Why Race-to-Finish is Energy-Inefficient for Continuous Multimedia Workloads

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

***Abstract*—It is often believed that a "race-to-finish" approach, where processing is finished quickly, is the best way to conserve energy on modern mobile architectures. However, from earlier work we know that for continuous multimedia workloads, the best way to conserve energy is to minimise processor frequency such that application deadlines are met. In this paper, we investigate the reasons behind this. We develop an original method to model dynamic and static power on individual power rails of the Tegra K1 by only measuring the total power usage of the board. Our model has an average error of only 8 %. We find that the way an application scales performance with frequency is very important for energy efficiency. We demonstrate a 37 % energy saving by minimising processor and memory frequency of a video processing filter such that a framerate of 20 FPS is met.**

I. Introduction

Modern mobile architectures such as NVIDIA's Tegra K1 SoC [5] have impressive power management capabilities. The Tegra K1 includes among other components a Low-Power (LP) core and a High-Performance (HP) quad-core application cluster. The operating system and applications can be hardware-migrated between the HP cluster and the LP core, the HP cluster cores can be turned on and off, and the processor and memory frequencies can be dynamically adjusted to meet an application's demands. A challenge for developers of such heterogeneous architectures is to understand the power usage of the platform because existing models are too simple and model the device as one unit, making it hard to optimise the energy usage of applications.

A particularly challenging type of workload is continuous multimedia processing, which must generate results according to a sequence of deadlines. For example, in a scenario where video is being recorded on a mobile phone, multiple filters are used for image stabilisation, debarreling, horizon detection, feature extraction, sharpening and finally encoding in order to produce the final video. These must be able to process incoming video frames at a certain framerate while consuming as little energy as possible. In our preliminary studies [10], we observed that for such workloads, energy can be saved by minimising CPU frequency such that application deadlines are still met. This contradicts the popular belief that a *race-to-finish* approach, where CPU frequency is maxed out until the processing is done, is the most energy-efficient alternative. Standard Linux frequency scaling algorithms are also easily outperformed following this heuristic.

In this paper, we investigate the reasons for this, i.e., where and how energy is lost under computation on the Tegra K1. We develop an original method to model individual static and dynamic power of different hardware blocks of the Tegra K1 based on extensive measurements of different parts of the heterogeneous system. This is challenging because it is impossible to install power usage sensors in series with the power rails. In this respect, we are the first to provide a method to model and quantify static and dynamic power on individual power rails (the HP cluster, LP core and memory rails) for the Tegra K1 by only measuring the total power usage of the board. Our experiments show that, for example, the HP cluster with one core active adds about 0.4 A to the leakage current on the HP rail. Each additional core adds between 0.15 and 0.20 A. This analysis is useful to understand static power loss which is always present, independently of the workload. Dynamic processor and memory power is workload-specific and must be re-modelled for each workload. Our model has an average error of 8 %.

We then implement a set of continuous multimedia workloads to study how processor and memory frequency impacts energy efficiency. We find that the way our workloads scale performance with memory and processor frequency is a key aspect to energy efficency. Dynamic power alone grows at least linearly with frequency, while application performance typically grows sublinearly. In practice, this means that a lot more power is needed for a marginal increase in performance, and performance per watt decreases. We demonstrate a 37 % energy saving for one of our workloads by minimising processor and memory frequency, and choosing the best number of cores active, such that a target framerate of 20 FPS is still met. This translates to a 37 % increased battery lifetime in a continuous video recording scenario.

II. Related Work

There are several works for Tegra SoCs and other embedded mobile platforms that study performance and power usage of different applications. Many of these consider the gain in terms of energy efficiency and performance of offloading computationally expensive tasks to different types of application processors, such as GPUs and DSPs. Wang et. al. [13] consider common image processing operations such as the fast fourier transform and matrix multiplication on the Tegra 2, Snapdragon S2 and OMAP platforms. Power usage and performance have been evaluated on the CPU, GPU and DSP, or a combination of these. Rister et. al. [8] optimise the scale-invariant feature transform by partitioning the workload between the CPU and the GPU of a Tegra 250. However, these studies have the limitation that, while they are attempting to investigate power-performance tradeoffs of applications using different processors, they do not delve deeply into the physical

aspects of modern heterogeneous processors and electrical components.

Castagnetti et. al. [1] investigate the power usage impact of voltage regulators and static and dynamic processor power on an Intel XScale mobile processor. They consider processor frequency and rail voltage and their relation to power usage. Dynamic and static power is modelled mathematically, as similarly proposed by Kim et. al. [3]. Both works are similar to ours, but we also model dynamic memory power. Pricopi et. al. [6] propose a performance-counter based model for power usage of the big.LITTLE architecture, as well as a cycles-per-instruction based performance model for applications. The same authors also propose a power management scheme [9] based on the same observations as we made in earlier work [10]. To save power, processor frequency should be minimised while meeting QoS requirements, such as a specific framerate. Compared to existing work, we first take a more fine-grained and quantitative approach to understanding power usage of the heterogeneous Tegra K1 by modelling power on individual power rails, and then, we investigate the effects that make workloads energy-inefficient.

III. System

A. Hardware Architecture

Insight into the relevant parts of the Jetson-TK1's hardware architecture is necessary for the methodology and results derived in later sections. The Jetson-TK1 is a development kit with an integrated NVIDIA Tegra K1 SoC and various supporting infrastructure. This includes different IO components such as HDMI and USB controllers, embedded buses, memory, cooling, a power management controller and other components. Because we focus on the CPU clusters and the memory controller, only the Tegra K1 SoC and certain details about the Jetson-TK1's power regulators affect our investigation.

The Tegra K1 consists of 20 power rails [4] which supply the different functional blocks of the SoC with power. Most of these are powered down. Figure 1 shows the most important rails, because they power the components that are utilised by our workloads. Only the power usage on those rails will vary.

- The *core rail* powers 40 clocks (most of which are idle), the LP core and additional shared circuitry between the LP and the HP cluster.
- The *HP rail* powers only the HP cluster and its clock.
- The *memory rail* powers the memory module and the memory clock.

The clocks drive the different co-processors, memory and buses. Higher clock frequency increases performance and power usage of the connected component. An important side-effect of clocking is that rail voltage increases with clock frequency, which also increases static power usage. For example, as processor frequency is increased, higher input voltages on the respective rails are needed to sustain the current throughput.

Table I shows a subset of the Tegra K1 clocks that are powered-on as well as their frequency and voltage ranges.

Clock	Rail	Description	Frequency		Voltage Range
			Steps	Range $[MHz]$	
cpu_g	HP Rail	HP cluster	20	$[204, 2320]$	$[0.80, 1.20]$
cpu_lp	Core Rail	LP core	9	$[51, 1092]$	$[0.80, 1.05]$
emc	Core Rail	Memory	9	$[40, 924]$	$[0.80, 1.01]$
pciex	Core Rail	PCIe	1	250	$[0.85]$
mselect	Core Rail	Crossbar	1	204	$[0.90]$
sbus	Core Rail	Unknown	1	204	$[0.85]$
host1x	Core Rail	Unknown	1	81	$[0.80]$

TABLE I: The Tegra K1 clocks, voltage and frequency ranges.

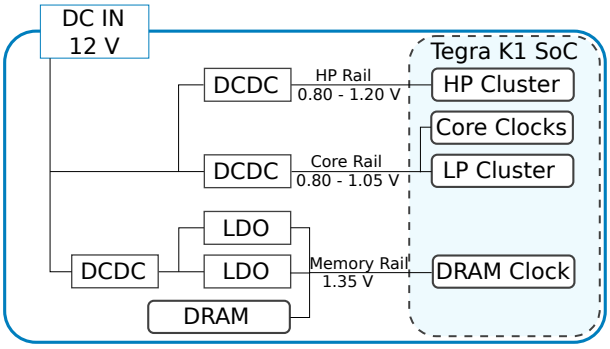


Fig. 1: Critical components of the Jetson-TK1 architecture.

Only the processor and memory clocks can vary; all other clocks are restricted to one frequency. The HP rail voltage is only governed by the HP core clock (cpu_g). However, the core rail drives more components, and therefore the core rail voltage is the maximum voltage required by any of its clocks at any point in time. Because the only variadic clocks on the core rail are the cpu_lp and emc clocks, and the mselect clock is set statically, requiring 0.9 V, it is the maximum voltage required by these that decides the actual core rail voltage (see Figure 6b). Each rail is powered by at least one regulator.

B. Power Measurement and Synchronisation

The Jetson-TK1 is not fabricated with any power measurement sensors. In previous work [10], we used a low-cost power measurement sensor attached to the main power rail. This approach had the disadvantage that the Tegra K1 had to poll the sensor for readings. To avoid this, we use a Keithley 2280S power source. In addition to supplying the Jetson-TK1 with power, the 2280S continuously measures output current with an accuracy of 0.05 % [2] and a configured sampling rate of 1 kHz. It also has a small internal circular buffer to store readings which can be queried over USB. Our experimental setup can be seen in Figure 2. We use an external logger machine to store readings. The Tegra K1 can start, stop and retrieve power measurements directly from the logger machine.

A challenge with this setup is that the measurements are not synchronised with the Tegra K1. For example, when initiating power measurements from the Tegra K1, up to 200 ms delay until the measurements start on the 2280S can be expected. The delay occurs as an effect of latency between the Tegra K1 and the logger machine, as well as between the logger and the 2280S. The effect can be seen in Figure 3. We therefore use a method suggested by Rice and Hay [7] to synchronise the measurements and the Tegra K1. After initiating measurements, the Tegra K1 cycles the platform between a low-power and a high-power state in set intervals of 500 ms. This creates

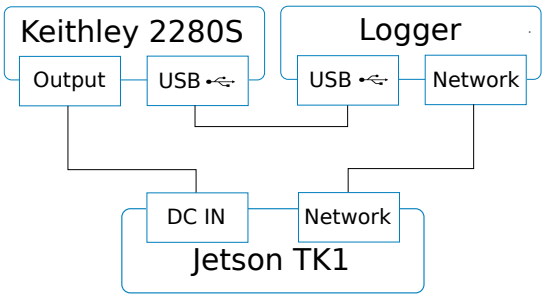


Fig. 2: Measurement Setup.

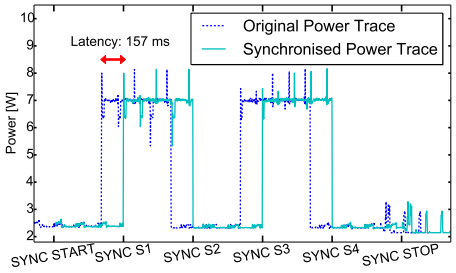


Fig. 3: A plot of a single synchronisation trace.

a power signature in the measurement trace as can be seen in Figure 3, which is used to calculate and compensate for the latency.

C. Workloads

Our workloads are continuous video processing operations that are performed on raw video streams stored in the YUV format. The operations are continuous and have per-frame deadlines in the sense that the repeat the same task on subsequent video frames. To achieve a framerate of 20 FPS, a frame must be processed within 50 ms of its arrival. Frames are read directly from RAM to avoid power usage due to disk activity. We have implemented image rotation, debarreling and the Discrete Cosine Transform (DCT) as workloads. In this section, we explain the operations in more detail.

1) Debarreling: Barrel distortion is an effect that occurs with wide [11]. Our "debarreling" workload computes a constant debarreling map which only needs to be calculated once and is subsequently applied to each frame. The debarreling filter is the least compute-intensive filter we consider.

2) Image Rotation: In the image rotation tests, each frame of a video stream is being rotated by a continuously increasing angle. This emulates the operation of video stabilisers. The pixel shifts need to be recalculated for each frame, which makes this filter more compute-intensive than debarreling.

3) DCT: While DCT in itself is not a useful video processing filter, it is a recurring part of others, for example compression. Our workload partitions each frame into macroblocks of 8x8 pixels and performs a naive 2D transformation.

IV. Methodology and Background

Building an accurate power model that separates the power usage of the Tegra K1's CPU clusters and memory is not trivial, as it is impossible to install power measurement sensors in series with the power rails. It is only possible to measure the total power usage of the Jetson-TK1. In this section, we outline the fundamental power models and summarise our methodology used to build the power models.

Processor and memory power usage can be divided into static and dynamic power [3], [12]. These are caused by leakage current and switching activity within the processor's or memory's transistors, respectively. An estimate of these power components for an idle system can be seen in Figure 4. The Jetson-TK1 has a base power usage caused by idle components (USB controllers, bus adapters etc.). The total power usage can be expressed by the following formula:

$$P_{jetson} = P_{cpu,dyn} + P_{hp,stat} + P_{core,stat} + P_{mem,dyn} + P_{base} \quad (1)$$

where $P_{cpu,dyn}$ is dynamic processor power (either on the core or HP rail, depending on the active processor), $P_{hp,stat}$ and $P_{core,stat}$ is static power on the core and HP rails, $P_{mem,dyn}$ is dynamic memory power and P_{base} is the power usage of all other components and rails, also including static power on the memory rail. It is important to note that the HP rail is powered down whenever processing is restricted to the LP core (i.e. $P_{hp,stat} = 0$).

A. Estimating Dynamic Power of Processor and Memory

Dynamic power for the processor or memory is modelled as follows [3], [12]:

$$P_{dyn} = \alpha C V_{rail}^2 f \quad (2)$$

where V_{rail} is the rail voltage and f is the frequency of either the processor or memory. The switching capacitance C and the switching activity α are unknown variables. C can be viewed as the potential maximum electric charge to be switched into the processor or memory per cycle, and has units of coloumbs per cycle. $alpha \in [0, 1]$ is workload-specific, and indicates (on average) how much of the maximum switching capacitance C is being switched through the circuitry at every cycle. We call αC the Dynamic Power Coefficient (DPC) of the processor or memory.

In principle, our methodology to find the DPC is based on the observation that increasing processor or memory frequency does not always increase the rail voltage V_{rail}, and therefore, regression can be used to estimate the DPC. For example, for the HP rail, the rail voltage is approximately 0.81 V between 204 to 1224 MHz (see Figure 5a and 6). In this interval, we

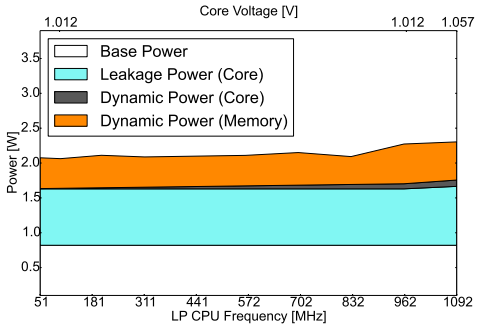
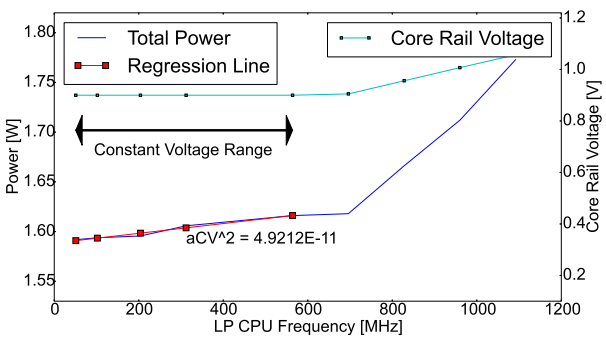
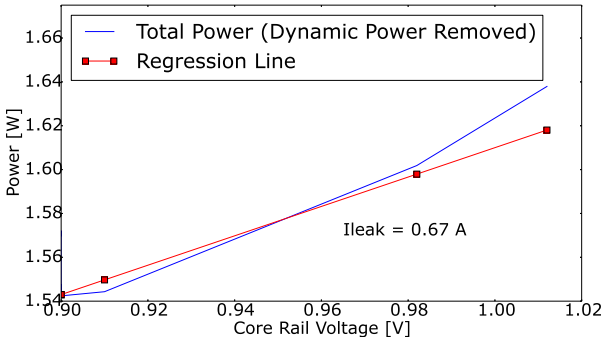


Fig. 4: Breakdown of different power components running the LP core (Tegra K1 idle, and HP rail off).

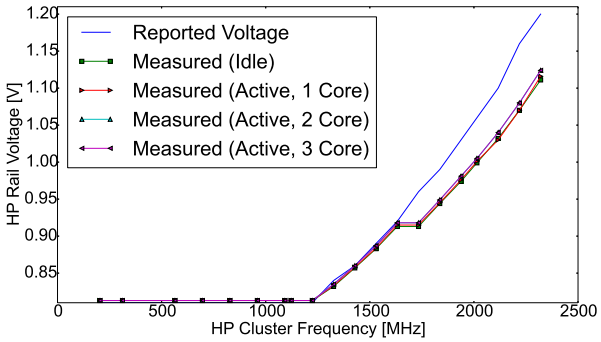


(a)

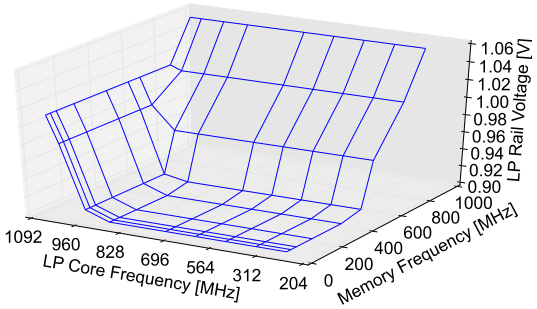


(b)

Fig. 5: Estimating DPC (a) and leakage current (b) using regression.



(a) For HP rail, including driver-reported value.



(b) For core rail.

Fig. 6: Measured rail voltages.

also see that the power usage grows linearly with frequency, which is in accordance with Equation 2. $R = \alpha C V_{rail}^2$ can therefore be estimated by applying single-variable linear regression over the slope where the rail voltage V_{rail} is stable.

The DPC αC can then be found by dividing the regression coefficient R over the rail voltage squared ($\alpha C = \frac{R}{V_{rail}^2}$). The DPC varies depending on workload and core configuration. For example, each workload utilises the hardware (α) in slightly different ways, also depending on the cluster and number of cores that are active. Therefore, the DPCs must be re-estimated for each workload and core configuration.

Our method to estimate the DPC so far ignores the fact that increasing processor frequency can affect memory hardware utilisation α_{mem}, and vice versa:

- If processor frequency increases while executing a workload, memory utilisation (and memory dynamic power) can also increase.
- If memory frequency increases, processor utilisation (and processor dynamic power) can also increase, for example if the processor is stalling, waiting for memory requests to finish.

If these effects are not taken into consideration, an estimated DPC may become too large (overfitting). In our initial experiments, we noticed that this happened when estimating processor DPC. Therefore, our workloads attempt to hide the effects of memory latency with multithreading, so that the rise in processor utilisation as memory frequency increases is negligible. The memory DPC can be estimated and dynamic memory power removed prior to estimating the processor DPC (see Section VI-A for more details).

B. Static and Base Power

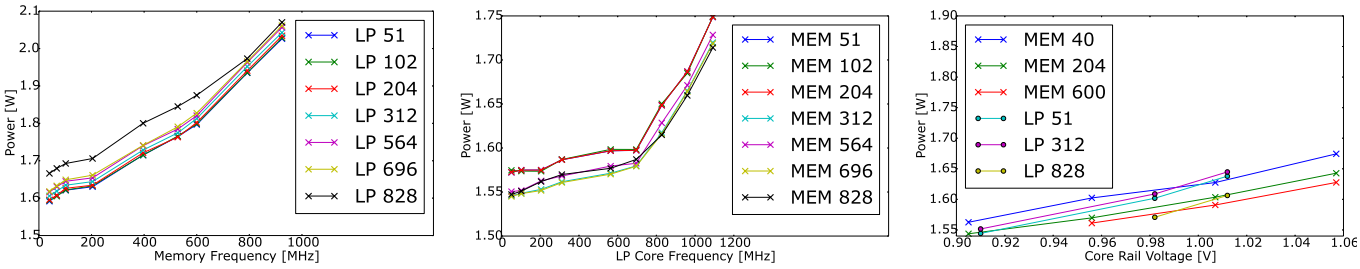
Static power P_{stat} on a rail is always present as long as the rail is powered. Static power can be described as [3]:

$$P_{stat} = I_{leak} V_{rail} \tag{3}$$

where I_{leak} is the leakage current on a power rail. Ignoring temperature effects, the leakage current is always constant and does not vary with workload. However, it varies when circuitry is power gated. For example, a workload can be restricted to the LP core, or running on the HP cluster, where it is possible to use up to four cores. Turning off an HP core effectively removes some leakage current, as that circuitry is power gated in hardware. Restricting processing to the LP core effectively disables the HP rail entirely. Therefore it is only necessary to estimate leakage current once for each core configuration, and not for each workload.

Leakage current can be estimated by observing the change in total power usage as the rail's voltage increases. However, dynamic (memory and processor) power must first be estimated and removed. A simplified example is shown in Figure 5b, where I_{leak} can be directly estimated by applying single-variable linear regression to the slope where the rail voltage V_{rail} is increasing.

Base power is found by subtracting the estimated dynamic and static power for the processor and memory from the total



(a) Over memory frequencies.

(b) Over LP core frequencies. Dynamic memory power has been removed.

(c) Over core rail voltage. All dynamic power has been removed.

Fig. 7: Idle power usage.

		Core Configuration				
		LP	HP-1	HP-2	HP-3	HP-4
Leakage Currents $[A]$	Core Rail	0.78	0.76	0.82	0.78	0.80
	HP Rail	0.0	0.38	0.54	0.68	0.80
Base Power $[W]$		0.82				

TABLE II: Estimated leakage currents and base power over the five different core configurations.

power usage. Estimation of leakage current on the memory rail is impossible, because the rail voltage is always 1.35 V and does not change with frequency. It is instead an implicit part of the base power.

V. Estimation of Leakage Currents and Base Power

In this section, we describe our experiments to estimate leakage currents and base power on the Tegra K1 using the methodology presented in Section IV. As already mentioned, leakage current estimation must be done for each core configuration (LP or HP cluster with up to four cores) because of power-gating, but it must only be done once because it is independent of the workload. The final leakage currents can be seen in Table II.

In the following experiments, we discovered that there is a discrepancy between driver-reported rail voltages and those measured with a voltmeter. This happens on both rails supplied with a DCDC regulator (core and HP rails, see Figure 6a and 6b). The best estimates for the leakage current are achieved by using voltage measurements on the rail.

A. Collecting Power Usage Data

We first have to collect power usage data for all possible combinations of processor and memory frequencies, in each of the five core configurations. There are a total of nine LP core and memory frequencies, as well as twenty HP frequencies (see Table I). We found that collecting this data when the Tegra K1 is idle achieves the best estimations. At each step, we let the Tegra K1 idle for five seconds, log the power usage over that time, and proceed to the next frequency combination.

B. Estimating Dynamic Memory and LP Core Power

Figure 7a shows the total idle power versus the memory frequency. The lines in the plot represent different LP core frequencies. The steeper climb of the curves at higher frequencies is due to increased static power, as voltage on the core rail is increased at these frequencies. For frequencies below this point, voltage is stable, and the memory DPC ($\alpha_{mem}C$) can be estimated, noting the following points:

- Between the third and the fourth memory frequencies of Figure 7a (102 MHz and 204 MHz), the increase in power is less than for the other frequencies.
- At memory frequencies above this, several driver-reported but undocumented clocks are automatically activated on the core rail, having a negative impact on the estimation.

Due to the different growth in power over memory frequency, we do regression over the three lowest memory frequencies (40 to 102 MHz), and then the next three (204 to 600 MHz) according to our methodology in Section IV-A. Figure 8 shows the estimated memory DPCs, which are plotted over the LP core frequencies.

Figure 7b shows the remaining total power over LP core frequencies when dynamic memory power has been removed. We see that the power grows linearly with LP core frequency until the rail voltage starts to increase. We repeated the process above to estimate the LP core DPC over different memory frequencies. The result can be seen in Figure 8, where the remaining power is plotted over memory frequencies. With these results, dynamic LP core power can be estimated and removed for all frequency combinations.

In Figure 8, we see that the DPC is approximately constant. Increasing processor frequency for an idle system does not increase memory utilisation, and vice versa. This may seem illogical because, even for an idle system with no workloads, there is still overhead in the running kernel. We believe this is due to caching. All data in memory operated on by the kernel

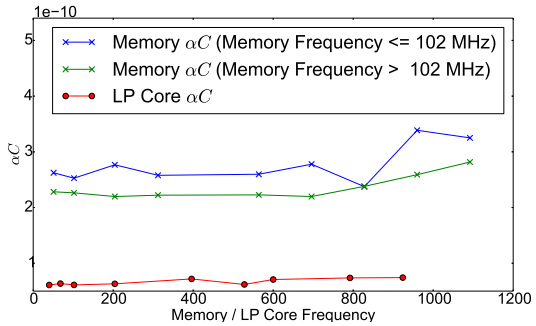
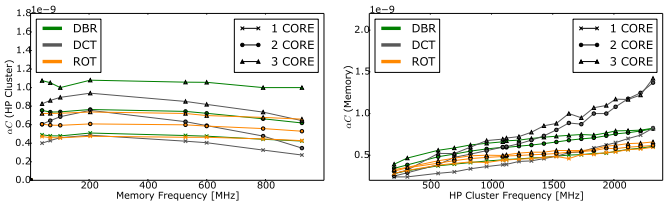


Fig. 8: DPC for Memory and LP Core (idle workload).



(a) Over memory frequencies. (b) Over HP cluster frequencies.

Fig. 9: Estimated DPC (αC) for processor and memory.

has most likely been cached in the processor's internal L1 and L2 caches, effectively making memory traffic negligible.

C. Estimating Leakage Current

When all dynamic power (memory and processor) has been removed, the leakage current can be estimated. In Figure 7c we show the remaining power as a function of the core rail voltage. We use regression over the curves to estimate the leakage current. We derive a leakage current on the LP rail of 0.78 A, with a standard deviation of 0.12 A. The standard deviation is high because the estimated leakage current over the memory lines in Figure 7c is closer to 0.9 A, while for the LP core lines, it is closer to 0.7 A. However, we believe that this estimate is reasonable, as the approximations for the core rail leakage current in the other core configurations are nearly the same.

D. Leakage Current on the HP Rail

The leakage current is also modelled on the HP rail, which requires that processing is restricted to the HP cluster. Dynamic memory and processor power is removed as in the previous section, but the rail leakage currents are estimated differently. The difference is that when HP cluster frequency is increased, the voltage of the *HP rail* increases. In practice this means that *only* the "memory" lines in Figure 7c must be used to estimate HP rail leakage, while the others must be used to estimate the core rail leakage. The resulting leakages can be seen in Table II.

VI. Energy Usage of Video Processing Filters

A. Estimating Dynamic Power of Workloads

We follow our methodology described in Section IV to estimate dynamic power for our workloads. Each workload is run over all possible frequency combinations, while logging average power usage for each run. We then estimate the DPCs for the HP cluster and memory (see Figure 9a and 9b). We see that the memory DPC increases with processor frequency, because the memory access rate also increases. Processor DPC, however, remains almost constant over the range of memory frequencies. Even when the memory frequency is low, memory latency is hidden by computation because of the multithreaded benchmarks, and so the processor utilisation does not go down.

B. Power Model Verification

To verify our model, we use Equation 1 and the estimated dynamic power coefficients, leakage currents and base power to predict total power. The result can be seen in Figure 10, where measured power (top row) and model error (bottom row) have been plotted over all frequency combinations. Due to space restrictions, we do not show the results for all benchmarks[1].

Studying Figure 10 we see that our prediction generally follows the measured total power. This is also true for the other benchmarks (rotation and DCT). Our model has an error over all frequency combinations of at most 13.4 %, but in most cases between 6-8 % (see Table III). A characteristic of our model is that it consistently overpredicts at least 0.15 W. We believe this is due to an overestimation of core rail leakage current because the LP core is off when the HP cluster is active. Looking at Table II, we see that each HP core (which is the same type as the LP core) adds between 0.15 to 0.20 A to the leakage current, translating to about 0.15 W.

C. Energy Efficiency

From earlier work [10], we know that energy can be saved compared to the standard Linux frequency scaling algorithms by minimising processor frequency so that performance requirements are met. We now add another dimension to the problem and consider memory frequency as well. We have already identified leakage currents as a source of energy inefficieny, because the power loss due to leakage increases with rail voltage at higher operating frequencies.

Figure 11 illustrates the Frame Processing Time (FPT) over all possible memory and processor frequencies for each benchmark (3 cores). In a live streaming scenario, a simple requirement is that frames are processed at a certain rate. We choose 20 FPS, which gives an FPT budget of 50 ms per frame. The frequency combinations that achieve this are marked with green, indicating a possible area of operation. This area is largest for the DCT benchmark, followed by debarreling and rotation. The reason for the differently sized areas of operation is that the benchmarks scale differently with operating frequencies. The DCT benchmark is for example highly optimised with SIMD instructions, while the rotation benchmark underutilises threads and recalculates pixel shifts for each frame.

We see that the decrease in FPT (Figure 9a) is highest for low processor and memory frequencies. Expressed differently, more performance is gained per increase in processor and memory frequency at low frequencies. From the experimental data, we can see that the performance is at best growing linearly with frequency. At high frequencies, the slope flattens out, and performance increases sublinearly. This is another source of energy inefficiency. Even when not considering power loss due to leakage currents, a doubling in processor frequency will at least double the dynamic power, but FPT is not necessarily halved in return.

D. Race to Finish Versus Frequency Minimisation

To see the difference between race-to-finish and frequency minimisation, we run our workloads on the minimum processor and memory frequency combination that satisfy the FPS requirement (the "optimal" strategy). We compare with two race-to-finish strategies, RTF-HP and RTF-LP. The results are

[1]All experimental data is available for download from http://folk.uio.no/krisrst/papers/mcsoc/experiments.ods or on request.

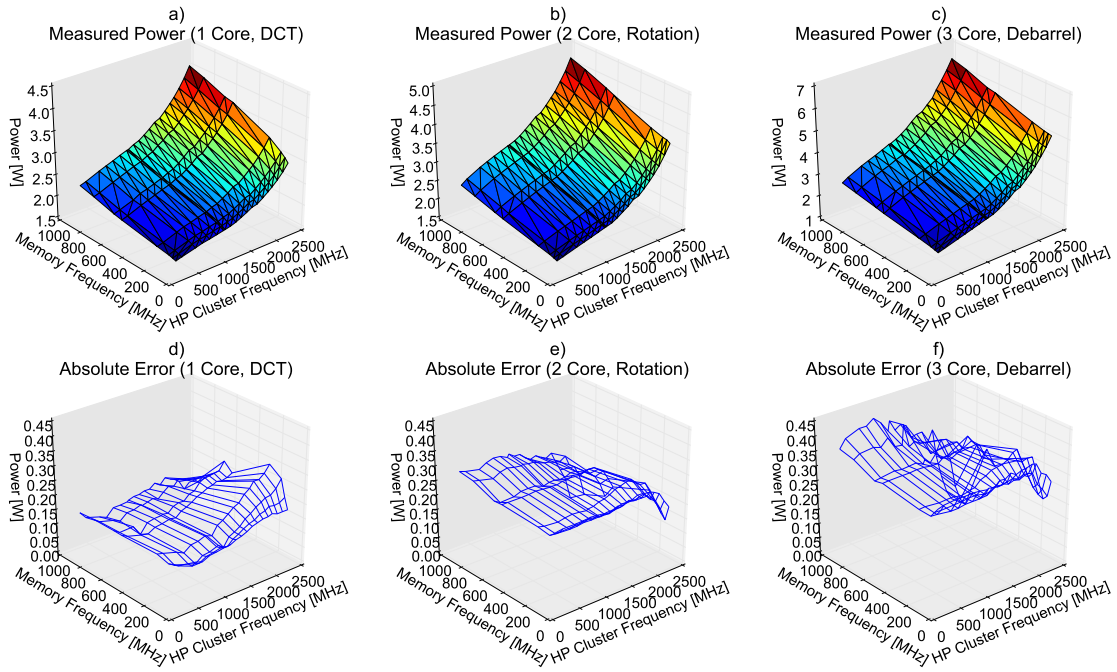


Fig. 10: Comparison of measured (top row) versus model error for some of the filters.

Benchmark		Debarrel			Rotation			DCT		
Cores		1	2	3	1	2	3	1	2	3
Minimum FPT $[ms]$		77.4	42.9	33.1	64.3	50.6	45.1	20.7	11.2	0.94
Model Error $[\%]$		6.1	7.0	9.6	5.1	8.9	13.4	6.2	7.0	8.8
Model Overprediction $[W]$		0.15	0.20	0.31	0.12	0.24	0.39	0.15	0.20	0.29
Measured EPF (RTF-HP) $[\mu Wh]$		N/A	69.81	69.15	N/A	N/A	68.27	37.92	36.04	36.36
Measured EPF (RTF-LP) $[\mu Wh]$		N/A	70.09	71.03	N/A	N/A	68.39	40.35	39.93	41.38
Optimal	CPU Frequency $[MHz]$	N/A	1224	696	N/A	N/A	1734	828	564	564
	Memory Frequency $[MHz]$	N/A	924	792	N/A	N/A	924	600	204	102
	EPF (Predicted) $[\mu Wh]$	N/A	49.99	48.16	N/A	N/A	57.80	34.11	33.46	35.13
	EPF (Measured) $[\mu Wh]$	N/A	46.81	42.96	N/A	N/A	52.56	32.88	30.38	31.21
	Improvement $(\%)$	N/A	32.9	37.8	N/A	N/A	23.0	13.2	15.7	14.16

TABLE III: Overview of model quality and most energy-efficient cpu and memory frequency (target framerate at 20 FPS).

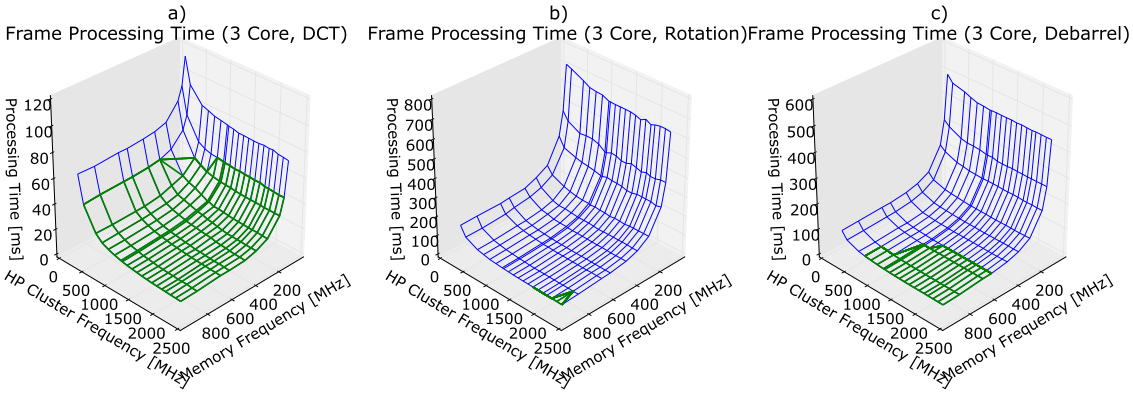


Fig. 11: Workload performance over memory and processor frequencies. The green lines mark configurations that achieve a frame processing time of 50 ms or below.

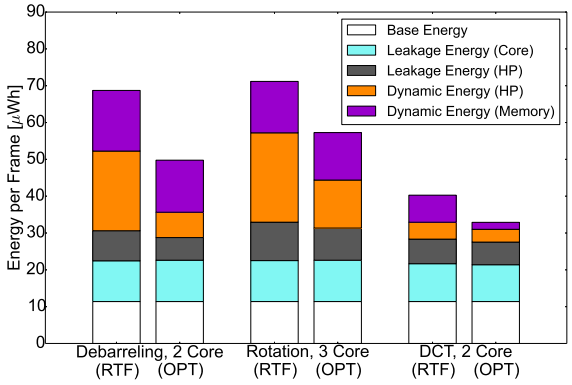


Fig. 12: Detailed breakdown of static, dynamic and base power.

shown in Table III, where energy usage is noted in Energy-per-Frame (EPF). In both strategies, memory and processor frequency is set to the maximum possible while a frame is being processed. When the frame processing is complete, RTF-HP scales down frequency to the minimum possible and sleeps until the next frame is available. RTF-LP additionally switches to the LP core. For some of the core configurations, the required framerate of 20 FPS could not be met. These cases are marked with N/A.

Table III shows that the RTF-LP strategy is consistently outperformed by the RTF-HP strategy in terms of EPF by up to $5\mu Wh$. The most likely reason for this is that switching between the HP cluster and the LP core incurs large transition overheads in terms of both energy usage and time. For example, the best- and worst-case switching overheads when migrating to the HP cluster is 1.4 to 7.0 ms. The overhead is strongly influenced by processor frequency. This result indicates that the LP core should not be used to save energy between frames.

Compared to the RTF-HP strategy, our approach to minimise frequency and power so that the target framerate of 20 FPS is met saves between 13.2 to 37.8 % energy. Based on our power model, Figure 12 gives a detailed breakdown of static, dynamic and base energy usage per frame between some of the benchmarks. The core rail leakage and base energy is roughly the same over the workloads. However, the RTF-HP strategy consumes more dynamic power, as well as more leakage energy on the HP rail. The increased HP rail leakage energy is due to the high voltage required to sustain the maximum frequency on the HP cluster. Furthermore, as explained in Section VI-C, the modest performance increase at high frequencies is not enough to compensate for the increase in dynamic power usage. This makes it more efficient to use lower frequency settings.

The debarreling and DCT benchmarks show that adding cores can have positive effects in terms of energy usage. For example, for the debarreling benchmark, using three cores is more energy efficient than two. The extra leakage current of adding a core pays for itself in that lower processor and memory frequencies can be used. This is not always true, however, when moving from two to three cores, the DCT benchmark actually increases in energy consumption.

VII. Conclusion

In this paper, we investigate the power usage of the Jetson-TK1 and how multimedia workloads can be energy-efficiently processed using the Tegra K1 heterogeneous multicore SoC. We look at three common video processing workloads and ask which memory and processor frequencies yield the best energy-efficiency. To accurately quantify energy consumption and leakage for these heterogeneous systems, we have developed an original method to quantify leakage currents on individual power rails by observing the increase in platform power while varying rail voltage and clock frequencies. We find that both leakage currents and architectural scaling (the workload's ability to scale performance with memory and processor frequency) is very important for energy efficiency. This is because lower memory and processor frequencies can be used to process the workload, while meeting the workload's performance requirements. Furthermore, our experiments show that the popular *race-to-finish* approach is inefficient in terms of energy consumption for continuous workloads. For our example workloads, we achieved between 13-37 % energy saving by minimising memory and processor frequencies such that a framerate of 20 FPS was met. Finally, the current system accurately models the cores and memories of the Tegra K1. However, modern systems also have various components like GPUs and DSPs for offloading. We are therefore currently extending our model to include such processors.

References

[1] A. Castagnetti, C. Belleudy, S. Bilavarn, and M. Auguin. Power Consumption Modeling for DVFS Exploitation. In *Proc of DSD*, pages 579–586. IEEE, 2010.

[2] Keithley. K2280S-32-6 Datasheet. Technical report.

[3] N. S. Kim, T. Austin, D. Blaauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage Current: Moore's Law Meets Static Power. *IEEE Computer*, pages 68–75, 2003.

[4] NVIDIA. Tegra K1 Circuit Schematics, Rev. 4.02. Technical report.

[5] NVIDIA. Tegra K1 Technical Reference Manual. Technical report.

[6] M. Pricopi, T. S. Muthukaruppan, V. Venkataramani, T. Mitra, and S. Vishin. Power-Performance Modeling on Asymmetric Multi-Cores. In *Proc of CASES*. IEEE, 2013.

[7] A. Rice and S. Hay. Decomposing Power Measurements for Mobile Devices. In *Proc of PerCom*, pages 70–78. IEEE, 2010.

[8] B. Rister, G. Wang, M. Wu, and J. R. Cavallaro. A Fast and Efficient SIFT Detector Using the Mobile GPU. In *Proc of ICASSP*, pages 2674–2678. IEEE, 2013.

[9] T. M. S. Muthukaruppan, M. Pricopi, V. Venkataramani and S. Vishin. Hierarchical Power Management for Asymmetric Multi-Core in Dark Silicon Era. In *Proc of DAC*, pages 1–9. ACM, 2013.

[10] K. R. Stokke, H. K. Stensland, C. Griwodz, and P. Halvorsen. Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC. In *Proc of MoViD*, pages 15–16. ACM, 2015.

[11] G. Vass and T. Perlaki. Applying and Removing Lens Distortion in Post Production. In *Proc of CGG*, pages 9–16, 2009.

[12] T. Vogelsang. Understanding the Energy Consumption of Dynamic Random Access Memories. In *Proc of MICRO*, pages 363–374. ACM, 2010.

[13] Y. C. Wang and K. T. Cheng. Energy and Performance Characterization of Mobile Heterogeneous Computing. In *Proc of SiPS*, pages 312–317. IEEE, 2012.