

F. Measuring Experience

For initial measurements, we have iterated over the free parameters of the prepared governor and compared their results with the Linux standard governors:

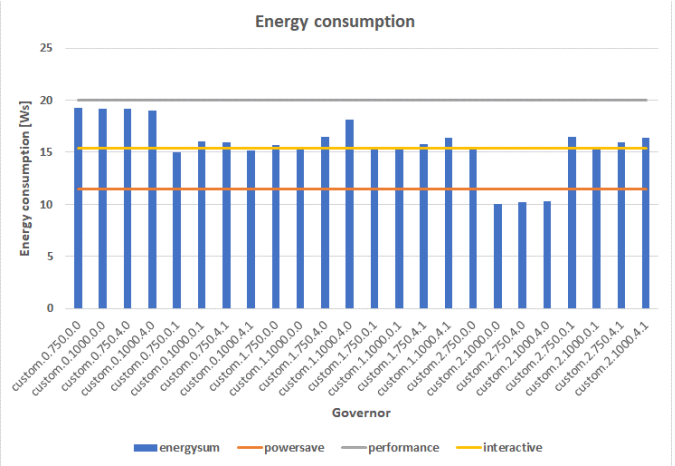

Fig. 10. Energy consumption during the initial tests

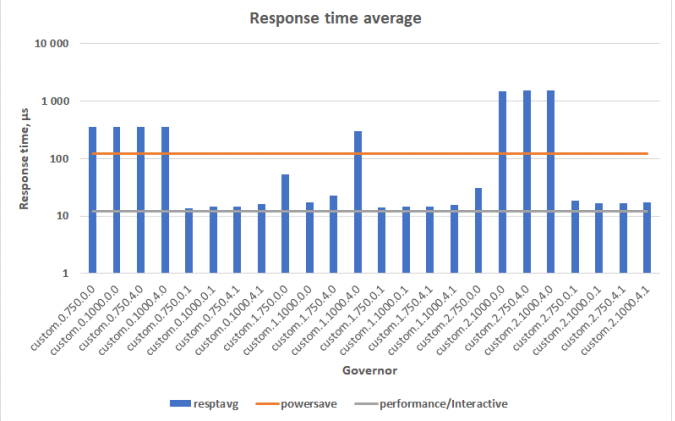

Fig. 11. Tesk task execution times during the initial tests

G. *Further experience*

The XU3 card's chiller cooling capacity does not allow long-term operation of the "big" cores at maximum frequency. The CPU temperature rises rapidly, and the system limits itself (the built-in protection mechanism begins to reduce the clock speed). This constraint, however, makes correct comparisons of test results impossible, so as a bypass solution, we have limited the maximum frequency of "big" cores at 2 GHz. At this frequency, the processor's temperature remains within the normal range.

Operation at a lower frequency results in energy savings, if:

$$\int p_i t_i > \int p_j t_j \quad (4)$$

Where "i" refers to the original settings and "j" to the operation after the frequency reduction. Based on the measurements, the XU3 card has a range where the above inequality is not met (see Fig. 1). The reason behind this is that the system has reduced the frequency but not the voltage. In the constant table used in the application, therefore, these work points have not been included.

V. Conclusions and Development Opportunities

Based on the first results, it can be stated that the custom-developed governor is worth of further refinement. The testing environment can efficiently be utilized for further testing as well.

The presented solution can be further developed in the following areas:

- Governor algorithms: developing new customization points in the algorithm
- Testing: The current test procedure makes use of a single algorithm that is based on fixed point calculations. It would also be useful to run other algorithms as well which represent a different load class (e.g. floating point calculations).

References

[1] ODROID-XU3 Wiki; https://wiki.odroid.com/old_product/odroid-xu3/odroid-xu3.

[2] Energy Aware Scheduling [EAS]; https://www.linaro.org/engineering/core/arm-power-management/eas/.

[3] Amit Kucheria; 2015; Energy Aware Scheduling Project; https://www.linaro.org/blog/energy-aware-scheduling-eas-project/.

[4] Hennessy, J. L. and Patterson, D. A., Computer Architecture: A Quantitative Approach., Morgan Kaufmann Publishers, Inc. San Mateo, CA., Fifth Edition, 2011.

[5] D. Sima, "Power Management of Processors v1.0", unpublished.

[6] ARM big.LITTLE Technology; https://www.arm.com/products/processors/technologies/biglittleprocessing.php.

[7] Peter Greenhalgh, Big.LITTLE Processing with ARM Cortex™-A15 & Cortex-A7, ARM White Paper, 2011

[8] Mobile Processor, Exynos 5 Octa (5422); https://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-5-octa-5422/

[9] D. Sima, "big.LITTLE technolog", unpublished.

[10] Linux kernel source 3.10.96; www.kernel.org

[11] W. Stallings, Operating Systems, Pearson Education, Harlow, England, Ninth Edition, 2018