Load Balancing of Multimedia Workloads for Energy Efficiency on the Tegra K1 Multicore Architecture

Kristoffer Robin Stokke
FLIR Systems, Oslo, Norway
University of Oslo, Norway

Håkon Kvale Stensland
Simula Research Laboratory, Norway
University of Oslo, Norway

Carsten Griwodz
Simula Research Laboratory, Norway
University of Oslo, Norway

Pål Halvorsen
Simula Research Laboratory, Norway
University of Oslo, Norway

ABSTRACT

Energy efficiency is a timely topic for modern mobile computing. Reducing the energy consumption of devices not only increases their battery lifetime, but also reduces the risk of hardware failure. Many researchers strive to understand the relationship between software activity and hardware power usage. A recurring strategy for saving power is to reduce operating frequencies. It is widely acknowledged that standard frequency scaling algorithms generally overreact to changes in hardware utilisation. More recent and original efforts attempt to balance software workloads on heterogeneous multicore architectures, such as the Tegra K1, which includes a quad-core CPU and a CUDA-capable GPU. However, it is not known whether it is possible to utilise these processor elements in parallel to save energy. Research into these types of systems are unfortunately often evaluated with the Performance Per Watt (PPW) metric, which is an unaccurate method because it ignores constant power usage from idle components. We show that this metric can end up increase energy usage on the Tegra K1, and give a false impression of how such systems consume energy. In reality, we show that it is much harder to save energy by balancing workloads between the heterogeneous cores of the Tegra K1, where we demonstrate only a 5% energy saving by offloading 10% DCT workload from the GPU to the CPU. Significantly more energy can be saved (up to 50 %) using the appropriate processor for different workloads.

CCS CONCEPTS

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Applied computing** → *Electronics*; • **Human-centered computing** → *Mobile devices*;

KEYWORDS

GPGPU, Multi- and Many-Core, Optimisation Techniques, Embedded Systems, Power Modeling

ACM Reference format:
Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. 2017. Load Balancing of Multimedia Workloads for Energy Efficiency on the Tegra K1 Multicore Architecture. In *Proceedings of MMSys'17, Taipei, Taiwan, June 20-23, 2017,* 12 pages.
https://doi.org/10.1145/3083187.3083195

This work has been performed in the context of the BIA project "Unified PCIe IO" (#235530) funded by the Research Council of Norway (RCN) .

MMSys'17, June 20-23, 2017, Taipei, Taiwan

ACM ISBN 978-1-4503-5002-0/17/06…$15.00
https://doi.org/10.1145/3083187.3083195

1 INTRODUCTION

Energy-efficiency is an important topic in modern, mobile computing, where the evolution in battery energy density has not followed the rapid development in transistor density and power requirements. Between 1990 and 2012, for example, transistor density in integrated circuits has increased by a magnitude of three while the energy density in lithium-ion based batteries has only tripled [15] (see Figure 1). This limits the utility of mobile devices such as smartphones, laptops or even drones, where the combination of a limited battery supply and power-hungry processors is evident. These developments have raised a necessity for rapid charging and external battery packs to power users' needs for gaming, social, work and other activities.

In the context of smartphones, much of the processing is related to multimedia. According to predictions from Cisco, by 2019, the sum of all forms of video will be in the range of 80 to 90 percent of the global consumer traffic, and of this, 14 percent will be mobile data traffic [4]. Many smartphone applications today such as Pokemon Go are actively processing live video streams, by dynamically drawing various creatures over a live, raw camera stream that the players can catch. Snapchat is another popular application that supports several video processing filters that, combined with mobile face recognition, can draw various display elements on top of still pictures or a video stream of a person. Much of this processing can be done on Digital Signal Processors (DSPs) in the devices; however, DSPs support only a static set of functions, and it is not given that they can support future demands for new video filtering operations. In these cases, it is important for mobile devices to have generically programmable processors.

The Tegra K1 is a mobile System-on-Chip (SoC) that provides the programmer with a heterogeneous multicore programming environment through a Low Power (LP) CPU core, a quad-core, High Performance (HP) CPU cluster and a 192-core, Kepler-based

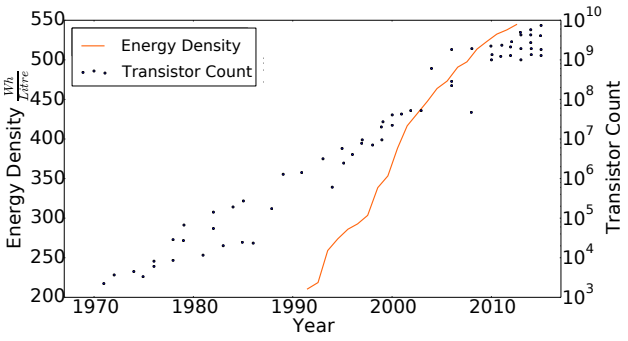


Figure 1: Evolution in transistor and battery energy density.

GPU. The platform also supports Dynamic Voltage and Frequency Scaling (DVFS) of all the compute elements and memory, to tune power and performance to the users' needs. Other examples of heterogeneous multicore SoCs are the OMAP and Exynos platforms, featuring dedicated digital signal processors and up to eight CPU heterogeneous CPU cores. However, the challenge for system architects and developers is to understand which processor cores should perform what video filtering operations, and at what frequency levels. Many researchers exploit the fact that processors are more energy-efficient at lower frequencies to improve the energy-efficiency of different multimedia systems. For example, it is easy to outperform standard DVFS algorithms by minimising processor frequency, such that for example a frame rate of 25 FPS is met [17]. However, in a context where the Tegra K1 is processing a video filter for a specific frame rate, an important question is whether it is possible to load balance the workload between the GPU and CPU, and save energy? Many researchers have approached this problem on various heterogeneous SoCs, and found that offloading processing to heterogeneous cores both boosts performance and increases energy efficiency. However, these authors usually only measure power usage while their experiment workloads are running, and consequently, the increased energy-efficiency is only caused by reduced contribution from idle platform power usage. We demonstrate this measurement pitfall in detail in Section 2.4.

In this paper, we focus on three video filtering operations: the Discrete Cosine Transform (DCT), motion vector search and Huffman coding (VLC). These resemble common video processing operations that are integral blocks of for example H.264 and Google's VP8. Processing a full HD video stream at 35 FPS, we show for example that it is possible to offload 10 % of the per-frame DCT workload from the Tegra K1's GPU to the CPU, saving 5 % Energy per Frame (EPF) compared to running the filter entirely on the GPU. Furthermore, using our high-precision power model for the Tegra K1 [16, 18], we show that the potential amount of energy-saving depends on the energy-efficiency of the CPU. For example, if the CPU is busy processing the motion vector search and Huffman coding, we show that it is no longer possible to save energy by offloading. This is because the CPU is operating at a higher frequency and voltage, reducing the energy-efficiency of that compute element. Instead, care should be taken to select the best processor for different workloads, where the difference in energy consumption can be up to 50 %.

The rest of this paper is organised as follows. In Section 2, we outline related work in energy-efficient multimedia processing, our power model for the Tegra K1 and the relationship between frequencies and voltages. This is necessary to understand how processor are energy-inefficient at higher frequencies. In Section 3, we outline our experimental methodology and discuss the results of our offloading experiments. We conclude the paper in Section 4.

2 BACKGROUND AND RELATED WORK

In this section, we describe the Tegra K1 SoC architecture. This is necessary to understand how the platform consumes energy under software workloads and lays the foundation for our work in later sections. We also survey related work in energy optimisation for island-style SoCs with special attention on the effects of frequency scaling and load balancing, and the dangers of measuring power efficiency using quantitative metrics.

2.1 Tegra K1 Mobile SoC

The Tegra K1 is a heterogeneous multicore SoC (see Figure 2) featuring several processing elements. Its CPU is composed of five ARM Cortex-A15 cores, with one Low Power (LP) core and a High Performance (HP) cluster comprised of four cores. All software can be migrated between the LP core and the HP cluster depending on application demand for processor power. The Tegra K1 also includes a fully programmable (CUDA) 192-core Kepler-based GPU (GK20A). This gives a rich computing environment where applications can utilise several heterogeneous cores with difference performance and power characteristics. The CPU and GPU share two 1GB DDR3 memory banks, where requests from these are arbitrated via an External Memory Controller (EMC) through a 32- and 64-bit interface, respectively.

All processing elements and components of the Tegra K1 (LP core, HP cluster, GPU, memory and others) are organised into several domains, where each domain draws power from a *rail* supplied by a voltage regulator (see Figure 2). This "island-style" organisation of SoCs is normal for mobile processors [2], and we have therefore chosen the Tegra K1 as a representative of these types of systems. The major challenge in evaluating power efficiency of these systems is to understand what is going on inside the processor, that is, where energy is consumed under different software workloads and how power management techniques such as clock and power gating affect energy consumption. Empirical experiments are usually restricted to measuring only the total power usage of mobile devices. In some cases, power can be measured using sensors on each power rail [1], but studies have shown that these are often inaccurate [5]. The Jetson-TK1 development kit, which we are using in our experiment, has no such sensors, and we are restricted to *measuring* the total power usage.

However, understanding the power usage of individual power rails on the Tegra K1 using only external "total" power measurements is impossible. This facilitates the need for a *power model* which is able to predict power on individual rails. Here, we will therefore use our own, highly accurate power model [18, 20] which has been shown to be close to 100 % accurate over all operating frequency ranges. The model and its methodology is too extensive to show here in detail, but we will briefly summarise the key points.

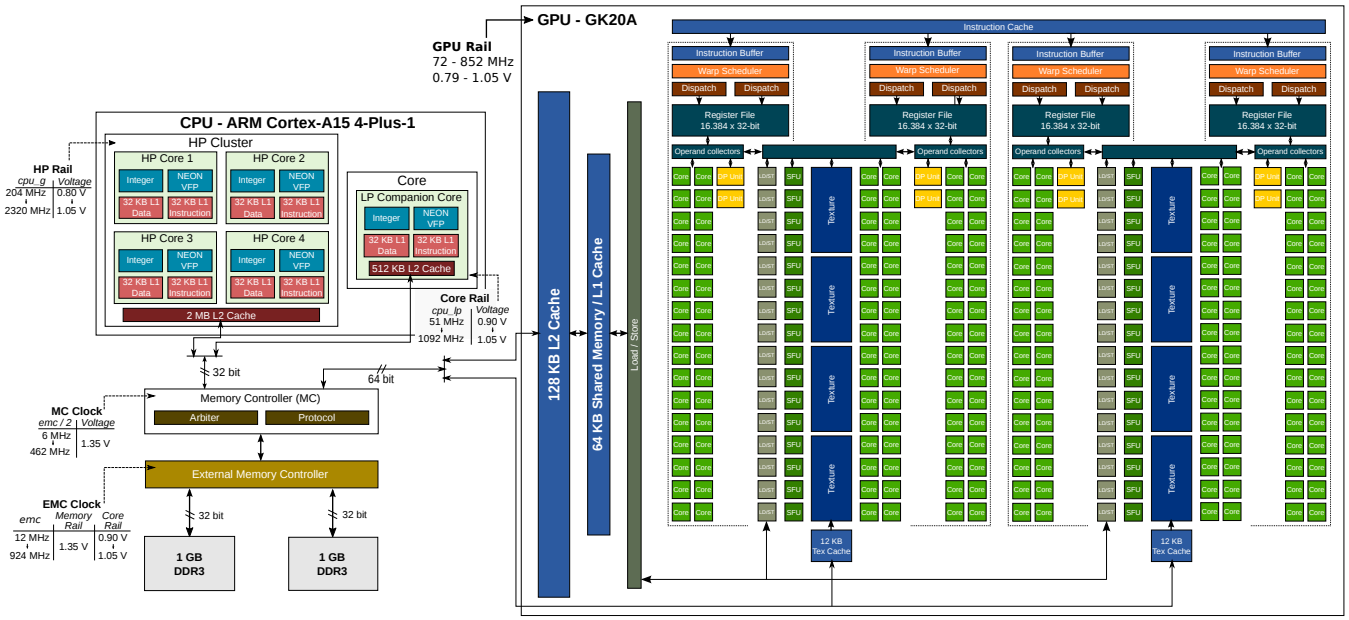


Figure 2: Overview of the Tegra K1 mobile processor.

Rail	Number	Description	Coefficient	Value
GPU	V_{gpu}	GPU voltage	$I_{gpu,leak}$	$0.27A$
	$\rho_{gpu,clock}$	Total clock cycles	$C_{gpu,clock}$	$2.10\frac{nC}{V}$
	$\rho_{gpu,L2R}$	L2 cache 32B reads	$C_{gpu,L2R}$	$10.79\frac{nC}{V}$
	$\rho_{gpu,L1R}$	L1 cache 4B reads	$C_{gpu,L1R}$	$8.90\frac{nC}{V}$
	$\rho_{gpu,L1W}$	L1 cache 4B writes	$C_{gpu,L1W}$	$8.43\frac{nC}{V}$
	$\rho_{gpu,INT}$	Integer instr.	$C_{gpu,INT}$	$41.11\frac{pC}{V}$
	$\rho_{gpu,F32}$	Float (32-bit) instr.	$C_{gpu,F32}$	$38.15\frac{pC}{V}$
	$\rho_{gpu,F64}$	Float (64-bit) instr.	$C_{gpu,F64}$	$115.33\frac{pC}{V}$
	$\rho_{gpu,CNV}$	Conversion instr.	$C_{gpu,CNV}$	$72.42\frac{pC}{V}$
	$\rho_{gpu,MSC}$	Miscellaneous instr.	$C_{gpu,MSC}$	$28.36\frac{pC}{V}$
Memory	$\rho_{mem,clock}$	Total clock cycles	$C_{mem,clock}$	$258.66\frac{pC}{V}$
	$\rho_{mem,CPU}$	CPU memory cycles	$C_{mem,cpu}$	$2.25\frac{nC}{V}$
	$\rho_{mem,OTH}$	GPU memory cycles	$C_{mem,oth}$	$2.17\frac{nC}{V}$
HP	V_{hp}	HP rail voltage	$I_{hp,leak}$	$59.8mA$
	$\rho_{hp,clk1}$	Cycles (first core)	$C_{hp,clk1}$	$395.65\frac{pC}{V}$
	$\rho_{hp,clk2}$	Cycles (second core)	$C_{hp,clk2}$	$270.40\frac{pC}{V}$
	$\rho_{hp,clk3}$	Cycles (third core)	$C_{hp,clk3}$	$261.89\frac{pC}{V}$
	$\rho_{hp,clk4}$	Cycles (fourth core)	$C_{hp,clk4}$	$213.95\frac{pC}{V}$
Core	V_{core}	Core rail voltage	$I_{core,leak}$	$633.7mA$
	$\rho_{core,clk}$	Cycles (LP core)	$C_{core,clk}$	$301.49\frac{pC}{V}$
Common	$V_{com,online}$	Core leakage	$I_{cpu,leak}$	$24.00mA$
	$\rho_{com,l1l2}$	Cache, L1-L2	$C_{com,l1l2}$	$2.35\frac{nC}{V}$
	$\rho_{com,l2ram}$	Cache, L2-RAM	$C_{com,l2ram}$	$2.29\frac{nC}{V}$
	$\rho_{com,ips}$	Instructions	$C_{com,ips}$	N/A
Other	P_{base}	Base power	-	$0.78W$

Table 1: Energy model predictors and coefficients.

All model coefficients can be seen in Table 1. The total power of the Jetson-TK1 and the Tegra K1 SoC can be described as the sum of power over all rails $R \in \mathbb{R}$:

$$P_{jetson} = \sum^{R \in \mathbb{R}} (P_{R,dyn} + P_{R,stat}) + P_{base} \quad (1)$$

There are many rails on the Tegra K1, but we only consider the HP

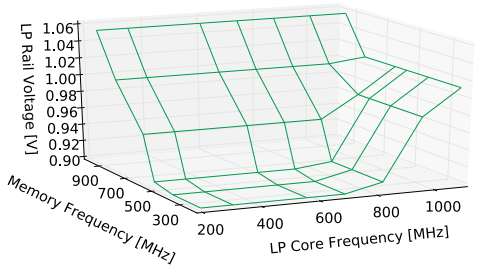


Figure 3: LP rail voltage versus processor and memory frequency.

core, GPU and memory rail. Power on a rail can be described using the standard CMOS equations for static and dynamic power [2, 11]:

$$P_R = P_{R,stat} + P_{R,dyn} \quad (2)$$

Dynamic power is caused by transistor switching activity. According to our methodology [18] we express dynamic power in terms of measurable hardware activity instead of straightforward application of the CMOS equation for dynamic power [19]:

$$P_{R,dyn} = \sum_{i=1}^{N_R} C_{R,i} \rho_{R,i} V_R^2 \quad (3)$$

where $C_{R,i}$ is the capacitive load (coloumbs per volt) per hardware event per second $\rho_{R,i}$. V_R^2 is the rail voltage, which varies depending on operating frequency in that domain. For example, Figure 3 shows the measure LP rail voltage for different combinations of memory

and processor frequency. The rail voltage is set to be the maximum required by and of the clock frequencies it supplies at any point in time. This is to ensure that the transistors switch fast enough with the high clock frequencies.

Each rail has N_R hardware events which reflect hardware activity (see Table 1). These are generally PERF and CUPTI hardware performance counters, but we also have specialised kernel drivers to log CPU core- and clock-gating. Note that due to a lack of hardware performance counters for the ARM CPU implementation, it is not possible to estimate capacitive loads per integer, floating point, data movement etc types of instructions, as is the case for the GPU. Instead, instruction power is estimated on a per-process basis [20]. Static power on a rail is just the product of leakage current $I_{R,leak}$ and voltage on each rail:

$$P_{R,stat} = I_{R,leak} V_R \tag{4}$$

where leakage current on the HP and core rails also varies depending on how many cores $cpu \in \mathbb{C}_R$ on that rail are online (not power-gated):

$$I_{HP/Core,leak} = I_{HP/core,leak} + \sum^{cpu \in \mathbb{C}_R} I_{cpu,leak} \tag{5}$$

2.2 Frequency Scaling and Rail Voltages

CMOS circuits have the advantage that they have very small leakage currents and dissipate most energy as dynamic power in transistor switching activity. Frequency scaling is therefore recognised as one of the key methods to conserve energy in processors, where the operating frequency of various computational elements can be changed to accommodate application demand for processor power [2]. The Tegra K1 can vary LP core, HP cluster, GPU and memory frequency to accommodate applications' demand for processor power. Frequency scaling has adverse effects on performance and power. At every CPU, GPU or memory clock cycle, an electrical charge is switched through the circuitry ($C_{R,clk}$ in Table 1), effectively dissipating energy as heat. Increasing frequency increases the rate at which this occurs and, consequently, the average power on that rail. Furthermore, rail voltage also increases with frequency [11], which is necessary for the circuitry to remain stable and deterministic (see Table 2). Dynamic power is proportional to rail voltage ($P_{R,dyn} \propto V_R^2$), meaning that increases in voltage have a considerable impact on the power of *all* switching activity on a rail.

It is well known in literature that standard frequency scaling algorithms in the Linux kernel generally overreact to changes in processor utilisation, leading to unnecessarily high CPU, GPU and memory operating frequency. As a result, they waste energy. In earlier experiments on the Tegra K1 [17], we found that by minimising CPU and GPU frequencies for a video processing filter such that a certain frame rate is met, we could save around 10 % energy. This is in accordance with the findings of You and Chung [22]. They develop a GPU frequency scaling algorithm that reduces memory frequency such that a target frame rate is met, demonstrating energy savings of between 11 to 21 % on the Exynos 4412 SoC. Pathania et. al. [13] conduct a detailed study of CPU, GPU and memory frequency effect on frame rate in several Android games on the Exynos Octa SoC, and design a frequency scaling algorithm that tunes these frequencies to meet specific frame rates. Jiao et. al. [10] study the impact of both core and memory frequency on power efficiency on a GTX 280 GPU under matrix multiplication, matrix transpose, and the Fast Fourier Transform. They measure power efficiency in performance per watt, or more directly, mega-bytes per second per watt. Interestingly, they find that increased frequency increases power efficiency compared to running the benchmarks on lower frequency levels. This contradicts the general belief that lower frequencies save energy. However, Jiao et. al. stop measuring power when the benchmarks complete, and as a result, the increased power efficiency is just a result of reduced energy consumption from idle (constant) power components. This is especially visible in their discussion for the matrix transpose benchmark, where the benchmark runtime is constant over memory frequencies due to the memory-intense nature of the task. Here, we can confirm that performance per watt is high at *low* frequencies, which occurs because the energy consumption of idle power components is constant. We discuss this phenomenon in detail in Section 2.4.

Clock	Rail	Description	Frequency		Voltage
			Steps	Range $[MHz]$	
cpu_lp	Core	LP core	9	[51, 1092]	[0.80, 1.05]
cpu_g	HP Rail	HP cluster	20	[204, 2320]	[0.80, 1.20]
emc	Core	Memory	7	[204, 924]	[0.90, 1.01]
gpu	GPU	LP core	15	[72, 852]	[0.79, 1.05]

Table 2: The Tegra K1 clocks, voltage and frequency ranges.

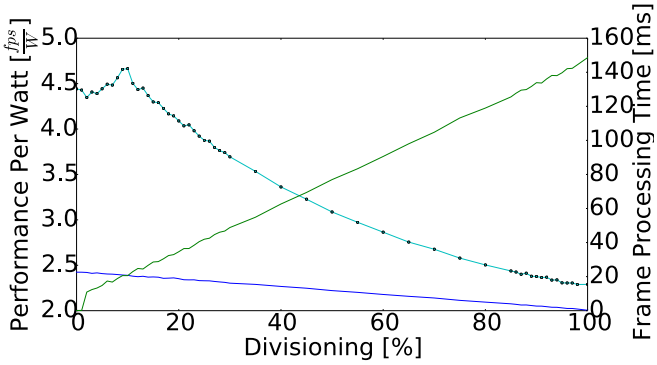
2.3 Load Balancing

Many authors are attempting to increase power efficiency of heterogeneous architectures by dividing workloads between the heterogeneous processing elements [3, 9, 12, 14, 21]. These target mobile SoCs such as the Tegra 2 [3, 21], Samsung S4, Samsung Note II, Google Nexus 7 and Tegra 250 [14], Tegra 3 [9] and Texas Instruments' OMAP 3530 platform [12]. A typical application area is SIFT [9, 14], but Huang and Lai [9] also experiment with BLAS benchmarks, mobile face recognition [3, 21] and face tracking [12].

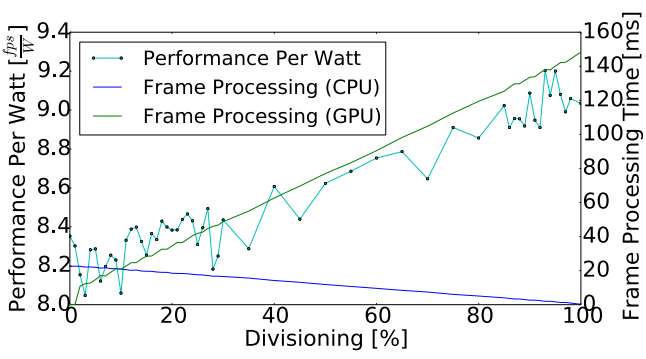
The common approach in these studies is to offload certain computation blocks entirely to the on-board GPU using the OpenGL ES graphics library. General purpose computing frameworks such as CUDA were not available until the Tegra K1 for these studies. With regards to what computation blocks are offloaded, this is rarely discussed in detail [14]. GPU-offloaded computational blocks for SIFT is for example gaussian scale space [14] or convolution, difference of gaussians, octave gradient and key description generation [9, 21]. In their face recognition system for the Tegra 2, Cheng and Wang offload the Gabor wavelet to the mobile GPU [3]. Lopez et. al. [12] design a full offloading scenario, where all mobile face tracking operations can be performed on the CPU or GPU, and in addition, the CPU and GPU can work in parallel on two independent frames.

2.4 Measuring Energy-Efficiency

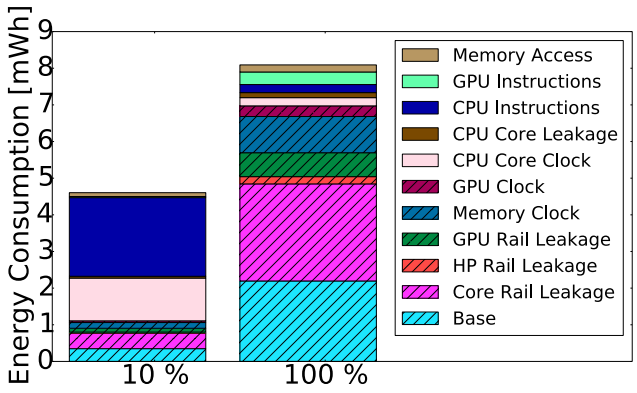
Authors who offload work to GPUs [3, 8, 9, 12, 14, 21] report performance speedups and increased energy efficiency when both the CPU and GPU are used. However, as explicitly noted by Wang and Cheng [21], the increase in energy efficiency is probably a result of *reduced execution time.* All electrical platforms have idle power



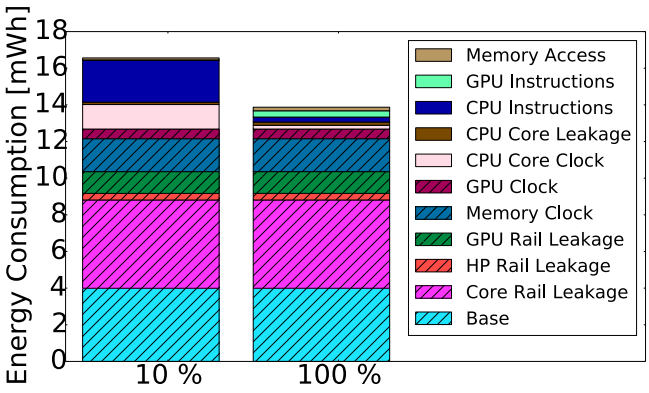
(a) Early finish.



(b) Constant time (20 s).



(c) Early finish.



(d) Constant time (20 s).

Figure 4: Evaluating energy-efficiency with different strategies for test runlength.

usage, caused by for example constant current draw of various passive components, leakage currents in transistors and various clocks which are not gated. When authors offload work to the GPU, execution time is normally reduced, further reducing time to completion for programs and leading to reduced energy consumption from idle components. This is similar to our discussion around the effects of DVFS and Jiao et. al. [10] in Section 2.2, and means that authors who *quantitatively* measure energy efficiency, for example in total runtime energy [3, 9, 12, 21] or Performance Per Watt (PPW) [7, 10, 14], observe increases in power efficiency which is potentially caused only by reduced energy consumption from *idle* power components. Therefore, interpreting the result that hybrid processing saves energy is dangerous. Idle power components is something we cannot affect, i.e., as long as a platform is on and powered, there will be some base power component which we can not remove without shutting down the whole system. When considering hybrid processing scenarios, the *real* questions are how much energy our workloads consume on each processing element, which one is most efficient, and whether energy can actually be saved by exploiting heterogeneous architectures. The risk of interpreting current results in this area is that the "increase" in power efficiency is solely a by-product of reduced idle energy, which effectively hides the answers to these questions.

To illustrate how different conclusions can be made when not considering idle power, consider the following experiment. We have implemented a video processing filter that is performing the Discrete Cosine Transform (DCT) on a sequence of frames on the Tegra K1. For each frame, a specific number of macroblocks (in percent) can be offloaded to the Tegra K1's GPU. Figure 4a shows the PPW (in frames per second per watt) as measured over varying offloading degrees from 0 to 100 % GPU offloading. In each test we process 72 HD frames and the CPU, GPU and memory frequency is set statically to 2320.5, 72.0 and 924.0 MHz, respectively. We see that the maximum PPW is about 4.7 FPS per watt at 10 % GPU offloading, indicating that it is very beneficial to combine the CPU and GPU. Judging from the experimental results, we almost increase power efficiency by 100 % compared to a full GPU offloading scenario, leading us to the conclusion that the GPU is very inefficient for this type of workload.

As mentioned above, the problem with measuring energy efficiency in this way is deeply coupled with idle power usage and the duration of each test. Like other authors, in each of our tests (offloading percentages) we measure power from the start until the test is done processing 72 frames. Consequently, as more per-frame workload is offloaded to the GPU, the benchmarks finish earlier. This is easily visible in Figure 4a, where the highest performance

(FPS) is achieved at 10 % GPU offloading. Increasing or decreasing offloading beyond 10 % degrades performance, because the GPU or the CPU takes longer to finish processing each frame. We now build a detailed energy breakdown of all power components in the system for 10 and 100 % GPU offloading using our detailed power model for the Tegra K1 [18, 20]. The per-frame energy breakdown can be seen in Figure 4c, where the lined blocks correspond to energy consumption from idle components. The rest (non-lined) components are directly related to only the DCT's hardware utilisation on each processing element. Comparing the offloading factors, we see that at 10 % there is a huge reduction in idle energy usage compared to 100 % offloading. This is caused by the short runtime of the test. However, the energy which is related to the processing of each frame is much larger with 10 % offloading than in the case where the entire frame is offloaded to the GPU. This indicates that full GPU offloading is in this case the most energy-efficient alternative, a conclusion which is a radically different than we can make from Figure 4a.

The problems of measuring energy efficiency using quantitative metrics can be overcome without detailed power breakdowns and models. All we need is to run each test for the same duration, taking care not to allow frequency scaling algorithms and power management change platform state during the tests. Considering for example Figure 4b, we see that the PPW is actually best at very high GPU offloading percentages. This is again confirmed with a detailed breakdown in Figure 4d, where it is clearer that the contribution to energy consumption from idle components is constant between the tests. The difference in energy usage per frame is solely due to the energy consumption of the Tegra K1's CPU and GPU under the DCT workload.

3 BALANCING SOFTWARE WORKLOADS ON HETEROGENEOUS CORES

In this section, we first discuss how SoCs are energy-inefficient on higher operating frequencies. Then, we outline the problem scope and method of distributing multimedia workloads between the Tegra K1's CPU and GPU, before we perform experiments on different multimedia workloads. Our workloads are different video processing filters, such as the DCT, motion vector search and Huffman coding. These represent common processing blocks of video encoders and decoders such as H.264 and Google's VP8 video standards.

3.1 Motivation

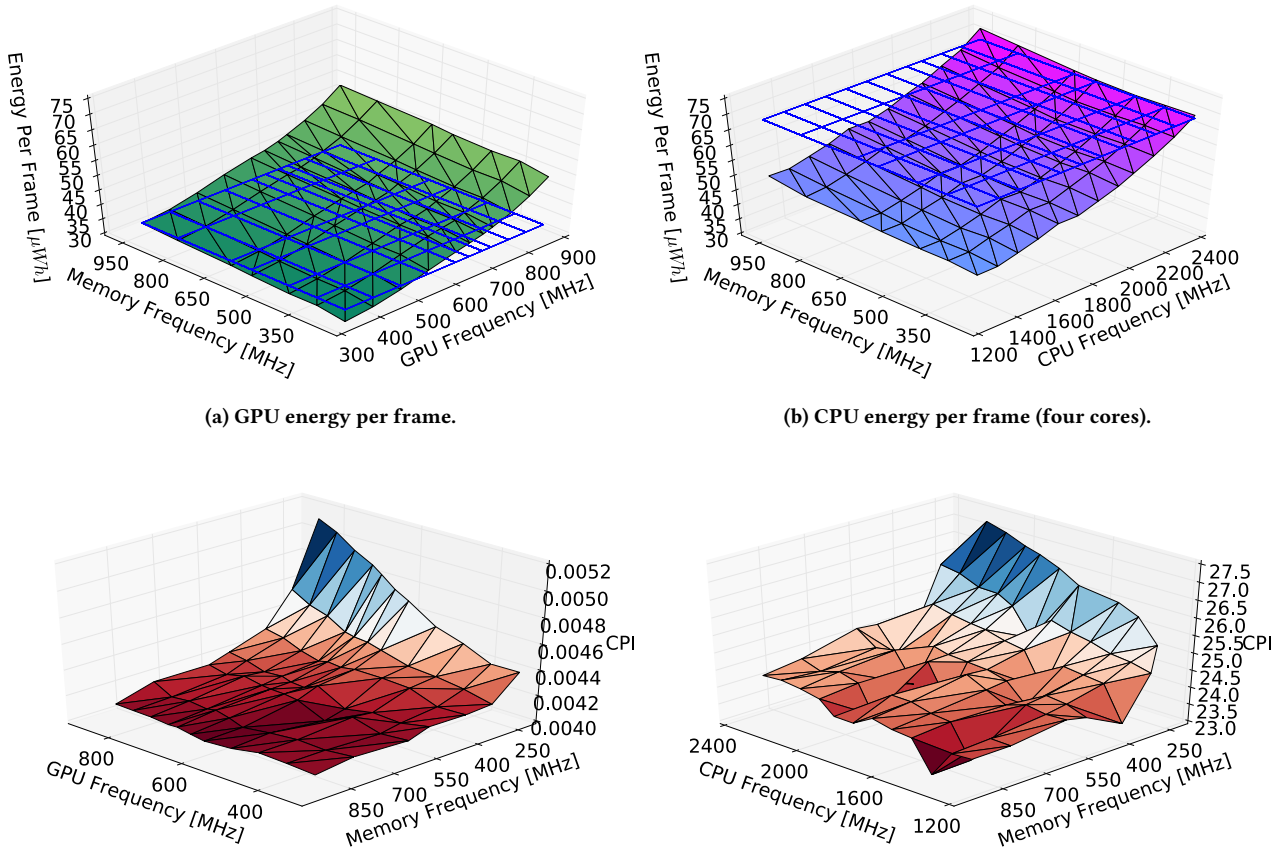
Recent research in energy-efficient, mobile computing often implicitly or explicitly make the assumption that processors are more energy-efficient at lower frequencies. We have also confirmed this phenomenon on the Tegra K1. Consider for example Figures 5a and 5b. These show the energy consumption per frame when processing the DCT on a full HD video stream at 25 FPS, when running the workload on the GPU and the CPU, respectively. The blue, transparent grid shows the energy per frame when the standard frequency scaling algorithms are tuning GPU, CPU and memory frequencies automatically. As we can see from the figure, frequencies should generally be minimised, while adequate frame rate can be delivered, to reach the lowest energy per frame. This heuristic easily outperforms the operation of standard frequency scaling algorithms in terms of energy consumption, without sacrificing performance. In our work on the Tegra K1, we have found two reasons behind this phenomenon. The first, and most important, source of energy-inefficency is voltage scaling. As for example GPU frequency is increased, the GPU's rail voltage also increases, which in turn increases dynamic and static power [11].

Besides energy-inefficiency caused by voltage scaling, there is also another effect which is more challening to observe directly. If we study the average required number of processor cycles required to complete each instruction (CPI) in Figures 5c and 5d, we can see that this fraction increases at low memory and high processor (GPU or CPU) frequencies. Considering GPU processing, at worst 0.0052 GPU cycles are needed per GPU instruction, where the best case CPI value lies at around 0.0040. This may seem counter-intuitive given the low number (only 0.0040) of GPU cycles required to complete each instruction. However, this occurs because the GPU has 192 cores that simultaneously execute instructions, and it is also capable of issuing two independent instructions per cycle. The high CPI value at low memory and processor frequencies means that more cycles, and in turn more dynamic power, is required to process the same amount of instructions. We can also make the similar observations when running the DCT on the CPU, as seen in Figure 5d. Here, the CPI value is at best 23.5 cycles per instruction, but 27.0 cycles per instruction on low memory and high CPU processor frequency. The effects of voltage scaling and CPI is also there on the CPU, and contributes to energy-inefficiency. However, this raises several interesting questions. Given that for example the Tegra K1's GPU is energy-inefficient at high frequencies; can we offload some of the work per frame to the CPU, reducing the current processor's frequency, and thereby save energy?

3.2 Problem Scope and Method

The problem of balancing a workload on two heterogeneous cores, where operating frequencies, CPU cluster and the number of CPU cores can vary is large. Our video processing filters (the DCT, motion vector search and Huffman coding) support fined-grained workload partitioning between the Tegra K1's CPU and GPU, where a percentage of the per-frame workload can be allocated to the GPU in 100 steps from 0 to 100 %. The CPU can be operating on the LP core or the HP cluster, in which case we can also control the number of potentially active cores, yielding an additional five configurations. There are 20 HP cluster frequencies, 7 memory frequencies and 15 GPU frequencies. For just a single workload, this yields over a million possible solutions and makes exhaustive searches unfeasible in practice.

In order to empirically investigate whether offloading can save energy we use the following methodology. First, we decide to focus on the HP cluster with four cores. This is because the Tegra K1 will have to utilise the HP cluster and a high number of CPU cores, as well as the GPU, to achieve required application performance. For example, on the Tegra K1, we found that the motion vector search filter scales much better, in terms of both performance and energy, on the CPU than the GPU. For the video encoder to meet the 35 FPS full-HD QoS requirement, the HP cluster therefore *must* be active with three or four cores to process the motion vector search



(a) GPU energy per frame.

(b) CPU energy per frame (four cores).

(c) GPU cycles per instruction.

(d) CPU cycles per instruction (four cores).

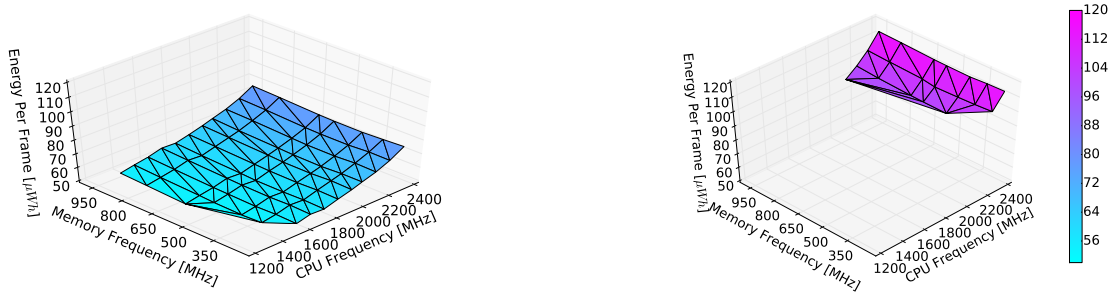
Figure 5: Running a DCT benchmark on different operating frequencies.

filter. Furthermore, to limit the number of frequency combinations to test, we first perform a *coarse* benchmark run. The coarse run always includes four to seven frequencies from each domain (CPU, GPU and memory) and 11 offloading factors. The coarse run always includes the maximum and minimum frequency from each domain. The frequency configuration (f_{mem}, f_{cpu} *and* f_{gpu}) which achieves the best EPF is then selected, and a *fine-grained* benchmark run is initiated. In this run, five frequencies and offloading configurations around f_{mem}, f_{cpu} *and* f_{gpu} are tested. We always focus our search on lower frequencies because we expect, in accordance with our own and other previous work [13, 17, 22], that more energy efficient configurations are found there.

In each frequency and offloading configuration, we run one of our video filters at 35 FPS, processing full HD video in raw YUV format for 80 frames. If the frame rate cannot be met, that configuration is marked as a failed, and it will not be considered. Power is measured using a high-precision, high sampling rate Keithley K2280S source and measurement unit. As described in our previous work [17], the measurements are taken using an external machine, and we take care to synchronise the power logs with the Tegra K1. Power is estimated at least every 100 ms on the Tegra K1 using our model from Section 2.1, CUPTI and PERF hardware performance counters as well as our own kernel tracing framework to track power and clock gating. This induces some additional but negligible overhead to the system which do not impact the results. The videos, output data and power estimation logs are stored in ramfs, meaning that the only hardware activity in the system is on the core, HP cluster, memory and GPU rails.

3.3 Effects of NEON Instructions

The video processing filters presented in this paper all support acceleration using NEON instructions. These are single instruction, multiple data instructions that increase the throughput of arithmetic operations on the Tegra K1's CPU by executing multiple computations concurrently. In the aim of optimising the performance and energy of the CPU as much as possible, we have therefore conducted experiments where we study the impact of these types of instructions. Consider for example Figure 6, where we show the EPF running the DCT filter at 25 FPS on the CPU with four active cores, over all processor and memory frequency combinations. Only the frequency combinations that reach 25 FPS



(a) With NEON instructions.

(b) Without NEON instructions.

Figure 6: Energy per frame for the DCT filter with and without NEON instructions.

are shown. Without NEON acceleration (see Figure 6b, the lowest EPF is 96.10μWh at $f_{cpu} = 2GHz$, $f_{mem} = 924MHz$. However, with NEON acceleration (see Figure 6a), the lowest EPF 51.92μWh for much lower frequencies at $f_{cpu} = 1.3GHz$, $f_{mem} = 600MHz$. These results have two important implications. First, the possible area of operation, in terms of possible processor and memory frequencies, is much larger with NEON acceleration. This is because the instructions accelerate performance, allowing the DCT filter to finish processing each frame faster, further allowing us to reduce frequencies until the frame processing time is close to 40 ms. Second, energy efficiency increases due to the reduction in platform frequencies. However, what can we say about the actual energy-efficiency of the NEON instructions?

In order to investigate the actual cost of NEON instructions, we continuously sample power usage over time, while running the DCT filter over all processor and memory frequencies on the Tegra K1's CPU. Using our power model presented in Section 2.1, we subsequently remove all the *generic* power components from these measurements. The remaining *residual* power should then reflect only the cost of CPU instructions; which is the only power component not captured by our model in Table 1. This is because the Tegra K1's CPU does not support fine-grained accounting of the type and number of CPU instructions executed by applications. It only has a single instruction counter, that counts *all* instructions executed by the CPU. Figure 7 shows the residual power components plotted over the model predictor, the $V_{cpu}^2 \rho_{ips}$ (square-voltage instructions per second). The slopes of the lines in the figure reflect the average capacitive load per instruction, in coloumbs per volt. With and without NEON acceleration, this cost is estimated to be 383 and 317 $\frac{pC}{V}$, respectively. This interesting finding means that, on average, the NEON instructions cost more than the normal instructions. This is reasonable given that each instruction excersises more of the underlying transistors in the processor through multiple arithmetic operations per instruction. However, fewer instructions are also needed to complete processing each frame, making it more energy-efficient than the alternative. Figure 8 shows a detailed energy breakdown for the best (in terms of EPF) frequency configurations of the DCT filter, with and without NEON acceleration. The figure confirms that the most substantial reduction in total energy consumption comes from a large reduction in instruction energy

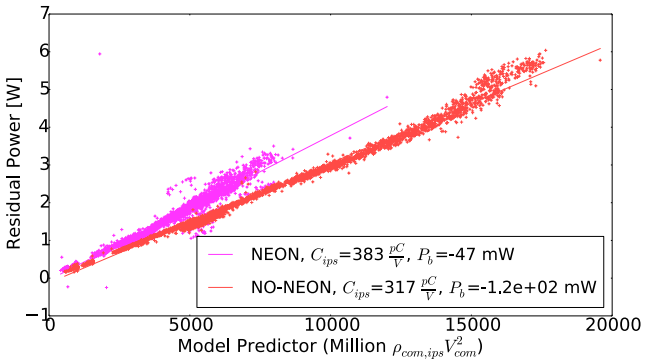


Figure 7: Capacitive load per instruction for the DCT filter with and without NEON instructions.

consumption. Some of this reduction can be attributed to reduced rail voltage, but as already stated the number of instructions required to finish processing each frame is also reduced. Additional reductions in cache and cycle energy (processor and memory) also contribute to the overall reduction in energy usage.

3.4 Offloading a Single Filter

We now focus on the DCT filter according to our methodology in the previous section. Figure 9a shows the measured DCT energy per frame for GPU offloading percentages. Each coloured line represents a single frequency combination, where the color intensity indicates the frequency level. The more intense (red) the color, the higher the frequencies, and the more cool (light-blue) the lower. Only the data points that reach the target frame rate are drawn. From the figure, we can see that for both these filters, the lowest measured EPF is found at very low frequency combinations (blue lines). This confirms the theory that in general, frequency should be minimised such that application requirements are met [13, 17, 22]. In the coarse test run, the frequency combination that achieved the lowest energy per frame (and 35 FPS) is $f_{mem} = 528MHz$, $f_{cpu} = 564MHz$ and $f_{gpu} = 756MHz$, where 90 % of the per-frame workload is processed on the GPU and the rest is processed on the four CPU cores. The

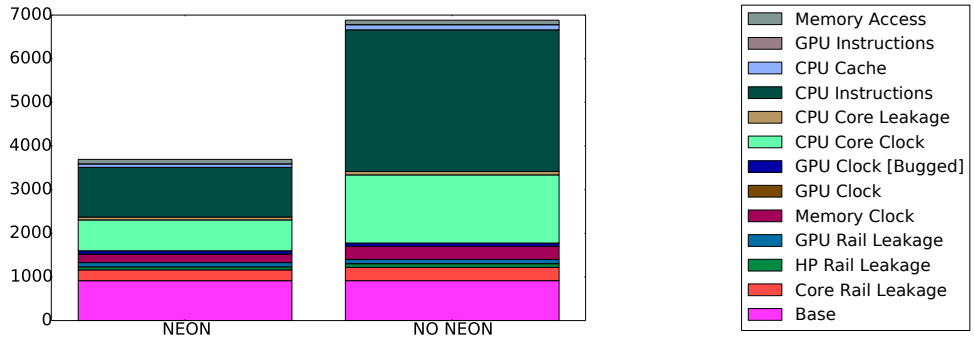
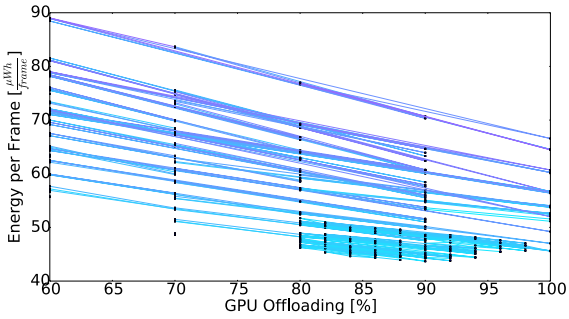
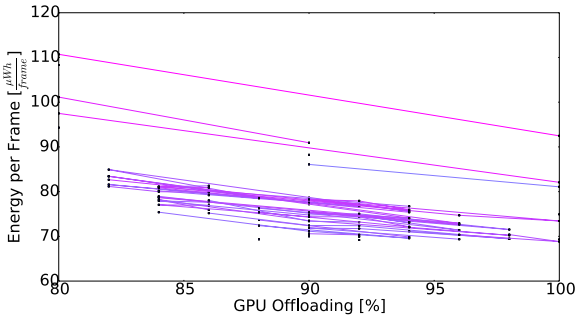


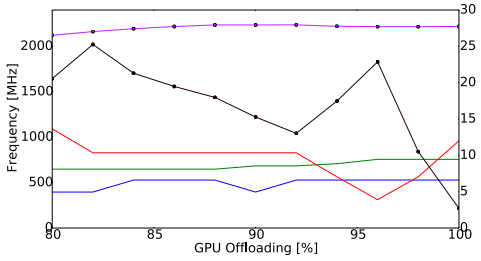
Figure 8: Comparison between NEON and non-NEON DCT filter energy breakdowns.



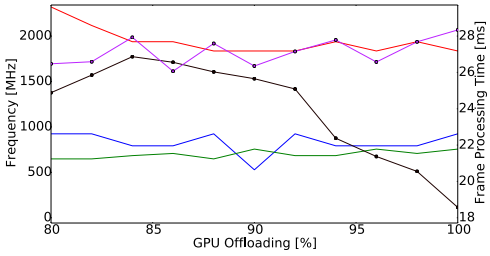
(a) EPF over offloading factors (DCT).



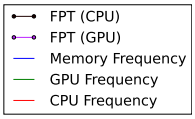
(b) EPF over offloading factors (all filters).



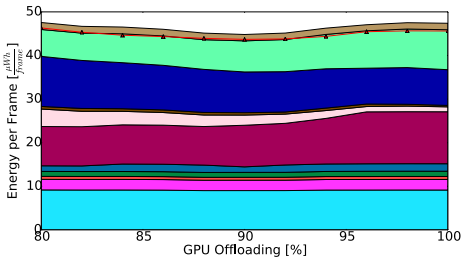
(c) Best frequency combinations (DCT).



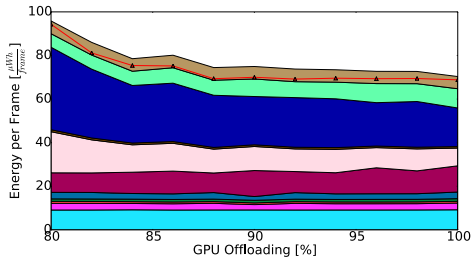
(d) Best frequency combinations (all filters).



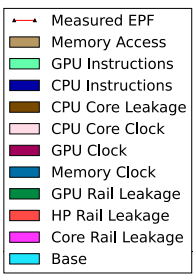
(e) A legend.



(f) Breakdown (DCT).



(g) Breakdown (all filters).



(h) A legend.

Figure 9: Offloading experiments.

energy per frame is $44.41\frac{\mu Wh}{frame}$. Following our methodology above, we now perform a fine-grained search around this frequency and offloading configuration. We focus on ten offloading configurations between 80 to 90 %, and attempt to set the memory, GPU and CPU frequencies close to the best point found above. We now find a better offloading configuration, where memory frequency f_{mem} has decreased to 324 MHz, CPU frequency f_{cpu} has been increased to 828 MHz and GPU frequency f_{gpu} is further reduced to 684 MHz. The energy per frame is also lower at $43.69\frac{\mu Wh}{frame}$.

Figure 9c shows, for any offloading percentage, the *best* frequency combination (in terms of energy per frame) and FPT at that point reaching 35 FPS. We can see that the GPU frame processing time is very close to the per-frame processing deadline of 28 ms. This is as expected, where it is part of our assumption that frequency should be minimised such that the frame processing deadline is just met. However, the CPU FPT is lower and generally varies between 10 to 20 ms. Theoretically, this means that the CPU frequency *could* be lowered, while still meeting the frame processing deadline of 28 ms. However, investigating this issue we found that some milliseconds are necessary to synchronise the CPU with GPU kernels. Therefore, if the CPU frequency is lowered, it takes too long to synchronise with the GPU. Consequently, the frame processing deadline is not met even if the actual GPU code finishes in time.

Further studying Figure 9c, we see that from the 100 % offloading scenario, GPU frequency is gradually reduced as more work is offloaded to the CPU. In turn, we see that CPU frequency generally stays at around 828 MHz over most offloading configurations, but that the CPU FPT gradually increases toward 28 ms as more work is offloaded from the GPU to the CPU. At 80 % offloading, CPU frequency has to increase to 1.0 GHz to reach the frame processing time. Memory frequency generally stays at either 396 or 528 MHz, but as we can see from the figure, there is some variance in which frequency is best and there is not any clear explanation as to why memory behaves this way. It is our hypothesis that, as more work is distributed between the cores, the memory controller becomes more efficient as it has both a 32 bit interface towards the CPU and a 64 bit interface to the GPU. More memory requests can be served per time, and therefore, the memory frequency shows some tendency to be reduced at 80 MHz.

To further understand what is happening within the SoC, and what is occurring within the Tegra K1's circuitry, we need to use our power model to understand what is happening on each rail in terms of static and dynamic power. Figure 9f shows, for the best frequency combination at each offloading factor, EPF broke down into the Tegra K1's individual power components. The red line shows the *measured* EPF, and the other blocks show the energy consumption of the Tegra K1 broken down into static and dynamic power components. We can see that the model prediction is decent, as the estimation generally follows the measured values. Of the energy components, base power, core, HP and GPU leakages as well as memory clock remain approximately constant over offloading configurations. The GPU clock energy, however, shows dramatic reduction because the GPU clock frequency is lowered from 100 towards 80 % GPU offloading. On the other hand, we can see that the CPU clock energy increases. This is as expected because more work is performed on the CPU. The same trend can be seen for instruction energy. The GPU instruction energy is reduced, and CPU instruction energy is increased, as more work is performed on the CPU. Interestingly, memory access energy remains constant over offloading configurations. We can conclude that it is possible to save energy by offloading 10 % DCT workload from the GPU to the CPU. This is because the energy saved in reduced GPU clock and instruction energy, and reduced GPU voltage, is larger than the additional cost of processing those 10 % on the CPU. Compared to running the filter 100 % on the GPU, 4.14 % energy per frame was conserved from $45.56\frac{\mu Wh}{frame}$ to $43.69\frac{\mu Wh}{frame}$. However, it is important to note that the actual saving on the SoC is larger due to the large base power component. Without it, the saving is 5.08 %.

3.5 Offloading Under Heavy Processing

In the previous section, we observed that it is possible to save five percent EPF by offloading 10 % of the DCT workload from the GPU to the CPU. This is possible because the total amount of energy saved by reducing the GPU operating frequency, conserving GPU clock energy and reducing GPU voltage, is larger than the additional cost of processing that workload on the CPU. However, in this scenario the CPU was idle with respect to other tasks. In a video encoding scenario, the CPU must also fulfill other tasks such as motion vector search and Huffman encoding. If the CPU is busy processing other tasks, the extra cost of offloading the DCT workload from the GPU may be too large to achieve a positive gain of the offloading. We now repeat the above experiment, but let the CPU process a motion vector search as well as Huffman encoding to file. The best EPF after the coarse run was here $67.25\frac{\mu Wh}{frame}$ at 100 % GPU offloading, at $f_{mem} = 924MHz$, $f_{cpu} = 1.8GHz$ and $f_{gpu} = 756MHz$. Searching for better (lower) frequencies around this configuration did not yield any better EPF. The results are shown in Figure 9b, where we see that while the 100 % offloading configuration is best, there are several configurations, such as at 86 and 92 %, that are very close to achieve the same EPF. Studying Figure 9d, we can see that the GPU frequency is lowered as was the case for the single-filter DCT experiment in the previous section. Additionally, the energy breakdown in Figure 9g shows that the GPU clock and instruction energy is reduced when more work is offloaded from the GPU to the CPU. However, the additional cost of processing on the CPU is larger than the saving of reduced GPU clock and instruction energy. This occurs because the CPU is already working at a very high processor frequency of 1.8 GHz, in order to process the motion vector search and Huffman filters at 35 FPS. Compared to running the CPU at well below 1.0 GHz, which was the case for the previous single-filter experiment, the CPU voltage has increased from 0.82 to 0.94 V. This increases dynamic and static power components such as instruction and clock energy on the CPU. It is worth noting, however, that the margins are small. At 90 % offloading, the EPF is $70.01\frac{\mu Wh}{frame}$, which is very close to the 100 % offloading EPF of $67.25\frac{\mu Wh}{frame}$.

3.6 Choosing The Right Core for the Right Job

In the previous sections, we have demonstrated concrete cases where balancing workloads between heterogeneous processors can

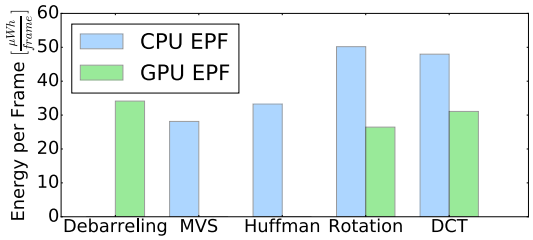


Figure 10: Lowest acheived energy per frame at 25 FPS between CPU and GPU software implementations. No bar means the implementation could not reach 25 FPS.

yield increases in energy efficiency. This occurs because the offloading itself allows us to reduce processor frequency on a core that may already be working heavily (at high frequencies). The reduction in processor frequency yields an energy saving (due to reduced rail voltage and clock power usage) that is greater than the additional energy usage of turning on an additional processor. However, the margins are slim. In Section 3.4, we achieved a 5 % energy saving by offloading some DCT processing to the Tegra K1's CPU. However, in Section 3.5, this was impossible because the Tegra K1's CPU was already burdened with other heavy processing. The additional enegy overhead of processing some of the GPU's DCT workload on the CPU was now greater than the energy saved on the GPU, because the CPU was already operating at much higher operating frequencies and rail voltages. This raises an important question: can offloading work between such heterogeneous processor be worth it in the end?

Dividing per-frame processing between cores is not trivial. First, it demands a significant effort on the programmer. Two, or maybe more, separate implementations are required, for example in CUDA, C and assembly. Second, the implementations have to be structured such that the correct offloading factors are actually computed on the respective cores. The filters in this article, for example, divide workloads according to sequential macroblocks in each video frame. Third, the divisioning of processing within consequtive frames may cause performance issues in terms of memory sharing and bottlenecks. For example, if the Tegra K1's GPU processes 85 % of the per-frame DCT workload, and the CPU relies on all that data for its own computation (such as the motion vector search), data has to be distributed correctly to achieve reliable computation. It is also simple to find scenarios where some later filter stage depends on only a small part of the previous filter stage's output data. All these points considered, the gain of offloading in terms of increased energy efficiency certainly seem small for such a large amount of work and possible points of failure.

In this work, we have implemented this functionality for a wide range of video processing filters. However, while we could not find concrete scenarios where offloading was considerably more beneficial, we have observed significant variations in the per-frame energy cost between CPU and GPU implementations of the different filters. Consider for example Figure 10. Here, the lowest EPF achieved betwen a CPU and a GPU software implementation for each filter is shown. This effectively means the minimum processor and memory frequencies that achieve an average frame processing deadline of 40 ms (see Table 3). The filters must reach a 25 FPS QoS requirement; otherwise, they are not shown. From the figure, it is already clear that the different processors are very different with respect to performance for the different workloads. Only the GPU is for example able to process the debarreling filter at 25 FPS, and only the CPU can process the MVS and Huffman filters. This occurs because, even at the highest memory and processor frequencies, and with the highest number of cores active, they are not able to reach the 40 ms frame processing deadline. Both the GPU and the CPU of the Tegra K1 are able to process the rotation and DCT workloads at the required 25 FPS. Here, we can see that the GPU significantly outperforms the CPU for both filters, with an increased energy-efficiency of between 50 to 60 %.

The significant difference in energy usage of processing the same type of algorithm on two different hardware implementations is closely tied to the performance of the processor. This is because each clock cycle consumes energy, and the fewer cycles are used, the less energy will be spent. Other power components, such as instruction execution and cache utilisation, also matter. However, these are architecture specific and it is difficult to discuss how these relate to each other across heterogeneous processors without further research. When two architectures are in principle fast enough to compute a workload within for example frame processing deadlines, the differences become very clear. Before they are implemented and tested, however, it is not clear from a programmer's perspective what processor will be best.

4 CONCLUSION

Energy is an important resource in modern mobile computing, and the evolution in battery energy density has not followed the rapid development in power requirements of processor technology. Modern SoCs such as the Tegra K1 offers the programmer unprecedented flexibility in terms of general purpose compute cores, such as an LP CPU core, a quad-core CPU cluster, as well as a CUDA-capable GPU. The challenge for system architects and developers is thus how to schedule and divide workloads on the Tegra K1's processors using the best possible frequencies. Many researchers take advantage of modern SoCs and their increased energy efficiency at lower operating frequencies, as well as the number of heterogeneous compute cores, to design more energy-efficient multimedia systems. However, the energy measurement methods used in this literature has a significant pitfall. Researchers usually only measure power usage for the duration of their experiments, and consequently, they often find that offloading to heterogeneous processor cores saves energy and boosts performance. However, when the base power usage of the platforms under study is not considered, the energy "saved" in these cases may only be a result of reduced idle energy usage (due to reduced processing time). This may lead to incorrect conclusions with respect to which offloading configurations are most energy-efficient. This problem can be easily avoided by simply running all experiments for the same duration. Alternatively, a detailed and

Workload	GPU			CPU			
	$f_{gpu}[MHz]$	$f_{mem}[MHz]$	EPF $\frac{\mu Wh}{frame}$	N_{cores}	$f_{cpu}[MHz]$	$f_{cpu}[MHz]$	EPF $\frac{\mu Wh}{frame}$
Debarreling	180	300	34.13	-	-	-	-
MVS	-	-	-	3	696	204	28.13
Huffman	-	-	-	4	828	204	33.28
Rotation	108	204	26.45	4	1200	396	50.19
DCT	324	204	31.08	4	1200	204	47.97

Table 3: Overview of the operating points and hardware configurations that reaches 25 FPS with different video filtering operations.

accurate energy model can be used to analyse in detail the power components.

Throughout this paper, we have considered several multimedia workloads that resemble common video processing operations, such as the DCT, motion vector search and Huffman coding, and studied how these could be distributed and scheduled on the Tegra K1 to minimise energy consumption. Through our experiments, we have shown that it is possible to conserve energy by load balancing work between the CPU and the GPU, where we demonstrate a 5 % energy saving by offloading 10 % DCT workload from the Tegra K1's GPU to the CPU. Our high-precision power model for the Tegra K1 reveals that the offloading allows us to reduce the GPU frequency, further reducing GPU clock and instruction energy, and that this saving is higher than the additional cost of processing 10 % DCT workload on the CPU. However, while we have shown that this offloading between heterogeneous processor cores is in principle possible, our experience is that it can be very hard to achieve in practice. For example, we have also attempted the same experiment where the CPU is not idle. Here, we observed that the offloading did not save energy. This is likely due to reduced CPU energy-efficiency caused by increased CPU voltage at higher CPU frequencies.

For further work, we are considering integrating the power models in a framework for distributed processing of interactive workloads such as the P2G framework [6]. In this framework, the high-level scheduler can use power consumption as one of the metrics when distributing and repartitioning workload during execution. We are also testing our power models on more recent SoCs such as the Nvidia Tegra X1 and X2.

REFERENCES

[1] Aaron Carroll and Gernot Heiser. 2010. An Analysis of Power Consumption in a Smartphone. In *Proc of USENIX ATC*, Vol. 14. 21–21.

[2] Andrea Castagnetti, Cécile Belleudy, Sebastien Bilavarn, and Michel Auguin. 2010. Power Consumption Modeling for DVFS Exploitation. In *Proc of DSD*. 579–586. https://doi.org/10.1109/DSD.2010.55

[3] Kwang Ting Cheng and Yi Chu Wang. 2011. Using mobile GPU for general-purpose computing a case study of face recognition on smartphones. *Proc of VLSI-DAT* (2011), 54–57. https://doi.org/10.1109/VDAT.2011.5783575

[4] Cisco. 2014. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019 White Paper. (2014).

[5] Mian Dong and Lin Zhong. 2011. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *Proc of MobiSys*. 335–348. https://doi.org/10.1145/1999995.2000027

[6] Håvard Espeland, Paul B. Beskow, Håkon Kvale Stensland, Preben Nenseth Olsen, Ståle Kristoffersen, Carsten Griwodz, and Pål Halvorsen. 2011. P2G: A Framework for Distributed Real-Time Processing of Multimedia Data. In *Proc of ICPPW*. IEEE, 416–426. https://doi.org/10.1109/ICPPW.2011.22

[7] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. 2013. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. In *Proc of ICPP*. 826–833. https://doi.org/10.1109/ICPP.2013.98

[8] Sunpyo Hong and Hyesoon Kim. 2010. An integrated GPU power and performance model. *ACM SIGARCH Computer Architecture News* 38, 3 (2010), 280–289. https://doi.org/10.1145/1816038.1815998

[9] Miaoqing Huang and Chenggang Lai. 2013. Accelerating Applications Using GPUs on Embedded Systems and Mobile Devices. In *Proc of HPCC_EUC*. 1031–1038. https://doi.org/10.1109/HPCC.and.EUC.2013.146

[10] Y. Jiao, H. Lin, P. Balaji, and W. Feng. 2010. Power and performance characterization of computational kernels on the GPU. In *Proc of GreenCom & CPSCom*. 221–228. https://doi.org/10.1109/GreenCom-CPSCom.2010.143

[11] Nam Sung Kim, Todd Austin, David Blaauw, Trevor Mudge, Krisztian Flautner, Jie S. Hu, Mary Jnae Irwin, Mabmut Kandemir, and Vijaykrisbnan Narayanan. 2003. Leakage Current: Moore's Law Meets Static Power. *IEEE Computer* 36, 12 (2003), 68–75. https://doi.org/10.1109/MC.2003.1250885

[12] Miguel Bordallo López, Henri Nykänen, Jari Hannuksela, Olli Silvén, and Markku Vehviläinen. 2011. Accelerating image recognition on mobile devices using GPGPU. *Parallel Processing for Imaging Applications, SPIE 7872* (2011), 78720R. https://doi.org/10.1117/12.872860

[13] Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra. 2014. Integrated CPU-GPU Power Management for 3D Mobile Games. In *Proc of Design Automation Conference*. IEEE, 1–6. https://doi.org/10.1145/2593069.2593151

[14] Blaine Rister, Guohui Wang, Michael Wu, and Joseph R. Cavallaro. 2013. A Fast and Efficient SIFT Detector Using the Mobile GPU. In *Proc of ICASSP*. 2674–2678. https://doi.org/10.1109/ICASSP.2013.6638141

[15] James F Rohan, Maksudul Hasan, Sanjay Patil, Declan P Casey, and Tomás Clancy. 2014. Energy storage: battery materials and architectures at the nanoscale. In *ICT-Energy-Nanoscale energy management concepts towards Zero-Power Information and Communication Technology*. Intech.

[16] Kristoffer Robin Stokke. 2017. *High-Precision Power Modelling and Optimisation of the Tegra K1 Heterogeneous Multicore Architecture*. PhD Thesis. University of Oslo.

[17] Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. 2015. Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC. In *Proc of MoViD*. 15–16. https://doi.org/10.1145/2727040.2727044

[18] Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. 2016. A High-precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads. In *Proc of MMMSys*. ACM, Article 14, 12 pages. https://doi.org/10.1145/2910017.2910591

[19] Kristoffer Robin Stokke, Håkon Kvale Stensland, Pål Halvorsen, and Carsten Griwodz. 2015. Why Race-to-Finish is Energy-Inefficient for Continuous Multimedia Workloads. In *Proc of MCSoC*. 57–64. https://doi.org/10.1109/MCSoC.2015.20

[20] Kristoffer Robin Stokke, Håkon Kvale Stensland, Pål Halvorsen, and Carsten Griwodz. 2016. High-Precision Power Modelling of the Tegra K1 Variable SMP Processor Architecture. In *Proc of MCSoC*. 193–200. https://doi.org/10.1109/MCSoC.2016.28

[21] Yi-Chu Wang and Kwang-Ting Cheng. 2011. Energy-Optimized Mapping of Application to Smartphone Platform - A Case Study of Mobile Face Recognition. In *Proc of CVPR*. 84–89. https://doi.org/10.1109/CVPRW.2011.5981820

[22] Daecheol You and Ki-Seok Chung. 2015. Quality of service-aware dynamic voltage and frequency scaling for embedded GPUs. *IEEE Computer Architecture Letters* 14, 1 (2015), 66–69.