CRITICALLITY STACKS IMPLEMENTED VIA KERNEL FREQUENCY GOVERNOR FOR POWER SAVINGS IN ARM MOBILE ARCHITECTURE

An Undergraduate Research Scholars Thesis

by

CESAR LOPEZ CARRASCO

Submitted to the Undergraduate Research Scholars Thesis program
Texas A&M University
in partial fulfillment of the requirements for the degree of

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Paul Gratz

May 2018

Major: Electrical Engineering

TABLE OF CONTENTS

ABSTRACT

Criticallity Stacks Implemented Via Kernel Frequency Governor for Power Savings in ARM Mobile Architecture

Cesar Lopez Carrasco
Department of Electrical Engineering
Texas A&M University

Research Advisor: Dr. Paul Gratz
Department of Electrical and Computer Engineering
Texas A&M University

With the advent of mobile computing devices such as smartphones, modern mobile processors are designed by compromising two opposite characteristics. In one hand, performance needs to be maximized in order to hold up with increasingly demanding applications such as virtual reality, real-time image processing, and intense multitasking. On the other hand, power consumption needs to be minimized, these devices typically run off portable batteries which have had unchanged capacities throughout the last half of the decade.

The two methods to have performance on demand, and regulate power consumption are having two different types of processors a.k.a big.LITTLE (one perfomance oriented and one energy saving oriented) and Dynamic Voltage-Frequency Scaling (DVFS). Ideally, performance scales linearly with the frequency of the processor; however, most of the systems on a SoC run slower than the CPU making an increase in CPU frequency not result in an increased performance because the processor is idle or waiting. On the contrary, power consumption increases as a function of the cube of the frequency. Determining how

to adapt frequency in order to maximize performance while optimizing energy consumption is the objective of DVFS. The gap between big and LITTLE processors is shrinking in modern mobile architectures, this necessitates more aggressive DVFS algorithms research. This study aims at adapting a DVFS algorithm that uses Criticality Stacks applied to scalable for the ARM microarchitechture using the Linux kernel. This algorithm has been tested in simulated hardware with generic benchmarks through pseudo-hardware implementation. A parallel research in Texas A&M is working into adapting this algorithm into a software implementation via the Linux kernel in a x86 architecture. Forking from that code, the algorithm was adapted to work on ARM on two separate Single Board Computers. As a result the algorithm was tested in multiple versions of ARM with a different range of frequencies and varying architecture complexities. Finally the Algorithm was adapted to be used to determine CPU migration mechanisms that are implemented in ARM big.LITTLE

The end result of this research is the verification of the previous simulated hardware which yielded 12% powersavings with 4% reduction in performance. Testing the algorithm in real silicon allowed to validate these results and as well as the feasibility and constraints of this implementation.

DEDICATION

"There is no threshold that makes us greater than the sum of our parts, no inflection point at which we become fully alive. We can't define consciousness because consciousness does not exist. Humans fancy that there's something special about the way we perceive the world, and yet we live in loops, as tight and as closed as the hosts do, seldom questioning our choices, content, for the most part, to be told what to do next. No, my friend, you're not missing anything at all." - Westworld

Break the loop.

To my father, whose effort has brought me this far.

ACKNOWLEDGMENTS

I will like to Acknowledge Bryan Elliot for giving me guidance while modifying his code. Dr. Gratz for patiently taking the time to discuss progress every week and for the flexibility and effort that have been necessary for the conclusion of this study. Finally I would like to acknowledge every person who have supported and encouraged me even when I've been to busy to spend more time together.

NOMENCLATURE

DVFS	Dynamic Voltage Frequency Scaling
SOC	System On a Chip
SBC	Single Board Computer
F	Frequency
V	Voltage/Volts
ARM	Advanced RISC Machine (Mobile Computer Architecture)
x86	Intel Designed ISA, popular on Desktops
A	Amperes (Current)

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

1.1 Motivation

Nowadays,Virtually every mobile device runs on the ARM mobile architecture. As the popularity of this architecture increased, so has increased the variety and the performance demands of the software that runs on it. Mobile phones are no longer limited to single threaded applications like light web browsing. High-end devices are designed to run in CPU demanding environments such as Virtual Reality applications and heavy multitasking. This has been achieved by requiring the demanding applications to rely strongly upon multithreading.

As performance demands increase, mobile phones have to deal with the fact that they are power constrained. As seen in figure 1.1, Eason [1] notes that from 2004 to 2011, the battery capacity has been almost constant. This changed several years after due to the introduction of larger phones which could accommodate a larger battery, not to improvement in battery technology. Eason [1] states that batteries improve very slowly while mobile processors improve in according to Moore's law. As a result, the performance has to improve while keeping the battery capacity constant. As a result, improving CPU efficiency is crucial for maintaining (and sometimes expanding) battery life, benefiting their mobile phone users. Taking all this into consideration, researching power savings techniques in mobile devices is justified.

With a majority of these mobile devices running the Linux kernel through Android; novel per-core frequency scaling mechanisms that take advantage of the available performance while minimizing power consumption can be implemented through Over-the-Air updates. The Motivation of this Research is to determine if applying these methods (namely a combination of Scalability and Criticality Stacks) translate into power savings

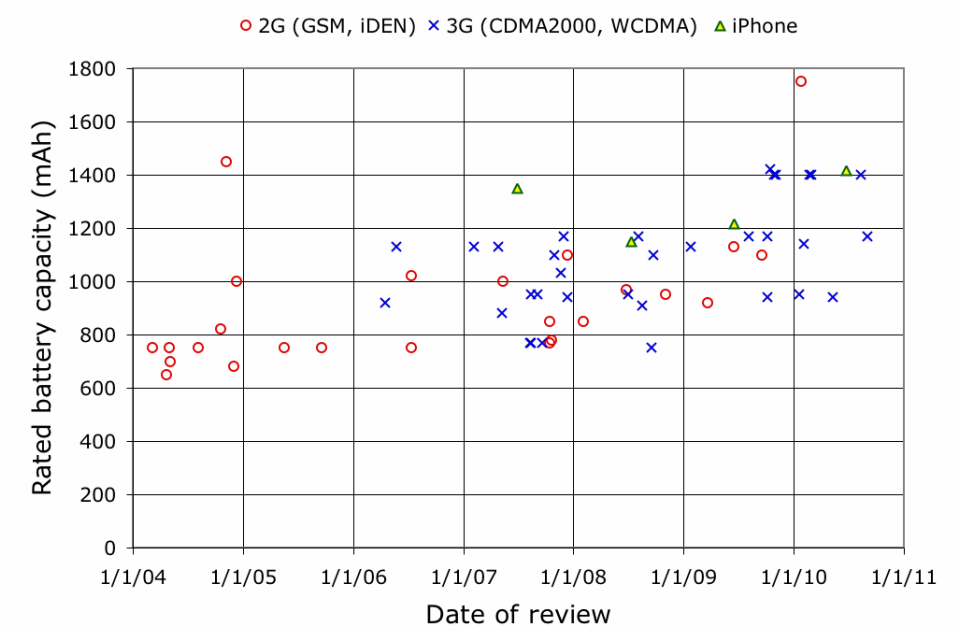

Figure 1.1: Battery capacity on smartphones throughout the years ©Eason 2012

with minimal performance loss.

Given that these concepts have already been proven to yield power savings through simulation, the expected outcome is to validate these simulations by implementing them through the Linux Kernel on real silicon chips and testing them using real world applications and synthetic benchmarks. This will be done by making modifications to the Linux Kernel which will eventually be submitted as a patch to the main Linux Kernel.

1.2 Power Savings, A Case for DVFS

1.2.1 Current Technologies

Walshe [2] states that ARM has been a fundamental part of mobile computing since the early 90's and currently it is an industry leader with 96% of total market-share. With such market-share ARM is designed around its power constraint. Combined with a simple architecture ARM implements heterogeneous processors a way of saving power. big.LITTLE

which consists of one part of the processor using performance optimized (but power hungry) cores and the other part uses energy optimized (although slower) cores. big.LITTLE uses one of these blocks at a time as a way of extending battery life when needed and delivering peak performance on demand. ARM [3] states that the power optimized cores are typically in use 95% of the time and performance optimized the other 5%.

In order to determine how to optimally use these two types of processors, ARM implements several migration methods. For the sake of simplicity we will assume that we are using Symmetrical Multiproccessors (SMP). SMPs consist of processors are designed on a symmetrical configuration (2 big+2 LITTLE or 4b+4L). As explained by Yoo [4], there is a switching cost as a result of the overhead when switching operation from one processor to another (Usually big to LITTLE or viceversa). The optimization of these switching methods is important. Using SMPs the two typical migration methods are Cluster migration and CPU Migration [5]. For Cluster Migration, the OS can only see two clusters with the same amount of cores, depending on the workload, it will decide to switch from one cluster to the other (Only one cluster is used at a given time). For CPU Migration, the OS will see *n* clusters (where n is the the amount of either big or little processors). Each cluster contains a big and LITTLE processor and it will change between them depending on the load. Only one of the processors of each cluster can be used at a given time. These two migration Mechanism are visually explained in figure 1.2.

1.2.2 Future Technologies

It is worth noting that in the coming years, ARM will introduce DynamicIQ which permits all cores to be used at a given as seen on figure 1.3. This will not only improve performance, but also it will permit a wider range of configurations without much switch overhead. Taking into account the reduction in power consumption produced by new manufacturing methods (like Samsung's 10nm), future high-end processors will feature more

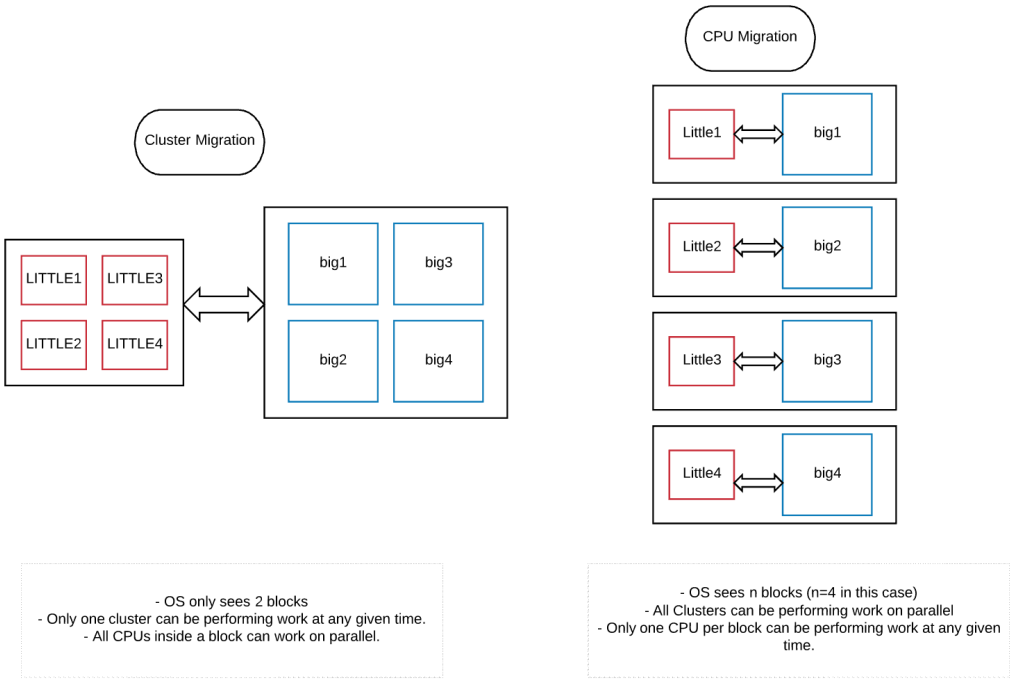

Figure 1.2: Migration Methods for SMP. Cluster Migration vs CPU Migration

big (performance optimized) and less LITTLE (power optimized) cores given that nowadays there is very little difference in the big vs Little Processors. This is evident in the case of processors with the Qualcomm 835 Chip which instead of using ARM designed LITTLE cores, it implements big.LITTLE with two blocks of big processors with a difference in frequency. In a non-distant future, it is likely that mobile processors can run on 8 homogeneous cores, which calls for other solutions for additional power savings.

1.2.3 DVFS and CPU Migration

Clarke proposes [6] that another method for dealing with power savings is extending Dynamic Voltage Frequency Scaling (DVFS). DVFS is a method of adjusting CPU frequency on a per core level in order to reduce processor iddle time. Clarke claims [6] that this should be leveraged with a wider gamut of frequencies so DVFS can also be used as a

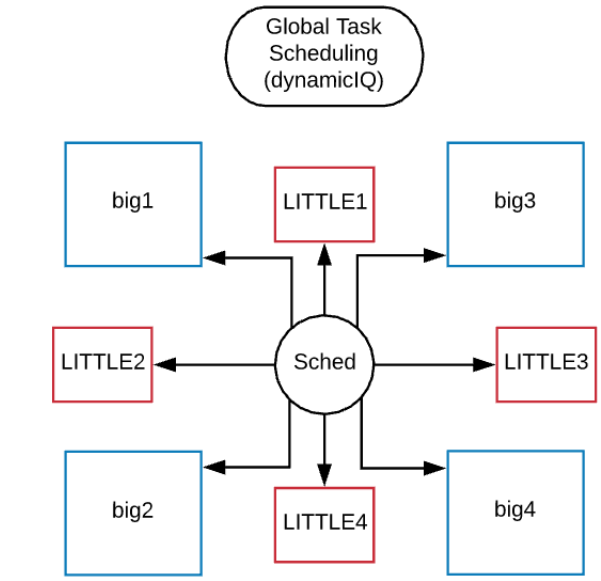

Figure 1.3: Dynamic IQ use of the CPU in 4+4 configuration

method to save energy. The top of the line processors in 2017 are based on the Cortex A73 architecture where the frequency scales up to around 2.4GHz, something unprecedented in mobile processors and very close to their desktop grade counterparts. This not only allows but necessitates the creation of more efficient frequency scalers that take advantage of the available frequency ranges. These new methods for implementing DVFS have to be adapted to high core count CMPs. Although some already exist in the Linux Kernel like the OnDemand Frequency Governor, different methods to maximize performance through frequency setting should be explored.

Kidd [7] explains how the power consumed in a digital circuit (i.e. Processor) obtained by changing the frequency follows a linear trend modeled by equation 1.1. while C *a is constant, the voltage and frequency can be set. For maximum performance, a processor will run at maximum frequency and operate at maximum voltage. This is very power demanding nonetheless, and ideally, it needs to be avoided. The voltage and the frequency are tied together, and a reduction in the frequency permits for the reduction in the voltage. This means that decreasing the frequency translates into a linear reduction in power consumption and allows for a reduction in the operating voltage. Decreasing the frequency allows a reduction in the Voltage which finally results in a cubic reduction of the power. As a result the Power is affected *exponentially* by the change of the frequency.

$$P = (C * a) * f * V^2 \tag{1.1}$$

With respects to workload migration, even though frequency setting on a homogeneous processor (or big/LITTLE cluster) is a direct benefit of DVFS, another added advantage of exploring more DVFS methods is workload determination. As explained before, the big.LITTLE processor will migrate between big or Little processors with respect to the load. The burden of the load is determined by DVFS [8], thus researching and contrasting new and current methods in the ARM architecture benefits more than direct frequency setting.

One of this methods was explored by Girdhar [9] who used the critically stacks concept introduced by Du Buois [et. al] [10] as a way to implement DVFS when the processor is running code that scales its performance with respect to core frequency. His results yielded the expected outcome while simulating in an ARM processor using GEM5. Performance dropped minimally (2-4%) while reducing power consumption by up to 12%. However this simulation was limited to a frequency range from 1.6-2.66GHz in synthetic bench-

marks. Testing this concept on real silicon with wider frequency ranges in both real world applications (code compilation) and synthetic benchmarks is a better method of testing this concept. At the same time, if the desired power savings are obtained, this ARM frequency scaler will be a useful addition to the linux Kernel.

2. PREVIOUS AND CONCURRENT WORKS

2.1 Criticallity Stacks

The concept of Criticallity stacks was introduced by Du Bois [et. al] [10] as a method of determine which thread is holding the others of performing work. This is done by determining which of the threads in a running program is more critical by assigning a "criticallity score" to each thread. A score is dynamically assigned to a thread depending on how many concurrent tasks are running. Du Bois [et Al] assume that if a thread is running by itself in a given time slice, while the rest are not performing any job, this means that this thread is performing work that is needed by the rest to continue, thus slowing them down. As a result, this thread will be "critical". On the oposite, if given a time slice, all threads are performing work, this means that they are equally important, thus they share the same criticallity is added to their stack.

In order to dynamically assign a score the time is divided into equal time slices. Each thread starts with a stack of 0, after the first time slice, the algorithm checks which threads are running, then it adds 1/(number of threads running) to their stack. A visual representation of the additions performed after each time slice is seen in figure 2.1. The representation of the stack after each time slice in figure 2.2

The underlying purpose of this method is to find the *Critical* thread. It is non-trivial for the hardware or the operating system to analyze which thread is more critical and has other threads waiting. It is claimed by DuBois [10] that current methods used to calculate criticality are based on cache misses and Bottleneck Identification and Scheduling; nonetheless, by using solely Criticality Stacks, the performance is increased as a result of a more accurate identification of the critical thread (s). In his tests, changing this algorithm resulted in speedups of 1.67x for 8 core systems and 2.16x for 16 core. These speedups

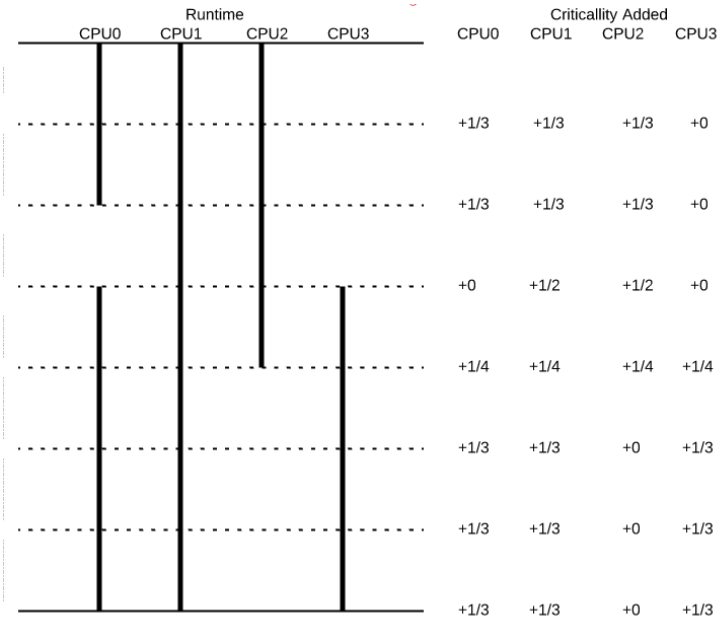

Figure 2.1: Criticallity stacks: Additions to stack after each time slice.

permit the reduction of the frequency of the non critical threads without impacting performance. By accelerating only the critical threads, performance was stable and power consumption was reduced by 3.8% and 3.2% for 8 and 16 cores respectively. speedup for

2.2 Scalability and Criticality Stacks Simulation

Although the Criticality Stacks algorithm has proven itself useful for identifying the critical thread. Parting from the fact that only accelerating one thread ignores the power savings obtained by reducing the other threads, Ghirdar [9] used this algorithm. In order to account for this, Ghirdar uses the criticality score difference between a given thread and the critical thread as a *Slack*. This slack ration is useful to identify how much the frequency can be reduced. For example, for an nth thread, slack is calculated in equation 2.1.

$$Slack = 100 * \frac{C_{critical} - C_n}{C_{Critical}} \tag{2.1}$$

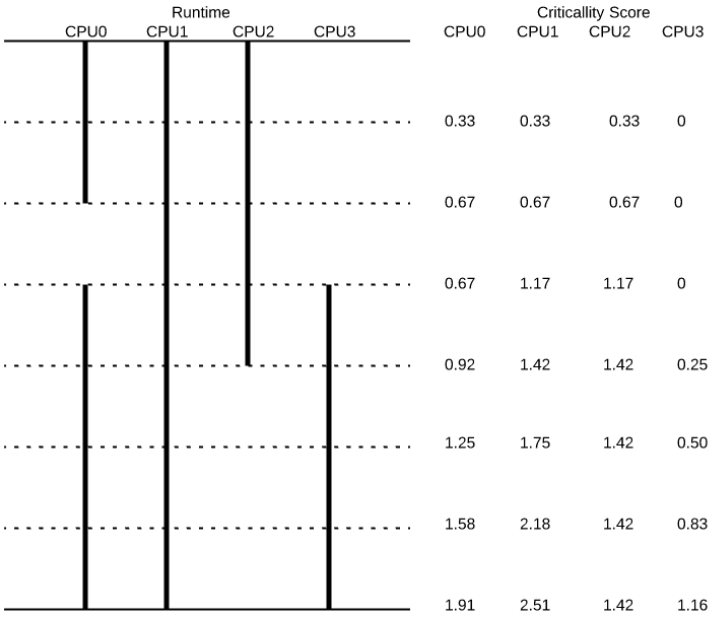

Figure 2.2: Criticallity stacks: Criticallity stack after each time slice

For the example in figure 2.2, we identify our Critical thread to be the one running on CPU1 given that it has a the largest criticality at any given time. Using 2.1, we can calculate the slack at the end of each time slice to be as shown in figure 2.3. In this figure we can see that the inactive CPU3 has an available slack of 100% until the moment that it starts running when its slack starts decreasing continuously. The same thing happens for CPU2 when its stops running, its slack continuously drops. In order to reset the stack and get a slack stabilized over time, Ghirdar accumulates the stack and makes a DVFS desision until a fixed control period time has passed. In his case it is 3 ms (whenever the OS Scheduled is called).

Ideally the original concept of criticality stacks would suffice in a world were the processor has 100% of cache hits, predicts correctly every branch and it is not waiting for any other slower subsystem inside the SOC. Lets assume a thread is determined to be most critical, however given the small caches in ARM, this thread spends a significant amount

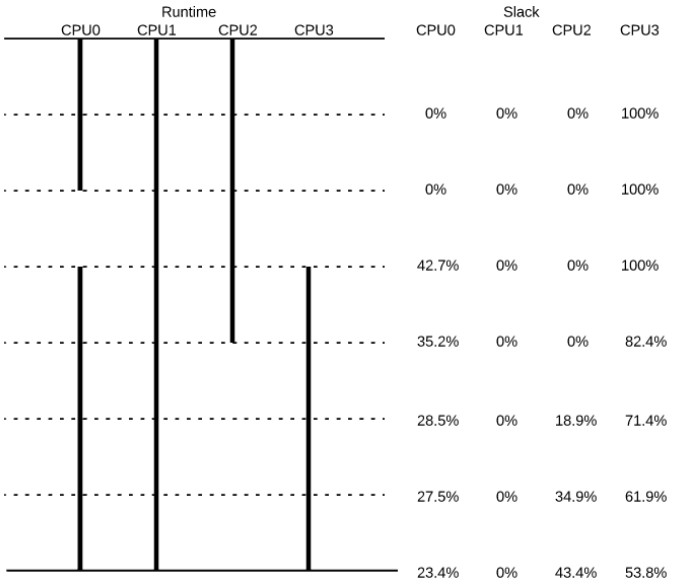

Figure 2.3: Frequency slack with respect to criticality

of time waiting due to cache misses, the original concept of criticallity stacks does not asses wether it would be worth or not to increase the CPU frequency, taking into account the exponential power penalty we have to pay for this increase in frequency.

This concept is also part of Ghirdar's Algorithm[9] under the name of Scalability. Scalability assigns a score to a CPU given the correlation between increases in frequency and increases in speed. The Scalability of a thread is determined by using the L1 and L2 cache misses and combining adding a weighted sum of them portrayed in equation 2.2. The ProActive Load Balancing Algorithm (PALBA) balances available slack and scalability of the code in order to asses assign a frequency to each core on the next controlled period.

$$S = 0.994 - 0.00000908 * L_{1,misses} - 0.000148 * L_{2,misses} \tag{2.2}$$

2.3 Hardware Testing Via Software Implementation

Given that DuBois proved the efficacy of the algorithm when running in multiple Threads, a concurrent study from Texas University by Bryan Elliot is translating this algorithm into a Linux Kernel Frequency Governor. Taking into account that the underlying nature of Linux multithreaded parallelism is that processes are ran as threads, instead of tracking the Criticality on a per CPU way, this patch calculates the Criticality and Salability for each process. This way, if the process is active, the OS can add the corresponding values to the Criticallity stack. When the Scheduler is ran, the OS calculates the slack. The scalability is ran in a similar fashion. The On-Chip performance counters (perf) are started for the CPU instructions and the two levels of cache. Every 10ms that the Scheduler is called, the algorithm calculates the performance as a ratio of 1000. Although the Ultimate goal of the study done by Mr. Elliot is corroborating these simulation in x86 with an abundance of processors, the goal of this study is to adapt and test this code for testing in ARM.

3. HARDWARE SELECTION AND TESTING METHODS

3.1 Hardware Selection

3.1.1 Raspberry Pi

The Rapsberry Pi 3 Model B was selected to conduct the initial round of testing of the frequency governor because it only has one block of A53 LITTLE processors. Thanks to this, Migration between blocks does not need to be accounted for because by default it does not happen. Secondly, both perf counters and two-level caches are available in order to calculate the scalability value. Finally the Pi has extensive documentation and support for compiling the linux kernel, making it ideal for a first round of testing. Detailed specs of the Pi as configure when testing are included in table 3.1. Finally, the simpler, smaller A53 is useful in order to see the effect of frequency scaling in simple processors

Table 3.1: Raspberry Pi Specs

Model	Raspberry Pi 3 Model B
Frequency Range (MHz)	600-1200
Processor	4 LITTLE (A53)
Instruction set	ARMV7 (32 bit)
L1 Cache	16KB
L2 Cache	128KB
RAM	1 GB

3.1.2 HiKey 960

Once the proof of concept was tested in the Raspberry Pi, there is a necessity of testing the new frequency governor in a processor similar to modern smart phones. Given its wider frequency range (Up to 2.4 GHz) the Hikey 960 was selected as a testing platform

given that it is built for running Linux. Initially the big processor will be used without CPU migration to replicate the test with a larger cache, larger frequency range but equal core count. Finally, the 960 has a big.LITTLE arrangement, in where DVFS can be tested for the diferent CPU migration mechanisms available. The full specs are included in table 3.2.

Table 3.2: Hikey 960 Specs

Model	Hikey 960
Frequency Range (MHz)	600-2400
Processor	4 LITTLE (A53) 4 big (A73)
Instruction set	ARMV8 (64 bit)
L1 Cache	64KB
L2 Cache	1MB
RAM	3 GB

3.2 DVFS in ARM

In order to be able to see the direct effect of power consumption, when performance is reduced, a generic DVFS driver already present in the Linux source was included in the Kernel mounted. This was done by modifying the default .config file from each board to include this driver and its respective scalers for each governor. This setting is changed by calling *make menuconfig* loading the default *.config* file from each board and enabling *Device Drivers-> Generic Dynamic Voltage and Frequency Scaling (DVFS) Support* as well as the governors inside it. This setting ensures that when the frequency is changed, the voltage is also changed to a lower voltage. This so the power consumption can follow the cubic relationship explained in equation 1.1.

3.3 Benchmarking Selection

3.3.1 Parsec

PARSEC was selected as a benchmark due to the natural parallelism of its different tests and the variety of tests available. One particular advantage of PARSEC is being able to simulate close to real life scenarios [11]. PARSEC includes several data-sets; however, given that testing was being done in real sillicon, the *native* data-set was selected, as it is large enough for extensive testing. Another benefit is the varying degrees of Data sharing and rates of Exchange. Data sharing has a direct correlation to L1, L2 cache Hit/Miss and Scalability. The Exchange rate are the number of Barriers and Conditions that stop other threads from running, hence creating Critical threads. The specialty of the Scalable Criticality Stacks algorithm is acceleration by identifying the *Critical Thread* and running it faster than the other ones. However given the absence of Critical threads, one can test the general efficacy of this algorithm in cases where the threads are completely Orthogonal like real life multitasking. PARSEC is built to test the this principle by having tests like BlackScholes that have 8 barriers (8 Critical Threads) to StreamCluster which has arround 130,000 barriers (Several thousands of Critical threads).

4. EXPERIMENTAL RESULTS

4.1 Setup

The only variable that we are interested in is the power consumption of the CPUs, nonetheless in SOCs it is virtually impossible to completely isolate this given that inside the same system, the Chip includes Memory, CPU , Networking, Image processors, GPUs... In order to create more accurate readings and taking into account the power consumption of these ever present systems, the setup was configured to make the least use of non-CPU components as possible. This meant turning off Networking modules, ISPs GPUs and limiting the system to operation via serial interfaces. Also in order to reduce internal thermal resistance buildup, Tests are ran with heatsinks over the board and a fan at a static speed directly blowing on such heatsinks.

In order to test power consumption an Arduino M0 that has the capability of logging 0-3.3V at a 12 bit resolution was used in conjunction with an INA169 circuit that paired with a 100Ω resistor makes a conversion of current from 1mA=1mV. The Raspberry Pi runs at 5V and the Kikey 960 runs at 12V. In order to get these voltages to a readable range, a potentiometer-calibrated voltage divider of V/4 was applied to the voltage before reading and multiplied after the reading inside the micro controller. The values were printed to a serial interface every 10ms where they were collected with their respective timestamp for further analysis.

4.2 Results

4.2.1 Control Testing in Raspberry Pi

In order to test the direct response of power consumption vs frequency, two different frequency governors were set while running both the BlackScholes and StreamCluster tests

from the PARSEC 3 benchmark suite. These governors set the frequency statically to the maximum, the minimum. The total time to perform this benchmark using the native data-set was taken as a measurement of the performance. As explained on previous section, the native data set was used and the tests were run spawning 4 parallel threads. This setup was selected as control for several reasons:

- Because the CPU power consumption cannot be fully isolated in the SBC, this test permits to test the effectiveness of DVFS on CPU power consumption.
- While being monitored, Powersave and Perfomance provide the minimum and maximum power consumption possible.
- Conversely, these tests show the direct impact of frequency on performance.

The results for Blackscholes and StreamCulster are shown in tables 4.1 and 4.2.

Table 4.1: Power Consumption vs frequency for BlackScholes

Governor	frequency (MHz)	Mean Power Consumption (mW)	Run time (s)
Performance	1200	7328	611
Powersave	600	6433	615

Table 4.2: Power Consumption vs frequency for StreamCluster

Governor	frequency (MHz)	Mean Power Consumption (mW)	Run time (s)
Performance	1200	8712	1609
Powersave	600	7456	1664

The transient data collection for the blackscholes benchmark is also shown in figures 4.1 and 4.2

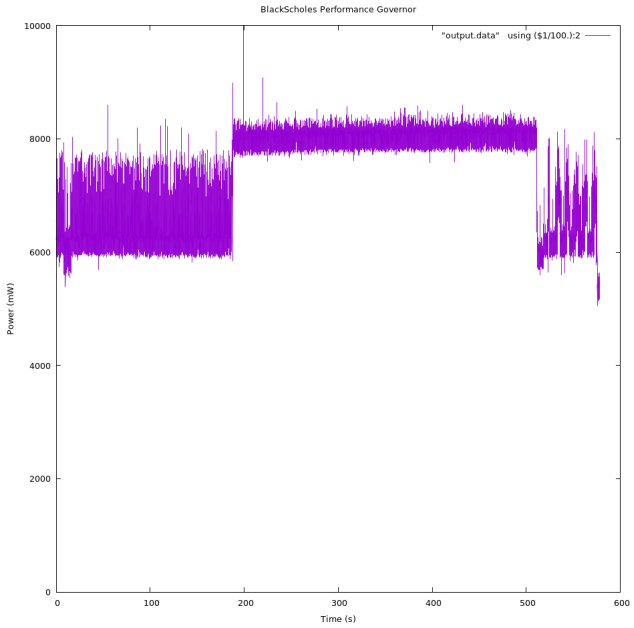

Figure 4.1: Blackscholes Perfomance

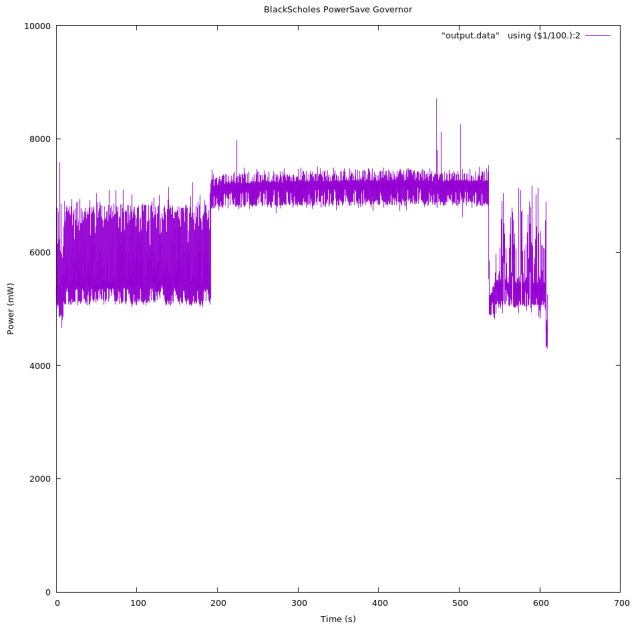

Figure 4.2: Blackscholes Powersave

4.2.2 Hikey960

Although, On Paper, the Huawei Hikey 960 should not have this same shortcomings (specially in the A73 cores), this board is extremely limited. Designed to be used as for Android Device Prototyping, the only implementation of a barebones linux system is an unstable version of Debian only accessible by a custom UART (1.8V) and without access to its built-in wifi, usb, usbc or m.2 ports. As a result the only software that can be loaded is through an SD card, and kernel configuration and crosscompiling documentation is nonexistent. Given said shortcomings and until this system fully supports debian, this testing is also unfeasible.

Although other boards were studied, there are no other Single Board Computers that are powered by recent big cores. This is a result of these boards being designed to run lightweight applications. High Performance processors are limited to Mobile phones or niche products like NVIDIA Development boards. These products usually contain more power hungry components (GPUs, Screens, 4g antennas...) that make power tracking virtually imposible.

5. SUMMARY AND CONCLUSIONS

5.1 Conclusion

In Conclusion, although in Simulation this load Balancing algorithm has shown to be promising, the existing tools are not adequate to be tested on real life. Given the inability of linux on ARM of being able to obtain real-time information of the CPU in which the thread is running at, Mr. Elliot's implementation would need to track the CPUs directly as opposed to tracking a thread. Although the Raspberry Pi is not versatile enough to be able to determine performance of a memory intensive test as a function of the frequency, the 960 should provide a better insight. The only shortcoming of this specific board is that it is not targeted and does not have a fully functional implementation of linux in which these tests could be performed.

5.2 Challenges

As mentioned in last section, the main issue was low availability of ARM based linux boards that are performanced centered. Also, the difficulty of isolating CPU power consumption inside a system limits the possibilities in which DVFS algorithms of applications can be tested.

5.3 Future Work

Given that the Kernel was not tested, a clear continuation is adapting the kernel code to read CPU info from within a thread indirectly. After that, and given the possibility of future availability of Linux for Performance Oriented SBC, validation of these results can be performed. Finally, after it is tested as a frequency scaler, it can also be tested as a CPU Migration Mechanism.

REFERENCES

[1] E. Eason, "Smartphone battery inadequacy,"

[2] B. Walshe, "A brief history of arm: Part 2," tech. rep., 2015. Accessed August 18 2017.

[3] ARM, "Arm big.little technology explained." https://www.youtube.com/watch?v=KClygZtp8mA, 2014. Accessed August 18 2017.

[4] S. Yoo, Y. Shim, S. Lee, S.-A. Lee, and J. Kim, "A case for bad big.little switching: How to scale power-performance in si-hmp," in *Proceedings of the Workshop on Power-Aware Computing and Systems*, HotPower '15, (New York, NY, USA), pp. 1–5, ACM, 2015.

[5] "big.little technology: The future of mobile: Making very high performance available in a mobile envelope without sacrificing energy efficiency," tech. rep., 2013.

[6] P. Clarke, "London calling: Are arm's big-little days numbered?," tech. rep., EE times, 2013.

[7] T. Kidd, "Why p scales as c*v^2*f is so obvious," tech. rep., Intel Developer zone, 2015.

[8] "Arm cortex -a series: Programmers guide for armv8- version 1.0a," tech. rep., 2015.

[9] N. Girdhar, "Palba: Pro active load balancing algoritghm using thread criticallity and scalability prediction," Master's thesis, Texas A&M University, 2016.

[10] K. Du Bois, S. Eyerman, J. B. Sartor, and L. Eeckhout, "Criticality stacks: Identifying critical threads in parallel programs using synchronization behavior," *ACM SIGARCH Computer Architecture News*, vol. 41, no. 3, pp. 511–522, 2013.

[11] C. Bienia, *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

APPENDIX

Figure 1 shows the setup between the voltage divider (Top Left), the INA169 (Bottom Left) and the Arduino M0 (Right). The power source comes from the bottom and the device being measured is connected to the wires leaving in the top of the picture. The current goint to the voltage divider is excluded from the INA169 and the entire divider has a resistance of 4k Ω, which only consumes 6.25mW @ 5V or 36mW @ 12V which is negligible given that the power Supplies are rated for 12,000mW and 25,000mW respecitvely.

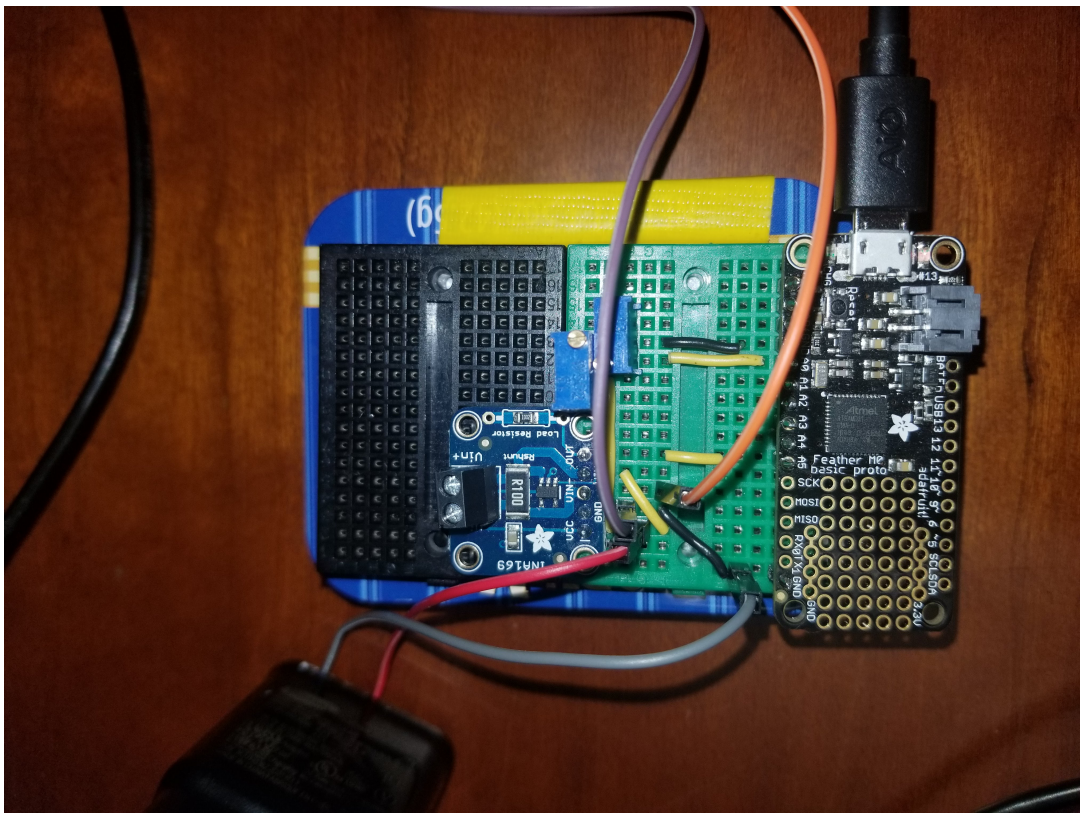

Figure 1: Power Measuring Tool