Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC

Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, Pål Halvorsen
Simula Research Laboratory & University of Oslo, Norway
{krisrst, haakonks, griff, paalh}@ifi.uio.no

ABSTRACT

Energy consumption is an important issue for mobile devices, as the technological development in battery technology has not kept pace with the power requirements of mobile hardware. This has led to innovative architectures with new power-saving capabilities, such as the Tegra K1 System-on-Chip (SoC). With a quad-core CPU, 192 CUDA cores and frequency scaling, the challenge in developing software for this architecture lies in optimising the energy consumption of mobile workloads while meeting performance requirements. In this paper, we consider the energy consumption of continuous multimedia workloads that are performed on video streams, such as image rotation. We take an empirical approach to find the most energy-efficient hardware and software configurations on a set of test videos. We find that the processor frequency should be minimised such that application deadlines are met. Using this heuristic, a reduction in energy consumption by up to 28% can be achieved compared to the standard Linux frequency scaling algorithms.

General Terms

Experimentation; measurement; performance

Keywords

multimedia workloads; power measurements; mobile SoC

1. INTRODUCTION

Battery capacity is a serious limitation of modern mobile devices. Despite improvements, the evolution in battery capacity does not keep up with the increasing power requirements of for example processors [6]. This manifests in rapid battery discharge, bothering the users of these devices [2] and limiting their usefulness. This has inspired new processing architectures, such as NVIDIA's Tegra K1 SoC [10], which provides more flexible power management capabilities. For example, the Tegra K1 allows offloading of parallel tasks to a CUDA-capable GPU as well as manual control of Dynamic Voltage and Frequency Scaling (DVFS).

MoVid'15, Portland, Oregon, USA

http://dx.doi.org/xxxxxx/xxxx ...$15.00.

Figure 1: The Jetson-TK1 mobile development board.

Offloading multimedia processing to the GPU has been shown to yield good improvements in terms of both performance and energy consumption, but the challenge of developing for the Tegra K1 SoC lies in implementing and scheduling it in such a way that energy consumption is minimized. Multimedia algorithms that have been studied in the literature include the scale-invariant feature transform (SIFT) [5, 13] and face recognition [3, 15]. However, these articles focus on *batch processing* scenarios: Their goal is to finish a task quickly using as little energy as possible. Our work differ in two main aspects:

1. We consider *continuous* multimedia workloads, where a user applies a pre-processing filter to a raw, live video stream. The requirement of such a workload is not to finish as quickly as possible, but to adhere to cyclic deadlines, e.g. deliver a processed frame once every 40 ms. Our paper explores software- and hardware-configurations for a small set of example workloads. We ask which configurations yield the best energy consumption, and whether a "race to finish" approach is the best.

2. We investigate more deeply the effects that the Tegra K1's power-saving capabilities have on continuous workloads, instead of optimising the workload itself. We focus especially on DVFS. We implement our own, "userspace-PSAV" DVFS algorithm for both the CPU and the GPU to study the frequency scaling's influence on energy efficiency. We compare out scheduler to the existing algorithms in the Linux kernel.

We find that by using our userspace-PSAV DVFS algorithm with the best frequency configuration, energy can be

saved in most cases compared to the standard Linux DVFS algorithms. We find savings of up to 28%, but we find also one case for a light-load scenario where the Linux DVFS algorithm is 2% better. Although we find that CPU-only execution is consistently most energy efficient, we suggest that the gain of GPU offloading depends greatly on the workload's ability to scale with input size on the CPU and GPU architectures.

2. METHODOLOGY

We conduct an empirical investigation based on the Jetson-TK1 [9] mobile development platform. The Jetson-TK1 contains the Tegra K1 SoC, along with several other peripherals. The platform itself does not include a power measurement sensor. Instead, we have equipped it with a custom I2C sensor (see Section 3.1 for details). The Tegra K1 SoC itself logs the power measurements. This introduces a small overhead to the system-under-test, which samples every 3 ms and logs to RAM. Furthermore, to minimise the overhead of the Operating System (OS) we have built a minimal Ubuntu distribution with as few system services as possible.

The tasks we use as workload represent video filtering operations that are typically done on a live video stream. Even though part of our examples may be handled more efficiently by dedicated SoC components, these workloads are meant to represent the arbitrary filters that we expect to see in the future. They simulate a scenario where the Tegra K1 processes arbitrary filters before executing, for example, hardware compression on a SoC dedicated component. The scenario is different from batch-processing jobs, because the task includes idle time and deadlines. A frame must be processed before the deadline marked by arrival of the following frame, but when work on one frame is complete, the Tegra K1 will idle until this deadline.

The Jetson-TK1 has a high, assumably constant power usage which stems from external peripherals, such as USB controllers and the cooling fan. According to NVIDA, this power usage is constant around 2 W [8]. In our power measurements, the exact number is impossible to verify due to the physical layout of the board. We therefore subtract the 2 W idle consumption from each measurement. In addition, each of our test has been run for the same duration. Under the assumption that the external peripherals' power usage is constant, the contribution to the total energy consumption from these components should always be the same in each run.

3. SYSTEM SETUP

Our system setup is comprised of a Jetson-TK1 mobile development kit [9] equipped with a Tegra K1 mobile processor [11] (see Figure 1). In this section, we give a summary of the details of our setup.

3.1 Power Measurement

The Jetson-TK1 is not pre-equipped with any power measurement sensor, but it does provide many GPIO interconnects for devices such as cameras and battery monitor units. Consequently, we equipped the board with an INA219 power sensor [14]. The sensor can provide measurements at a rate of 4000 samples per second over the Tegra K1's I2C bus, using a kernel driver we have developed for this purpose.

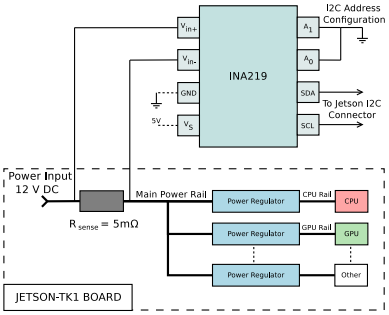

Figure 2: INA219 configuration.

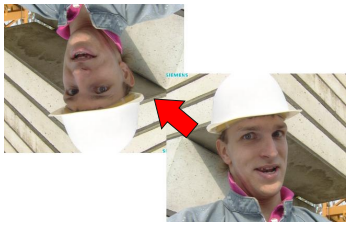

Figure 3: Video stream rotation.

The INA219 is a power rail monitor that works by measuring the voltage drop across a sense resistor. This is the conventional way to measure power [16, 17, 4]. The power measurement circuit used in our setup is shown in Figure 2 and works as follows. The INA219 is continuously measuring the voltage drop $U_{R_{sense}}$ over the power rail sense resistor, R_{sense}. The rail's current draw I_{rail} can be calculated by Ohm's law:

$$I_{rail} = \frac{U_{R_{sense}}}{R_{sense}} \quad (1)$$

The rail's power consumption P_{rail} can be calculated as:

$$P_{rail} = U_{rail} I_{rail} \quad (2)$$

In Equation 2, U_{rail} is the rail voltage which is also continuously measured by the INA219.

3.2 Continuous Workload

Our workload represents continuous video processing operations that are performed on raw video streams. The tests have deadlines and are continuous in the sense that the repeat the same task on subsequent video frames. Deadlines are imposed by the framework, e.g. to achieve a framerate of 25 fps, a frame must be processed within 40 ms of its arrival. We have implemented image rotation, motion vector search and compression as multimedia workloads. To test the effect of GPU offloading, we provide three different execution configurations. In this section, we will explain the tests and how they are run in more detail.

3.2.1 Image Rotation

In the image rotation tests, each frame of a video stream is being rotated by a different (continuously increasing) angle θ (see Figure 3). The algorithm treats the frame as a cartesian coordinate space centered in the middle of the frame. The reference pixel positions (u, v) are calculated by multiplying

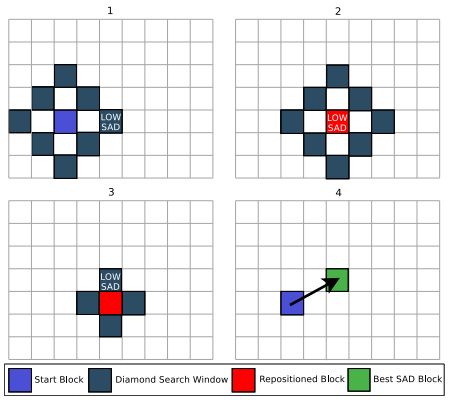

Figure 4: An illusatration of the operation of the diamond search algorithm.

each original pixel coordinate (x, y) by the *rotation matrix* as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} cos-\theta & -sin-\theta \\ sin-\theta & cos-\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (3)$$

Subsequently, each reference pixel at position (u, v) is put at its corresponding frame location $(x, y$.

3.2.2 Motion Vector Search

In the second test, we apply Motion Vector Search (MVS) on the raw video stream. MVS is a common technique in video encoding to reduce the amount of information that has to be stored with each frame. In our case, it works by dividing each frame into a set of macroblocks of 8x8 pixels, and then attempting to estimate each block's displacement (the vector) relative to the previous frame.

We have implemented the Diamond Search (DS) algorithm [18]. DS estimates the displacement of each macroblock by computing the Sum of Absolute Difference (SAD) of the current macroblock and the eight surrounding macroblocks in the previous frame (as shown in Figure 4). At every step, as long as the macroblock with the lowest SAD is not in the center of the "search window", the window will be re-centered at the macroblock with the lowest SAD. After three iterations, the pattern changes to a smaller diamond with only four surrounding macroblocks, where the one block with the lowest SAD is estimated to be correct.

3.2.3 Compression

The third and final test is MJPEG video compression. In this test, the video is compressed by removing high-frequency components from each frame. The image compression algorithm transforms each macroblock of 8x8 pixels into the frequency domain using the discrete cosine transform (DCT):

$$M_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} m_{x,y} cos[\frac{\pi}{8}(x+\frac{1}{2})u] cos[\frac{\pi}{8}(y+\frac{1}{2})v] \quad (4)$$

In Equation 4, $u, v \in [0, 7]$ are the DCT output coordinates, $M_{u,v}$ are the frequency components, $m_{x,y}$ are the original pixel values in the macroblock, and $\alpha(w)$ is a normalising function. After the DCT transformation, the high-frequency components are scaled down by quantisation:

$$B_{x,y} = \frac{M_{x,y}}{Q_{x,y}} \quad (5)$$

where $B_{x,y}$ is the remaining image information after quantisation, and $Q_{x,y}$ is the quantisation matrix, which is constant for each channel.

3.3 GPU Offloading

To investigate the energy- and performance-impact of GPU offloading, we provide three different schemes with varying degrees of GPU offloading. In the first scheme, the GPU is left *idle*, and the CPU processes the Y, U and V frames. In the second scheme, the CPU is left idle, while the GPU processes the Y, U and V frame. In the third scheme, the CPU processes the V frame only, while the GPU processes the Y and U frame. See the following table for a full overview.

Scheme	Offloading		Frequency Settings	
	CPU	GPU	CPU_{min}	GPU_{min}
CPU-ONLY	YUV	Idle	204000 MHz	72000 MHz
GPU-ONLY	Idle	YUV	204000 MHz	72000 MHz
HYBRID	V	YU	1326000 MHz	72000 MHz

3.4 Frequency Scaling Algorithms

DVFS is a method to scale the power, voltage and frequency of processor cores. The relationship is given by the following formula [1]:

$$P_{core} = \alpha C V_{core}^2 f + I_{leak} V_{core} \quad (6)$$

In Equation 6, P_{core} is the processor power usage, V_{core} is the core voltage, f is the core frequency, C is the core switching capacitance, I_{leak} is the transistor leakage and α is the core utilisation. The Linux kernel provides several DVFS algorithms. In this section, we will explain the DVFS algorithms used throughout the experiments in this paper.

3.4.1 Linux DVFS Governors

The Linux kernel is shipped with five standard CPU DVFS algorithms, called governors [12][1]. The five governors are as follows:

- Userspace. Frequency can be set manually by userspace applications.
- Powersave. Frequency is statically scaled to the lowest possible value.
- Performance. Frequency is statically scaled to the highest possible value.
- Ondemand. Adjusts the frequency to the maximum possible value in response to large increases in workload. Lowers the frequency more slowly in response to workload reduction (see Pallipadi and Satrikovskiy [12] for details).
- Conservative. Similar to the ondemand-governor, but reacts slower to changes in current workload.

Of the five governors, the two latter are the most recent developments meant to be used in mobile, energy-constrained systems such as laptops.

[1]For additional background, refer to the Linux kernel documentation.

Algorithm 1 The userspace frequency scaling algorithm. It is used to test the impact of different frequency settings in our experiments.

```
function PROCESS_STREAM(stream, freq, flim_ms)
    deadline_ms = flim_ms
    while yuv_frame = NEXT_FRAME(stream) do
      START_MEASURE_TIME( )
      rawy, rawu, rawv = GET_RAW_COMP(yuv_frame)
      SET_CPU_SCALING(freq->cpu_max)
      SET_GPU_SCALING(freq->gpu_max)
      gpu_thread = START(gpu_scale_thread, [rawy, rawu])
      cpu_thread = START(cpu_scale_thread, [rawv])
      JOIN(gpu_thread)
      JOIN(cpu_thread)
      elapsed_ms = STOP_MEASURE_TIME( )
        if elapsed_ms < deadline_ms then
            remaining_ms = deadline_ms - elapsed_ms
            SLEEP_MS(remaining_ms)
            deadline_ms = flim_ms
        else
             deadline_ms -= (elapsed_ms - flim_ms)
        end if
    end while
end function
function GPU_SCALE_THREAD(yuv_frame[] frames)
      PROCESS_GPU(frames)
      SET_GPU_SCALING(freq->gpu_min)
end function
function CPU_SCALE_THREAD(yuv_frame[] frames)
      PROCESS_CPU(frames)
      SET_CPU_SCALING(freq->cpu_min)
end function
```

The Tegra K1's GPU is also under DVFS control, but there is just one governor that can be enabled or disabled (to allow userspace control). We have been unsuccesful in finding documentation for the governor itself.

3.4.2 *Userspace-PSAV*

Our own power saving implementation can be seen in Algorithm 1. This algorithm is used to test the impact of frequency scaling using the workload described above. The algorithm takes five main parameters. These are the maximum and minimum frequency of the CPU and the GPU, as well as the per-frame deadline. The algorithm works as follows:

1. As long as either the CPU or the GPU are busy processing, their respective frequency is scaled to *max*, a user-specified parameter.
2. When either the CPU or GPU runs out of work, their respective frequency is lowered to *min*, a user-specified parameter.
3. If the processing ends before the frame deadline (e.g. 40 ms for a framerate of 25 FPS) the CPU sleeps for the remaining time.
4. If the processing ends after the frame deadline, the deadline for the next frame is lowered accordingly.

4. EXPERIMENTAL RESULTS

In our experiments, we run the multimedia workloads described in Section 3.2. The goal is to reach a target framerate of 25 FPS using as little energy as possible. We first try to find the most energy efficient frequency settings using the Userspace-PSAV algorithm (see Algorithm 1), and compare with the four standard frequency scaling governors. We use three different video resolutions under the three execution configurations CPU-only, GPU-only and hybrid. The achieved energy consumption in each test represents the average of ten runs. The results from the frequency scaling experiments is shown in Figure 5 (see also Table 1). In most cases, more optimal frequency settings could be achieved using our Userspace-PSAV algorithm over the standard linux governors, by an amount up to 28 %.

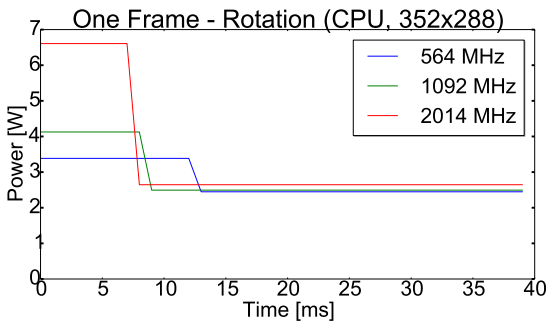

Figure 6: Power usage during a single frame encoding interval. The slopes in the three curves corresponds to points where the CPU is done processing, and goes from 100 to about 0 % utilisation.

4.1 Execution Configuration

In most of our tests, the CPU configuration is the best. The only exception is the rotation tests, where the GPU is most energy efficient for the highest input resolution. This is because the CPU and the hybrid configurations doesn't reach the 25 FPS requirement, and as such the GPU version does not receive any competition for this image resolution. In other words, the GPU version scales better to higher input sizes. However, the most energy efficient offloading choice does not depend solely on the workload's ability to scale, but also electrical characteristics tied to the target processor (such as the model constants in Equation 6.

4.2 Frequency Settings

When it comes to the most optimal frequency settings for the CPU and the GPU, it is clear that the best way to conserve power is not a "race-to-finish" strategy. For the CPU, the most optimal frequency configuration is *the minimum frequency setting that meets the acceptable framerate*. In other words, the sleeping period between each frame should be minimised. There may be several reasons for this. One is *power regulator loss*, which can grow quadratically with the CPU's current consumption [7]. Another is excessive system overhead, where the CPU stays at less than 100% utilisation in high frequency states. The third is that a doubling in frequency does not necessarily mean a reduction in runtime by half, which can easily be seen in Figure 6. Here, the reductions in power usage corresponds to the time where the CPU is done processing a frame. A doubling in frequency from 1 to 2 GHz yields roughly only a runtime improvement of around 10 %, but the power usage doubles. The CPU power usage increases linearly with frequency, but the runtime is

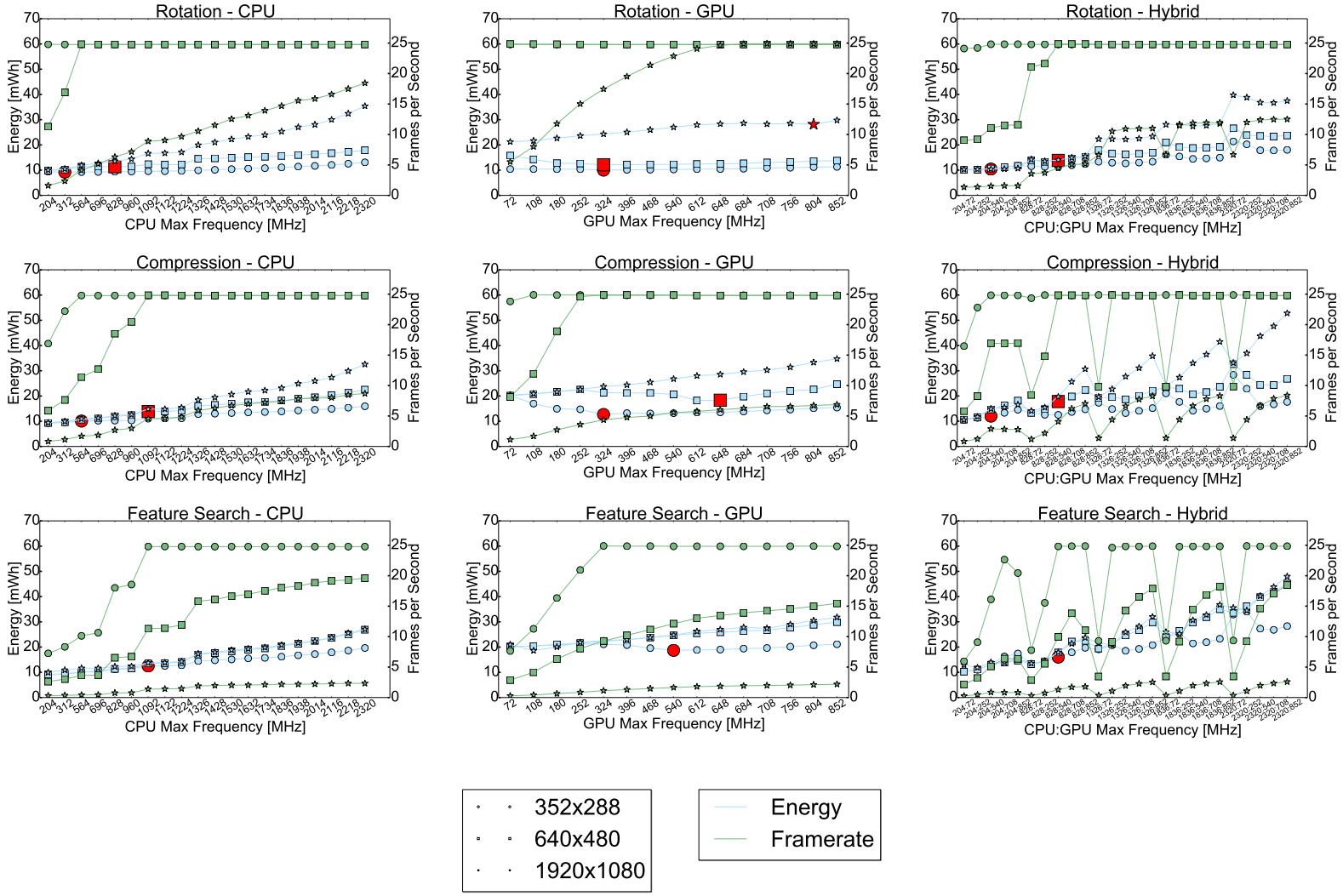

Figure 5: Frequency scaling tests (Userspace-PSAV). The most energy efficient configurations, that reached a framerate requirement of 25 FPS, are marked with red.

Benchmark	Resolution	Userspace P-SAV			Linux Governor			Improvement
		Core	Max Frequency	Energy	Governor	Core	Energy	
Rotation	352x288	CPU	312 MHz	9.22 mWh	Ondemand	CPU	9.02 mWh	-2.2 %
	640x480	CPU	828 MHz	11.33 mWh	Ondemand	CPU	12.86 mWh	11.9 %
	1920x1080	GPU	468 MHz	26.00 mWh	Ondemand	GPU	29.16 mWh	10.8%
Compression	352x288	CPU	564 MHz	9.31 mWh	Ondemand	CPU	12.93 mWh	28.0 %
	640x480	CPU	1092 MHz	13.82 mWh	Conservative	CPU	19.05 mWh	27.4 %
	1920x1080	-	-	-	-	-	-	-
MVS	352x288	CPU	1092 MHz	12.49 mWh	Ondemand	CPU	13.59 mWh	8.1 %
	640x480	-	-	-	-	-		
	1920x1080	-	-	-	-	-		

Table 1: The most energy efficient frequency configurations under the different DVFS algorithms (Userpace P-SAV and standard Linux governors).

not reduced by a corresponding amount.

The GPU shows a similar trend, but it is not as clear as for the CPU. The most energy efficient frequency can generally be found a little higher up the range, instead of the minimum that meets the FPS requirement. Additionally, once the 25 FPS requirement is met, the energy consumption *flattens out*, rather than increasing. We argue that the heuristic above is still valid for the GPU, but the gain is smaller, and we have not been able to give any good indication of the reasons behind this trend.

5. CONCLUSION

In this paper, we have attempted to minimise the energy consumption of continuous multimedia processing. We have taken an empirical approach by measuring the total platform power of the Jetson-TK1 development kit while testing the effects of GPU offloading, DVFS algorithms and frequency settings. Our major finding is that "race-to-finish" approaches, where the CPU is kept at the maximum frequency settings to finish work and sleep as fast as possible, are inefficient for continuous multimedia workloads. The standard Linux frequency scaling algorithms can be outperformed by up to 28 % by minimising the CPU and GPU frequency, such that the application deadlines are met.

6. REFERENCES

[1] A. Castagnetti, C. Belleudy, S. Bilavarn, and M. Auguin. Power consumption modeling for DVFS exploitation. In *Proc. of Euromicro DSD-AMT*, pages 579–586. IEEE, 2010.

[2] CAT Phones. New research reveals mobile users want phones to have a longer than average battery life. Press Release, November 2013. http://www.phonearena.com/news/Survey-shows-battery-life-to-be-the-single-main-gripe-of-todays-mobile-phone-user_id49818.

[3] K.-T. Cheng and Y.-C. Wang. Using mobile GPU for general-purpose computing–a case study of face recognition on smartphones. In *International Symposium on VLSI-DAT*, pages 1–4. IEEE, 2011.

[4] M. Dong and L. Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proc. of MobiSys 2011*, pages 335–348. ACM, 2011.

[5] M. Huang and C. Lai. Accelerating applications using gpus on embedded systems and mobile devices. *Proc. of IEEE HPCC*, 2013.

[6] K. Lahiri, S. Dey, D. Panigrahi, and A. Raghunathan. Battery-driven system design: A new frontier in low power design. In *Proc. of ASP-DAC 2002*, page 261. IEEE, 2002.

[7] W. Lee, Y. Wang, D. Shin, N. Chang, and M. Pedram. Power conversion efficiency characterization and optimization for smartphones. In *Proc. of ISLPED'12*, pages 103–108. ACM, 2012.

[8] NVIDIA. Jetson-tk1 technical brief. Web. `http://developer.download.nvidia.com/embedded/jetson/TK1/docs/Jetson_platform_brief_May2014.pdf`.

[9] NVIDIA. Jetson-TK1 Embedded Development Platform. `http://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html`, 2014.

[10] NVIDIA. NVIDIA Tegra K1 - A New Era in Mobile Computing. Technical report, 2014.

[11] NVIDIA. Tegra K1 Next-Get Mobile Processor. `http://www.nvidia.com/object/tegra-k1-processor.html`, 2014.

[12] V. Pallipadi and A. Starikovskiy. The ondemand governor. In *Proc. of the Linux Symposium*, volume 2, pages 215–230. sn, 2006.

[13] B. Rister, G. Wang, M. Wu, and J. R. Cavallaro. A fast and efficient sift detector using the mobile gpu. In *Proc. of IEEE ICASSP*, pages 2674–2678. IEEE, 2013.

[14] Texas Instruments. INA219 Power Rail Monitor. `http://www.ti.com/product/ina219`, 2014.

[15] Y.-C. Wang and K.-T. Cheng. Energy-optimized mapping of application to smartphone platform—a case study of mobile face recognition. In *Proc. of IEEE CVPRW*, pages 84–89. IEEE, 2011.

[16] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. A system-level model for runtime power estimation on mobile devices. In *Proc. of GREENCOM-CPSCOM'10*, pages 27–34. IEEE, 2010.

[17] Y. Xiao, Y. Cui, P. Savolainen, M. Siekkinen, A. Wang, L. Yang, A. Ylä-Jääski, and S. Tarkoma. Modeling energy consumption of data transmission over wi-fi. *IEEE Trans. on Mobile Computing*, 13(8):1760–1773, 2013.

[18] S. Zhu and K.-K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Trans. on Image Processing*, 9(2):287–290, 2000.