Doctoral Dissertation

High-Precision Power Modelling and Optimisation of the Tegra K1 Heterogeneous Multicore Architecture

Kristoffer Robin Stokke

August 2016

Submitted to the Faculty of Mathematics and Natural Sciences at the University of Oslo in partial fulfilment of the requirements for the degree of Philosophiae Doctor

Abstract

Modern, mobile devices such as smart phones, laptops or even drones fulfill many of the users' need for games, social, work and other activities. Much of the processing done on these devices is related to multimedia, for example with applications such as Snapchat, Pokemon Go, Ingress and YouTube. However, mobile devices are constrained with a limited energy supply, where the evolution in battery energy density has not followed the power requirements of modern processors. These integrate Systems-on-Chip (SoC), such as the Tegra K1 SoC, that feature a range of different hardware accelerators, general purpose processors and advanced power management mechanisms such as frequency scaling. The challenge for system architects and programmers is thus to understand how to develop, distribute and schedule multimedia workloads across the heterogeneous, multicore processors. However, manually measuring power usage of running software on different processors is unfeasible, and separating the power usage of an application's instructions and memory usage from other processes and power components is very hard. Models for power are therefore necessary to understand the relation between energy consumption, software activity and power saving mechanisms. However, the state-of-the-art power modelling methods that exist in the literature fail to give an accurate view of the energy consumption of SoCs such as the Tegra K1. In this thesis, we therefore develop an improved power modelling methodology that is able to predict power usage of the Tegra K1 with close to 100 % accuracy. By tracking platform frequencies and voltages, power- and clock-gating as well as fine-grained hardware activity measurements, our model provides detailed insight into static and dynamic power components of the Tegra K1's heterogenous processors and memory. By using our power model, we identify main reasons behind energy-inefficiency in processors, analyse and optimise the energy consumption of kernel drivers, and study the effects of programming using different types of instructions, non-coherent caches and load-balancing between heterogeneous processors.

Acknowledgements

The work presented in this thesis was started on the fifth of June, 2014. I always recall this as a warm, sunny day, getting my hands on the Tegra 3 "Kayla" development platform. After just a few weeks, we had learned that understanding how these devices consume energy is far from trivial. In fact, it can be very challenging at times, when mechanisms that are often far beyond the scope of easy understanding and reach complicate even the simplest of tasks. Therefore this thesis is about *understanding* these devices. Without my research group in Simula, located in the old airport terminal in Fornebu, I do not think this task would even be manageable. While some of the days here have indeed been long and rough, I look back at this time with a smile when I remember our airconditioning unit, the Petter Solberg posters, ice cream in the sun (and occasionally snow and rain), running after Vamsi in all kinds of hikes, travels around the world and unvoluntary trips around volcanoes. I think the environment is the most important thing for a PhD student to thrive, and to be encouraged and motivated. The Media group at Simula has and continues to provide a great social community, while still being able to guide and listen to us when things are not going so well. I would like to give a huge thank you to Dr. Håkon Kvale Stensland, Prof. Pål Halvorsen, Prof Carsten Griwodz and Prof. Tor Sverre Lande. You have not only been my supervisors, but also travel companions and better friends than I could ask for. I would also like to thank my office colleagues; Kristian, Preben, Vamsi, Ragnhild, Olga, Minoo, Jonas, Michael, Andreas, Konstantin, Lilian, Georgios, Iffat and last but not least, Kjetil. Special thanks to Marius, Cathrine, Jim, Kaja and Tor Martin: UiO was never the same without you, so I am very grateful that we are still in touch today. Finally, dearest of appreciation to my father, Per (the toolbuilder), my sister Helle, Trond and Anette, Liv and all the rest of my family for their continued support and unlimited patience over these years. This thesis is for you.

Contents

List of Figures

List of Tables

Abbreviations

A/D ..... Analog-to-Digital

ACTMON ... Activity Monitor

API ..... Application Programming Interface

BIOS ..... Binary Input Output System

CMOS ..... Complementary Metal-Oxide-Semiconductor

CPI ..... Cycles Per Instruction

CPU ..... Central Processing Unit

CUDA ..... Compute Unified Device Architecture

CUPTI ..... CUDA Profiling Tools Interface

DCT ..... Discrete Cosine Transform

DSP ..... Digital Signal Processor

DVFS ..... Dynamic Voltage and Frequency Scaling

EMC ..... External Memory Controller

EMMC ..... Embedded MultiMedia Card

EPF ..... Energy per Frame

FPS ..... Frames per Second

GPU ..... Graphical Processing Unit

HD ..... High Definition

HDMI ..... High Definition Multimedia Interface

HP ..... High Power

HPC ..... Hardware Performance Counter

LP ..... Low Power

MMU ..... Memory Management Unit

MPEG ..... Motion Picture Experts Group

MVS ..... Motion Vector Search

OS ..... Operating System

PCI ..... Peripheral Component Interconnect

PERF ..... Performance Counters for Linux

PMIC ..... Power Management Integrated Circuit

PPW ..... Performance per Watt

QoS ..... Quality of Service

RAM ..... Random Access Memory

RTC ..... Real Time Clock

RTL ..... Register Transfer Language

SAD ..... Sum of Absolute Differences

SFU ..... Special Function Unit

SIMD ..... Single Instruction, Multiple Data

SMP ..... Symmetric MultiProcessor

SoC ..... System-on-Chip

TLB ..... Translation Lookaside Buffer

UEFI ..... Universal Extensible Firmware Interface

USB ..... Universal Serial Bus

VFP ..... ARM Floating Point Architecture

VLIW ..... Very Long Instruction Word

Part I

Overview

Chapter 1

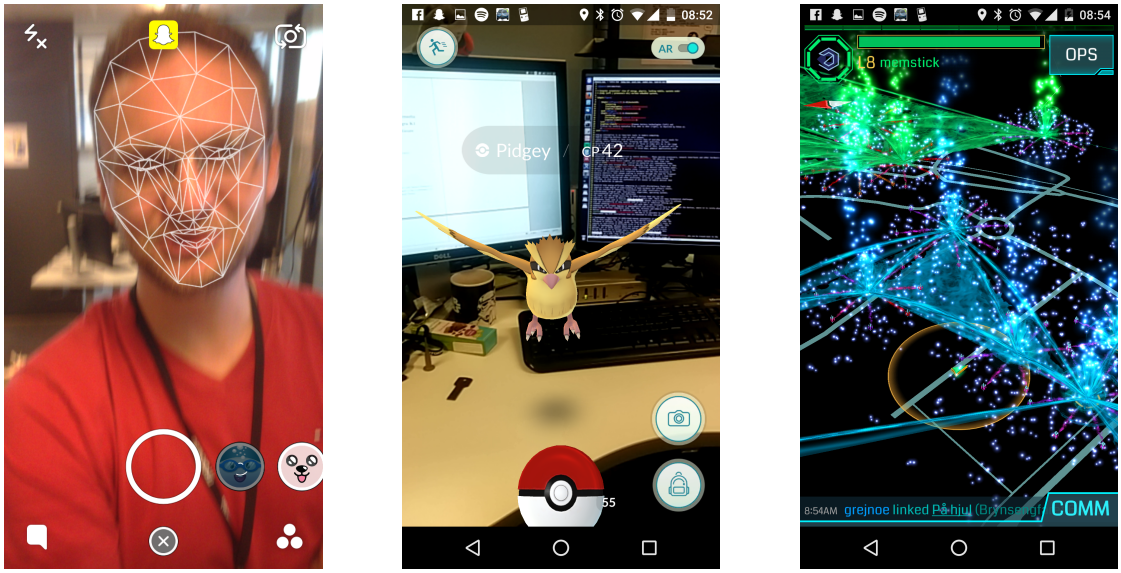
Introduction

Energy is an important resource in mobile environments, where the small evolution in battery energy density constrains the usefulness of devices such as smart phones, laptops, sensor nodes or even drones. Motivated by this limitation, many researchers attempt to analyse and optimise the power usage of for example multimedia systems on different mobile devices. However, it is difficult to gain insight into the power usage of different architectural units, such as different heterogeneous, multicore processors and memory, when limited to measuring the total power usage of the device. Power models are necessary to bridge the gap between software activity and the power usage of hardware, but unfortunately, the state-of-the-art methods that exist to model power all have individual weaknesses that limit their accuracy. In this thesis, we motivate and develop a high-precision power modelling method that can capture the power usage of a modern SoC with almost 100 % accuracy, and subsequently, use this model to analyse the power usage of a modern SoC. The remainder of this section is outlined as follows. Section 1.1 gives an introduction to the topic of energy modelling and motivates the need for improved power modelling methodologies. The problem statement of this thesis is given in Section 1.2. Limitations and research method is outlined in Sections 1.3 and 1.4, respectively. Finally, we state the main contributions of this thesis in Section 1.5.

1.1 Background and Motivation

Mobile computing devices, such as smart phones, sensor nodes, laptops or even drones fulfill many of everyday users' needs for gaming, social networking, working, browsing and other activites. Multimedia workloads account for a substantial part of the processing being performed on these devices. In fact, according to Cisco, by 2019 the sum of all forms of video will be in the range of 80 to 90 percent of global consumer traffic, and of this, 14 percent will be mobile data traffic [13]. Youtube is an example application that incurs both heavy networking usage and video processing, where users can download and upload videos directly using their smart phones. Facebook is another application that automatically downloads and plays back videos from the news feed. The video processing in these applications is performed on dedicated hardware encoders and decoders (DSPs), which today typically support H.264 or Google's VP.8 encoding standards.

While integrated DSPs are able to encode and decode full High Definition (HD) video at framerates above 25 Frames Per Second (FPS), they only provide fixed functions to



(a) Snapchat. (b) Pokemon Go. (c) Ingress.

Figure 1.1: Games and social media applications that perform video processing.

the programmer. Therefore, additional rendering and processing in general purpose processors may be needed. Snapchat is here a good example (see Figure 1.1a). In addition to taking and transmitting single photos and short videos, the application has a face recognition feature and several *filters* that draw sunglasses and other dynamic elements on the person's face. This additional processing must be done on the device's application processors, unless the device has intrinsic DSP support for some of the features such as face recognition. Pokemon Go, a virtual reality game where users walk around the real world to catch make-believe animals, is hot in the news these days (see Figure 1.1b). Here, the creatures are drawn directly into a live video recording taken by the device's own camera. Ingress is another virtual reality game where 3D-objects are drawn directly on top of a real-world map (see Figure 1.1c) using GPS coordinates. While today's and future's mobile applications and services will continue to add new and original filters to process multimedia, these may not be supported by dedicated DSPs. As a consequence, some additional processing will have to be done on general purpose processors.

To provide programmers with a flexible computing environment, modern mobile devices implement powerful application processors and multimedia accelerators, such as DSPs, integrated on a SoC. The SoC also implements many other components, such as USB host controllers, various data buses (camera, RAM, I2C), Power Management Integrated Circuits (PMIC), clock generators and voltage sensors. The Tegra K1 [43], for example, is a SoC meant to be used in mobile devices. This SoC is used as a case study in this thesis because it provides excellent programmability and flexibility for multimedia processing through many heterogeneous cores and DSPs. It has two main Central Processing Units (CPU), with a dedicated Low Power (LP) core and a quad-core High-Performance (HP) CPU for demanding workloads. The operating system and applications can be migrated seamlessly between the LP core and the HP cluster, and all five CPU cores support Single Instruction, Multiple Data (SIMD) instructions that accelerate multimedia processing. In addition to this, the Tegra K1 has a fully programmable, 192-core

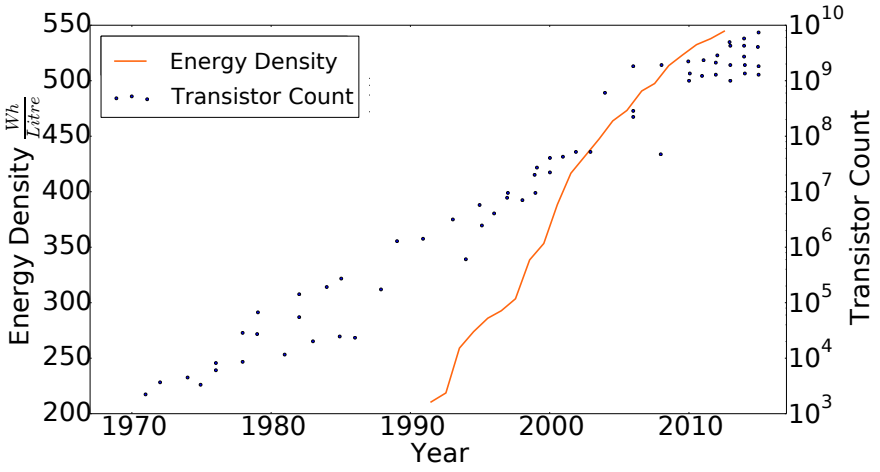


Figure 1.2: Evolution in the number of on-chip transistors between 1970 and 2015 and lithium-ion battery technology between 1990 and 2012 (data as provided by Rohan et. al. [55]).

Kepler-based Graphical Processing Unit (GPU) for computationally heavy multimedia processing. The Tegra K1's GPU is programmable through the Compute Unified Device Architecture (CUDA) framework. On the Jetson-TK1 development kit we are using, the Tegra K1 is also equipped with 2 GB of DDR3 RAM. The clock frequencies of the various components, such as the LP core, HP cluster, GPU and memory and over 20 other components can be dynamically adjusted to meet performance and power requirements of applications. This offers system architects and programmers with a range of design choices as to what cores should process different multimedia workloads, and which operating frequencies should be used.

The Tegra K1's rich and flexible computing environment, however, also comes with concerns of a limited energy supply. The isolated Tegra K1 SoC, without a network interface or a display, can for example draw up to 10 W alone under heavy computation, which would deplete a normal cell battery of 15 Wh in little more than an hour. The high power usage of the Tegra K1 and other SoCs may be attributed to the rapid evolution in on-die transistor density. In 1965, Gordon Moore published a paper [37] where he predicted that the number of transistors in an integrated circuit would double every year. This prediction has turned out to be relatively accurate until around 2012 (see Figure 1.2). The transistor gate widths are reduced, increasing the potential number of transistors that can reside on a die. This allows for more complex, higher-performance circuits to be implemented on a die with components such as CPUs, GPUs, DSPs, caches and memory. However, this also increases the power usage caused by switching activity and leakage currents. Additionally, as transistor gate widths decrease beyond 250-300 nm, transistor leakage currents increases significantly and becomes a major obstacle for integration [30].

While the potential number of transistors on a die has increased by a *magnitude* of three between 1991 and 2012, the evolution in energy density of lithium-ion batteries has only tripled [55] (see Figure 1.2). This does not mean that the power usage of SoCs has increased by a magnitude of three; but it does show that the current battery technology has not been able to keep up with the developments in large-scale integration. It is difficult

to state the design choices that the SoC vendors make in the face of this challenge, but it is likely to be a compromise between the number and type of transistors, what types of compute capabilities to provide and power usage, not to mention costs. What is easy to see, however, is that the uneven scaling between performance, power usage and energy storage is evident through the need for frequent battery charging and external battery packs to power all the user's need for gaming, social, browsing and other activities. In scenarios where for example a user or a drone is filming a football game or a concert, one can imagine several interesting video processing *filters* that post-process a live, raw video stream to for example increase the visibility of the football or perform debarreling of image frames to compensate for lens distortion. In terms of the Tegra K1, the challenge for system architects and developers is to decide what cores should process the different filters, and what platform frequencies should be used, to meet timing constraints while using as little energy as possible.

The need to understand the energy consumption of modern, mobile devices and increase power-efficiency has become increasingly apparent to researchers in computer science in recent years. However, conducting research in this field is challenging for a number of reasons. Research into energy-efficient computing is a multi-disciplinary field that, in its entirety, requires a high degree of understanding of physics and energy dissipation in transistors, battery chemistry, the complexities of operating systems, software applications and instrumentation. Perhaps the most complex issue is the lack of ways to *measure* the power usage of a device, and to bind this power usage to software activity. The availability of sensors to measure power is usually restricted to development versions of devices and a few smart phone vendors, such as HTC and Google. Research has shown that the quality of such sensors, for example in terms of their sampling rate and accuracy, is lacking [15]. Additionally, such sensors are usually restricted to measuring the total power usage of the device. This hinders insight into the actual power usage of the device's subcomponents, such as the CPU, GPU, display and RAM. Manually instrumenting production devices can reveal the power usage of these components, but this is usually impossible due to the board layout, and at best cumbersome, for example by inserting current sense resistors [20] in series with the device's internal components [10].

Challenged with the problems of measurements, researchers have in the past ten to fifteen years resorted to moving away from measurements and rely on *models* for power. These are completely necessary to bridge the gap between software activity and energy consumption of a device. It is for example not feasible to measure directly an individual process' power usage on a CPU. An application may utilise one of several cores, caches and buses integrated on a SoC. These are impossible to measure directly, and even if it was possible, it would require cost-expensive sensors with sampling rates capable of capturing the power usage of instructions that elapse in a matter of microseconds. For these reasons, researchers have in recent years resorted to *modelling* techniques that capture the power usage of a platform as a function of hardware activity. There are three common modelling methodologies found in the literature; state-, rate- and CMOS-based models. Of these, rate- and CMOS-based models are the most commonly used to describe the power usage of SoCs. Through our initial research, we have discovered that these can mispredict power usage of a platform such as the Tegra K1 substantially.

Rate-based power models [15, 28, 34, 70, 71, 75] are common in the literature. These assume that power grows proportionally with hardware activity. Hardware activity can

for example be instruction throughput or CPU utilisation. While such models have been shown to yield relatively good accuracy above 90 %, rate-based power models have a fundamental weakness in the way power is modelled. Power is assumed to grow proportionally with hardware utilisation, but the models neglect changes in voltage. As for example CPU frequency is increased, the CPU voltage also increases to sustain the current throughput of the CPU's circuitry. Voltage changes have dramatic effects on dynamic power, which grows proportionally with the square voltage. In effect, rate-based models can therefore mispredict power usage substantially. In our own research [64, 65], we have studied the effects that frequency scaling has on the power estimation accuracy of common types of power models. Not surprisingly, depending on the Tegra K1's clock frequencies, the estimation error compared to real measurements can be substantial up to 20 % on the CPU, and 50 % on the GPU. It can only be speculated that the reported estimation accuracies above 90 % are because the devices' frequency scaling algorithms keep the clock frequencies on the platform at similar levels, when subjected to model training and verification software activity. While this may not be a problem as long as the device under study is operating under the default Dynamic Voltage and Frequency Scaling (DVFS) algorithms, the lack of operating frequencies and voltage in rate-based models render them useless to yield any insight into how DVFS affects power usage on a platform. The inaccuracy of rate-based models is not only limited to frequency settings. Without exception, they also ignore the effects of core and rail power gating [61], variable cost of executing different instructions and contention of hardware resources such as caches [4].

In contrast to rate-based models, *CMOS-based* models take voltage into account and correlate power with the *operating frequency* of components such as processors and memory. There are few such models, but some have been proposed [11]. While being theoretically well-founded compared to rate-based models, CMOS-based models do not consider in detail the hardware utilisation of, for example, CPU caches and different arithmetic-logic units. The cost per clock cycle will vary depending on how different software workloads excersise the underlying hardware components. As for example CPU clock-frequency increases, the memory utilisation may also increase as a result, but CMOS-based models do not account for this. By correlating power usage directly with only clock frequency, they implicitly assume that the increase in power as for example CPU frequency increases, is solely due to increased power usage of that component. The error of these models can, because of this, be 20 to 30 % on the Tegra K1's CPU, and up to 50 % on the GPU, and as for rate-based models, the estimation error varies significantly depending on operating frequencies. *State-based* models is another modelling methodology. Here, the power usage of components such as wireless interfaces and processors is associated with the state that they are in, such as whether the processor is on, sleeping or off. However, these models neglect both voltages and dynamic power completely and, as a result, can mispredict power with up to 60 %. It is clear that there is a lack of power models that have been extensively verified over the possible ranges of operating frequencies on a platform, and that successfully capture the detailed power usage of individual components on the device such as the CPU, GPU, memory and their subcomponents, such as compute logic and caches. This is necessary to bridge the gap between power usage and the unique ways that different applications excersise the underlying hardware. In this thesis, we aim to investigate the power usage of a modern SoC such as the Tegra K1 and develop power models for its various computational elements, that can successfully reflect accurately not

only the total power usage of the platform, but also how an application consumes energy in these individual elements.

1.2 Problem Statement

Modern SoCs for mobile devices, such as the Tegra K1, provide developers and system architects with a flexible multimedia-processing environment through several heterogeneous, multi-core processors and DSPs. Mobile devices such as smart phones, laptops or even drones, however, consume non-negligible energy and are limited by the lack of high-density energy storages, restraining the usefulness of such devices. Power models are necessary to bridge the gap between software activity, frequency scaling, gating mechanisms and the energy consumption of SoCs, where physical measurement is otherwise impossible. This can be used to estimate the energy consumption of processes, identify candidates for energy-optimisation and eliminate the need for cost- and space-expensive energy consumption sensors. Existing modelling methodologies, such as rate- and CMOS-based models, have never been verified over the span of possible operating frequency ranges, and consequently end up mispredicting energy consumption in the various components of the SoC. Rate-based models, which are commonly used on devices such as smart phones, are especially susceptible to this problem because their accuracy depends on the operating frequencies that are in use on a platform. Additionally, rate-based models ignore changes in voltage that occurs as a result of frequency scaling. CMOS-based models capture changes in frequency and voltage, but fail to consider the power usage at the fine-grained level of, for example, floating point conversion instructions, cache hierarchies and power gating mechanisms. It is therefore a clear need for a new power modelling methodology that is able to bind the power usage of heterogeneous processors and memory to software activity with higher accuracy than state-of-the-art methods.

In this thesis, we use the Tegra K1 SoC as a case study. This is partly due to its compute capabilities. It has many heterogeneous cores and power management mechanisms that enable us to investigate in detail how different software workloads consume energy on different processors and hardware configurations. While we consider only one SoC, we have worked on several heterogenous processors, such as the Tegra K1's LP CPU core, the HP CPU cluster, the GPU and memory. The Tegra K1 is also the first mobile SoC to provide a CUDA-programmable GPU. Other SoCs also have GPUs, but the only method of programming these was through framworks that were not originally intended for general purpose programming, such as OpenGL [22, 54, 73]. Additionally, as explained in the next section, working on one SoC has allowed us to delve deeply into the power usage of one platform, and to discover how complex this topic can be. We investigate whether more accurate power modelling methodologies can be developed for the Tegra K1's heterogeneous processors and memory under the following hypotheses:

> **Hypothesis 1:** To achieve a high degree of accuracy, a power model for the Tegra K1 SoC must combine chip-, hardware- and software-level knowledge into dynamic and static power terms for all of the SoC's architectural units.

> **Hypothesis 2:** The average capacitive load per CPU instruction remains the same over time for repetitive workloads.

We have approached the problem of developing more precise power models for the Tegra K1 in the following steps:

- We show that existing modelling methods are not accurate and fine-grained enough to estimate in detail the power usage of individual components of the Tegra K1 SoC, such as the two CPUs, GPU and memory.
- Combining CMOS- and rate-based models, we develop a more accurate power model for the Tegra K1 by accounting for frequency scaling, variable voltages and more fine-grained accounting of hardware activity and power management mechanisms such as clock-, core- and rail-gating.
- Given the lack of Hardware Performance Counters (HPC) to measure the number and types of instructions that are executed on the Tegra K1's CPU, we show that it is possible to estimate the CPU's instruction costs on a per-process basis.

The focus of this thesis is to show how we can use all the heterogeneous processing elements, such as the CPU, GPU and memory, as well as power management mechanisms, such as frequency scaling, on the Tegra K1 to energy-efficiently process multimedia workloads while meeting Quality-of-Service (QoS) constraints, such as a framerate. In this aspect, we use our developed power model to investigate the power usage of the SoC in terms of the following steps:

- We show how frequency scaling affects the total energy consumption of multimedia workloads operating under some given QoS constraint, such as a framerate, and how the Tegra K1's heterogeneous processors are energy-inefficient at high processor frequencies.
- Given that the Tegra K1's heterogenous processors are more energy-inefficient at higher frequencies, we perform preliminary experiments to energy-efficiently balance multimedia workloads across these.
- Through our power model, we show how we can analyse the total power usage of the Tegra K1 SoC, for example in terms of static and dynamic power components in its processors and memory.
- The Tegra K1's CPU and GPU provide the developer with different types of instructions and cache modifiers. We show how these can be used to further reduce the energy consumption of multimedia workloads.

1.3 Limitations

In our study, we have only used the Tegra K1 as a case study of mobile SoCs. This is because it takes time and effort to understand and model power accurately. There are many mechanisms, and indeed bugs, that are functionally transparent from a programmer's perspective, but visible when modelling power. For example, the Tegra K1 increases the CPU voltage when temperatures fall below a specific point, effectively increasing CPU power usage. This occurred randomly above a ventilation vent in our server room, which

we used to avoid changes in static power as a result of temperature variations. Some CUDA libraries disable power management on the Tegra K1's GPU, meaning that the GPU rail will never be suspended, and the GPU clock is never gated. This will not cause CUDA programs to fail, but we experienced that our model's accuracy decreased. While confirmed by NVIDIA as a power management bug, we have never received any fix to this problem. These issues are indeed very hard to see without physically measuring rail voltages. For this and several other reasons, we have limited our study to the Tegra K1. While this has only allowed us to focus on one SoC, it has enabled us to go extremely deep into this architecture. However, we believe that our methods, model predictors and experiences can be of use to model future SoCs and further investigate the generality of our results.

The software workloads introduced in this thesis are strictly multimedia applications that process live video. In part, this choice was made because video processing is arguably a substantial part of modern, mobile computing where many applications are manipulating live video feeds from the network or a camera. However, they also have the advantage that they have a QoS constraint in terms of a framerate that they must reach. This allows us to measure the workloads' performance when we tune, for example, platform frequencies on the Tegra K1, while at the same time observing the changes in total energy consumption. As the filters are essentially performing the same work over and over on consecutive frames, this also allows us to verify whether the average capacitive load per CPU instruction remains the same over time.

This thesis focuses on mobile devices. Stationary devices and desktop environments are left out in a large part due to time constraints. However, working on mobile SoCs offers some advantages that simplifies research in this field. At the SoC level we are working very close to bare-metal hardware. There is no BIOS or UEFI handling low-level device management and everything is essentially running in the Linux kernel that is mostly open-source. For the Tegra K1, some driver code handling the GPU is, of course, proprietary, but the Jetson-TK1's circuit layout is open to the public. In a stationary computer, the actual equivalent to the Tegra K1 SoC would be the main processor circuitry, but many functional parts such as external peripherals, north- and south-bridges, memory controllers etc. would be scattered across the motherboard and potentially hard to monitor in terms of hardware and software activity. Furthermore, energy is a fundamental issue in mobile environments where users rely on batteries to function as long as possible.

1.4 Research Method

In 1989, the ACM Education Board approved a report [14] made by the Task Force on the Core of Computer Science that characterised the structure of computing, and how researchers in this field approach their work. In this report, it was recognised that computer science is essentially at a crossroads between the central processes of applied mathematics, science and engineering, reflected in the paradigms of *theory*, *abstraction* and *design*. Theory is concerned with characterising the objects under study, formulate and hypothesize possible relationships among them, determine whether the relationships among them are true and then interpret the results. Abstraction, or modelling, is rooted in the experimental scientific method. Through the investigation of a phenomenon, the

researcher forms an hypothesis, constructs a model, designs experiments and collects data, and then analyses the results. The third paradigm, design, has its roots in engineering. For a given problem, the researcher states requirements and state space for a solution, designs and implements the system, and then tests the system.

The three paradigms in computer science are intertwined.For example, in the course of our work on this thesis, a substantial amount of design has been done to implement the various systems to measure hardware activity in processors and memory, to synchronise external power measurements with the events on the Tegra K1's processors, to process video and develop prototypes for live video encoding on the Tegra K1's computational elements [62]. Requirement and state space engineering has been an integral part of this process. However, the various pieces of software we have built is not the main contribution of this thesis, although they have been necessary to reach our conclusions. The paradigm of theory shares some similarities with our work. Indeed, we have characterised the objects under study, the Tegra K1's two CPUs, GPU and memory, and we have formulated a possible relationship between the hardware and software activity in these units, their supply voltages, and the total energy consumption of the platform in both state-, rate- and CMOS-based power models. However, we do not seek to conclusively prove these relationships, for example for all SoCs in the world. Many of the parameters we use to measure hardware activity, such as active memory cycles spent serving GPU requests, are possible because the Tegra K1 has intrinsic support for them. It is not given that these will be available for different SoCs. Finally, the paradigm of abstraction is the one that shares most similarities with our work. Our contributions in this thesis have indeed been founded on our hypothesis, which is that the accuracy of power models can be improved by compensating for variations in rail voltages and correlate dynamic and static power with fine-grained hardware activity measurements in all parts of the SoC. The innovation lies in the addition of voltages in the equation. Coupled with fine-grained hardware- and software-activity measurements in the Tegra K1's CPUs, GPU and memory, it is *formulated* into an original model for power usage. The model is subsequently trained, and the number of electrons being switched through the circuitry on various hardware and software events and leakage currents is estimated. The model is subsequently subjected to extensive *verification* over all possible platform frequencies with several workloads. Finally, we *analyse* the estimation accuracy of the model and use it to study the power usage of various multimedia workloads.

1.5 Main Contributions

The main research question in Section 1.2 states the challenge of utilising the Tegra K1's hereogeneous multicore processors to process multimedia workloads in an energy-efficient manner. In our initial work [61], we experimented with a video encoder ("Codec 63"), several video filtering operations such as frame rotation, Huffman encoding, the Discrete Cosine Transform (DCT) and Motion Vector Search (MVS), and the effects that frequency scaling had on the energy-efficiency of these multimedia operations. We have also investigated in detail the energy consumption of video encoding, subject to framerate constraints, over different types of DVFS algorithms [58]. In our live demonstration [62], users could dynamically adjust processor frequency, choose the CPU to use and the number of active cores, as well as several task partitioning schemes (CPU-only, GPU-only and

hybrid) while the Tegra K1 was encoding a live video feed. The change in performance (framerate) and energy consumption was displayed live on a dedicated computer. From these and other studies [61], we observed that a good heuristic when performing the DCT, MVS, debarreling or rotation on a video feed was to minimise processor frequency such that application requirements (for example a framerate) was just met. In Section 2.5, we have extended these results to also minimise memory frequency, finding that between 10 to 40 % energy can be saved when minimising processor and memory frequency compared to the standard DVFS algorithms. But *why* is this such an effective strategy? What is physically happening within SoCs that makes frequency minimisation such an effective strategy?

The aforementioned questions are demanding to answer. Very little can be said about the energy consumption of a SoC's internal logical circuits and compute elements, as it is essentially a "black box" containing potentially billions of transistors. The last point where it is physically possible to measure the energy consumption of these elements are the *power rails* providing the various components, such as CPUs and GPUs, with energy. This is also rarely feasible in practice, however, and researchers are therefore limited to measuring the total power usage of the SoC. Our master student, Øyvind Skjøld [57], built a rate-based power model for the Tegra K1 and demonstrated through experiments that it had the potential for substantially mispredicting power usage by up to 50 % in different hardware configurations. This was in accordance with our findings in this thesis, where we show that state-of-the-art power modelling methods have the potential for serious misprediction of power usage over different processor and memory operating frequencies. This thesis' main contribution is the introduction of a power modelling methodology that is able to estimate running SoC power usage with close to 100 % accuracy [64, 65]. The improvement over state-of-the-art methods, that can incur estimation errors between 10 to 70 percent, was made possible by two main contributions. First, we correlate transistor switching activity (dynamic power) with fine-grained hardware activity measurements. Our power model for the Tegra K1 captures dynamic power at a granularity that has never been done for any SoC:

- We measure the number and type of instructions executed on the GPU, for example integer, single- and double precision and conversion instructions, as well as GPU cache utilisation.
- We associate cache utilisation and application-specific instruction power on the CPU, which has only one single generic instruction counter.
- We measure the number of active memory cycles spent serving CPU and GPU memory requests through our own kernel tracing framework.
- We measure active LP core, HP cluster, GPU and memory cycles, subject to clock-gating.
- We consider rail voltages of the LP core, HP cluster, GPU and memory, based on the current platform frequencies in use.

This allowed us to estimate in detail the dynamic energy consumption of running processes as they execute code on both the Tegra K1's CPU and GPU, as well as the

pressure on external (off-chip) memory. Second, our model also considers leakage currents. We have developed a special kernel tracing framework that is able to track when and for how long individual CPU cores are power-gated. Combining this with rail voltages and rail-gating, we are able to estimate leakage currents of all the four main power rails on the Tegra K1 as well as individual CPU cores. Finally, our model has been verified over all CPU, GPU and memory operating frequencies, which has never been done before for any other power model, and demonstrated to yield significant accuracy over all of them. By taking all these factors into account, our model yields unprecedented insight into the energy consumption of a complex SoC. We have also presented our model to Nvidia's Tegra product management team [59, 60].

To demonstrate the insight given by our power model, we have also applied it to our video processing filters through a series of extensive experiments. First, we demonstrated that the increase in energy-efficiency as platform frequencies are reduced such that the running application's QoS requierments is met, is mainly due to reduced rail voltages. However, our model also revealed another critical insight: the number of processor cycles required to complete an instruction increases on some processor and memory frequency combinations, additionally reducing energy-efficiency. This detail would be impossible to see without our fine-grained power model. Second, we took the role of a system architect and investigated the power usage of a running idle system. We identified the critical power components, such as memory clock power, which is never clock-gated, and demonstrated how our methodology can be used to estimate idle instruction power and optimise system services and drivers. Third, we showed that, in their evaluation of energy-efficiency, many researchers do not consider the effects of *base power* in their experiments. Consequently, they risk optimising away base power that it is impossible to avoid, and reduce the energy-efficiency of the actual workload. Finally, as we observed that the Tegra K1's processor cores are energy-inefficient at higher frequencies, we exploited this by offloading video processing between the Tegra K1's CPU and GPU. By offloading workload from the GPU, we could for example reduce its processor frequency and demonstrate an additional 5 % energy saving per processed frame.

1.6 Outline

The outline of this thesis is as follows. To understand the power model development in later sections, we introduce the Tegra K1 mobile SoC as a generic multimedia processor in Section 2. We outline the power distribution circuitry, as well as the effect of DVFS, on our Jetson-TK1 development platform. Furthermore, we introduce our multimedia workloads as well as preliminary experiments with frequency scaling and energy consumption. Section 3 is a literature review for state-of-the-art power modelling methodologies. We outline the three common methods; state-, rate- and CMOS-based models, and also investigate the accuracy of such models on the Tegra K1. In Section 4, we motivate and introduce our main contribution, that is, the high-precision power modelling methodology. We outline the various predictors of the model, which includes fine-grained hardware- and software-activity measurements, power- and rail-gating statistics and rail voltages. Furthermore, our methodology to train the models is introduced, which includes specific model training benchmarks that stress specific components of the Tegra K1's CPU, GPU and memory. We conclude the chapter by verifying that the accuracy of the model is

close to 100 percent over all processor and memory frequencies. In Section 5, we use our multimedia workloads as case studies and investigate the power-inefficiency that occurs at high frequencies. Additionally, we use our model to analyse and optimise idle system power usage. Finally, Section 6 concludes the thesis.

Chapter 2

The Tegra K1 as a Mobile Multimedia Processor

In this chapter, we introduce the Tegra K1 mobile SoC, which is integrated on a Jetson-TK1 development kit, as a mobile multimedia processor. In Section 2.1, we outline the Tegra K1 device architecture in terms of its power distribution framework, which is important to understand our modelling efforts presented in Chapter 4. An introduction to power measurement methods, and our solutions for power measurement on the Jetson-TK1 development kit, is given in Section 2.2. In Section 2.3, we introduce the fundamental CMOS equations. These relate static and dynamic power usage of transistors with platform voltages and clock frequencies. They are important to understand both our evaluation of state-of-the-art power modelling methods in Chapter 3, and the foundations of our modelling methodology in Chapter 4. We also verify the CMOS equations with power measurements. In Section 2.4, we introduce our video processing filters that are used throughout the experiments in this thesis. We study the effects that frequency scaling has on the energy consumption of our video proessing filters in Section 2.4. We summarise this chapter in Section 2.6.

2.1 Device Architecture

The Tegra K1 is a mobile SoC featuring a range of heterogeneous, multicore processors, DSPs and various peripherals. In this section, we give an introduction to the hardware architecture of the Tegra K1, and how power is distributed to the various hardware resources on the SoC. This is important to understand the various compute capabilities that the platform provides, as well as our modelling efforts in Chapter 4.

2.1.1 Functional Description

The Tegra K1 is a SoC that provides many heterogeneous, multicore processors to the programmer. It features a CUDA-programmable GPU and two CPUs split into a low-power core (LP core) and a quad-core high performance compute cluster (HP cluster). On the Jetson-TK1, it is additionally equipped with 2 GB of DDR3 RAM (see Figure 2.1). The LP core and the HP cluster are seen as two separate CPUs, where processing is only restricted to one of them at any point in time. The Jetson-TK1 is running Ubuntu 12 with

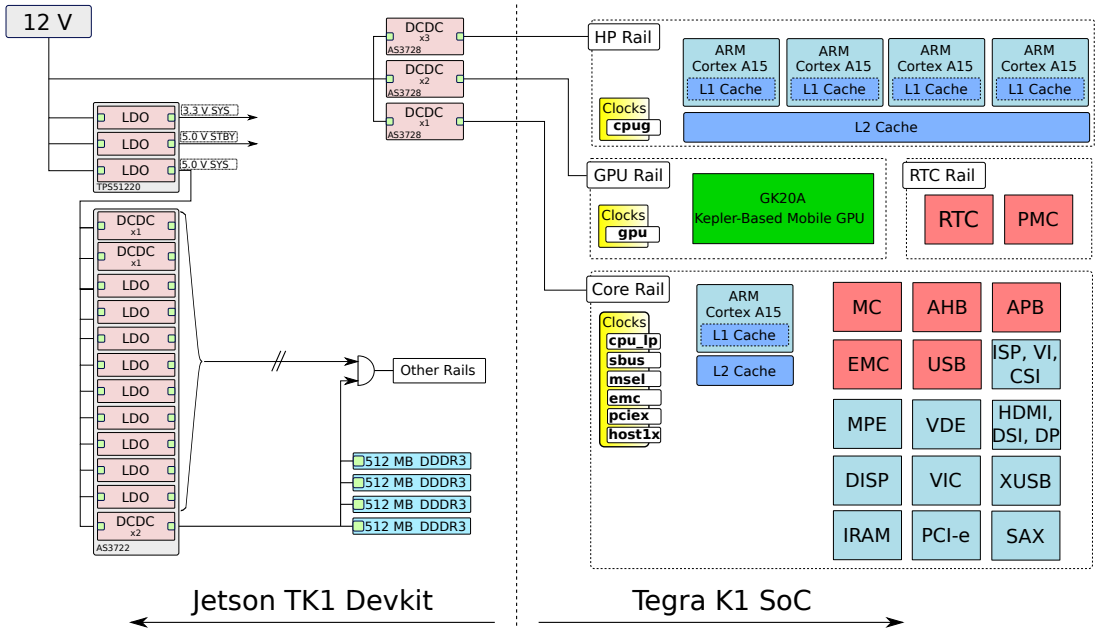


Figure 2.1: Jetson-TK1 device architecture.

Linux as its default Operating System (OS), where Linux kernel drivers manage migration between the CPUs. The CPU cores are nearly identical ARM Cortex-A15 cores. There are only small differences between these in terms of external memory access latency (three cycles instead of two [44]), as well as smaller L2 cache (512 MB vs. 2048 MB). The Kepler-based NVIDIA GPU is a fully programmable processor consisting of 192 compute cores. This gives developers good flexibility to run workloads on different, heterogeneous cores. Furthermore, the operating frequencies of the LP core, HP cluster, GPU and memory can be scaled to meet power and performance requirements. From a power management perspective, it is therefore an interesting SoC because developers and system architects have good flexibility to schedule their workloads on several processors, while at the same time fine-tuning frequency levels in all components of the SoC.

2.1.2 Energy Distribution of an Island Architecture

The Tegra K1 [44] implements an *island-style* device architecture (see Figure 2.1). From the perspective of an engineer integrating the SoC on a device, it is just a small chip with many input pins. Most of these pins are buses, or signaling pins, for communication with external devices. Some of them, however, power the various subcomponents of the SoC, such as its two CPUs, GPU and RAM. These are referred to as individual *rails* supplying these units with power. The Tegra K1 has five main power rails. These, and other, less important rails are part of a complex *power distribution circuit*, which is implemented by the PMIC. The PMIC manages power on the platform. It is not integrated on the Tegra K1 SoC itself, but is built around the SoC as part of the Jetson-TK1's board layout. The PMIC in turn takes an input voltage, which for the Jetson-TK1 is 12 V, and downconverts it to the voltages required for the various power rails of the SoC. The downconversion is done by *regulators* [32], which can be of two types: DC-DC (buck, or switching, converters) and Linear DropOut (LDO) regulators. The regulators waste

energy as heat in the downconversion. While we in this thesis do not model these effects, such as for example Castagnetti et. al. [11] and Lee et. al. [32], it is important to be aware that the loss is there. Buck-regulators can have superior energy-efficiency, which depends on its output current and configuration, but the output voltage is noisy because energy is switched out of an inductance in specific intervals that decide the output voltage. The energy loss of LDO-regulators is always proportional to the current draw, but the output voltage is less noisy. This makes LDO-regulators necessary for some of the more delicate electronic components.

As mentioned above, the Tegra K1 has five power rails. The *RTC rail* is *always* on. It powers an internal PMIC on the Tegra K1, which is unfortunately poorly documented, and a Real-Time Clock (RTC). The internal PMIC most likely controls at least power-gating to the CPU cores, and maybe also the Tegra K1's GPU. A *core rail* powers the LP core and its L1 and L2 caches, the boot processor, IRAM, a range of over 20 different clock generators, DSPs and various peripherals such as USB host controllers and the PCI-express root complex. It also contains the External Memory Controller (EMC), which arbitrates accesses to off-chip memory. The core rail is the most *complex* rail in terms of the high degree of functionality integrated on it. This rail is also always on. There is support to turn off more units within the core rail, reducing leakage and clock power, by cutting off the voltage supply to additional "subrails" (**Other Rails** in Figure 2.1). Additionally, the core rail supports a deep-sleep mode where the rail is turned off, and waked up after a predefined amount of time from the RTC rail. These functions, however, are out of scope for this thesis. The *HP rail* powers the HP cluster and its clock generator. This rail is turned off when processing is restricted to the LP core. Otherwise, it is on and supplying the HP cluster with energy. The process of dynamically shutting off supply voltage to electrical circuits, such as the entire HP rail, is commonly referred to as *gating*, or more specifically in this case, rail-gating. Similarly to the HP rail, the *GPU rail* powers the Tegra K1's GPU. The GPU rail is also normally rail-gated when the GPU is idle for a predefined amount of time. This duration can be specified in *sysfs*. Sysfs is a RAM file system for kernel parameters in Linux. However, the Tegra K1's GPU rail is subject to a power management bug that disables power gating on that rail (see Section 4.2.2). For this reason, unless otherwise specified, the GPU is always on and consuming GPU cycles throughout the experiments in this thesis.

2.2 Power Measurement

The most basic requirement to understand energy usage of devices is to measure power. This process is at best cumbersome with modern devices. There is generally a lack of sensors to do this, forcing researchers to resort to complex laboratory setups and advanced techniques to overcome problems related to for example synchronisation [53]. In this section, we outline the most common approach to measurement, the sense-resistor, used in both laboratory setups and production devices. We also discuss the technical challenges we encountered in using production-device sensors, alternative approaches and design-tradeoffs. We outline the solutions we have used to measure power usage on the Jetson-TK1.

2.2.1 Current Sensing

By far, the most common approach to measure power is based on inserting [53] or using existing sense resistors [10] in series with relevant electrical components, such as the Tegra K1's CPU, GPU and memory rails. The voltage drop across it, U_{sense}, can then be measured using an Analog-to-Digital (A/D) converter. Given the sense resistor value R_{sense}, instantaneous power can be calculated by straightforward application of Ohm's law:

$$P_{rail} = U_{rail}\frac{U_{sense}}{R_{sense}} \tag{2.1}$$

where U_{rail} is the rail voltage. Many authors of power models focus on production devices that have current sense capability. Typical smart phone examples that have such sensors are HTC and Google, but these sensors are only capable of measuring the total power usage of the board. Dong and Zhong [15] and Yung et. al. [28] show in their experiments with the Nokia N95, Lenovo ThinkPad T61 and Google Nexus One that such sensors can incur significant latency, and as a result, the power measurement samples as read by the device's CPU can be delayed by several seconds. This occurs because some sensors are actively averaging readings to achieve better accuracy, effectively behaving as a low-pass filter.

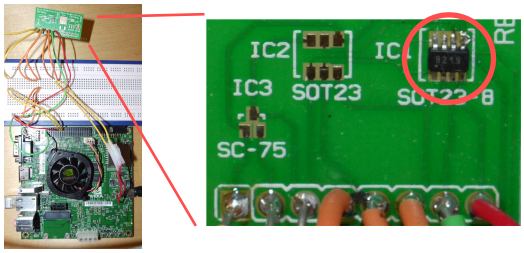
The common case, however, is that devices do not have integrated sensors, and more manual labour is needed. In their experiments on the OpenMoko Freerunner smart phone, Carroll and Heiser [10] show that it is possible to insert sense resistors in series with most of the device's power rails. They then use A/D converters to measure the voltage drop across these and are able to build a detailed breakdown of all power components in the system (CPU, GPU, DSP, RAM, WiFi etc.). However, it is rare for production devices' circuit layout to be built for this purpose. It is also challenging to open such devices without breaking them, and they cannot be used normally after this because of all the external wiring (see for example Figure 2.2a).

Most researchers have to use external measurement meters [34, 46, 69, 70] connected directly to the main power supply of the device. Monsoon power meters[1] are commonly used [71], but unfortunately, researchers often do not mention what type of measurement units they are using and other details that are important such as accuracy and sampling rate. An issue which is almost always neglected and raised by Rice and Hay [53] is that of measurement synchronisation. When using external measurement tools, the measurement readings must be synchronised with the events on the device, in particular the CPU. Rice and Hay propose to solve this by triggering a specific *pattern* in the power measurement log by turning on and off energy-consuming devices, which can be used for synchronisation (see Section 2.2.4 for details).

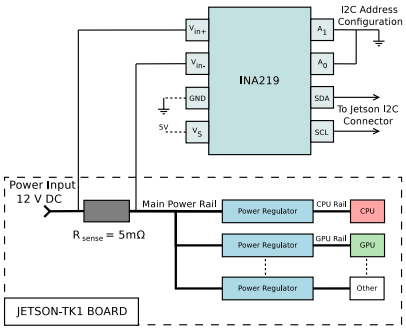
2.2.2 Other Approaches

Several other measurement methods exist, but compared to the sense-resistor based approach these are more rarely used. Xu et. al. [71] exploit the drop in battery terminal voltage as the battery load current changes, to estimate total device current using the

[1] `https://www.msoon.com/`



(a) A surface-mounted INA219 rail current sensor (circled).



(b) Circuit schematics.

Figure 2.2: Integrating the INA219 as a power measurement sensor on the Jetson-TK1.

internal battery voltage sensor. This scheme has the advantage that most battery-driven devices have voltage sensors. However, it is unclear what type of requirements this solution places on existing voltage sensors in terms of accuracy and sampling rate. Hergenroder and Furthmuller [20] present a range of different measurement methods intended for embedded devices. Dutta et. al. [16] propose to equip buck-regulators with a microcontroller to count the switching frequency of the converter, and thus be able to count the total energy switched into the circuitry. This solution has been used in some sensor node implementations such as Quanto [18]. Some manufacturers have also included their own solutions. For example, Nokia supported the Nokia Energy Profiler for their smart phones in earlier years, which has been used by many researchers [50].

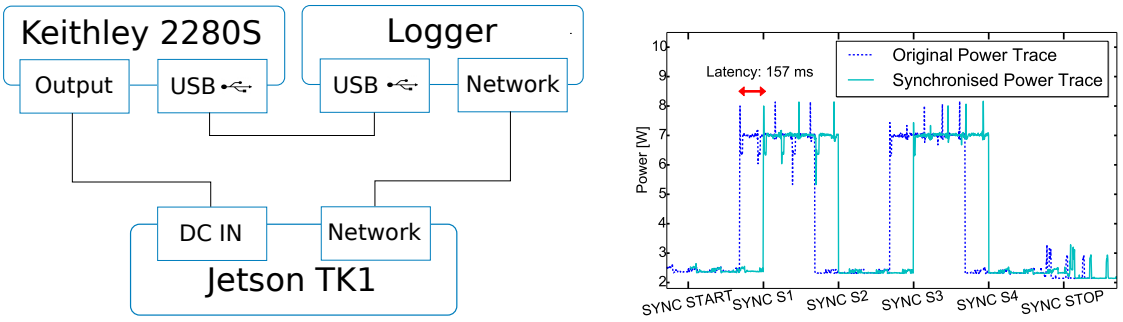
Instead of direct measurement, power can also be estimated based on other tools such as Wattch [8], SimplePower [72] and SimpleScalar [3]. Kalla et. al. [29] use publicly available Register Transfer Language (RTL) code to estimate the power usage of individual processor components - such as the core, MMU, register file, execution unit, integer unit and memory interface unit - of a Sun MicroSparcIIep processor. This solution has the advantage that it achieves very fine-grained insight into the power usage of a commercial processor. It is, however, not based on real measurements.

2.2.3 Failed Attempts

In our experience with many NVIDIA products, we have had several challenges working with integrated power measurement sensors. This is a short summary of these:

- Many of NVIDIA's desktop GPUs implement current sensors, monitoring power of the three PCI Express power rails. This is possible to see by dumping VGA BIOS tables [49], revealing that single-rail INA219 or triple-rail INA3221 sensors are often used. These can be programmed using NVIDIA's own virtual I2C-over-PCI-E framework. Attempting to do so however only resulted in hanging the host OS, and we have also experienced that a GTX 570 GPU broke from these attempts.

- The Kayla development kit for the Tegra 3 SoC, SECO mITX, contains kernel configuration data for INA219 sensors. However, as we were unable to communicate



(a) Measurement setup.

(b) Measurement synchronisation.

Figure 2.3: Keithley 2280S as a power measurement device.

with them, our support contact suggested that they were probably removed from the final hardware revision.

- The Jetson-TX1 development platform and the Shield TV (Tegra X1) has on-board INA3221 sensors. However, communicating directly with NVIDIA's own Jetson-TX1 development platform manager we found that these are not working on the current hardware revision, because the sense resistors are removed. The Shield TV is in turn challenging to use because there are no official kernel sources for it and more limited IO connections.

- The Jetson-TK1's power regulators have current sense capability. However, the datasheet was unavailable and closed to the public at the start of the work in this thesis. Additionally, the recently released datasheet does not provide the sense resistor value.

Due to our challenges with finding on-board measurement sensors, we therefore resorted to design our own power measurement solutions.

2.2.4 Solutions for the Jetson-TK1

As mentioned in the preceding sections, the Jetson-TK1 has current sensors integrated with the buck regulators supplying the CPU clusters and GPU with power. However, these can not be used because the sense resistor value is unknown. Initially, we used a custom solution with a single surface-mounted INA219 [66] (see Figure 2.2a) and an existing $5m\Omega$ sense-resistor mounted on the main power rail (see Figure 2.2b). This allowed us to measure the total power of the board by polling the Jetson-TK1's I2C interface. We later changed this solution to a Keithley 2280S which both sources and measures power of the board. In addition to having a better measurement error (0.02 %) and current resolution (between $10nA$ and $10\mu A$), this solution also removes the overhead of logging power during experiments.

Figure 2.3a shows how the Keithley 2280S is used to measure power of the Jetson-TK1. It can simultaneously measure output voltage and current, but this effectively halves the measurement rate because the 2280S has only one A/D conversion unit. Because the 2280S is configured to deliver a stable 12 V to the Jetson-TK1, and only the current varies

significantly, we disable voltage measurement to improve the reading rate. The 2280S is not designed to make long-running measurements, where current readings are stored in a ring buffer with space for 2500 samples. Therefore, an external machine (`logger` in Figure 2.3a) is needed to continuously retrieve these and store them locally using SCPI commands. Logging can be started, stopped and transferred remotely. Initially, we attempted to communicate with the 2280S over USB through the USBTMC protocol, but this often broke for unknown reasons. The 2280S however runs a Standard Commands for Programmable Instruments (SCPI) server on port 5051, and we managed to develop a workaround that communicates with this server instead. This effectively made the system more stable. The system is configured to provide one sample per millisecond.

As stated by Rice and Hay [53], synchronisation is needed to match the timestamps of the (external) power measurements with the events of the device under test. Our requirement is millisecond-synchronisation, where our workloads are always targeting frame processing deadlines between 20 to 40 ms. Inspired by Rice and Hay's solution we create a *power pulse* (see Figure 2.3b) which can be used to synchronise the logs. At the beginning of any experiment, the Jetson-TK1 starts the power logging remotely. Then, it starts the synchronisation process as follows:

1. At SYNC START in Figure 2.3b, the Tegra K1's CPU:
   - Minimises all clock frequencies on the platform.
   - Restricts processing to the HP cluster with all (four) cores on.
   - Initiates four threads that are doing dummy vectorised (NEON) floating point operations.
   - Sleeps for a predefined amount of time.
2. At SYNC S1, S2, S3 and S4, the Tegra K1 cycles the platform between high-and-low power states by maximising and minimising CPU and memory frequencies, respectively.
3. At SYNC STOP, the synchronisation pulse is done.

At the end of an experiment, the power log is retrieved from the logger. The Tegra K1 then scans the log for the synchronisation pulses, calculates the offset in time and makes the necessary shifts to the power log. For example, the offset in Figure 2.3b has been determined to be 157 ms. This solution only compensates for delay between starting the measurements on the Tegra K1 until they are physically starting on the logger. It can not compensate for clock drifts between the logger and the Tegra K1. However, we assume that this drift is negligible where our experiments are run for very short durations (typically less than one minute), and synchronisation is done for every experiment.

2.3 The Fundamental CMOS Equations

We have outlined how power is distributed through the Jetson-TK1's power distribution framework to the Tegra K1 SoC, and how we measure power using the Keithley 2280s high-precision measurement unit. Understanding how the actual SoC consumes power,

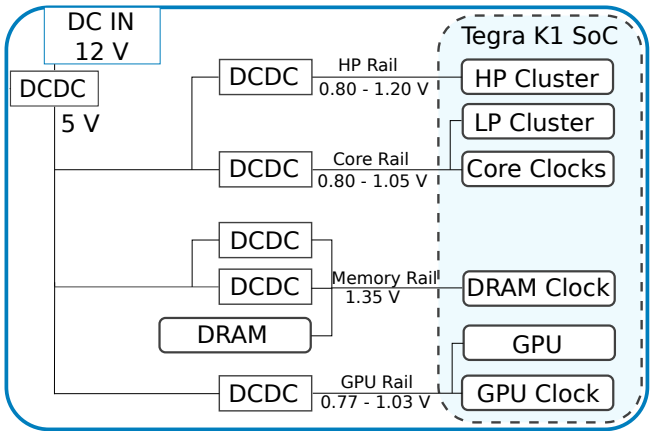


Figure 2.4: General overview of the Tegra K1 island-style architecture.

however, is challenging. This is not only because it is hard to measure the power usage on each of the Tegra K1's power rails, but also due to limited insight into how the Tegra K1 SoC functions internally. In this section, we will introduce and validate the fundamental CMOS equations. These equations describe the power usage of CMOS transistors in relation to operating frequencies and rail voltages on the platform, and are important to understand the clock architecture of the Tegra K1 as well as our modelling methodology in Chapter 4.

The power usage of CMOS circuits can be described as the sum of static and dynamic power [30]:

$$P_{cmos} = P_{stat} + P_{dyn} \tag{2.2}$$

Static power is caused by leakage currents in transistors and other electrical components. It is modelled as the product of the leakage current, $I_{R,leak}$, and voltage V:

$$P_{stat} = I_{leak}V \tag{2.3}$$

Dynamic power is caused by switching activity in transistors as these change state, for example on clock cycles, instruction execution, cache accesses and other forms of hardware utilisation. The canonical formula for dynamic power is as follows:

$$P_{dyn} = \alpha C V^2 f \tag{2.4}$$

where C can be viewed as the maximum capacitive load (in coloumbs per volt per cycle) being switching through a circuit f times per second operating at voltage V. $\alpha \in [0, 1]$ is an environmental factor which decides how much of the maximum capacitive load C is switched through the circuit at every clock cycle.

The Tegra K1 is an island-style SoC with several *rails* powering different parts of the system (see Figure 2.4):

- The *HP rail* supplies the quad-core HP CPU and its clock-generator.
- The *core rail* supplies the LP core and most of the Tegra K1's clock generators.
- The *GPU rail* supplies the GPU and its clock generator.

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	$[0.80]$
gpu	GPU	LP core	15	$[72, 852]$	$[0.79, 1.05]$

Table 2.1: The Tegra K1 clocks, voltage and frequency ranges.

- The *memory rail* supplies off-chip memory and its clock-generator.

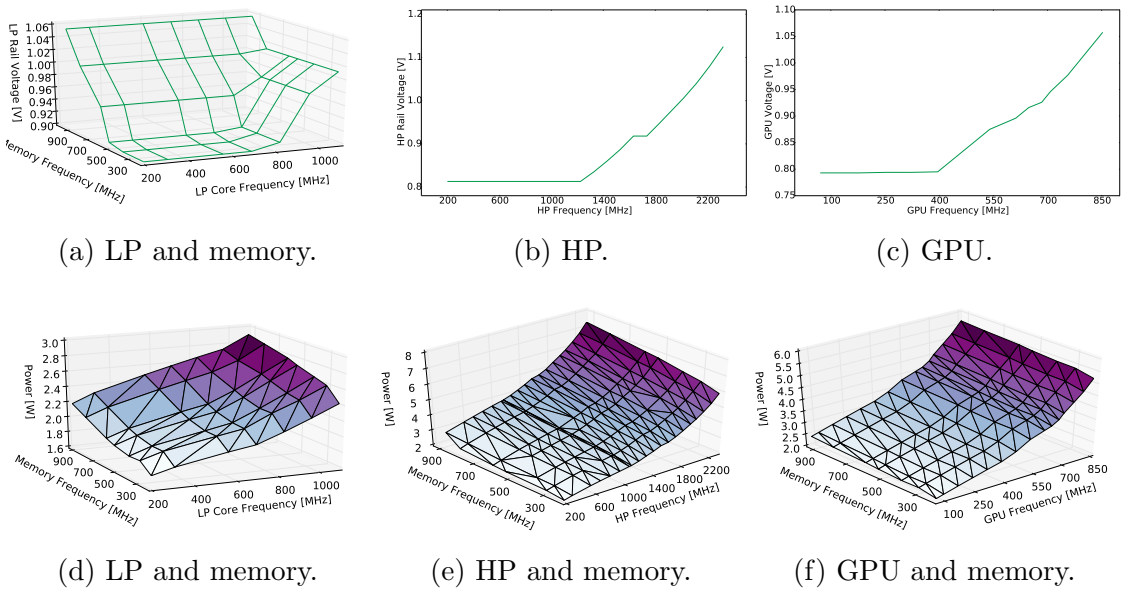
Equation 2.3 and 2.4 can be readily applied to these rails [11]. Their key insight is that rail power is proportional to the square rail voltage ($P_{rail} \propto V_{rail}^2$). For example, a doubling in voltage essentially increases dynamic power by a factor or four. Rail voltage in turn increases and decreases with clock frequency on that rail to sustain the current throughput:

$$f_R \propto \frac{(V_R - V_{th})^\alpha)}{V_R} \tag{2.5}$$

where f_R is the clock frequency on rail R, V_R is the rail voltage, V_{th} is the transistor threshold (minimum operating) voltage and α is an experimentally derived constant for the current technology [30]. The relation between operating frequencies and rail voltages for the Tegra K1 can be seen in Figure 2.5a for the core rail, 2.5b for the HP rail and 2.5c for the GPU rail. It is important to be aware that there is a lower threshold where supply voltage cannot be further reduced, even if frequency allows for it. The GPU rail voltage is for example never reduced below 0.77 V. This is because the transistors in the circuit cannot operate on lower voltages.

The relevance between the CMOS equations and power usage of the Tegra K1 can all be confirmed. For example, Figure 2.5a, 2.5b and 2.5c show measured voltage over the core, HP and GPU rails over the relevant clock frequencies. The core rail voltage in Figure 2.5a is the most intricate. It supplies over 39 different clock generators. Most of these are powered down or idle, but for the remaining clocks (see "core-rail" clocks in Table 2.1), the core rail voltage is the maximum required by any of these at any point in time. The PCI express, crossbar and unknown core rail clocks are therefore fixed to relatively low values to avoid impacting the rail voltage. Only variations in the LP core and memory clock frequency impacts core rail voltage (see Figure 2.5a).

Figure 2.5d shows the impact that frequency adjustments have on average power while running a CPU-intensive workload. We see that measured power increases linearly with frequency when rail voltage is approximately constant (0.9 V below 600 MHz LP core and 500 MHz memory frequency). In this frequency region, measured power grows linearly with frequency because dynamic power is proportional to clock frequency. Above these operating points, core rail voltage increases. This has a profound effect on power, because dynamic power is proportional to the square rail voltage. Additionally, static power will increase. As a result, we see that the slope of the curve in Figure 2.5d in these areas is much steeper than when rail voltage is constant.



(a) LP and memory. (b) HP. (c) GPU.

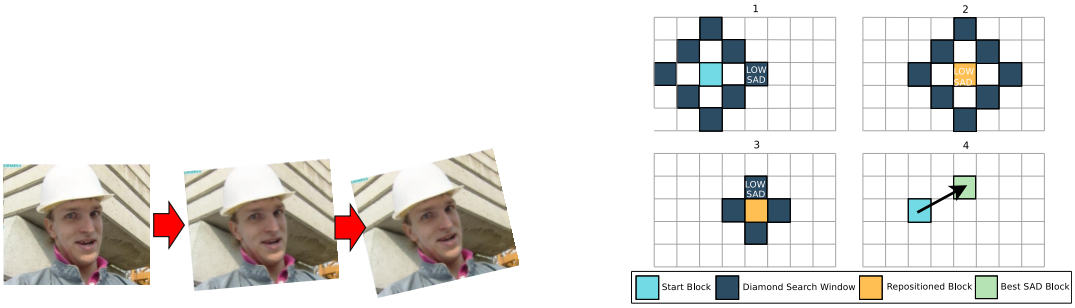
(d) LP and memory. (e) HP and memory. (f) GPU and memory.

Figure 2.5: Measured voltage (top row) and power (bottom row) under various frequency ranges.

The HP and GPU rails show the same trends, and are simpler because they power their own clocks. Above 1.3 GHz HP and 400 MHz GPU frequency, voltage climbs above the threshold voltage levels (see Figure 2.5e and 2.5f). Similarly to the core rail, we see that the slope of the average measured power curve increases superlinearly with frequency above these points due to increased voltage. It is worth noting that the variation in memory frequency in these plots has an effect on both the core and memory rails. On the core rail, it varies the voltage. But the off-chip memory and its clock is driven on the memory rail, therefore increasing or decreasing dynamic power on that rail. The memory rail voltage is always 1.35 V.

2.4 Workload: Video Processing Filters

As described in the introduction, multimedia applications and in particular video processing consitutes a substantial part of the processing done in modern, mobile SoCs. Throughout this thesis, we use several multimedia workloads (video processing filters) to evaluate energy efficiency. Some of these are stand-alone filters that can be applied directly to any raw video stream in YUV 422 format, such as rotation and debarreling. However, we also have filters that are integral parts of video encoding, such as MVS, DCT and Huffman encoding. These were originally part of a video encoder on the Tegra K1 [62], but are mostly used as stand-alone filters throughout this thesis. All of our filters are implemented in C on both the Tegra K1's CPU and GPU using CUDA. The CPU implementations are multithreaded, where six concurrent threads simultaneously process a pixel area of the size of a U or V frame. This divisioning is reasonable given that every Y frame can be divided into four areas of roughly the size of a U or V frame. Furthermore,



(a) Video stream rotation.

(b) An illustration of the operation of the diamond search algorithm.

Figure 2.6: Illustration of our multimedia workloads.

all of our workloads can be *offloaded* between the cores. For example, it is possible for the GPU to process 20 % of the per-frame MVS workload, while the CPU processes the remainin 80 %. The divisioning is based on the number of 8x8 pixel macroblocks to process, and can be adjusted in 100 steps from 0 to 100 % GPU offloading. For the remainder of this section, we introduce the video processing filters used throughout this thesis.

2.4.1 Debarreling

Barrel distortion is an effect that occurs with different lenses [68]. Our "debarreling" workload computes a constant debarreling map for one type of lens. This map only needs to be calculated once and is subsequently applied to each frame. The debarreling filter is the least compute-intensive filter we consider, and is majorly memory-intensive.

2.4.2 Image Rotation

In the image rotation tests, each frame of a video stream is being rotated by a continuously increasing angle θ (see Figure 2.6a). The algorithm treats each frame as a cartesian coordinate space centered in the middle of the frame. Reference pixel positions (u, v) are calculated by multiplying each original pixel coordinate (x, y) by the *rotation matrix* as follows:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} cos-\theta & -sin-\theta \\ sin-\theta & cos-\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.6}$$

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

2.4.3 Motion Vector Search

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

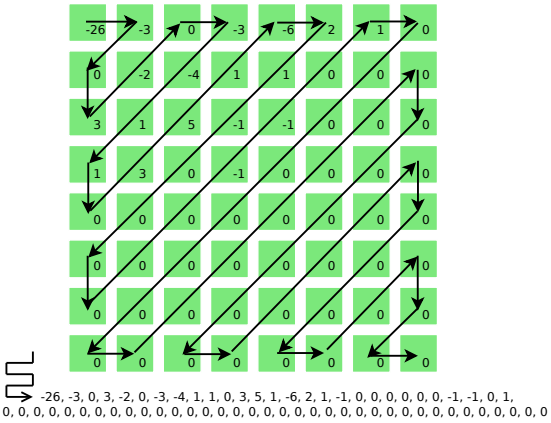


Figure 2.7: Illustration of variable length coding.

We have implemented the diamond search algorithm [76]. Diamond search estimates the displacement of each macroblock by computing the sum of absolute difference (SAD) of the current macroblock and the eight surrounding macroblocks in the previous frame (as shown in Figure 2.6b). At every step, as long as the macroblock with the lowest SAD is not in the center of the "search window", the window will be re-centered at the macroblock with the lowest SAD. After three iterations, the pattern changes to a smaller diamond with only four surrounding macroblocks, where the one block with the lowest SAD is estimated to be correct.

2.4.4 DCT

The DCT is used to convert macroblocks of incoming YUV video frames to the frequency domain. It is a necessary step in compression techniques to remove high-frequency components from each frame, reducing video size. Normally, in video encoding, quantisation is done after the DCT. This is left out from this benchmark, which only transforms each macroblock of 8x8 pixels into the frequency domain according to the following equation:

$$M_{u,v} = \gamma(u)\gamma(v)\sum_{x=0}^{7}\sum_{y=0}^{7} m_{x,y} cos[\frac{\pi}{8}(x+\frac{1}{2})u] cos[\frac{\pi}{8}(y+\frac{1}{2})v] \tag{2.7}$$

In Equation 2.7, $u, v \in [0, 7]$ are the DCT output coordinates, $M_{u,v}$ are the frequency components, $m_{x,y}$ are the original pixel values in the macroblock, and $\gamma(w)$ is a normalising function. The DCT is implemented in two versions; one version with NEON optimisation and one version without.

2.4.5 Variable Length (Huffman) Coding

Huffman coding, or variable length coding, is a lossless compression technique developed by David A. Huffman [23] in 1952. Given some input data, Huffman coding represents source symbols as variable length codes. The codes are extracted from a table where frequently-occurring source symbols are represented by shorter codes. The source symbols in the case of for example video encoding can be residual frequency components from a quantised image macroblock (see Figure 2.7).

2.5 Effects of Dynamic Voltage and Frequency Scaling

Not all workloads, such as video processing filters, need to process data as fast as possible. Especially for multimedia workloads that operate on videos, it is enough that they reach a framerate that is fast enough to avoid quality degradations in the delivered material. With a framerate requirement of 25 FPS, for example, the requirement is that a video processing filter finishes processing each frame within 40 ms, or at least 40 ms on average, if some delay in the delivery of the frames can be tolerable. The framerate requirement allows us to measure the performance of our video processing filters, and further allows us to experiment with the operating frequencies that are used on the Tegra K1's CPUs, GPU and memory. In this section, we investigate how frequency scaling on the Tegra K1 impacts performance and energy consumption of our video processing filters under a framerate constraint.

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 [11]. The Tegra K1, for example, can vary LP core, HP cluster, GPU and memory frequency to accommodate applications' demand for performance. 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, effectively dissipating energy as heat. Increasing the clock frequency increases the rate at which this occurs and, consequently, the average power usage on that rail. Furthermore, rail voltage also increases with frequency [30], which is necessary for the circuitry to remain stable and deterministic. Dynamic power is proportional to rail voltage ($P_{R,dyn} \propto V_R^2$), meaning that variations in voltage have a considerable impact on the power of *all* switching activity on a rail (see Section 2.3).

It is well known in literature that standard frequency scaling algorithms in the Linux kernel overreact to changes in processor utilisation. This leads to unecessarily high CPU, GPU and memory operating frequencies. As a result, the frequency scaling algorithms waste energy. In our experiments on the Tegra K1 [61, 62], we found that by minimising CPU and GPU frequencies for a video processing filter such that a certain framerate is met, we could save around 10 % energy. This is in accordance with the findings of for example You and Chung [74]. They develop a GPU frequency scaling algorithm that reduces memory frequency such that a target framerate is met, demonstrating energy saving of between 11 to 21 % on the Exynos 4412 SoC. Pathania et. al. [48] conduct a detailed study of how CPU, GPU and memory frequencies affect the framerate of several Android games on the Exynos Octa SoC. They also design a frequency scaling algorithm that tune these to meet specific framerates. Bortolotti et. al. [6] develop a mobile transmission gateway for body sensor networks on the Exynos 5422 SoC. The gateway compresses body sensor data and transmits it to a receiver. This SoC is equipped with two CPUs comprised of a quad-core Cortex-A15 cluster as well as a quad-core Cortex-A7 cluster. They also observe that processor frequency should be minimised such that application requirements, in this case the average received signal to noise ratio at a receiver, is met.

Jiao et. al. [26] study the impact of both processor and memory frequency on power

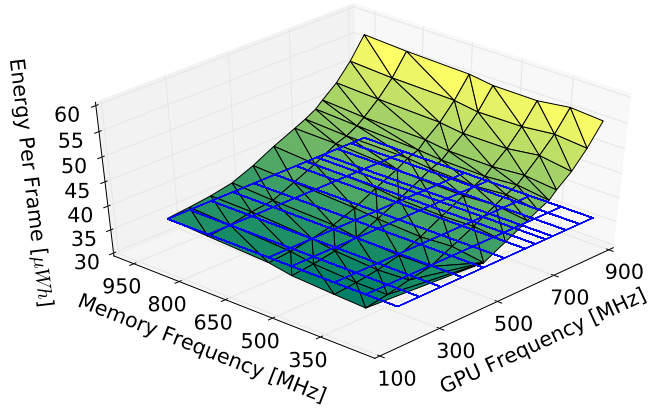
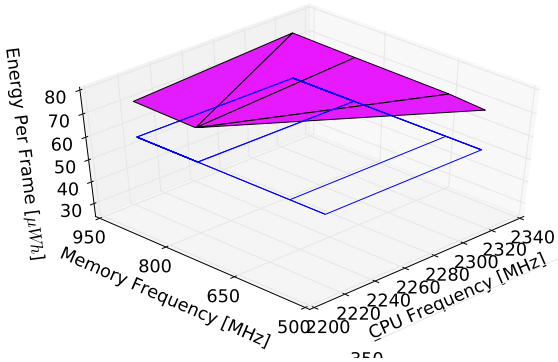


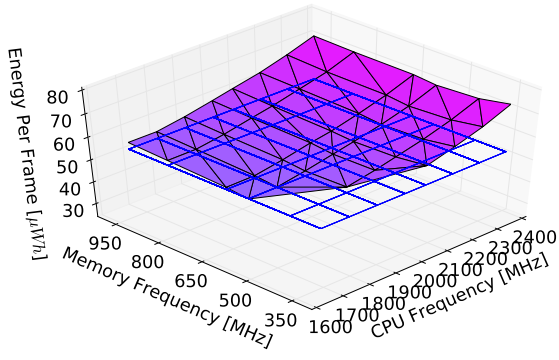
Figure 2.8: Debarreling energy per frame. Only the GPU achieved a framerate of 25 FPS. The blue, transparent grid indicates the EPF under the standard DVFS algorithms.

efficiency on a GTX 280 GPU under workloads such as 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 an increased frequency increases the power efficiency compared to running the benchmarks on lower frequency levels. This contradicts to the general belief that lower frequencies save energy. However, as many authors do [12, 22, 35, 54, 73], Jiao et. al. stop measuring power when the benchmarks complete, and as a result, the increased power efficiency may be a result of reduced energy consumption from idle (constant) power components. This is confirmed in their discussion for the matrix transpose benchmark, where the benchmark runtime happens to be 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 happens because the energy consumption of idle power components is constant. We discuss this phenomenon in detail in Section 5.2.1. In this section, we study how these results behave on the Tegra K1, that is, how frequency scaling affects energy usage and performance of various multimedia workloads.

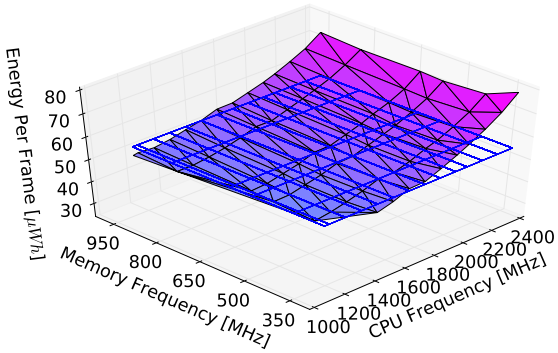
Our workloads (see Section 2.4) resemble video processing operations. Some of these, such as the rotation and debarreling filters, resemble stand-alone pre-processing operations that can be applied before encoding stages of a video processing pipeline. Other filters, such as the DCT, MVS and Huffman encoding are common components found in a complete video encoding pipeline. In these experiments, we let each of these filters process 80 HD frames, separately on the CPU and the GPU, at a rate of 25 FPS. The results are shown in Figures 2.8, 2.9, 2.10, 2.11 and 2.12. The Energy per Frame (EPF) is plotted over all memory and processor (CPU or GPU) frequencies. Those frequency combinations that do not reach 25 FPS are not shown in the figures. The EPF achieved by the standard DVFS algorithms are shown as a blue, transparent grid in the plots. In the GPU experiments, the CPU is fixed to process on the LP core, disabling DVFS on this processor and fixing the CPU frequency to 1 GHz. Otherwise, in the CPU experiments, the *ondemand* [45] DVFS governor is active. All input data (raw YUV-format video or Huffman-ready compressed frames) are pre-loaded into ramfs to avoid accesses to persistent storage.



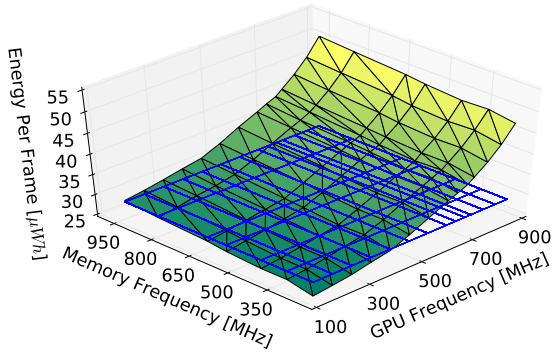
(a) Two HP cores (CPU).



(b) Three HP cores (CPU).



(c) Four HP cores (CPU).

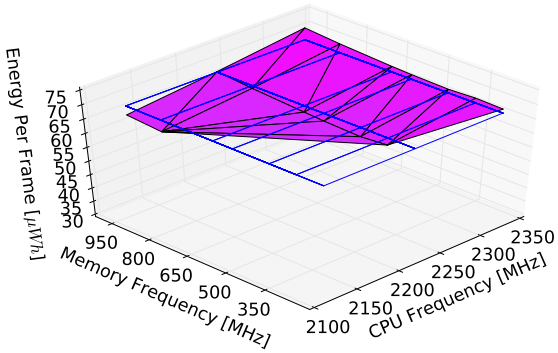


(d) GPU.

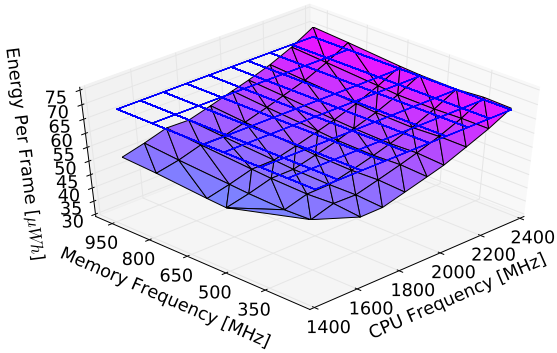
Figure 2.9: Rotation energy per frame on the GPU and the CPU. The Jetson-TK1 is operating under normal DVFS conditions. Red-violet colouring indicates CPU benchmarks, while green-yellow colouring indicates GPU benchmarks. The blue, transparent grid indicates the EPF under the standard DVFS algorithms.

The EPF for the debarreling experiment is shown in Figure 2.8. For this filter, only the GPU managed to process the full-HD live-stream at 25 FPS. The CPU's frame processing time is always above 40 ms, even if four cores are active at the maximum operating frequency. This occurs because the debarreling operation is *memory-intensive*. At every frame, every Y, U and V pixel must be shifted to form a new (lens-corrected) raw image. The GPU is naturally better than the CPU at performing this operation, because it has a larger number of concurrently active threads that shift individual pixels. Additionally, the memory interface is 64-bit, whereas on the CPU, it is only 32-bit, leading to a higher memory throughput on the GPU. The best EPF is $34.13 \mu Wh$, located at $f_{gpu} = 180 MHz$ and $f_{mem} = 300 MHz$. At this frequency combination, the EPF is better than when the DVFS algorithms are operating ($37.01 \mu Wh$), which represents an energy saving of 7.78 %.

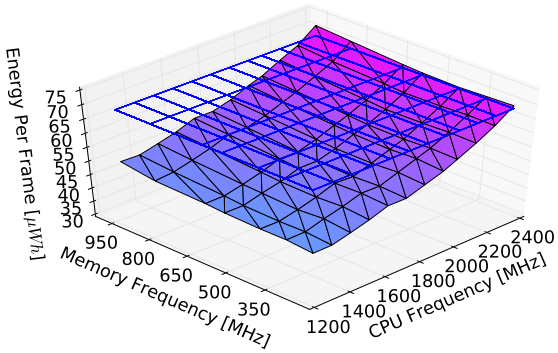
The results of the rotation filter is shown in Figure 2.9, where red-violet colouring indicates CPU benchmarks, while green-yellow colouring indicates GPU benchmarks. The CPU is now capable of processing at 25 FPS when operating on two, three or four cores. Studying Figure 2.9, we can see that the CPU is most energy-efficient on four cores (Figure 2.9c). When a fewer number of cores are active, the EPF increases. The fact that EPF is reduced when more CPU cores are active, means that the extra cost of adding



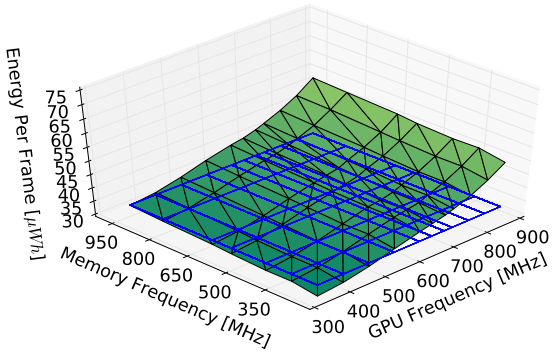
(a) Two HP cores (CPU).



(b) Three HP cores (CPU).



(c) Four HP cores (CPU).

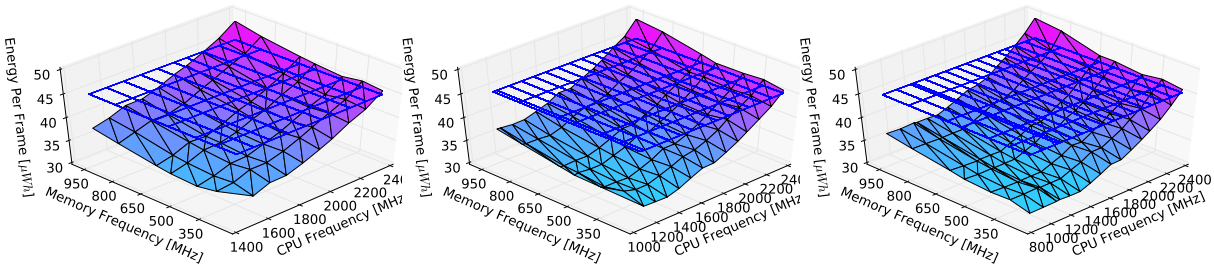


(d) GPU.

Figure 2.10: DCT energy per frame on the GPU and the CPU. The Jetson-TK1 is operating under normal DVFS conditions. Red-violet colouring indicates CPU benchmarks, while green-yellow colouring indicates GPU benchmarks. The blue, transparent grid indicates the EPF under the standard DVFS algorithms.

compute cores is smaller than the saving we achieve by reducing operating frequencies: the possible (25 FPS) frequency range increases as more cores are used. This indicates that *scalability* is important for energy usage on the Tegra K1. With two cores, the CPU frequency has to be above 2.1 GHz to reach 25 FPS. With three cores, the CPU frequency can be reduced to 1.6 GHz, and with four cores, the frequency can be as low as 1.2 GHz. With four CPU cores operating at $f_{cpu} = 1.2GHz$ and $f_{mem} = 396MHz$, the EPF is only $50.19\mu Wh$. This is again smaller than $57.33\mu Wh$, which is the EPF operating under the standard DVFS algorithms. The energy saving is 12.45 %. However, the GPU in Figure 2.9d achieves a much better EPF, $26.45\mu Wh$, at $f_{gpu} = 108MHz$ and $f_{mem} = 204MHz$. This again, is lower than what the standard GPU DVFS algorithm achieves, which is an improvement of 11.77 % from $29.98\mu Wh$. Even if both the CPU and the GPU *can* process the rotation filter at 25 FPS, the GPU outperforms the CPU in terms of the EPF by 47.30 %. This suggests that the GPU should be used for this workload.

Similar observations can be made for the DCT benchmark in Figure 2.10. Both the CPU and the GPU reach 25 FPS using two, three or four HP cores (Figure 2.10a, 2.10b and 2.10c, respectively). As more CPU cores are used, the potential CPU and



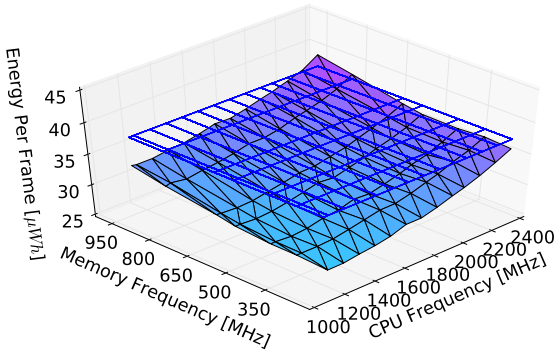
(a) Two HP cores. (b) Three HP cores. (c) Four HP cores.

Figure 2.11: Huffman energy per frame over all core configurations, CPU and memory frequencies. The Jetson-TK1 is operating under normal DVFS conditions. The blue, transparent grid indicates the EPF under the standard DVFS algorithms.

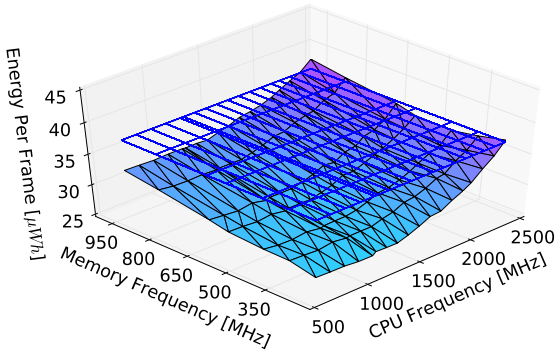
memory frequency ranges that reach 25 FPS increase, with the lowest EPF of $47.97\mu Wh$ at $f_{cpu} = 1.2GHz$ and $f_{mem} = 204MHz$. The improvement compared to the standard CPU DVFS algorithm is substantial at 31.88 % from $70.43\mu Wh$. The GPU reaches the lowest EPF of $31.08\mu Wh$ at $f_{gpu} = 324MHz$ and $f_{mem} = 204MHz$, again outperforming the CPU. However, the difference compared to the CPU's lowest EPF on the rotation filter is smaller, with a 35.20 % improvement. Compared to the standard GPU DVFS algorithms, we see that we can outperform it in terms of energy savings by 12.00 % from $35.32\mu Wh$.

The results for the Huffman filter is shown in Figure 2.11. For this filter, the GPU is not able to reach 25 FPS at any frequency combination. This is not surprising given that Huffman (or variable-length) coding is not easily parallelisable. The CPU, however, reaches 25 FPS when more than two cores are active. As before, we can also see that the EPF decreases as more CPU cores are active, reducing the lowest frequency which reaches a framerate of 25 FPS. The lowest EPF of $33.28\mu Wh$ is reached at $f_{cpu} = 828MHz$ and $f_{mem} = 204MHz$ with four cores active, which is 27.39 % better than the DVFS EPF at $45.84\mu Wh$.

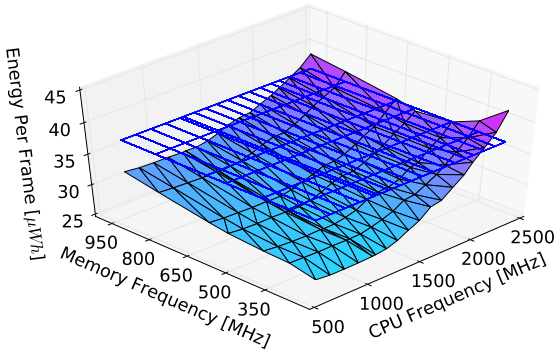
Finally, the results for the MVS filter is shown in Figure 2.12. Similarly to the Huffman filter, the GPU is not able to sustain a throughput of 25 FPS. However, the CPU is able to perform the MVS filter on all the HP cores as well as the LP core (not shown). On the LP core, the best EPF is $32.94\mu Wh$ at $f_{cpu} = 1.0GHz$ and $f_{mem} = 396MHz$. The HP cluster, however, performs better, but the margins are different. So far, in general, we have seen that the best EPF is found at *the lowest processor and memory frequencies that reach the QoS requirement of 25 FPS*. However, if we study Figure 2.12d carefully, we can see that at the lowest frequency point the EPF is $29.26\mu Wh$. Increasing CPU frequency above this effectively lowers the EPF by a marginal amount with for example $28.60\mu Wh$ per frame at 696 MHz CPU frequency. We have also observed that the best EPF for the CPU is found with a higher number of cores. For the MVS filter, one, two, three and four cores reach minimum EPF values of 29.18, 28.36, 28.13 and $28.60\mu Wh$, respectively. The memory frequency which achieves this throughput is always 204 MHz, but the CPU frequency is reduced from 1.0 GHz (at one HP core) to 696 MHz at four HP cores. The MVS filter has the lowest observed EPF on three HP cores. However, the EPF values are so similar that using four cores, or slightly increasing the CPU frequency,



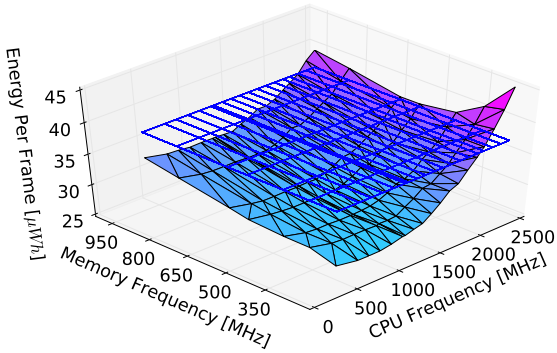
(a) One HP core.



(b) Two HP cores.



(c) Three HP cores.



(d) Four HP cores.

Figure 2.12: MVS energy per frame over all core configurations, CPU and memory frequencies. The Jetson-TK1 is operating under normal DVFS conditions. The blue, transparent grid indicates the EPF under the standard DVFS algorithms.

makes negligible difference.

The lesson learned from these experiments, is that standard DVFS algorithms operating on the Tegra K1 can be easily outperformed in terms of energy consumption. By minimising processor and memory frequency, such that application requirements and QoS constraints are met, it is trivial to increase energy-efficiency of our video processing filters by up to for example 27 % in the case of the Huffman filter. For the MVS filter, this strategy is not optimal with respect to what we have observed, but the margins are small enough that it makes negligible difference. It is not simple to state *why* frequency minimisation is an effective strategy. To some extent, it can be argued that reduced voltage as platform frequencies are reduced, impact energy-efficiency. However, one cannot neglect the possiblity that other factors, such as power management mechanisms, pipeline efficiency and memory access stalls, also impact energy-efficiency. Without the possibility to measure directly the power usage of individual compute units, power models are necessary to understand how the Tegra K1's CPUs, GPU and memory consume energy under these workloads.

2.6 Summary

In this section, we have introduced the Tegra K1 mobile SoC and the heterogeneous processors it provides to the programmer. We have outlined how we measure power usage on the Jetson-TK1, and how energy is distributed to the Tegra K1's five main power rails: the RTC, core, HP, GPU and memory rails. This insight is important to understand how the SoC consumes energy and our high-precision power modelling methodology in later sections. Furthermore, we have introduced the fundamental CMOS-equations. These describe the dynamic and static energy dissipation in these rails as a function of clock frequencies and voltage. We have put special emphasis on how rail voltages are affected by DVFS, and verified that the Tegra K1's total power usage increases superlinearly with frequency as rail voltages increase. This confirms that the CMOS-equations are valid for our platform.

We have also studied how DVFS affects the energy consumption of multimedia processing. We have observed that, in terms of energy-efficiency, the Tegra K1's heterogeneous compute cores are best at processing specific workloads. For example, in terms of the energy consumption per frame, the GPU is better than the CPU to process the DCT filter. The CPU is best at processing the MVS and Huffman filters. This is a challenge for developers, where care must be taken to implement the filter on the appropriate processor. Additionally, we discovered that processor (CPU or GPU) and memory frequencies should be minimised such that application requirements are met to save energy. This corresponds to observations made by other researchers on different SoCs. However, the application must allow for flexibility in terms of performance reduction. Video processing filters are here a good example, because they must adhere to deadlines to produce frames at a specific framerate. The fact that the processor is running slower is of little importance, as long as this framerate is met.

An interesting question is here how energy consumption can be reduced for other types of workloads. For example, what about shorter, and more intermittent types of workloads, that do not have any specific or natural deadlines to adhere to? This could for example be kernel drivers or software libraries that stand-alone applications rely on. The requirement is now that that the supporting drivers and libraries do not cause any dependent applications to miss their own processing deadlines. Stand-alone applications may also not have intrinsic deadlines to accommodate. In this scenario, the issue becomes much more complex and, in some cases, psychological. From a user perspective, it is of best interest that application performance is decent, and that as little time as possible is spent waiting for operations to complete. In this respect, allowing a mobile device to operate at full speed may allow the user to finish quickly, putting the device into standby earlier, and saving energy. However, operating at high frequencies has been shown to be energy-inefficient. Therefore, it becomes more interesting to find some threshold frequencies that achieves decent performance while being able to save some energy. Another important aspect that we have not investigated in detail is that of variance in frame processing time. Our filters generally use the same amount of time to finish processing each frame over a video stream. If for example the time taken to finish processing each frame varied significantly, a frequency scaling framework that consistently attempted to minimise platform frequencies might cause a filter to miss frame processing deadlines. This could for example occur while processing I-, P- and B-frames in an MPEG video encoder. It is also easy to

imagine scenarios where even a filter with uniform frame processing time would have this type of problem, if for example other processes in the system posed significant contention on shared hardware resources.

It is hard to conclude *why* frequency minimisation is such an effective strategy. It is for example not possible to measure power usage of the Tegra K1's individual processors and memory, and break this measurement down into static and dynamic power components. In this aspect, *power models* are necessary to provide insight into how the Tegra K1's processors and memory consume energy, for example under the influence of software workloads and frequency scaling. In the next chapter of this thesis, we will discuss and evaluate state-of-the-art power modelling methods found in the literature.

Chapter 3

Evaluation of State of the Art Power Modelling Methodologies

In the previous chapter, we introduced the Jetson-TK1 development kit, and the Tegra K1 SoC, in terms of its device architecture and compute capabilities. Studying the impact of DVFS on the energy-efficiency of our video processing filters, we found that processor and memory frequency should be minimised such that application requirements are met. However, it is hard to understand *why* this occurs, because we are limited to measuring the total power usage of the SoC. In this aspect, power models are useful because they provide insight into the energy consumption the various hardware components, such as the Tegra K1's CPUs, GPU and memory.

Many power models have been introduced in the literature. These are generally used for a range of hardware components such as processors, SoCs and wireless interfaces, and can be classified into one out of three categories: state-, rate- and CMOS-based models (see Figure 3.1). Instruction-based models are also often referred to in the literature. These can be considered as rate-based models because of their tendency to correlate power with the throughput of instructions. State- and rate-based power models are by far the most simple and commonly used models, seeing a wide range of adoption to investigate power usage of processors and especially network interfaces. Introducing both a simple state- and a rate-based power model for the IEEE 802.11 predecessor, the Lucent Wavelan network interface, Feeney and Nilsson [17] are often at the source of these works. The three model types differ mainly in terms of the abstraction level that they use to attribute power or energy to a system, and have individual strengths and weaknesses.

In this section we outline the differences between the three main power modelling methodologies. In Section 3.1, 3.2, 3.3 and 3.4, we introduce CMOS-, state-, rate- and instruction-based models found in the literature. These often cover more exotic hardware components, such as DCDC and LDO regulators. In Section 3.5 we build power models according to the three power modelling methodologies on the Tegra K1. We show that their accuracy can vary significantly depending on the operating frequency levels on the Tegra K1. Finally, we conclude our observations in this chapter in Section 3.6.

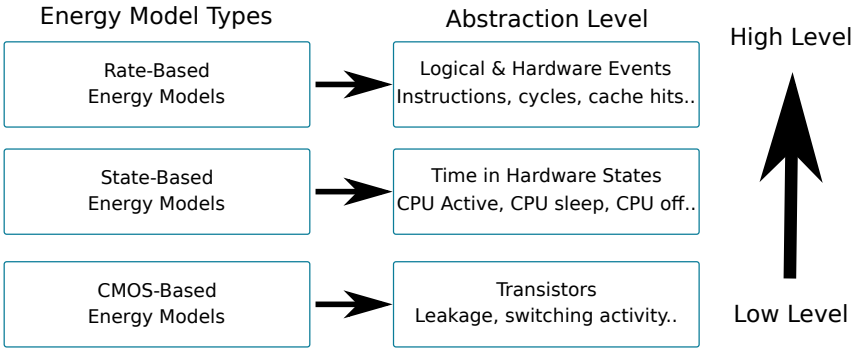


Figure 3.1: Classification of modelling methodologies.

3.1 CMOS-Based

The CMOS equations (Equations 2.3 and 2.4 in Section 2.3) represent the deepest abstraction levels used in power modelling, and several authors attempt to use these directly to model power. Castagnetti et. al. [11] model both dynamic (αC) and static ($I_{cpu,leak}$) power of an Intel XScale SoC by running a CPU- and memory-benchmark on five frequency configurations with fixed CPU, bus, memory and SDRAM frequencies. The model achieves an accuracy above 90 %. Pathania et. al. [47] model an ARM variable SMP ARM CPU (quad-core Cortex-A7 + quad-core Cortex-A15) and a GPU (on the Odroid-XU+E development kit) for mobile gaming workloads. The SoC used is the Exynos-5. They propose to use a processor's utilisation factor as a substitue for α and estimate the maximum capacitive load for the CPU and GPU (C_{cpu} and C_{gpu}) using regression, achieving an accuracy above 85 %. However, their method assumes that capacitive load per cycle is constant and independent of workload, which is not necessarily true. Power management mechanisms such as clock-gating and different types of instructions will excersise the processor pipeline and switching activity in different ways depending on software activity. This fact is not recognised by other authors in the field.

In our own research on the Tegra K1 [63] we also modelled power using the CMOS equations, but we attempted to separate more accurately the static and dynamic power terms. For example, we ran many experiments when the Tegra K1 was idle to estimate leakage currents for each cluster and for each core, and we estimated average capacitive loads over more rails (HP, core and memory) for three multimedia workloads. We also used more CPU and memory frequency combinations, where the Tegra K1 is a more modern chip with more frequency levels and clock domains. Despite careful development our model had a worst-case error of 13 %. The average error over all CPU and memory operating frequencies and benchmarks is 8 %, which is similar to that reported by Pathania [47] and Castagnetti [11].

It is interesting that CMOS-based models seem to share the same average error over all operating frequencies of about 6 % for such diverse SoCs and electrical implementations. Because it is impossible to physically measure and verify the power of each rail, it is difficult to conclusively prove why CMOS-based models fail to model power more accurately. However, we believe that the main cause of the error is the implicit assumption of linearity between capacitive loads in one domain and frequency in another. For example, increasing memory frequency under a memory-intensive workload is likely to increase

switching activity in both the CPU and memory rail. It therefore becomes unreasonable to assume that the increase in power as for example memory frequency is increased, is solely due to increased switching activity on that rail, but this is assumed by for example Castagnetti [11]. The same argument can be made for Pathania [47], where CPU and GPU capacitive load per cycle is estimated based on only the maximum frequency levels, or in research where regression is used directly over experiments where both memory and CPU frequency is varied [63].

3.2 State-Based

State-based power models are more high-level than CMOS-based models. These models abstract hardware platforms into components with associated states, where each state is attributed a constant power draw. Lee et. al. [24], for example, propose to abstract an ARM926EJ-S CPU into *idle* and *active* states, where each state has an associated, constant power draw. Some models also propose to incorporate transition cost between states [5]. Transition costs can for example be related to changing clock oscillator frequency, but these effects are rarely considered in practice. Formally, the energy consumption of a component is given by the following equation:

$$E_{comp} = \sum^{s \in \mathbb{S}} T(s)P(s) + \sum^{n \in \mathbb{S}} \sum^{m \in \mathbb{S}} N(n,m)C(n,m) \tag{3.1}$$

where $\mathbb{S}$ is the set of hardware states for the component, $T(s)$ is the time spent in state s, $P(s)$ is the constant power draw of state s, $N(n,m)$ is the number of state transitions and $C(n,m)$ is the energy cost per transition between state n and m. The total platform energy usage is the sum of energy of each component.

State-based models are typically more extensively used to predict power of networking interfaces, and not only processors. Negri et. al. [38–40] and Cano et. al. [9] have for example used this model type for the Bluetooth low-power, transmission and reception states. Rantala et. al. [52] build a state-based power model for a WiFi interface. Others consider a more complete systems perspective. For example, Perrucci et. al. [50] use a Nokia N95 as a case study and derive state costs for the CPU, display, memory, Bluetooth, WiFi, 2G and 3G. Measurements are primarily done with the Nokia energy profiler and verified using an external power meter. Their methodology is based on isolating the power usage of individual components to the extent possible by turning off others. The CPU is divided into five different hardware states based on the utilisation rate. Pathak et. al. [46] focus on the HTC Magic, Touch and Tytn II, abstracting power states for the CPU, disk and WiFi. The CPU's power states is based on the minimum and maximum utilisation levels.

Fonseca et. al. [18] develop an energy profiler for wireless sensor networks with an underlying, state-based power model. The model is based on datasheets and exposes the *current draw* of hardware component states. To find the power draw of each state, the current must be multiplied with the component's voltage supply and it is therefore one of the few state-based models that consider voltages. This resembles static power and it can therefore appear to be closer in nature to CMOS-based models, but the authors refer to it in terms of states and there is no explicit mention of dynamic power. The CPU is

here divided into states depending on whether it is fully active or in any of five low-power modes. The authors also show how power models can be built on-line with regression, which is done by augmenting drivers to expose the time each component spends in each of its states.

3.3 Rate-Based

Rate-based modelling is typically used to build more complete power models for devices such as smart phones. In this methodology, power usage is correlated with hardware utilisation. Hardware utilisation is typically measured with any available performance counters (HPCs) on the device under study, and has units of hardware events per second. Formally, power is modelled as:

$$P_{platform} = \beta_0 + \sum_{i=1}^{N_p} \beta_i \rho_i \tag{3.2}$$

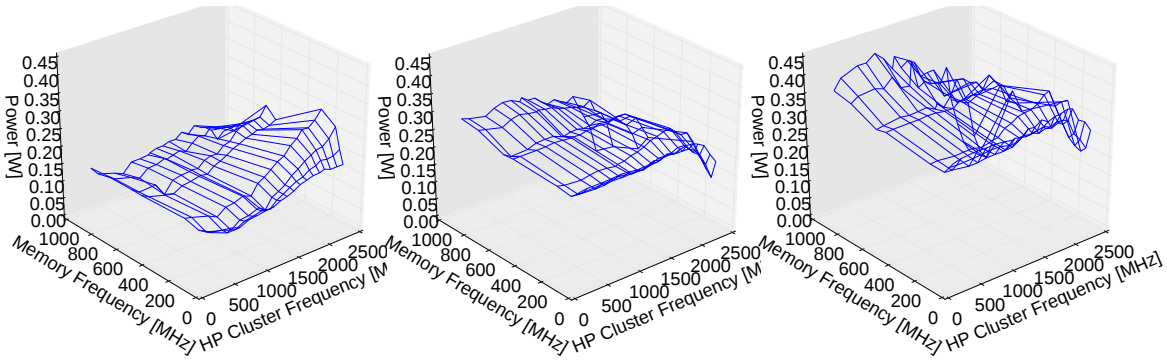
In Equation 3.2, N_p is the number of events, ρ_i is a predictor in events per second, β_i is the power cost in watts per event per second and β_0 is the constant, always-present base power. Some authors [21, 25, 33, 69] have slightly different interpretations of Equation 3.2, and replace β_i with the maximum power usage of a component $P_{i,max}$ ($\rho_i \in [0, 1]$). The predictors β_i normally reflect utilisation of various hardware components, such as CPU, GPS, display, storage and wireless network interfaces [15, 28, 34, 70, 71, 75]. Multivariable, linear regression is commonly used to estimate the model coefficients β_i over training benchmarks. Some authors, such as Limin et. al. [34] use neural networks to estimate the model coefficients. In later years, researchers have established a new category of rate-based models where smart phone power models are built on-line using existing power measurement sensors [15, 28, 71, 75]. Whether external measurement setups are necessary is a topic for dicussion. Dong and Zhong [15], for example, show that on-board sensors can be inaccurate and subject to signal processing such as averaging (see Section 2.2.1). In these cases it becomes important to characterise the on-board sensors such that the appropriate design decisions can be taken to guarantee accurate measurements over time. Nevertheless, self-constructive models are a promising development in power modelling that is necessary to tackle the fundamental problem of device heterogeneity. No devices, even of the same type and brand, are guaranteed to have the same energy consumption characteristics [36]. Self-constructive modelling is therefore a promising technique to overcome this issue in production devices.

Many rate-based models focus primarily on CPUs and GPUs. Vatjus-Anttila and Hickey [69] build a rate-based model for the Tegra 2's OpenGL ES 2.0 graphics pipeline by correlating power usage to the utilisation of triangle, rendering and texture 3D primitives. Hong and Kim [21] and Leng et. al. [33] build power models for CUDA-capable desktop GPUs. Their models are based on the hardware utilisation in the GPU's various computational units, such as the floating point, integer and special function units, register file, shared memory, texture and constant caches and GPU RAM. Utilisation in these hardware domains can be traced by executing instructions that excersise the specific hardware units [21]. Pricopi et. al. [51] focus on an ARM variable SMP CPU (dual-core Cortex-A15 + three-core Cortex-A7) on a development kit, and build what

is one of the most detailed rate-based power models for this type of architecture by using fine grained accounting for instructions and cache accesses. Power is correlated with individual instructions, such as floating point and integer execution. However, different CPU implementations may not include support for measuring detailed hardware utilisation. This is up to the manufacturer of the SoC, with the exception of certain HPCs. For example, the armv7 architecture [2] must always provide an HPC to measure the number of elapsed active CPU cycles. On the Tegra K1, there is no support for HPCs that count the number and type of instructions executed. Pricopi et. al. also build detailed performance models that are capable of estimating cache misses and branch mispredictions on the heterogeneous CPU cores. Xiao et. al. [70] build a full-system power model for a tablet, similarly to self-constructive models in the previous paragraph. The power of an ARM 1136 CPU, WLAN and display is considered, where the authors consider 17 different HPCs for the CPU model. These HPCs reflect utilisation in data and instruction caches, instructions executed, TLB misses, stalls and active cycles. There is however a limit to the number of HPCs that can be tracked simultaneously (three, where one is always the cycle counter). This is a common issue in rate-based modelling. Most authors solve this by running the same model training benchmarks as many times as needed to collect the event rates [27]. Xiao et. al. propose to use the HPCs with the highest estimated power cost per event per second. The resulting CPU HPCs are the cycle counter, data cache writeback and TLB misses. To some extent, it can be argued that the model implicitly considers RAM because data cache writebacks will evict old cache lines to RAM. We agree in the choice of the cycle counter as an important model predictor, where we expect especially the cycle counter to correlate strongly with power. However, the coefficients of these hardware events is not a good measure of how important that event is for power usage. That depends on how often that particular event occurs, which can vary depending on workloads and resource usage.

3.4 Instruction-Level

A modelling method which is related to rate-based models is *instruction-level* models. These are similar because they attempt to correlate power with individual instructions. However, they often consider more sophisticated and low-level techniques to model power of microprocessors that requires much deeper hardware knowledge. Instead of modelling utilisation in hardware units (integer, floating point) as a result of executing specific instructions, power is directly attributed to instructions and how these exersise the underlying processor pipeline. Tiwari et. al. [67] propose to build power models for the Intel 486DX and Fujitsu SPARClite CPU, assuming that every instruction i residing in the processor pipeline causes a constant, base current draw per clock cycle $I_{i,base}$. The core idea is that the additional effects in terms of of inter-instruction dependencies, instruction operand bits, pipeline stalls and cache misses can be modelled on top of the base current draw. It is for example shown that prefetch buffer stalling cycles draw 250 mA, and that pipeline stalls due to cache misses draw 216 mA. Kalla et. al. [29] build an instruction-level power estimation tool called SEA for the Sun MicroSparcIIep processor. For a set of instructions $\mathbb{I}$, they describe the energy of the processor as:



(a) DCT, one core. (b) Rotation, two cores. (c) Debarrel, three cores.

Figure 3.2: Error for the filters in our preliminary, CMOS-based model [63].

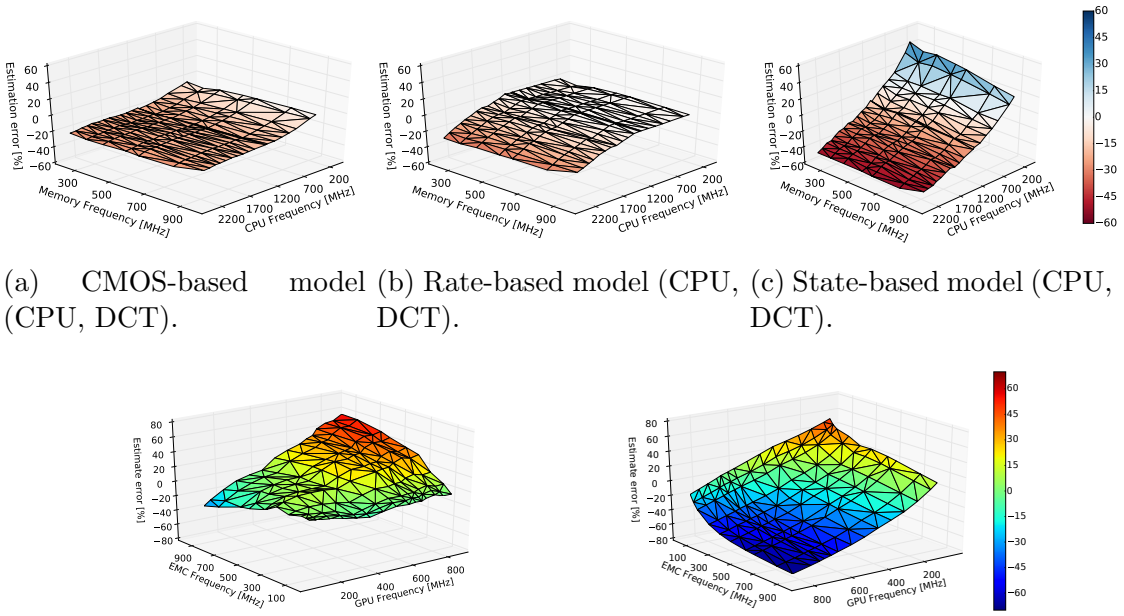
$$P_{cpu} = ((\sum^{i\in\mathbb{I}} P_{i,avg} n_{i,act}) + (\sum^{i\in\mathbb{I}} n_{i,stall}) P_{stall}) T \tag{3.3}$$

where i is an instruction in the set of instructions $\mathbb{I}$ executed on the processor in the time interval T, $n_{i,act}$ and $n_{i,stall}$ is the number of active and stalling cycles, $P_{i,avg}$ is the average power and $P_{i,stall}$ is the average stall power. Tiwari and Kalla agree that stall power is similar across instructions (independently of the source of the stall). Based on register transfer level power estimations of the processor, Kalla et. al. build a power database where $P_{i,avg}$ can be extracted automatically for any instruction i. Their method to build this database is their main contribution and compensates for power variations as a result of data dependencies and inter-instruction effects. Unlike Tiwari et. al., they find that some pairs of instructions (such as `add-and`) *lower* the power usage of the processor (0.94 W) compared to running for example only `add` in a loop (1.08 W). Despite using fine-grained power estimations based on register transfer level abstractions, the authors do not discuss the reasons behind this but instead argue that Tiwari's assumptions, that inter-instruction effects are always added to the base power of instructions, are not correct for this processor.

Brandolese et. al. [7] focus on an Intel i80486DX processor. They break the processor down into components, such as fetch-and-decode, arithmethic and logic unit, write register, load, store and branching units. For any instruction executing in the pipeline, they estimate the average current per processor cycle in each of these units. This approach is similar to Lee et. al. [31], where the authors estimate the energy (in joules) per cycle in each component of an ARM7TDMI processor using regression. Sami et. al. [56] extend this method to VLIW cores. This is more challenging because the number of inter-instruction and other effects is much higher due to the data parallelism in these types of cores. The authors' methodology to model power of such architectures is therefore based on distinguishing the processor components where the rate of energy consumption is independent (their power usage do not depend significantly on other components).

3.5 Model Accuracy

In our first modelling attempts [63], we aimed to build a CMOS-based model for the Tegra K1's CPU clusters and RAM under our video processing filters. The prediction error



(a) CMOS-based model (CPU, DCT). (b) Rate-based model (CPU, DCT). (c) State-based model (CPU, DCT).

(d) CMOS-based model (GPU, MVS). (e) Rate-based model (GPU, MVS).

Figure 3.3: Power estimation error for common model types on the Tegra K1's CPU (top row, four cores) and GPU (bottom row).

of this model, for each filter, can be seen in Figure 3.2, plotted over all CPU and memory frequencies. Three different core configurations (one, two or three CPU cores powered) are shown. While the average error is between 6-7 % over all frequencies, we observed more substantial modelling errors at different frequencies and core configurations. For example, our model overestimated power for the rotation filter running on three cores by an average 13.4 %, but on one core, the average error was only 5.1 %. Furthermore, at the lowest CPU and memory frequency points, the error is much smaller. The lesson from these results is that, despite extensive efforts to build an as accurate power model as possible, the estimated error may vary significantly depending on the number of cores and frequencies used. It is rare that authors of power models perform such extensive validation over hardware configurations, and especially frequencies.

Motivated by the large variations in prediction error over different hardware configurations and frequency levels, we built power models for all the major categories of models: state-, rate- and CMOS-based models. The models are built individually for the CPU and GPU. Through extensive verification over the different core configurations and operating frequencies, the results show significant variation in accuracy on our platform. Consider for example the CMOS-based model in Table 3.1. The training data for this model is generated by running our filters over all possible CPU, GPU and memory frequencies. Average capacitive load per CPU, GPU and memory clock cycle, as well as leakage currents on these rails, are estimated directly using multivariable, linear regression. We notice that GPU rail leakage is now negative. This indicates that there are already some assumptions that are incorrect. Furthermore, the coefficients that are directly comparable to our more accurate hybrid model presented in Section 4.7 are much higher, such as the

	CMOS-Based		
	Predictor	Coefficient	Description
CPU	$\rho_{core,clk}$	$591.9\frac{pC}{V}$	LP core clock
	V_{core}	$172.6mA$	Core rail leakage
	$\rho_{hp,clk1}$	$579.1\frac{pC}{V}$	HP core clock (first core)
	$\rho_{hp,clk2}$	$514.4\frac{pC}{V}$	HP core clock (second core)
	$\rho_{hp,clk3}$	$413.7\frac{pC}{V}$	HP core clock (third core)
	$\rho_{hp,clk4}$	$317.6\frac{pC}{V}$	HP core clock (fourth core)
	V_{hp}	$377.4mA$	HP rail leakage
	$\rho_{mem,clk}$	$342.1\frac{pC}{V}$	Memory clock
	P_{base}	$1.08W$	Base power
GPU	$\rho_{gpu,clk}$	$3116.7\frac{pC}{V}$	GPU clock
	V_{gpu}	$-2239.4mA$	GPU rail leakage
	$\rho_{mem,clk}$	$528.1\frac{pC}{V}$	Memory clock
	P_{base}	$3.65W$	Base power

Table 3.1: CMOS-based model predictors.

average capacitive load per cycle. This is because the CMOS-based models neglect hardware utilisation in for example the GPU's integer and floating-point units. The cost of utilising these are now estimated as an integral part of the average cycle cost. Our refined model in Section 4.7 captures switching activity at a more detailed level, attributing it to individual instructions, cache usage and memory activity instead of only the average capacitive load per cycle.

The accuracy of the CPU- and GPU-model is shown in Figure 3.3a and 3.3d. The models are verified using our DCT and MVS filters over all frequency combinations. We see that the accuracy of the CMOS-based CPU-model generally remains constant with an underestimation of power at around 20 %-30 %. At the lowest CPU frequencies, the error is close to 0 %. For the GPU-model, the error is generally below 15 % but it can also be very high up to 50 % at very high GPU and memory frequencies. On average, the model predicts reasonably well but these plots show that it might also severely mispredict power usage. The most important point from these results is that the observed accuracy dependends on the frequency levels in use at the time of the verification. If this is not controlled, the frequencies set by the DVFS algorithms are used.

With limited opportunity to measure rail power, the underlying reason for the prediction error of CMOS-based models is hard to determine. However, they implicitly assume that the capacitive load per clock cycle in one domain, such as the memory rail, is independent of the frequency in others, such as the CPU. This is because CMOS-based models use multivariable, linear regression to estimate the average capacitive load per clock cycle in different architectural units. Another way to interpret this is that the models assume that increasing memory frequency will only increase switching activity on the memory rail. It is trivial to imagine scenarios where this assumption may fail. For example, when executing a memory-intensive workload, it is reasonable to expect that the switching activity per clock cycle on both the CPU and memory rail will increase with memory frequency. Furthermore, CMOS-based models assume that the average capacitive load per clock cycle in different domains is constant, independently of the workload. This assumption is also questionable. Authors of instruction- and rate-based models, such as Tiwari et. al. [67] and Pricopi et. al. [51] show that power varies depending on how applications excersise the underlying hardware, such as processor pipeline stages and hardware units. In our CMOS-based model [63], we took this into account and modelled switching capacitance

	Rate-Based		
	Predictor	Coefficient	Description
CPU	$\rho_{core,clk}$	$397.0\frac{W}{eps}$	LP core clock
	$\rho_{hp,clk1}$	$499.2\frac{pW}{eps}$	HP core clock (first core)
	$\rho_{hp,clk2}$	$415.2\frac{pW}{eps}$	HP core clock (second core)
	$\rho_{hp,clk3}$	$377.9\frac{pW}{eps}$	HP core clock (third core)
	$\rho_{hp,clk4}$	$319.7\frac{pW}{eps}$	HP core clock (fourth core)
	$\rho_{mem,clk}$	$582.0\frac{pW}{eps}$	Memory clock
	$\rho_{cpu,l1l2}$	$-60.5\frac{nW}{eps}$	Cache maintenance, L1 and L2
	$\rho_{cpu,l2ram}$	$51.0\frac{nW}{eps}$	Cache maintenance, L2 and RAM
	$\rho_{cpu,ips}$	$79.0\frac{pW}{eps}$	Global instructions
	P_{base}	$1.50W$	Base power
GPU	$\rho_{gpu,L2R}$	$-18.6\frac{nW}{eps}$	L2 cache 32 B reads
	$\rho_{gpu,L1R}$	$0.0\frac{nW}{eps}$	L1 cache 32 B reads
	$\rho_{gpu,L1W}$	$-3.7\frac{nW}{eps}$	L1 cache 32 B writes
	$\rho_{gpu,int}$	$62.4\frac{pW}{eps}$	Integer instructions
	$\rho_{gpu,f32}$	$66.5\frac{pW}{eps}$	Single-precision instructions
	$\rho_{gpu,f64}$	$279.8\frac{pW}{eps}$	Double-precision instructions
	$\rho_{gpu,cnv}$	$326.9\frac{pW}{eps}$	Conversion instructions
	$\rho_{gpu,msc}$	$-300.3\frac{pW}{eps}$	Miscallaneous instructions
	$\rho_{core,clk}$	$887.0\frac{pW}{eps}$	LP core clock
	$\rho_{cpu,ips}$	$1.4\frac{nW}{eps}$	Global instructions
	P_{base}	$2.83W$	Base power

Table 3.2: Rate-based model predictors.

individually for our different filters.

We also built rate-based models for the Tegra K1's CPU and GPU. The predictors and estimated coefficients can be seen in Table 3.2. The CPU model here considers individual core clock frequencies, cache maintenance operations and the number of (globally executed) instructions. The GPU model is intended to be more similar to Pricopi's model [51], where we disregard the clock but allow for instruction, L1 and L2 cache utilisation to be measured. The error of these models is shown in Figure 3.3b and 3.3e, where we see that the models still mispredict substantially across frequencies. For the CPU, the estimation error is closer to 0 % at CPU frequencies below 1.2 GHz. The error increases towards higher CPU frequencies, reaching a maximum of around 20 %. The GPU model is worse than its CMOS-based equivalent. The model mispredicts up to 70 % on the maximum and minimum GPU and memory frequency combinations.

The strength of rate-based models is that they capture consumption of energy that occurs as a result of hardware utilisation. Therefore, they can estimate accurately the power usage of for example integer and floating-point units in processors. However, they disregard variations in voltage. This may not be a problem for devices where voltage does not change. However, we believe that for modern SoCs such as the Tegra K1, it has a profound effect on the model's accuracy. The GPU coefficients, for example, represent an average over a set of extensive tests where all GPU and memory frequencies are varied. This is why the model shows good accuracy "in the middle" (green area of Figure 3.3e):

- Increasing GPU and memory frequency beyond the green area increases voltage on the GPU and core rails, effectively increasing dynamic and static power, and causing the model to underpredict.

- Decreasing GPU and memory frequency below the green area decreases voltage on

	State-Based		
	Predictor	Coefficient	Description
CPU	P_{hp}	$164mW$	HP rail on
	P_{core}	$367mW$	Core on
	P_{base}	$1.50W$	Base power

Table 3.3: State-based model predictors.

the GPU and core rails, effectively decreasing dynamic and static power, and causing the model to overpredict.

Similar observations can be made for the CPU model. Below 1.2 GHz CPU frequency, the model error is very close to 0 %. However, increasing frequency above this also increases CPU voltage, again increasing static and dynamic power and causing the model to underpredict. The error plot shows little effect on changes in terms of memory frequency. It is possible that the increase in core voltage as memory frequency increases is covered by the relatively large cost per memory clock cycle.

We also built a state-based model for the CPU. The GPU, which doesn't expose more than a single hardware state (rail on or off), is ignored. For the CPU, we have a base state where processing is restricted to the LP core. The HP rail exposes two states: on and off. Additionally, power gating is tracked as normal on each individual core (on the HP and core rails), which also exposes on- and off-states. The coefficients can be seen in Table 3.3. As expected, this model performs badly (see Figure 3.3c), overpredicting power by 60 % at low memory and CPU frequencies, and underpredicting by 60 % at high frequencies. This may be caused by the same reasons as for the rate-based models expressed in the previous paragraph. State-based models additionally predict badly because they ignore both changes in voltage and dynamic power. They are instead naturally good at capturing the quiescent power related to the state that components are in, which is directly related to especially leakage currents.

3.6 Summary

In this section, we have reviewed the state-of-the-art power modelling methodologies. State-, rate- and CMOS-based power models are used to describe the power usage of mobile devices and SoCs, where rate-based models are the most prominently used for devices such as smart phones. We have seen that regression is the most widely used method to estimate power model coefficients. Self-constructive models, where the power model is built by the same device as it is to be used for, have been introduced in recent years. These models are promising because no device, even if it is of the same brand and type, is guaranteed to share the same physical characteristics in terms of energy consumption.

We have built power models according to each of the main power modelling methodologies of state-, rate- and CMOS-based models. While these have their individual strengths, they also have weaknesses that cause them to mispredict power on the Tegra K1. **State-based models**, for example, are good at reflecting static power, which is always present depending on the hardware state and supply voltage. They do not, however, take into account the effects of dynamic power as a result of hardware utilisation, for example as a

result of executing instructions. Most of these also do not consider voltage levels on the SoC. **Rate-based models** are good at capturing dynamic power, because they correlate the power usage of a platform with hardware activity. Active processor cycles, cache misses and instruction execution are all examples of popular predictors for rate-based models. However, they do not consider changes in voltage levels on the SoC or leakage currents. Also, there are no rate-based models that take into account the effects of complex power-saving mechanisms such as core- and clock-gating. **CMOS-based models** have strong theoretical foundations. These models take into account both static and dynamic power, as well as voltage variations. However, they correlate dynamic power with operating frequency of the various components (CPU, GPU or memory) of the SoC. Because every process is composed of a unique instruction mix, the way in which they excersise the underlying hardware and transistors is different. Therefore the estimated capactive load per cycle will vary depending on the workload that was used in the modelling phase. In summary, the three main categories of models can mispredict power usage of the Tegra K1 substantially because of these fundamental issues.

It is important to emphasise that the model accuracies presented in this chapter were evaluated on the Tegra K1. They do not represent the accuracy of the models found in the literature. However, what *can* be noted about these works is that the models do not usually undergo extensive testing over different operating frequencies and workloads. This is also hard to test without access to the various devices and SoCs found in the literature. Older devices and SoCs may have been simpler, and this may be why the models have not been validated more extensively. The Intel XScale [11], for example, has only five CPU frequency operating points. It is also possible that frequency and/or voltage scaling has not been implemented, in which case a state-based model may have performed adequately. The results presented in this chapter are meant to demonstrate that the mathematical abstractions, the core ideas behind these, and the predictors typically used in literature, can mispredict power on a modern platform such as the Tegra K1. Power management mechanisms such as core-, clock- and rail-gating are for example never considered in related work, and the lack of rail voltages in rate-based models can cause these to mispredict substantially. An original modelling methodology is necessary to successfully capture static and dynamic power usage of the various compute elements on the Tegra K1 SoC, and bind this power usage to software activity.

Chapter 4

High-Precision Power Modelling

From our discussion in the previous section, we have seen that state-, rate- and CMOS-based models predict power with varying degrees of accuracy on the Tegra K1. This is because they all have their own strengts and weaknesses, where none of them are able to capture *all* the aspects that are required to model power with high accuracy. In this chapter, we motivate, develop and evaluate our high-precision power modelling methodology for the Tegra K1. Our method is able to capture the power usage of the Tegra K1 with an accuracy close to 100 %, achieves unprecedented insight into the power usage of the Tegra K1 SoC, and bridges the gap between software activity, power management mechanisms and energy consumption in the Tegra K1's various compute elements and memory. The improvement in modelling accuracy is made possible by correlating dynamic power with fine-grained hardware- and software-activity measurements, while at the same time accounting for voltage, in all of the Tegra K1's processors and memory. Additionally, static power usage is carefully modelled by considering rail- and core-gating mechanisms that shut off leakage current in various parts of the SoC. Capacitive loads and leakage currents are estimated based on linear, multivariable regression through carefully developed model training benchmarks. The resulting power model is evaluated using our video processing filters.

The outline of this chapter is as follows. In Section 4.1, we start by explaining the fundamental concepts behind our model. In Section 4.2, we outline how hardware and software activity is measured in the various components of the SoC. Our regression-based method, as well as our model training benchmarks, are explained in Section 4.3. We also review some major design issues that have impacted the development of our model in Section 4.4. Individual models for the Tegra K1's GPU and CPU are built and evaluated in Section 4.5 and 4.6. Full-hybrid models for all compute cores on three Tegra K1 development kits in Section 4.7.

4.1 Concept and Derivation

In the previous chapter, we have evaluated the three state-of-the-art power modelling methodologies for modern, mobile devices. We have argued that these have individual strenghts and weaknesses that cause them to mispredict power on the Tegra K1 SoC. CMOS-based models, for example, are theoretically well founded, considering both static and dynamic power as well as variations in voltage levels. However, the methods that

are used to build them fail because they assume independency between clock frequency and switching capacitance per cycle in different power domains. Additionally, they often assume that switching capacitance per cycle is constant and independent of the workload. We expect switching activity to vary depending on how software exercises the underlying hardware, for example by executing different instruction types and utilising different hardware components. In principle, rate-based models are good at tackling exactly this problem because they correlate hardware activity with power. Their strength is therefore their ability to capture dynamic power. However, they ignore both rail voltage and static power. State-based models are in turn naturally good at capturing static power, because they abstract hardware into components with associated states. However, these models again ignore dynamic power, and, like rate-based models, voltage. Additionally, we have seen that among the vast range of models for different types of devices in Section 3, there is considerable variety in the type and number of predictors that are used to model power. These all ignore many effects that are related to gating of rails, cores and clocks. For a heterogeneous architecture such as the Tegra K1, it is important to take all these factors into account in the resulting model.

The total power usage of the Tegra K1 can be expressed as the sum of power usage of the core (P_{core}), HP (P_{hp}), memory (P_{mem}) and GPU (P_{gpu}) rails:

$$P_{total} = P_{core} + P_{mem} + P_{gpu} + P_{base} \tag{4.1}$$

where base power P_{base} is assumed to be a constant power draw of idle components within the Tegra K1 SoC and on the Jetson-TK1 development board. The power usage of the networking interface and persistent memory (embedded 16 GB EMMC) is assumed to be constant, where we run all experiments in a RAM filesystem with as little network communication as possible.

Following our discussion on power modelling, we will now develop an improved model based on the CMOS equations. The core improvement is that we will express dynamic power on each rail based on measurable hardware events occurring on that rail. This is similar to rate-based models, but we will also compensate for rail voltage. Dynamic power on a rail is expressed as:

$$P_{R,dyn} = \sum_{i=1}^{N_R} C_{R,i}\rho_{R,i}V_R^2 \tag{4.2}$$

where, on rail R, $\rho_{R,i}$ is a hardware event (in events per second), $C_{R,i}$ is the capacitive load per event per second (in coloumbs per volt), N_R is the number of hardware activity predictors on the rail and V_R is the rail voltage. Expressed differently, we model dynamic power on a rail as the sum of dynamic power contributors on that rail. The contributors are based on measurable hardware activity, where every hardware event is assumed to trigger transistor switching activity with a specific capacitive load. The total power usage of the Jetson-TK1 becomes:

$$P_{jetson} = \sum^{R\in\mathbb{R}} (P_{R,dyn} + P_{R,stat}) + P_{base} \tag{4.3}$$

where $P_{R,stat}$ is the leakage current on that rail (see Equation 2.3) subject to power management mechanisms such as rail- and core-gating. The resulting expression is linear

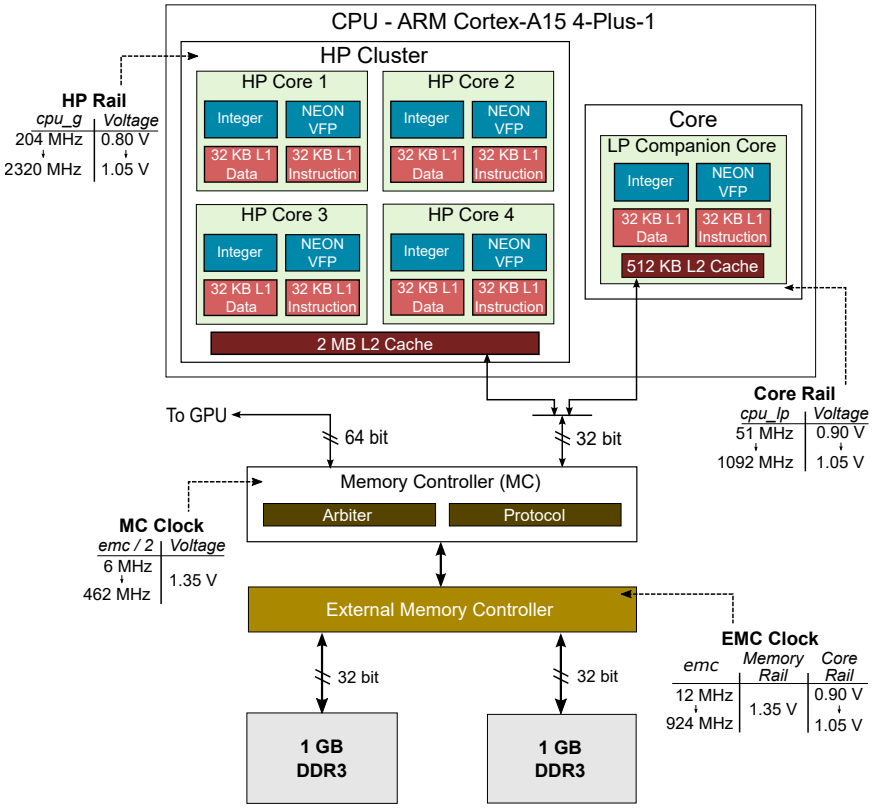


Figure 4.1: Overview of the Tegra K1 CPU architecture.

where the unknown coefficients are the leakage currents on each rail $I_{R,leak}$, capacitive loads $C_{R,i}$ and base power P_{base}. We solve this equation for the unknown variables with multivariable, linear regression.

4.2 Measuring Hardware Activity

To build an accurate power model for the Tegra K1 and its various subsystems such as the LP core, HP cluster, GPU and RAM, we want to express dynamic power in terms of measurable hardware activity in each subsystem. This constitutes a major task in understanding how software excersises the underlying hardware and which factors are important to model power usage. Hardware activity can be measured using standard HPC collection libraries such as PERF and CUPTI. However, we had to implement our own systems to measure activity in hardware units such as the external memory controller. In addition to measurements that reflect dynamic power, it is also important to understand when and how power management mechanisms turn off individual components, impacting both dynamic and static power. In this section, we outline the how we measure hardware activity in the Tegra K1's processors and memory.

4.2.1 CPU

The Tegra K1 is composed of two "clusters" of CPUs: the single LP core situated on the core rail and the quad-core HP cluster on the HP rail (see Figure 4.1). As described in

Chapter 2, all five cores are of the type ARM Cortex-A15 [1]. The Tegra K1 technical reference manual [44] only specifies slight differences in L2 cache size and memory access latency between the LP and HP cluster's CPU cores. Therefore, the CPU cores are assumed to be otherwise electrically identical. This simplifies modelling because factors such as leakage current, cache and instruction cost can be assumed to be similar across the cores. Hardware activity for ARM cores is popularly tracked with the Linux PERF library. PERF provides direct access to various performance counters, HPCs, and insight into hardware activity within each CPU core. However, we also found it necessary to implement spesialised kernel drivers to model the effects of power management.

Every clock cycle in every CPU core triggers switching activity in the core's transistors. The number of `clock cycles` per second is therefore a classical predictor, and has been shown to have strong correlation with power in other research [70]. However, authors never consider the effect of gating mechanisms. The Tegra K1's CPU cores are subject to active power management. The HP rail with its quad-core cluster and clock generator is for example only powered when the HP cluster is active. Otherwise, the HP rail is off, effectively removing static and dynamic power on that rail entirely. When powered but idle, cores are either clock- or power-gated. Clock-gating shuts off clock distribution through the core circuitry, removing dynamic power entirely from that core. Core power gating additionally blocks voltage supply to individual cores and removes `leakage current`. Due to these effects, the leakage current of the LP or HP rail is therefore:

$$I_{R,leak} = \delta(I_{HP/core,leak} + N_{HP/core} I_{cpu,leak}) \tag{4.4}$$

where $I_{HP/core,leak}$ is the base leakage current on the HP or core rail, $\delta \in [0, 1]$ indicates whether the rail is on or off, $N_{HP/core}$ is the number of active cores on each rail and $I_{cpu,leak}$ is the individual leakage current of each core. Unfortunately, there are no standard HPCs that track when and for how long individual cores on individual rails are power-gated. Such information is not exposed through standard APIs or HPCs. Instead, we implemented our own kernel tracing framework that expose the total time (in μs) each individual core is power-gated. This information is then used to track each core's uptime in any time interval. Note that this approach is likely to overestimate the actual core uptime, but exactly by how much is impossible to know. This is because there is some extra overhead between our kernel tracing framework measuring the time when the Tegra K1's power management drivers are asked to power-gate cores, until the cores are physically gated in hardware. Similarly, the same reasoning can be made when the cores are *un*-gated until our tracing framework detects this in software. To measure clock-gating, we originally developed another kernel tracing framework to measure the time when the Tegra K1's power management driver requested individual CPU cores to be clock-gated. However, in the course of our modeling efforts, we discovered that the clock cost was modelled inaccurately (see for example the T-17 model in Section 4.7). This occurs because the CPU cores are also clock-gated in hardware, and not only by software requests. To accurately measure clock-gating, we therefore resorted to use the `active cycles` PERF HPC.

`Caches` are also often assumed to have non-negligible effect on power. Consequently, researchers use HPCs to monitor cache activity as well. The Tegra K1 is comprised of a two-level cache hierarchy (see Figure 4.1). Each core has a private 64 kB L1 cache divided into two equally-sized 32 kB instruction- and data-cache banks. The HP cluster and LP

core has 2 MB and 512 kB L2 cache, respectively, serving any type of access. The ARM HPC implementation [2] defines three types of HPCs reflecting hardware utilisation in these caches:

- *Accesses* count the number of read or write accesses to the respective cache. There are two counters counting instruction- and data-access to the L1 cache.
- *Writebacks* count the number of memory accesses that cause a writeback of data towards off-chip memory. These have separate counters for data- and instruction-accesses, with a total of four counters.
  - L1 cache writebacks count writebacks to L2 or RAM.
  - L2 cache writebacks count writebacks to RAM.
- *Refills* count the number of memory accesses that cause a refills of data towards L1 cache. These have separate counters for data- and instruction-accesses, as well as Transaction Lookaside Buffer (TLB) refills, with a total of five counters.
  - L1 data cache refills count refills from L2 or RAM.
  - L2 data cache refills count refills from RAM.
  - L1 instruction cache refills.
  - L1 data TLB refills.
  - L1 instruction TLB refills.

There are in total 11 HPCs tracking all types of cache utilisation in each core. With a total of six available HPCs per core, it is impossible to track these simultaneously. This is also not reasonable given that we do not know the internal workings of the cache hierarchies, and how these HPCs affect power usage. It is likely that most of these events contribute to power usage. However, we made several observations that were important to the cache power model. For example, we discovered that cache refills, that are triggered by accesses to empty cache lines, are usually accompanied by cache writebacks. This occurs because the Tegra K1's CPU cache eviction policy is least-recently-used: Over time, all cache lines are filled, and accesses that hit empty cache lines cause a writeback of the cache contents to outer memory hierarchies. This is to create new space in the L1 cache for the stale data. In practice, this means that the *rates* at which the cache writebacks and refills occur, is approximately the same in any time interval. Therefore, it becomes impossible to isolate the cost of both using regression. This important detail conserves HPC space, as we only need to track either refills or writebacks on the L1 and L2 caches. Consequently, however, the estimated cost of these events will reflect the cost of both a refill *and* a writeback. We therefore model cache utilisation in terms of *communication* between the cache hierarchies as follows:

$$\rho_{com,l1l2} = (N_{l1d,rf} - N_{l2d,rf}) + N_{l1i,rf} \tag{4.5}$$

$$\rho_{com,l2ram} = N_{l2,rf} \tag{4.6}$$

In Equation 4.5 and 4.6, $\rho_{com,l1l2}$ reflects cache communication between L1 and L2 and $\rho_{com,l2ram}$ reflect cache traffic between L2 and RAM. $N_{l1d,rf}$ is the number of L1 data

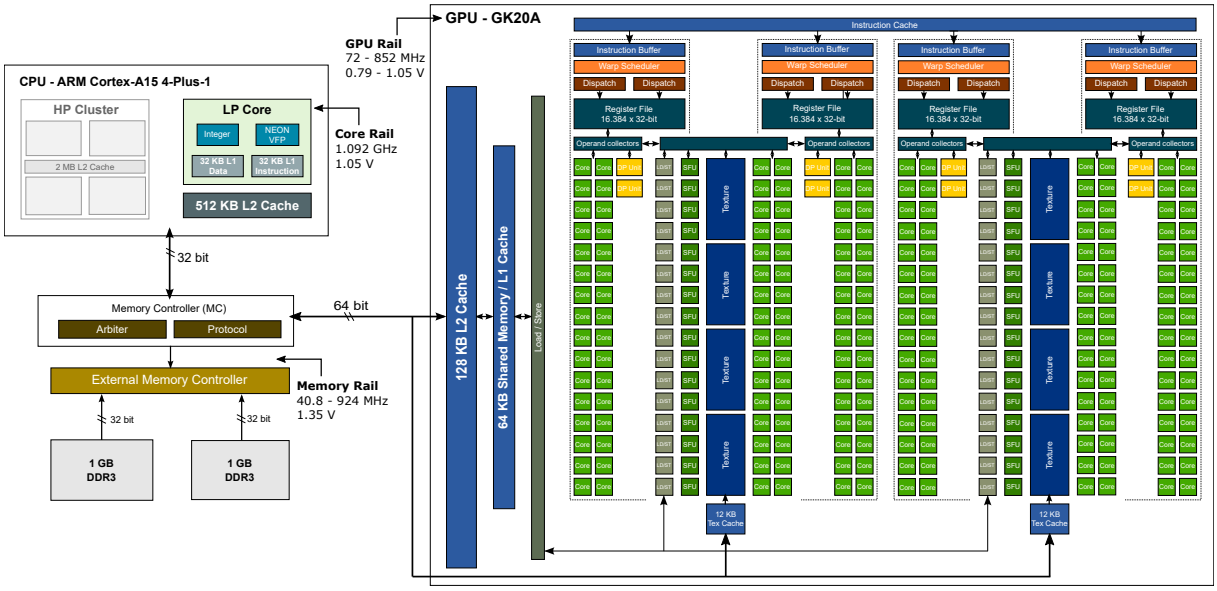


Figure 4.2: Overview of the Tegra K1 GPU architecture.

cache refills from L2 or RAM, $N_{l2d,rf}$ is the number of L2 data cache refills from RAM, and $N_{l1i,rf}$ is the number of instruction cache refills to L1. Because the L1 data cache refill HPC in Equation 4.5 reflects both refills that occur from L2 and RAM, it is important to remove the refills from RAM to L2 cache from the equation. We have also added the number of L1 instruction cache refills. Originally, we also attempted to include TLB refills. However, this significantly reduced the quality of the resulting model. We do not include cache access costs, assuming this to be an integrated part of the CPU instruction cost.

As software executes `instructions` on the cores, they excersise the underlying hardware in various architectural units such as the integer, NEON and VFP units in Figure 4.1. Pricopi et. al. [51] suggest to use the individual types of instructions (integer, VFP, NEON, data movement etc.) executed as model predictors for an ARM big.Little CPU architecture. However, the PERF HPCs counting these types of hardware events are not implemented on the Tegra K1. The only HPC reflecting instruction execution on the Tegra K1 is the `total instructions exectued` counter. This essentially complicates modelling. Without the capability to count the number and type of each instruction executed on the cores, the model looses generality. However, the PERF HPCs can be counted on a per-process basis. We therefore assume that, over time, the capacitive load per *generic* instruction executed per process will remain approximately constant. We believe this to be a valid assumption especially for video processing filters, where the same work is essentially being done repeatedly over consecutive frames in our video processing filters. The downside of this approach is however that instruction power must be carefully evaluated over different processes in the system, and specific care must be taken to vary the processes' instruction throughput in accordance to each other, in order for regression to estimate meaningful coefficients to these events.

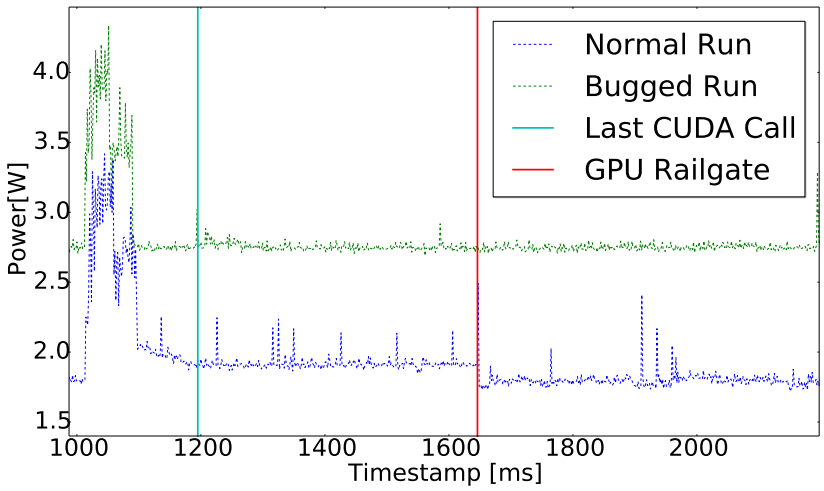


Figure 4.3: Effects of GPU rail- and clock-gating timers.

4.2.2 GPU

The Tegra K1 contains a Kepler-based [42] 192-core NVIDIA GPU. Differently from the CPU, the Tegra K1's GPU is a more parallel architecture. While the CPU is only able to run a maximum of four threads in parallel, the GPU is able to run four groups of 32 concurrent threads called *warps* at the same time through its four warp schedulers (see Figure 4.2). GPU threads are concurrently utilising various hardware units, such as the compute cores, load/store and Special Function Units (SFUs). Due to the high number of concurrently active threads, memory pressure is substantial. If for example 128 concurrent threads access a unique element in an array of 128 8 B double precision floating point values, memory throughput easily becomes a bottleneck. Furthermore, threads often cooperate and share data. For these reasons, the Tegra K1's GPU has a 64-bit interface with RAM (where the CPU only has 32-bit), a 128 kB L2 cache and flexible 64 kB L1 cache that can be used not only to cache data but also to share data between threads.

Like the Tegra K1's CPU, the GPU is also subject to power management. However, there is no publicly available information about when this occurs and what hardware units are affected. Early versions of the Jetson-TK1's GPU kernel drivers exposed two files in sysfs: `railgate_delay` and `clockgate_delay`. These parameters were set to 500 and 50 ms, respectively. The effects of these timers is shown in Figure 4.3. The plot shows the measured Jetson-TK1 power usage over time, with the final CUDA API call at approximately 1200 ms. The figure reveals that after approximately 500 ms, there is a significant drop in power where the GPU rail is powered down, eliminating static power usage on this rail. These types of gating effects are easy to observe, but it is very hard to ascertain whether individual GPU units, such as the compute cores, are subject to for example clock-gating during runtime.

In our work with the Tegra K1, we use CUPTI HPCs to track hardware utilisation in the GPU. However, after some time, we discovered that there is a power management bug involved in using these which is only possible to see if you directly measure the GPU rail voltage. *Any* call to the CUPTI library disables power management on the GPU. It is no

longer clock- or rail-gated for the remaining uptime of the Tegra K1. As can be seen in Figure 4.3, while this bug is active, there is no longer any visible change in power usage 500 ms after the last CUDA call. Additionally, the platform power usage is very high, approximately 0.84 W higher while the GPU rail is powered but idle (between 1200 and 1650 ms). Incidentally, this number closely matches the estimated GPU clock power in Section 4.7. This test was run at GPU frequency 540 MHz, where the GPU rail voltage is 875 mV, with an estimated GPU clock power of 0.87 mW. This indicates that the GPU clock is in fact stuck, and never gated, after CUPTI calls. This unfortunately forces us to ignore the effects of clock- and rail-gating on the Tegra K1's GPU, but simplifies `GPU clock` and `GPU leakage` modelling. Because the GPU clock and the GPU rail is never gated, estimation of these power contributors is a simple matter of logging the GPU clock frequency and rail voltage.

The Tegra K1's `GPU cache` is a complex memory unit. With almost a hundred CUPTI HPCs, the challenge is to identify those that reflect hardware utilisation as generically as possible. For example, the HPCs `gld_throughput` and `gst_throughput` reflect RAM read and write throughput. However, the source of the read or write is transparent. It can reflect a read or write operation from or to actual RAM, L2 and L1 cache. We expect the costs for each of these types of memory accesses to be different. Pricopi et. al. [51] use these counters directly. In contrast, our goal is to track these types of hardware utilisation at a more fine-grained level where the cost of each memory access types, such as L1 and L2 reads and writes, can be found.

L2 cache utilisation can be tracked by two CUPTI HPCs: `l2_subp0_total_read_sector_queries` and `l2_subp0_total_write_sector_queries`. These counters count the raw number of 32 B reads and writes to that cache. However, we have been unable to estimate a capacitive load per 32 B L2 cache write. The estimated load has a higher standard deviation than the other coefficients, and is always much smaller than the other coefficients. We believe this indicates that the GPU's L2 cache is writeback. There is no evidence of this in any publicly available documents for the Tegra K1, but if true means that the GPU's memory write operations are immediately evicted to RAM with negligible switching activity on the GPU rail. As a result we do not estimate any cost to L2 cache writes. Instead, hardware activity associated with RAM writes is measured directly as an integrated part of active GPU memory cycles (see Section 4.2.3).

For the L1 cache, there is no possibility to measure directly the physical number of reads and writes per second (as for the L2 cache). Instead, L1 hardware utilisation must be approximated through other HPCs that reflect the various means of using it. The GPU's L1 cache is a flexible cache which is used for several purposes. First, it is used to cache *local memory*. Exactly what type of memory this is and what is being cached is unclear and poorly documented from NVIDIA. However, it is at least used to cache data related to GPU code (such as code and function parameters) as well as local register spills (which occurs when GPU kernels are out of available registers during computation). Local memory accesses can be tracked with the `l1_local_store_hit` and `l1_local_load_hit` HPCs. These counters increment by one for each transaction. L1 cache is also used to cache *RAM reads*, not writes. This type of utilisation can be tracked with the `l1_global_load_hit` HPC. In this case the cached data is not coherent with RAM.

L1 cache is also used as *shared memory*. From a developer's perspective, this is the

most common and flexible way of using L1 cache, providing a convenient way to share data between groups of threads. Shared memory accesses can be tracked through the HPCs `l1_shared_load_transactions` and `l1_shared_store_transactions`. However, capacitive loads cannot be attributed directly to these counters. For example, in the event that a group of 32 threads read the same shared memory location, that value is broadcasted to all threads of that warp. The value is *physically read* once, but the HPC counting the number of shared loads will inrease by 32. Therefore, the counters must be multiplied with the `shared_efficiency` HPC:

$$L1_{shared,load/store} = L1_{shared\_\{load/store\}} \times shared\_efficiency \tag{4.7}$$

Where the `shared_efficiency` HPC is the ratio between actual shared memory read or write transactions and the requested. However, this solution has drawbacks. The `shared_efficiency` HPC does not distinguish between reads and writes. If the capacitive load per L1 cache shared read or write is different, it becomes impossible to estimate the energy cost related to these events. Fortunately, as we will see in Section 4.7, the estimated capacitive load per shared read and write is similar.

In summary, L1 cache utilisation is approximated as follows:

$$L1_{reads} = L1_{local,read} + L1_{RAM,read} + L1_{shared,load} \tag{4.8}$$

$$L1_{writes} = L1_{local,write} + L1_{shared,store} \tag{4.9}$$

where $L1_{local,*}$ is the number of local memory reads or writes and $L1_{RAM,read}$ is the number of cached RAM reads.

`Instruction` power is easier to track on the GPU because the CUPTI library provides access to all types of instructions executed. The following HPCs are used to estimate instruction power:

- `inst_fp_32 / inst_fp_64` counts floating point operations on different datatypes (32-bit or 64-bit operations).
- `inst_integer` counts integer operations.
- `inst_bit_convert` counts bit conversion instructions.
- `inst_control` counts control flow instructions, such as branching and jumps.
- `inst_misc` counts miscellaneous instructions, such as register moves and NOP instructions.

In our experiments, we could not attribute a positive capacitive load to control instructions. Additionally, the last category of instructions, `inst_inter_thread_communication`, is not used in our multimedia workloads. We therefore removed these components from the model.

4.2.3 RAM

RAM is a computing resource which has never been directly considered in literature. To some extent, it can be argued that RAM power is indirectly considered in certain HPCs. Cache misses [70], for example, are likely to trigger hardware activity in off-chip RAM. However, no authors have attempted to estimate directly the cost of RAM hardware utilisation from various components in the SoC. This is not only limited to the CPU and GPU, but also other hardware peripherals such as PCI devices that also need to communicate with the CPU through this channel.

By default, there is no way of measuring the number of RAM reads and writes. It may, however, be possible to approximate RAM utilisation with *local* HPCs. For example, the CPU has HPCs that count L2 cache refills and writebacks. However, this would occcupy four HPCs (instruction + data) for tracking all types of data traffic to and from RAM. As the Tegra K1's CPU HPC implementation only supports seven concurrent HPCs to monitor, this should be avoided to free up space for other counters. The Tegra K1's GPU again does not have any means to count RAM read and write traffic. Some NVIDIA GPUs have counters for this type of hardware utilisation, but these have a dedicated pool of GPU RAM. The Tegra K1 has only one RAM pool which is shared between the GPU and the CPU. We believe this is why this CUPTI HPC is not implemented on the Tegra K1.

The Tegra K1's EMC does, however, have a specialised means of tracking `utilisation of RAM`. A dedicated hardware monitoring unit called the *activity monitor* (ACTMON) counts two important metrics:

- The total number of active memory cycles serving any system component (CPU, GPU or other).
- The total number of active memory cycles serving GPU or other system components (not including the CPU).

While these counters do not separate between reads and writes, the activity monitor does provide an essential way to track RAM utilisation. We modified the driver for the activity monitor to expose the total number of active memory cycles serving the CPU or GPU (and other) units. Using this as a measure of RAM hardware activity, we estimate capacitive loads to each of these hardware events. In addition to direct hardware utilisation, RAM continuously maintains the consistence of its contents at every `clock cycle`. It is never gated. Therefore, we also use the number of RAM clock cycles per second as a hardware activity predictor.

4.3 Methodology

Our methodology to estimate coefficients (capacitive loads and leakage currents) of the various components on the Tegra K1 is based on multivariable, linear regression. This is similar to the majority of related work. In this regard, our goal is to design a set of benchmarks that stress the various architectural units of the Tegra K1 in such a way that meaningful (non-negative) coefficients can be estimated based on our power modelling predictors. Negative coefficients in a power model, such as those we developed in

Benchmark	Description	Components / instructions under explicit stress										
		CPU	RAM (CPU)	GPU	RAM (GPU)	L2	L1	INT	F32	F64	Conv.	Misc.
Idle CPU	GPU off, CPU in idle state.	✓										
CPU-workload	GPU off, CPU processing.	✓	✓									
Idle GPU	GPU on and idle, CPU in idle state.	✓		✓								
L2 Read	Stresses L2 cache reads only.	✓		✓		✓						
L1 Read	Stresses L1 cache reads.	✓	✓	✓			✓					
L1 Write	Stresses L1 cache writes.	✓	✓	✓			✓					
RAM	Stresses RAM activity (GPU memory cycles).	✓		✓	✓							
Integer	Stresses integer arithmetic unit.	✓		✓		✓		✓				
Float32	Stresses floating point unit.	✓		✓		✓		✓	✓			
Float64	Stresses floating point unit.	✓		✓		✓		✓		✓		
Control	Stresses conversion instructions.	✓		✓		✓		✓			✓	
Misc	Stresses miscellaneous instructions.	✓		✓		✓		✓				✓

Table 4.1: GPU model training suite.

Section 3.5 and reported by for example Limin et. al. [34], are confusing because they ultimately mean that power can be *gained*. Some researchers, such as Xiao et. al. [70], use regression with non-negative coefficients to overcome this issue. However, we argue that an accurate regression-based model should *naturally* produce non-negative leakage currents and capacitive loads. If not, there should be clear explanations as to why they are there. For example, with respect to static power, this is evident in that a closed, non-powered circuit consumes energy when supplied with voltage. We should also be able to attribute positive capacitive loads to hardware events (dynamic power), under the assumption that the hardware event reflects the total switching activity in a group of transistors.

Throughout the model development in this thesis, we have in many cases encountered negative coefficients. These are almost exclusively dynamic power coefficients. Examples are the costs of CPU cache refills and writebacks in Section 4.2.1, as well as the issues attributing capacitive loads to GPU L2 cache writebacks and control instructions (Section 4.2.2). In most of these scenarios, we discovered or hypothesised reasons behind these phenomenon. You *cannot* attribute capacitive loads to CPU cache refills and writebacks, because the event rates do not vary enough when compared to one another. The assumption that the GPU's L2 cache is *writeback*, that explained why no switching activity could be attributed to these events, was partly confirmed by NVIDIA executives. Critically pursuing positive coefficients has also enabled us to discover bugs and other issues that broke the model training phases, such as the CUPTI power management bug (Section 4.2.2), or the fact the the rail voltages on the Tegra K1 is increased to 1.0 V when the SoC is sufficiently cold. In summary, it has helped us develop and argue *which* model predictors to use, as well as *why* we use them, to the extent possible. This is not done by any related modelling efforts.

When developing our power model on the GPU and the LP core [64], we initially attempted a straight-forward approach where we executed some of NVIDIA's own sample kernels over all combinations of GPU and memory frequencies. However, we quickly discovered that this approach had a disadvantage. Logging the power modelling predictors, we observed that several of the CUPTI HPCs occur at a *rate* which is proportional to the frequency level used. For example, halving the GPU frequency halved the number of

Benchmark	Description	Components under explicit stress						Instruction Cost
		RAM (CPU)	L1 - L2	L2 - RAM	INT	FPU	NEON	
Idle CPU	CPU idle.		✓					$2.50\frac{nC}{V}$
L1-WB	L1 writeback stress.		✓					$0.15\frac{nC}{V}$
L1-RF	L1 refill stress.		✓					$0.18\frac{nC}{V}$
MMUL-INT	Matrix multiply (integer operations).	✓	✓	✓	✓			$0.45\frac{nC}{V}$
MMUL-INT-VOL	Same as above, volatile memory.	✓	✓	✓	✓			$0.27\frac{nC}{V}$
MMUL-F32	Matrix multiply (floating point operations).	✓	✓	✓		✓		$0.36\frac{nC}{V}$
MMUL-NEON	Matrix multiply (NEON floating point operations).	✓	✓	✓			✓	$0.44\frac{nC}{V}$

Table 4.2: CPU model training suite.

L2 cache reads per second, but also halved the number of GPU integer instructions per second. Since the predictors (hardware access rates) do *not exhibit enough variation between each other*, it is therefore impossible or at least challenging to estimate meaningful coefficients (capacitive loads) to these events.

An important design goal, which is never mentioned by other authors, is therefore to design training benchmarks such that we guarantee that the *hardware access rates* are diverse enough between training samples to produce meaningful estimations. This can only be achieved by careful development of training benchmarks that are designed with this in mind. For our power models, we developed a methodology where we stress as few architectural units as possible before adding units on top (in a "pyramid-fashion", see the GPU benchmarks in Table 4.1 and the CPU benchmarks in Table 4.2). This is challenging in a number of ways. For example, it impossible to stress only integer arithmetic without also stressing L1 or L2 cache utilisation. This is because the kernels *must* perform global loads or stores. Otherwise, the GPU runtime driver detects that the kernel is not performing meaningful computation, and optimises it away. Imagine for example that we create a GPU kernel with one million threads, where each thread *only* reads one value from an array of one million integers. The CUDA runtime management will detect that these do not yield any meaningful output, and they will therefore never be launched.

The only GPU component which can essentially be stressed alone on the Tegra K1 SoC is the L2 cache. For example, it is possible to have some million threads read the same integer value from RAM, if that value is written back in an if-condition which we know never evaluates to true. Subsequently, other units can be added on top by executing integer, floating point or data movement instructions. RAM activity can be stressed by using GPU assembly instructions that access volatile memory pointers. The resulting benchmarks for the GPU are shown in Table 4.1. Similarly, for the CPU, the same reasoning can be made. However, as outlined in the previous section, PERF exposes a limited set of instruction HPCs and it is therefore simpler to design training benchmarks (see Table 4.2). Here, we implemented benchmarks that stresses cache traffic between the L1, L2 and RAM memory hierarchies. As mentioned in the previous section, the CPU only has a single, global instruction counter. Therefore, our training benchmarks exercise

different types of hardware units, such as integer, floating point and NEON instructions, where we estimate "workload-specific" capacitive loss per instruction per benchmark.

For each of the five core configurations (LP core or any of the four HP cores on), the CPU benchmarks in Table 4.2 are run over all CPU and memory frequencies. Similarly, the GPU benchmarks in Table 4.1 are run over all GPU and memory frequencies, with only the LP core online at 1 GHz. This approach has several advantages:

- It creates natural variation in hardware access rates. Changes in frequency in one domain will vary the rate at which events occur in other domains.
- Rail voltages vary with frequency. This also helps to create diversity in model predictors (see Equation 4.2).
- It is necessary to estimate leakage currents, which can only be modelled by observing changes in power while varying rail voltages. In this respect, it is important to note that the memory rail is always set to 1.35 V. Therefore, it is impossible to model leakage current on this rail.
- It helps trigger clock- and power-gating in individual CPU cores.

The hardware activity predictors and rail voltages are logged in periodic intervals, where the rail voltages are measured and stored as hard-coded, frequency-indexed tables. At the end of each test (processor and memory frequency tuple), the power log is downloaded from the power measurement unit, synchronised with the Tegra K1's local event log, and a profile in csv-format is generated (see Section 2.2.4). The resulting training datasets take less than 24 hours to generate and are composed of between 30 000 and 40 000 samples. We believe that training time can be substantially reduced if fewer frequencies are used, where we essentially only need to cover frequency ranges such that rail voltages are varied. Using matlab, we take care to process the training data as follows:

- For every rail $R \in \mathbb{R}$, all dynamic power predictors (hardware access rates) $\rho_{R,i}$ must be multiplied by the corresponding square rail voltage V_R^2.
- Static power terms (leakage currents) must be taken into account, taking care to compensate for CPU gating (Equation 4.4).

The model predictors are then passed to a regression solver and correlated with measured power. The GPU and CPU models were initially built separately, where the resulting coefficients can be found in Table 4.4 for the CPU and Table 4.3 for the GPU. The GPU model was built before the CPU model, and as a result, most of the CPU predictors in Table 4.4 are missing from the GPU model. In the next sections, we evaluate and discuss these models separately.

4.4 Design Issues

To estimate power or collect power modelling training data, we need to collect HPCs from many sources (see Section 4.2):

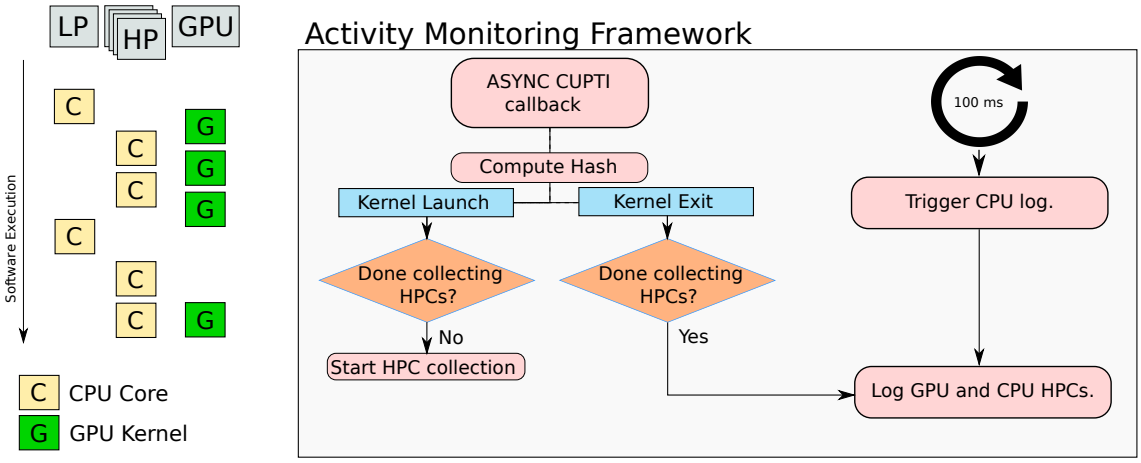


Figure 4.4: Activity monitoring framework.

- We use PERF to track HPCs for the ARM cores. PERF is a Linux HPC implementation that can track different types of hardware and software events. We track several, raw HPCs provided by the ARM CPU cores to measure instruction throughput, active clock cycles and different types of cache utilisation.
- CUPTI is NVIDIA's own HPC framework for GPU kernels, capable of counting almost 200 hardware events related to kernel execution. We measure the number of different instruction types (integer, floating point, etc), GPU cache utilisation and active cycles.
- Our own kernel tracing framework is built into the Linux kernel to expose when CPU power-gating takes place and how many active memory cycles are spent serving CPU and other (GPU) memory requests.
- Additionally, we control and track several platform-specific details:
  - Which rails (HP and GPU) are on at any time.
  - How many HP cores are on, if any.
  - What CPU, GPU and memory frequencies are used.
  - Rail voltages are stored in static voltage-frequency tables in our framework.

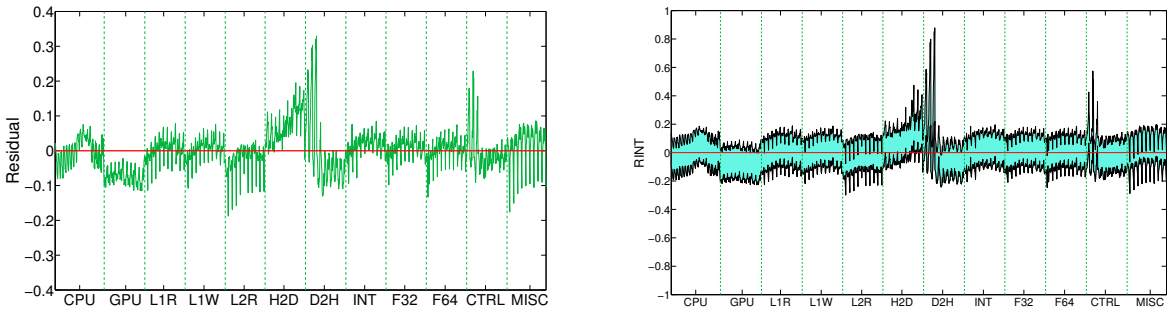
The main design goal of our framework is to impose as small overhead as possible on normal processes running on the CPU and the GPU, as well as *reliable* accounting of hardware activity (see Figure 4.4). This is trivial in the context of the two latter bulletpoints above. Our kernel tracing framework exposes counters in `sysfs` to measure core power gating, and our framework itself controls the CPU cluster, number of cores online and the platform frequencies. However, with regards to PERF and CUPTI there are several issues. A major headache is the *lack of documentation* and, in terms of PERF, *lack of error reporting*. The Tegra K1's CPU cores, for example, each support seven PERF HPCs, where one is always occupied by the `active cycles` HPC. We initially tracked a total of five HPCs for five system processes:

- The full system (aggregate count over all CPU activity).
- Our activity monitoring framework.
- The three ACTMON kernel drivers.

With these processes, we must track a total of 25 HPCs. This number is larger than the number of available HPC slots on each core. It is easy to assume, incorrectly, that this is not a problem. Because the five "processes" all share the same five unique HPCs, one easily expects that these counters are shared when the PERF subsystem is tracking the HPCs over different processes. However, after many failed modelling attempts, we realised that some HPCs for *processes* were never counted (failed without error), but *the full system counters* counted as normal. Consulting the PERF developers, we were made aware that even if the HPCs can in principle be shared in hardware, the PERF system does not support that functionality. PERF instead uses one HPC slot on the CPU per process to track, even if the same HPC is measured over processes. If there is not enough HPC space, the HPCs are multiplexed, *but* priority is given to system-wide counter collection. For our CPU model, this had an adverse effect in that we could not differentiate CPU instruction power between our video processing benchmarks and system services as originally intended. In the end, we just counted system-wide HPCs. These issues have heavily influenced model development, as we originally intended to separate instruction power for different processes in addition to different CPU benchmarks.

Our understanding of CUPTI has also developed over time, and we have learned several insights that have influenced our design choices. To count HPCs for kernel launches, we use *callbacks* that are triggered on function call entry and exit (see Figure 4.4). For example, as our code calls `cudaLaunch(..)` to execute kernels on the Tegra K1's GPU, a callback is executed *on that thread* on function call entry and exit. But *how* is that callback invoked? It can be *serially*, on the thread that executes `cudaLaunch(..)`, or it can be *asynchronously* in a separate thread. The CUPTI documentation does not state how the callbacks are implemented. The reason why this is important is related to *synchronisation*. We must know exactly what kernel was executed at what time, and for how long it was running. Initially, we used a type of synchronisation primitive that *blocked* until all pending GPU activity had finished inside these callbacks. However this expensive blocking call added several hundred milliseconds to kernel launches, which is not desirable. In the end, we talked with several of the CUPTI engineers at NVIDIA's own GPU technology conference, and we learned that the callbacks are executed serially on the thread that invokes `cudaLaunch(..)`. This insight enabled us to avoid expensive blocking synchronisation primitives by using stream synchronisation and CUPTI events to track timing for kernel launches.

Logging of hardware activity is done every 100 ms, or at the end of any kernel launch (see Figure 4.4). PERF is light-weight and easy to use; there is no overhead of collecting them apart from reading special HPC files. However, the CUPTI HPC collection incurs a substantial amount of overhead and not all the HPCs (for types of instructions, cycles, cache utilisation) can be tracked simultaneously. A total of four runs are needed to track HPCs for each kernel. It is trivial to understand that an overhead of 200 ms for a kernel that normally takes 20 ms to finish is not tolerable. Our solution is to avoid tracking HPCs at every kernel launch, and instead measure the GPU HPCs once for every unique



(a) Residual error.

(b) Outliers.

Figure 4.5: Training data statistics for the GPU model.

kernel. For example, our framework tracks HPCs for every new kernel, the first four times that kernel is launched. To implement this, we compute a hash for each kernel at `cudaLaunch(..)` invocations. The hash is a function of the kernel's symbol name and execution configuration. On subsequent kernel launches, new CUPTI HPCs are measured. Once HPC collection is complete for a kernel, that kernel will be logged at every exit from `cudaLaunch(..)`. This solution avoids the need to continuously measure HPCs, but it also has disadvantages. For example, the HPC measurements (integer instructions, cache utilisation, etc) can vary depending on kernel launch parameters. Our rotation filter is an example workload where hardware utilisation will vary depending on the angle to rotate the video. However, in the case of most our filters, it is enough to distinguish the kernels depending on their launch configuration.

4.5 GPU Model

In this section, we discuss our developed GPU model. In this model, only the CPU's LP core is activated and operating at 1 GHz. The CPU is only running our profiler and system services, where the activity monitor kernel drivers are the most significant processes. Inspired by Xiao [70] we attribute capacitive loads to the number of CPU instructions per cycle and the number of cycles per instruction. This is different from the CPU model in Section 4.6, where we attribute capacitive loads to the number of instructions per *second*. The effects of CPU gating, cache traffic and variable instruction power are not considered (see Section 4.6). This design choice was made based on the assumption that the CPU consumes negligible power compared to the GPU, which was the main focus of the study. We first discuss the quality of the resulting power model, in terms of a statistical analysis, in Section 4.5.1. Then, we evaluate the accuracy of the model under our video processing filters in Section 4.5.2.

4.5.1 Regression Analysis

The model training data is collected as described in Section 4.3 using the specialised GPU model training benchmarks in Table 4.1. The final dataset is composed of 1800 samples, where each sample contains measured hardware utilisation and average power

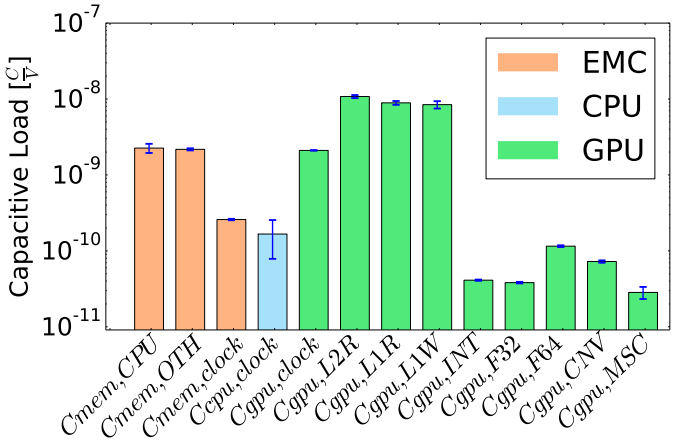


Figure 4.6: GPU power model dynamic power coefficients.

usage over all possible combinations of GPU and memory frequencies. The residual error of the model can be seen in Figure 4.5a. Each of the 1800 residuals are grouped into the model training benchmark they belong to. Studying the figure, we see that the residual error generally remains below 100 mW. Some benchmark groups, in particular the pure memory-copy ones (host-to-device and vice versa), include worst-case residual errors up to 300 mW. However, the mean observed residual is only $8.0810^{-16}W$, with a standard deviation of 57 mW. This is very close to zero, indicating that the correlation between observed power usage, and the model predictors (capacitive loads and leakage currents), can be described as a linear function. Figure 4.5b shows residual confidence intervals, where intervals which do not contain zero (the red line) are likely to be outliers. The plot shows that there are very few of these, where 96.2 % of the samples contain zero.

From the GPU model coefficients in Table 4.3, we see that all predictors are above zero. This is a good initial result because it indicates that we have successfully separated and captured individual static and dynamic power coefficients across the platform. Most of the GPU coefficients, such as clock and instruction costs, yield very small 95 % confidence intervals (see Figure 4.6). This is good because we can generally expect these coefficients to be very close to their estimated values (based on the observations in the training data). The estimated cost for miscallaneous instructions ($C_{gpu,MSC}$) exhibits higher variation than the others, but it is difficult to establish exactly why this occurs. One possible reason is that miscallaneous instructions actually reflect a range of different instruction types, each of which excersise different hardware units (similarly to the CPU's global instruction counter). For example, the CUDA documenation [41] places different instructions such as `NOP`, `VOTE`, `S2R` and other barrier instructions in this category. Additionally, sources claim that the miscellaneous instructions HPC counts other instruction types as well, such as `MOV`. This is a fact we can confirm on the Tegra K1's GPU as well. Because the CUDA compiler can compile our benchmarks into any of these miscellaneous instructions (or more), it is possible that the estimated coefficient reflects the cost of all of these. Consequently, it would explain the larger variation in the coefficient estimate for this group of instructions.

Rail	Predictor	Description	Coefficient	Value	95 %
GPU	V_{gpu}	GPU voltage	$I_{gpu,leak}$	$0.27A$	$[0.24, 0.30]$
	$\rho_{gpu,clock}$	Total clock cycles per second	$C_{gpu,clock}$	$2.10\frac{nC}{V}$	$[2.08, 2.12]$
	$\rho_{gpu,L2R}$	L2 cache 32B reads per second	$C_{gpu,L2R}$	$10.79\frac{nC}{V}$	$[10.29, 11.28]$
	$\rho_{gpu,L1R}$	L1 cache 4B reads per second	$C_{gpu,L1R}$	$8.90\frac{nC}{V}$	$[8.38, 9.41]$
	$\rho_{gpu,L1W}$	L1 cache 4B writes per second	$C_{gpu,L1W}$	$8.43\frac{nC}{V}$	$[7.49, 9.36]$
	$\rho_{gpu,INT}$	Integer instructions per second	$C_{gpu,INT}$	$41.11\frac{pC}{V}$	$[40.35, 41.87]$
	$\rho_{gpu,F32}$	Float (32-bit) instructions per second	$C_{gpu,F32}$	$38.15\frac{pC}{V}$	$[37.27, 39.04]$
	$\rho_{gpu,F64}$	Float (64-bit) instructions per second	$C_{gpu,F64}$	$115.33\frac{pC}{V}$	$[112.23, 118.43]$
	$\rho_{gpu,CNV}$	Conversion instructions per second	$C_{gpu,CNV}$	$72.42\frac{pC}{V}$	$[69.83, 75.01]$
	$\rho_{gpu,MSC}$	Miscellaneous instructions per second	$C_{gpu,MSC}$	$28.36\frac{pC}{V}$	$[23.21, 33.52]$
Memory	$\rho_{mem,clock}$	Total clock cycles per second	$C_{mem,clock}$	$258.66\frac{pC}{V}$	$[253.01, 264.30]$
	$\rho_{mem,CPU}$	CPU busy memory cycles per second	$C_{mem,cpu}$	$2.25\frac{nC}{V}$	$[1.94, 2.57]$
	$\rho_{mem,OTH}$	GPU busy memory cycles per second	$C_{mem,oth}$	$2.17\frac{nC}{V}$	$[2.10, 2.24]$
Core	V_{cpu}	CPU voltage	$I_{cpu,leak}$	$0.79A$	$[0.62, 0.95]$
	$\rho_{cpu,cpi}$	CPU instructions per cycle	$C_{cpu,cpi}$	$3.72\frac{mC}{Vs}$	$[1.78, 5.66]$
	$\rho_{cpu,acl}$	CPU active cycles per second	$C_{cpu,acl}$	$166.62\frac{pC}{V}$	$[78.52, 254.72]$
Other	P_{base}	Base power	1	$0.78W$	$[0.62, 0.94]$

Table 4.3: GPU model.

Coefficients related to CPU execution show broader estimation variation than the GPU coefficients. This is expected where we have not modelled the CPU as accurately as the GPU. For example, the cost of active CPU and GPU memory cycles is similar (2.25 and 2.17 $\frac{nC}{V}$, respectively), but the CPU memory cycle cost has a broader 95 % confidence interval of about $\pm 0.30\frac{nC}{V}$. The active GPU memory cycle cost has a better interval which is closer to the estimate at $\pm 0.07\frac{nC}{V}$. This is because we are not intentionally stressing active CPU memory cycles enough through our benchmarks, especially when compared to active GPU memory cycles. Also, the off-chip CPU memory activity observed in our training data is caused by for example driver accesses to various other buses and units on the SoC. The coefficients on the core rail in Table 4.3 show significant variation in leakage and the cost of LP core clock cycles and instructions per second. The large confidence intervals on the CPU side of the platform indicates that we do not capture accurately enough the hardware activity occurring here. For instance, in this model, we have ignored CPU gating effects. Also, modelling CPU instructions relative to the number of cycles executed as proposed by Xiao [70], is not a good design choice. The rationale behind this must be that, as more cycles are needed to execute a single instruction, the lower the average power will be. In other words, instruction power is expressed as a function of CPU cycles. However, the faster a CPU is operating (in terms of clock frequency), the faster the rate of instruction execution will be, regardless of how many cycles it takes to complete a single instruction. In this regard, it seems reasonable to expect that power also increases with the *speed* that the processor is operating at.

Studying the dynamic power predictors in Figure 4.6, we see that the hardware events corresponding to cache loads and stores are two orders of magnitude more expensive than generic compute instructions. This result indicates that cache memory usage should be avoided when and if possible. This can be done by for example using local register space to store data instead of shared memory, or by disabling caching at different levels on data movement instructions. We perform additional experiments on this topic in Section 5.4.1. The cost for reading and writing L1 and L2 cache is generally similar at around $10.0\frac{nC}{V}$. This result is promising if we assume that the same type of transistors are used in these cache hierarchies.

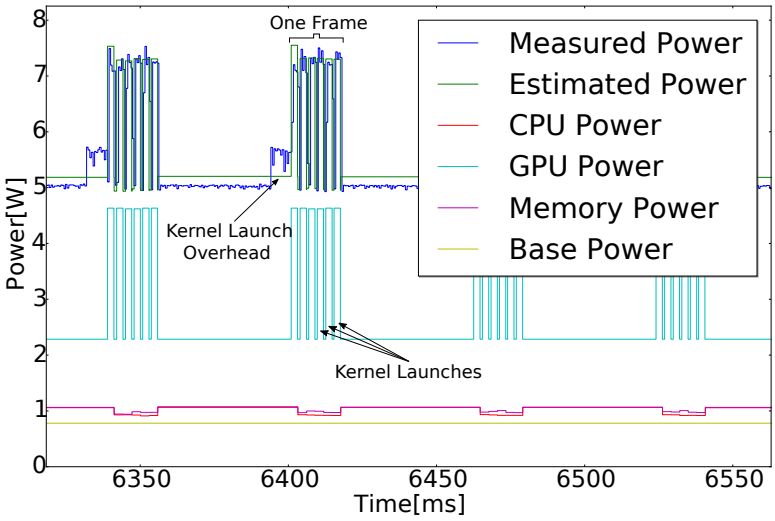


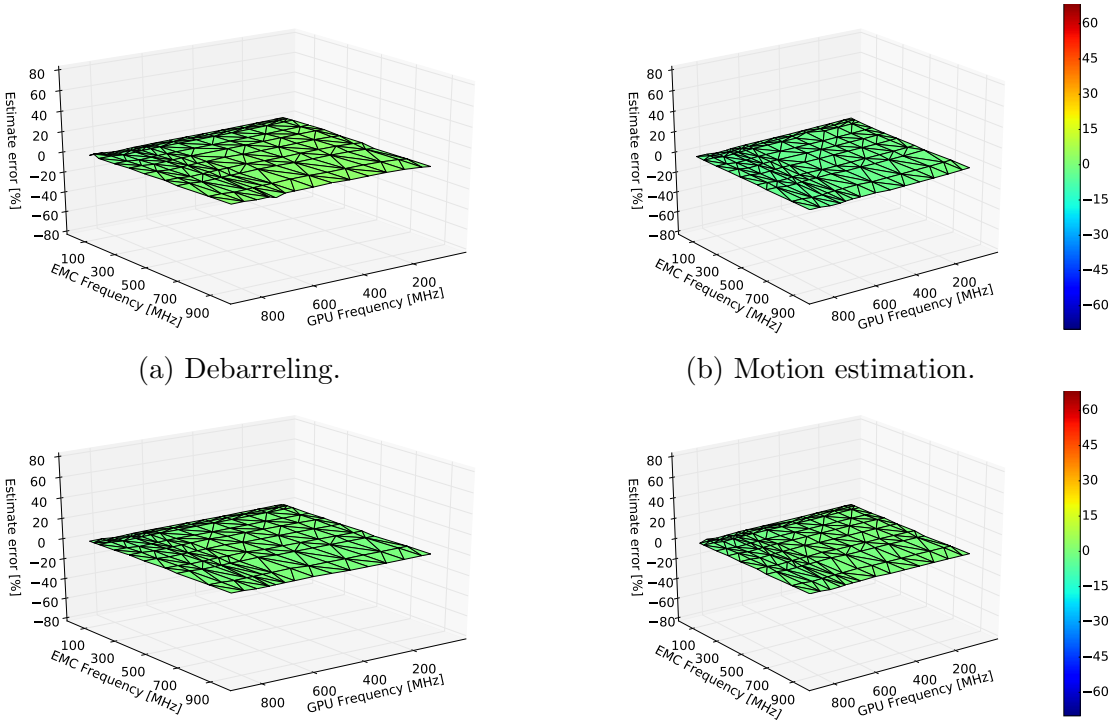
Figure 4.7: Power over time for a single DCT frame.

The capacitive load per GPU clock cycle is $2.10\frac{nC}{V}$, which is an order of magnitude higher than the CPU and memory clock. The fact that the GPU clock estimate is higher than on the CPU can be attributed to the large number of compute cores and other processing elements on the GPU. However, it is important to note that we are comparing with a dedicated low power CPU core. In a real application scenario, the HP cluster is also likely to be on, with four individial cores that may have even higher individual clock costs. Furthermore, as GPU power management is broken in these experiments (see Section 4.2.2), it is possible that the cycle cost in a "non-nugged state" is lower.

Integer, floating point, conversion and other types of instructions have the smallest costs per event, between 40.0 to $115.0\frac{nC}{V}$. The cost of the different instruction types exhibit more variation. Single-precision floating point operations are less expensive than double-precision operations, which is expected. Interestingly, the estimated cost per integer instruction is similar to a single-precision floating point instruction, but this does not mean that the operations cost the same amount of energy. This depends on the time taken to complete each instruction. Integer operations are for example often assumed to be faster than floating point instructions. In this case, less GPU processor cycles will be consumed to complete the operation, and the integer instruction will draw less energy (when considering the whole processor).

4.5.2 Model Accuracy: Video Processing

In this section, we discuss and evaluate our GPU model's accuracy. To illustrate the type of insight achieved from the GPU model, consider Figure 4.7. This figure shows measured and estimated power components in a single DCT frame encoding interval, where the peaks in the plot represent kernel launches (six peaks, or kernel launches, per frame). We see that the total estimated power is very accurate over time and close to 100 %. The model allows us to see how much energy is being consumed on each of the rails, without the need for specialised measurement circuitry. As expected, the CPU and memory power



(a) Debarreling.

(b) Motion estimation.

(c) DCT.

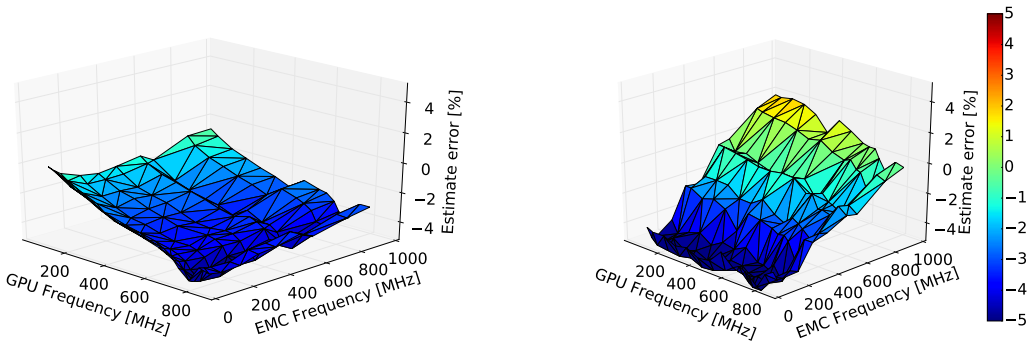
(d) Rotation.

Figure 4.8: GPU model prediction error for different filters.

components are relatively constant over time where there is not much activity on these rails. The model is very accurate *while* kernel launches take place, but appears less precise in the periods between frames. During this time, frames are read and written from and to RAM, and there is a non-negligible kernel launch overhead before the first kernel launch at the beginning of each frame. This overhead is represented as an average since the exit of the last kernel of the previous frame. Furthermore, as stated in the previous section, the CPU model coefficients have larger confidence intervals (less precise), which can worsen the prediction accuracy between consecutive frames.

Figure 4.8 shows the model error in % for a total of 70 processed frames over all GPU and memory frequency combinations, for the debarreling, MVS, DCT and rotation workloads. The estimation is very accurate. Not considering the MVS filter, the accuracy of our model is over 99 % on average and always above 96 %. This demonstrates that our model, which is built using entirely synthetic benchmarks, is able to consisely capture the power usage of kernels comprised of a more complex mix of instructions. Furthermore, it is a significant improvement over the accuracy of the rate- and CMOS-based models in Section 3.5, where the model error could be as large as 50 %.

The MVS filter shows the worst prediction error of all our benchmarks of up to 4 %. For this reason we now focus our discussion on this filter with a closed-up plot of its prediction error in Figure 4.9a. We see that, in general, estimation is good (between zero to one percent error) at low GPU frequencies. Moving toward the lowest memory and GPU frequencies, estimation accuracy tends to degrade toward its lowest point at 4 %. This trend is true for all the filters, but hard to see in Figure 4.8. Furthermore, we see that at 756 MHz GPU frequency, estimation accuracy shows a significant drop.



(a) Normal.

(b) No kernel launches.

Figure 4.9: Power prediction error for the motion estimation kernel.

It is important to note that there is a non-negligible amount of time where the GPU is *not* actively processing, but the CPU is busy reading the next frame or writing results in memory (for example between the kernel launches in Figure 4.7). When error starts to increase, we see that prediction error during this period is larger than when the GPU is active. Consider for example Figure 4.7, where a single peak (Y-frame motion estimation kernel) is shown. For this test, the error was 4 %. It is clear that the estimation during the kernel launch is very accurate. However, in the period between frames, the estimation is worse, leading us to the conclusion that the CPU model is not well enough designed and developed. We also believe that inaccuracy in HPC counting for the MVS filter can reduce model accuracy. This GPU workload excersises shared memory more, and runs for 5 to 26 times longer than for the other filters, depending on frequency settings. We found that the CUPTI HPCs for shared memory transactions do not appear to be reliable, which can cause our model to mispredict more than the other filters. For the U and V frames, CUPTI is for example not able to count shared memory transactions at all. Only shared memory transactions for the Y frames are actually counted.

To further test the accuracy of the CPU and memory model, we run a differential test on the MVS benchmark. No GPU kernels are executed, but frames are read and results written as normal. To avoid compiler optimisations that detect the absence of kernel launches and consequently optimise code away, we conditionally execute the MVS kernel in an if-statement which never evaluates to true. The result can be seen in Figure 4.9b. Compared with the normal experiment in Figure 4.9a, where GPU execution is enabled, we see that the model still mispredicts down to 4 % even when the GPU is not used. As memory frequency increases, estimation accuracy also increases and is close to 100 % accurate at 924 MHz. Again, this result indicates that the CPU and memory models may not be modelled precisely enough. However, the model still succeeds to capture most of the relations. We believe that a better power model for the CPU is necessary to increase prediction accuracy.

In this section, we have evaluated our GPU model for the Tegra K1. The model was built using carefully designed, synthetic benchmarks that stressed individual architectural units on the Tegra K1's GPU. We have shown through our regression analysis that our modelling methodology successfully captures the relationship between measured power

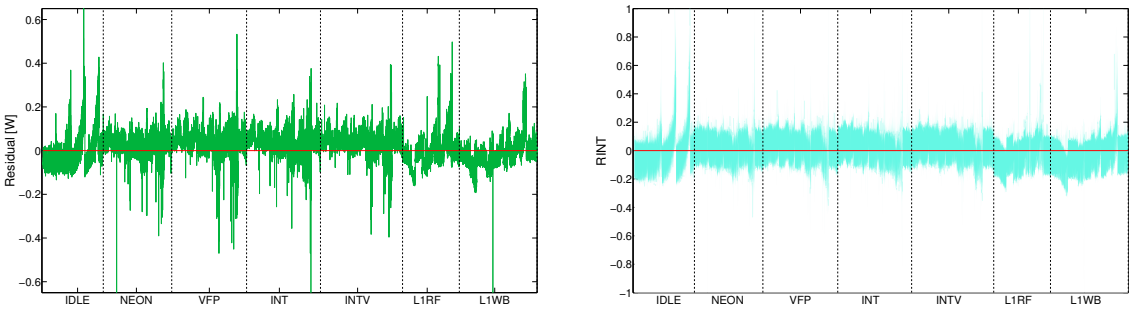
usage of the Tegra K1, and our dynamic and static power usage predictors. We have also discovered a power management bug on the Tegra K1, and that the GPU's L2 cache is write-back. Furthermore, we have evaluated and discussed the accuracy of the platform over all GPU and memory frequencies, using our video processing filters. The model accuracy is close to 100 % with worst-case errors of only 4 %. This extensive validation over different platform frequencies has not been done before in the literature. However, as we have seen by removing GPU execution from the MVS experiment, modelling errors of up to 4 % can occur. For the MVS filter, we also encountered issues with inaccurate CUPTI HPC counting of shared memory transactions. This can lead to reduced model accuracy. We believe that an improved CPU model and further investigations into the reliability of CUPTI HPCs to be good starters for improving the modelling accuracy.

4.6 CPU Model

In this section, we discuss our developed CPU model for the Tegra K1. The CPU model is built according to the same methodology as the GPU (presented in Section 4.3). For each core configuration (LP core or HP cluster with one, two, three or four cores active), the CPU model training benchmarks listed in Table 4.2 are run over all possible memory and CPU frequency combinations. Model predictors are sampled at 100 ms intervals for a total of 38 000 samples. As discussed in Section 4.2 there is a lack of PERF HPCs to track hardware utilisation in the various compute units of the Tegra K1's CPU. Pricopi et. al. [51], for example, have access to individual HPCs measuring data movement, integer and floating point instructions. The Tegra K1 only has one generic instruction counter. This complicates modelling, because we expect the capacitive load per instruction to vary between the different system processes, and how these excersise the underlying hardware. We therefore assume that each individual process will exhibit the same average instruction cost over time, and use this to model the average capacitive load per generic instruction per process in the system. This can pose a problem with regards to varying the instruction throughputs enough to estimate meaningful (non-negative) per-process capacitive instruction loads. However, we found this to be more trival than on for example the Tegra K1's GPU, because the way system processes and their threads are scheduled is arbitrary. For example, the Linux scheduler, conditionals, interprocess communication, blocking synchronisation primitives and sleeping all contribute to varying the rates of cache accesses and instruction throughput across samples. In this section, we first discuss the quality of the developed CPU model in terms of a statistical analysis in Section 4.6.1. In Section 4.6.2, we model application-specific instruction power for our video processing filters. We evaluate the accuracy of our CPU model with real measurement of the platform in Section 4.6.3.

4.6.1 Regression Analysis

In this section, we discuss the quality of our CPU model in terms of a statistical analysis. Figure 4.10a shows the residual error over the training data, categorised into the groups of CPU training benchmarks. The mean residual is $6.624210^{-15}W$, which is very close to zero. This indicates that the correlation between observed power usage, and the model predictors (capacitive loads and leakage currents), can be described as a linear function.



(a) Residual error. (b) Outliers.

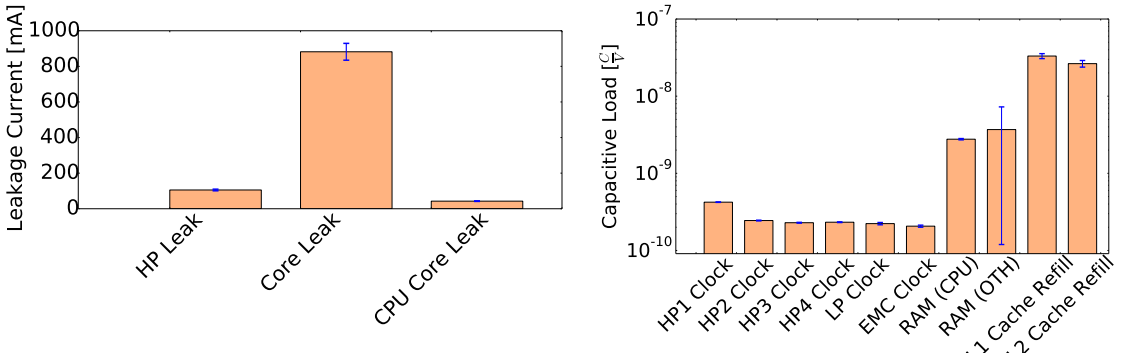
Figure 4.10: Training data statistics for the CPU model.

Rail	Predictor	Description	Coefficient	Value	95 %
HP	V_{hp}	HP rail voltage (when powered)	$I_{hp,leak}$	$105.41mA$	[100.81, 110.01]
	$\rho_{hp,clk1}$	Active cycles (one core)	$C_{hp,clk1}$	$422.72\frac{pC}{V}$	[419.74, 425.71]
	$\rho_{hp,clk2}$	Active cycles (two cores)	$C_{hp,clk2}$	$244.35\frac{pC}{V}$	[241.71, 247.00]
	$\rho_{hp,clk3}$	Active cycles (three cores)	$C_{hp,clk3}$	$228.94\frac{pC}{V}$	[226.17, 231.71]
	$\rho_{hp,clk4}$	Active cycles (four cores)	$C_{hp,clk4}$	$232.43\frac{pC}{V}$	[229.90, 234.96]
Core	V_{core}	Core rail voltage	$I_{core,leak}$	$881.91mA$	[834.66, 929.17]
	$\rho_{core,clk}$	Active cycles (LP core)	$C_{core,clk}$	$222.86\frac{pC}{V}$	[215.11, 230.61]
Common	$V_{com,online}$	CPU Core leakage	$I_{cpu,leak}$	$42.66mA$	[41.19, 44.13]
	$\rho_{com,l1l2}$	L1 data / instruction cache refills	$C_{com,l1rf}$	$33.13\frac{nC}{V}$	[30.67, 35.58]
	$\rho_{com,l2m}$	L2 data cache refills	$C_{com,l2rf}$	$26.40\frac{nC}{V}$	[23.79, 29.00]
	$\rho_{com,ips}$	Instructions (workload-specific)	$C_{com,l1tlb}$	See Figure 4.14	
Memory	$\rho_{mem,clk}$	Clock cycles	$C_{mem,clk}$	$206.60\frac{pC}{V}$	[200.24, 212.97]
	$\rho_{mem,CPU}$	CPU memory cycles	$C_{mem,cpu}$	$2.76\frac{nC}{V}$	[2.71, 2.82]
	$\rho_{mem,CPU}$	Other memory cycles	$C_{mem,cpu}$	$3.69\frac{nC}{V}$	[0.11, 7.26]
	$\beta_{mem,204}$	Power offset at 204 MHz	$P_{mem,204}$	$-3.10mW$	[-7.17, 0.97]
	$\beta_{mem,300}$	Power offset at 300 MHz	$P_{mem,300}$	$53.38mW$	[50.21, 56.56]
Other	P_{base}	Base power	1	$671.91mW$	[634.69, 709.14]

Table 4.4: CPU model.

The standard deviation is 69.4 mW, which is similar to the GPU model. Figure 4.10b shows the residual interval, where samples that do not include zero are indications of outliers. We see that there are very few samples where this occurs. With a coverage 96.73 % samples that contain zero, we do not remove any samples from the model training data.

The coefficients for the CPU model are shown in Table 4.4. Studying the base power component P_{base} we see that it has been estimated to be 0.67 W. This is lower than in the GPU model (see Table 4.3), where the base power was estimated to 0.78 W. Additionally, the 95 % confidence interval is much lower between 0.63 and 0.70 W, where for the GPU model, it was between 0.62 and 0.94 W. This may be due to the number of modelling improvements (clock- and power-gating, per-process instruction cost, etc.). Additionally, the GPU rail is completely off, removing disturbance from GPU rail static and dynamic power. Figure 4.11a shows the leakage currents (static power contributors). The core rail has an estimated leakage of 881.91 mA. This is substantial compared to for example the HP rail leakage, which is estimated to be only 105.41 mA. Additionally, the core leakage current estimate has a higher 95 % confidence interval than the other leakage currents (see Figure 4.11a). The large difference in leakage currents between the rails, is reasonablethe given that the core rail includes many additional hardware components (see Section 2.1).



(a) Leakage currents.

(b) Dynamic power.

Figure 4.11: Generic CPU model coefficients. Error bars show the 95 % confidence interval.

The core rail powers more functional units, such as two display controllers, various DSPs for video and audio processing, PCI-express- and USB-controllers, buses and more. The HP rail only powers the HP rail cluster clock. An important question here is thus whether it is possible to power off additional circuitry on the core rail (such as clocks). Finally, the per-core leakage is estimated to be 42.66 mA. This leakage adds to the static power usage of the SoC when the core is not under power-gating. Interestingly, the base power and core rail leakage estimates are similar to those made in our first CMOS-based modelling attempts on the Tegra K1 [63]. Here, base power was estimated to be 0.82 W, with a core rail leakage current of about 800 mA. However, this is also where the similarities end. The per-core leakage current, for example, was estimated to be around 150 mA, which is over three times higher than the current estimate. The results are not directly comparable, however, as this model was based directly on the CMOS equations. Compared to our more refined model in this section, it is too simple in terms of the number of predictors to accurately model power usage of the Tegra K1.

The generic model coefficients are shown in Figure 4.11b. These are assumed to be *generic* in that they are not expected to vary depending on the workload. From the figure we see that the HP cluster clock coefficients vary depending on which core they power. The first HP clock, for example, has an estimated capacitive load per clock cycle at $422.72\frac{pC}{V}$. The three other clocks have more similar cost between 228.94 and $244.35\frac{pC}{V}$. It is reasonable to expect that the actual switching activity per clock cycle in *each core* is similar at around $240.00\frac{pC}{V}$, but that the cost per cycle of having the HP clock generator *on* adds to the capacitive load per cycle when only one HP core is on. This is reasonable if the clock-generator itself is shutdown entirely when only one HP core is active, and that core is clock-gated. The cost per active CPU memory cycle is estimated to be $2.76\frac{nC}{V}$. This is higher than what was estimated in the GPU model ($2.25\frac{nC}{V}$), but the 95 % confidence interval is much smaller (between 2.71 and 2.82 instead of 1.94 and 2.57). In the GPU model, only system services such as drivers were dominating execution time on the LP core, and CPU memory activity was not stressed to the extent done here. It is possible that whatever active CPU memory cycles triggered in the GPU model, constituted lower-

overhead accesses to other computational units on the Tegra K1, such as DSPs and PCI-E devices. Consequently, the cost per "driver-access" can be thought of as potentially lower. The CPU memory cycle cost in the CPU model, however, reflects application-specific memory accesses and the cost per active cycle is shown to be higher. Interestingly, for the cost of other active memory cycles is now at $3.69\frac{nC}{V}$ with a significantly large 95 % confidence interval between 0.11 and $7.26\frac{nC}{V}$. This is again because there are very few active memory cycles stemming from other sources, where the GPU is not on. L1 and L2 cache refill events are estimated to cost $33.13\frac{nC}{V}$ and $26.40\frac{nC}{V}$, respectively. With the exception of the other memory cycle estimate, all the generic model predictors for this model are credible with very small 95 % confidence intervals.

4.6.2 Instruction Power

Modelling instruction power on the Tegra K1's CPU is more challenging than for the GPU. This is because the CPU cores do not implement HPCs to count the types and number of instructions executed, such as floating point, integer and data movement operations. The CPU cores only implement an HPC that counts the total number of executed instructions. This essentially complicates modelling. However, instruction power on the Tegra K1's CPU can still be estimated if we assume that a process' average capacitive load per instruction remains the same over time. In this section, we show how CPU instruction power can still be estimated on a per-process basis by observing the changes in *residual power*. Residual power is the remaining power usage of the Tegra K1 after all other (*generic*) power components, such as leakage currents and cache power, have been removed.

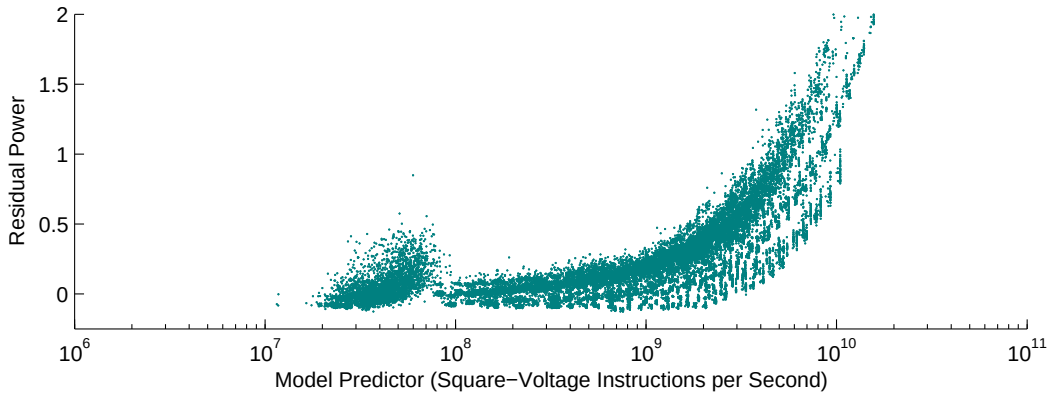
To analyse instruction power, we separately estimate the average capacitive load per CPU instruction per benchmark. Figure 4.12 illustrates the basic observations behind the idea. Assume that most of the CPU model coefficients in Table 4.4 are *generic*, that is, they do not vary depending on the current CPU workload. Further assume that instruction capacitive load is *process-specific* and varies depending on how each process excersises each CPU core. For every sample $n \in \mathbb{N}$, where $\mathbb{N}$ is the set of 38000 CPU model training samples, the total power usage of each sample can be described as:

$$P_n = P_{n,gen} + P_{n,res} \tag{4.10}$$

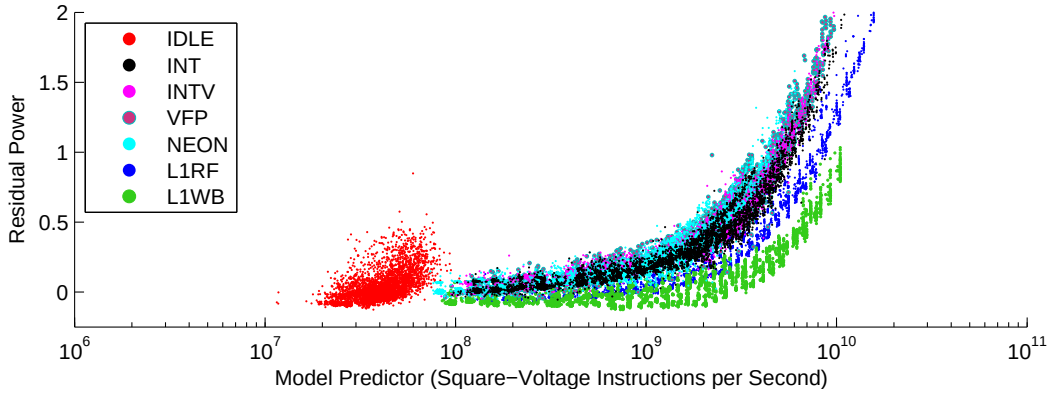
where P_n is the measured average power of sample n, $P_{n,gen}$ are the generic power components stemming from clock, cache and leakage terms for sample n and $P_{n,res}$ is a *residual* power component for sample n. Residual power $P_{n,res}$ is assumed to be related to the rate of instruction execution as follows:

$$P_{n,res} = \rho_{n,ips} C_{n,ips} V_{com}^2 + P_b + e \tag{4.11}$$

where $\rho_{n,ips}$ is the number of instruction per second executed by sample n, $C_{n,ips}$ is the unknown average capacitive loss per executed instruction, and V_{com} is the CPU voltage (depending on frequency settings and which cluster is active). We abbreviate $rho_{n,ips} V_{com}$ as V^2IPS, which can be interpreted as the number of square-voltage instructions per second. P_b is an intercept (for $V^2IPS = 0$) and e is a modelling error which follows the same distribution as seen in Figure 4.10a. Figure 4.12a shows the residual power component $P_{n,res}$ for every sample in the training data set plotted versus the model predictor



(a) Over all instructions.



(b) Over workload-specific instructions.

Figure 4.12: Residual power (workload-dependent instruction power).

($\rho_{n,ips}V_{com}^2$, V^2IPS). In other words, if our assumptions are correct, the residual power in Figure 4.12a should reflect *only* the cost of instructions across all the samples in the model training data set. Figure 4.12a does not distinguish between types of instructions (benchmarks). We see that the residual power lies in the region between 0 and 2 W. Some of the samples also have negative residuals below zero, but the majority are above. The presented data indicates that there is a large variation in residual power. At one billion "square-voltage instructions per second" the potential difference in residual power is as large as 300-400 mW. It is possible to fit a regression line to this data to estimate a generic capacitive load per instruction over all our benchmarks. However, this solution would incur non-negligible modelling error because of the large variation in the dataset. In Figure 4.12b, we have grouped the samples (colored) after which benchmark group they belong to. If we study the groups carefully we can see that the instructions issued by specific benchmarks generally follow more specific trend lines, where the variation in residual power is much lower. This strengthens our hypothesis that capacitive loads can be attributed to specific processes in the system. It is important to note that, even if residual power can be negative for some benchmarks, this is assumed to be due to the model error e. Ultimately, it is the *slope* of the line that will decide the estimated contribution

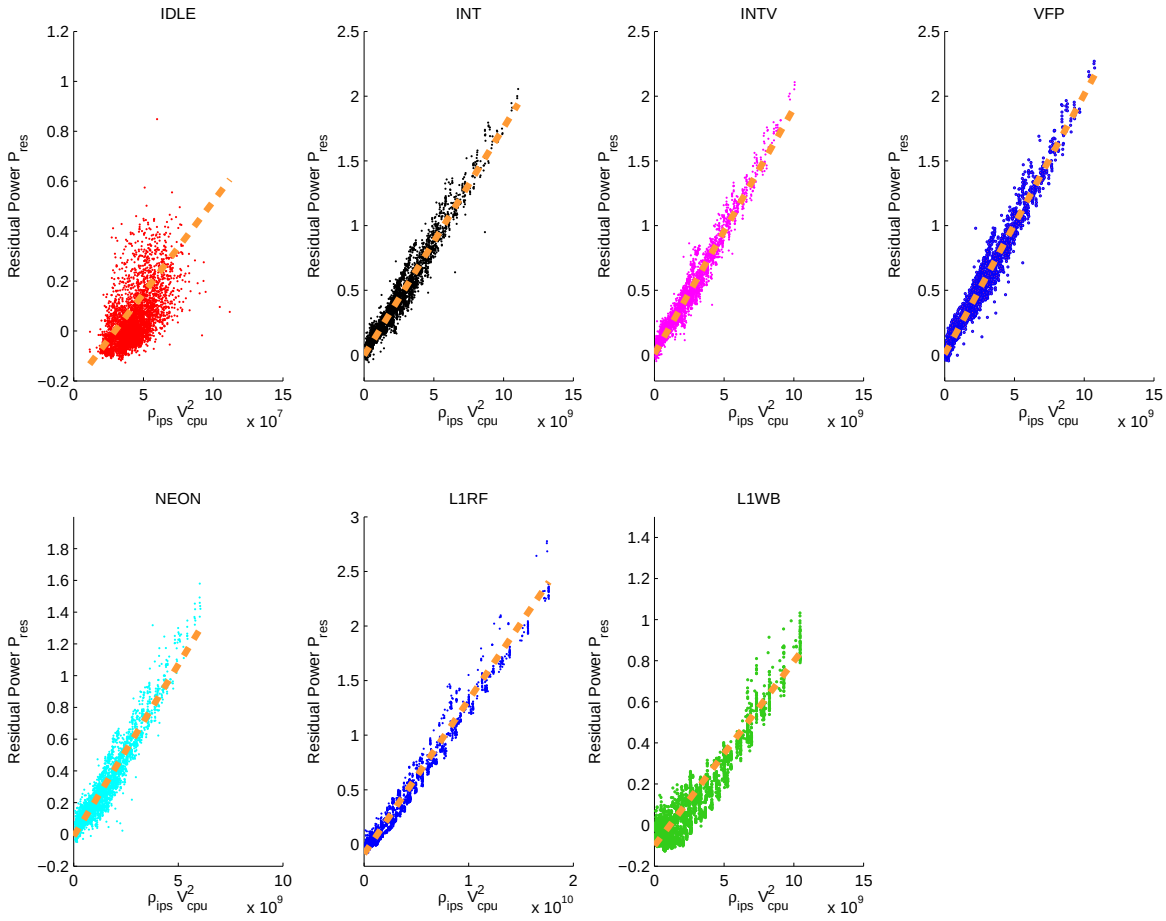


Figure 4.13: Per-benchmark residual power versus square-voltage instructions per second.

to dynamic power from that group of instructions.

Figure 4.13 shows the residuals for each of the seven CPU training benchmarks in closed views. A regression line is also drawn through the samples. The gradient of this line is the estimated capacitive load per instruction. Note that the gradients of the regression lines in this example do not represent the capacitive loads in Figure 4.14, but are provided for illustrative purposes. The actual modelled instruction costs are shown in Figure 4.14. We can see that there is a very good fit for most of the CPU training benchmarks. The residuals in these examples clearly show a *linear trend* towards the model predictor (V^2IPS), indicating that it is possible to get good estimates for instruction cost. The residuals for the IDLE benchmark, however, are significantly scattered and it is not easy to ascertain where the regression line should be. Despite this, the residual power increases with the square-voltage instructions per second and yields a positive capacitive load. It is difficult to conclusively argue why this occurs. One explanation can be that the modelling error is larger than the cost of IDLE instructions. The V^2IPS throughput is three orders of magnitude smaller than for the benchmarks that are actively processing, and consequently, the power usage of those instructions is also very low and hard to capture. Consequently, the residuals may "drown" in noise stemming from error. The idle instructions can also belong to any process in the system; therefore it may be possible to

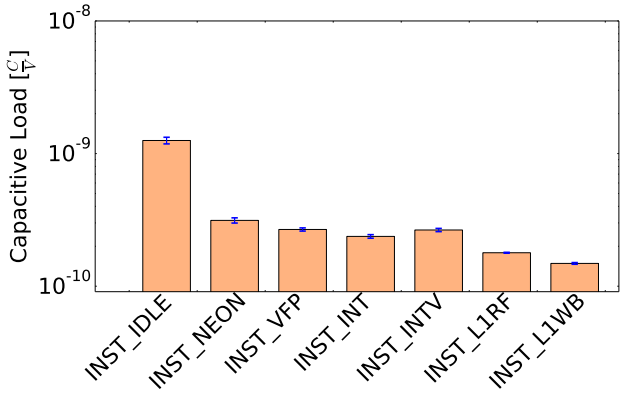


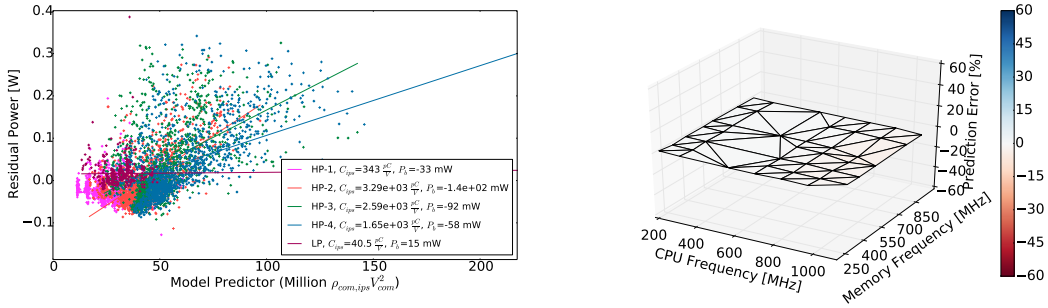
Figure 4.14: Workload-dependent instruction coefficients (CPU model).

get better estimations by studying the instruction throughputs of different system services, or per core. The per-benchmark instructions costs are shown in Figure 4.14, where we see that per-instruction cost varies significantly between applications. The NEON benchmark instructions, for example, cost twice as much as L1WB instructions, which have the overall lowest cost per instruction. The NEON instructions also cost more than VFP instructions. These benchmarks both calculate matrix products of single-precision floating-point values, but the NEON instructions process more data per instruction. From this perspective, NEON instructions appear more energy-efficient. However, the actual energy cost will also depend on how long time it takes to complete the instructions, how many CPU cycles are consumed in that time, and other factors.

4.6.3 Model Accuracy: Video Processing Filters

In this section, we study the instruction power and accuracy of our model for four distinct test cases. The Tegra K1 is either idle or actively processing one of our workloads, the DCT, MVS, debarreling and Huffman encoding filters. The methodology to estimate instruction cost per benchmark is similar to our discussion in the previous section. Each of the four bencmarks is run over all possible core configurations, memory and CPU frequencies. Power is estimated in set intervals of 100 ms. After each test, we remove the *generic* power components from the samples. The remaining *residual power* should reflect only the cost of CPU instructions (see Equation 4.10 and 4.11). These residuals are subsequently used to estimate the cost of instructions, and add this cost to the generic power predictors. The *measured* power is subtracted from the final *estimated* power to verify the model with real measurements.

The idle-system residuals are shown per core configuration in Figure 4.15a. As we observed in the previous section, we see that there is not any clear correlation between the rate of instruction execution and residual power. Although some of these residuals are larger than 100 mW, the majority are "clustered" in a region between 0 and 100 mW at 20 to 60 V^2IPS. As debated in the preceding paragraphs, this is assumed to be an effect of noise (error) where the regression error in Figure 4.10a generally lies in the same region.



(a) Capacitive instruction load.

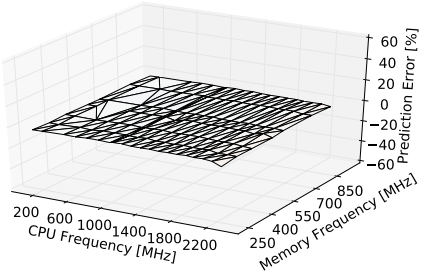
(b) Model error (LP core).

Figure 4.15: Idle system.

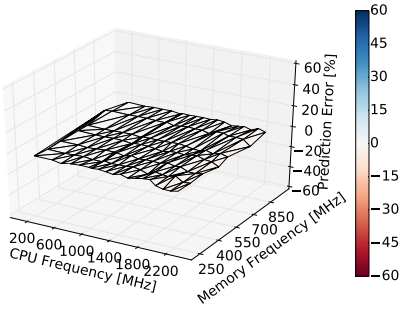
If we draw regression lines through each of the CPU core configurations (in Figure 4.15a) we can see that these yield very different results. The resulting base (P_b in Equation 4.11) is negative for all the four HP core configurations. The capacitive load per idle instruction on the HP cluster also exhibits large variations between 343.00$\frac{pC}{V}$ and 3.29$\frac{nC}{V}$, depending on how many cores are active. If we take the average of these, the cost per instruction on the HP cluster is 2.05$\frac{nC}{V}$. At a rate of 50M V^2IPS, this corresponds to an instruction power of 100 mW. The idle instruction cost on the LP core is only 40.50$\frac{pC}{V}$, corresponding to an instruction power of only 2 mW. We investigate the Tegra K1's idle system power further in Section 5.3.

We now use the estimated capacitive loss per instruction and base power (see Figure 4.15a) to estimate total power (generic and workload-dependent, see Equation 4.10) over all idle system scenarios. This includes all memory and CPU frequency combinations and core configurations. The result can be seen in Figure 4.15b for the LP core and Figure 4.16 for the HP cluster. For all five core configurations, the absolute average model error is below 0.34 % and the standard deviation is below 2.70 %. These are good initial results that show that the model captures idle power accurately. For the LP core and one HP core, the worst case (maximum observed) error is less than 3.72 %. However, for two, three and four HP cores, the worst-case error is much larger with underpredictions down to 9.43 %.

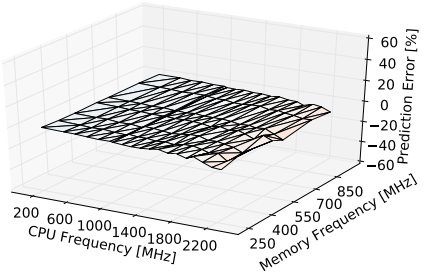
Studying the error plots in Figure 4.16b, 4.16c and 4.16d, we can make two important observations. First, the modelling error worsens at higher CPU frequencies (slightly red area above 1.6 GHz CPU frequency). Second, the model accuracy worsens when more cores are active. This is not immediately distinguishable from Figure 4.16, but the standard deviation increases from one to four cores at 0.82, 1.42, 2.67 and 2.70 %. During the development of the CPU model, we have worked intensively to increase the accuracy of the idle test cases. We discovered that the HP rail voltage in an *idle* system exhibits high variation which is challenging to capture in static frequency-voltage tables in our code. Figure 4.17 shows the difference in measured HP rail voltage depending on how many HP cores are active. When the Tegra K1 is *actively processing*, that is, continuously processing for example a video filter, the voltage is the highest observed. At 2.3 GHz, for example, it is above 1100 mV. But as the Tegra K1 idles, and the number of HP cores decreases, the voltage also decreases. With only one HP core active, the HP rail voltage is as low as 870 mV. The rail voltages shown in Figure 4.17 are based on



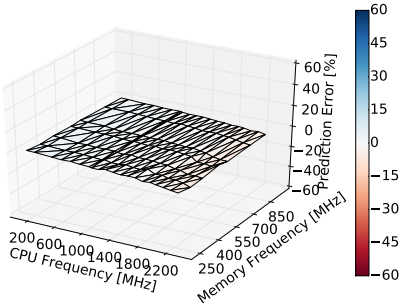
(a) Model error (one HP core).



(b) Model error (two HP cores).



(c) Model error (three HP cores).



(d) Model error (four HP cores).

Figure 4.16: Idle system model error (HP cluster).

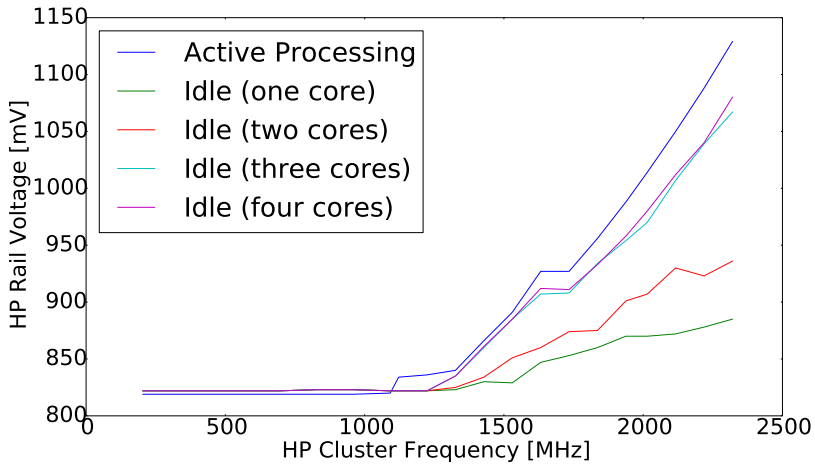
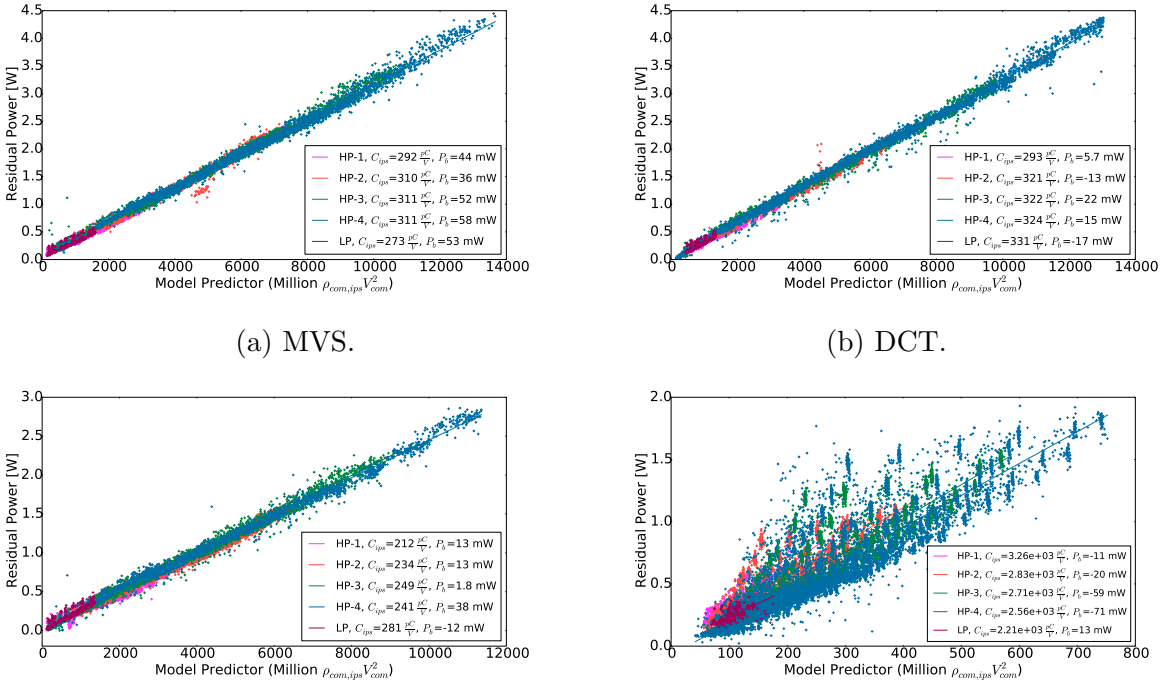


Figure 4.17: Problems measuring HP rail voltage in an idle system.



(a) MVS. (b) DCT.

(c) Huffman. (d) Debarreling.

Figure 4.18: Instruction (residual) power plotted versus V^2IPS for our test filters.

real measurements taken after 10 s of idling. The measurements were taken using the Keithley 2110 which averaged one second of voltage readings per sample. In general, the voltage appears to be more stable across cores when the Tegra K1 is actively processing. Initially, we only used the voltage curve for an active system which caused large idle-system prediction errors above 1.4 GHz. Incidentally, this is also the point where the rail voltages start to vary depending on how many HP cores are on. The plots shown in Figure 4.16 were validated with specific voltage tables that match the characteristics of Figure 4.17. However, we have not compensated for this in the model training data. We believe better estimations can be achieved by compensating more accurately for changes in rail voltage. Ideally, platform voltages should be measured during model training to achieve the best estimations possible.

We now focus our discussion on the model accuracy of our video processing filters (MVS, DCT, Huffman encoding and debarreling). Figure 4.18 shows the residual power for all of the filters plotted versus the V^2IPS for that benchmark. It is important to note that V^2IPS is a measure of the number of instructions per second multiplied with the square CPU rail voltage. We can see that instruction power can be substantial up to 4.3 W for the MVS and DCT filters in Figures 4.18a and 4.18b. The Huffman and debarreling filters in Figures 4.18c and 4.18d exhibit lower instruction powers of 2.7 and 1.7 W. The V^2IPS can be as high as 14000M V^2IPS, with the exception of the debarreling filter at only 700M V^2IPS. The low instruction throughput of the debarreling filter can be attributed to stalling memory operations that continuously shift pixels at subsequent HD frames. For the MVS, DCT and Huffman filters, we can observe that the residuals follow the same trend line. This is a promising result, as we can expect this to occur

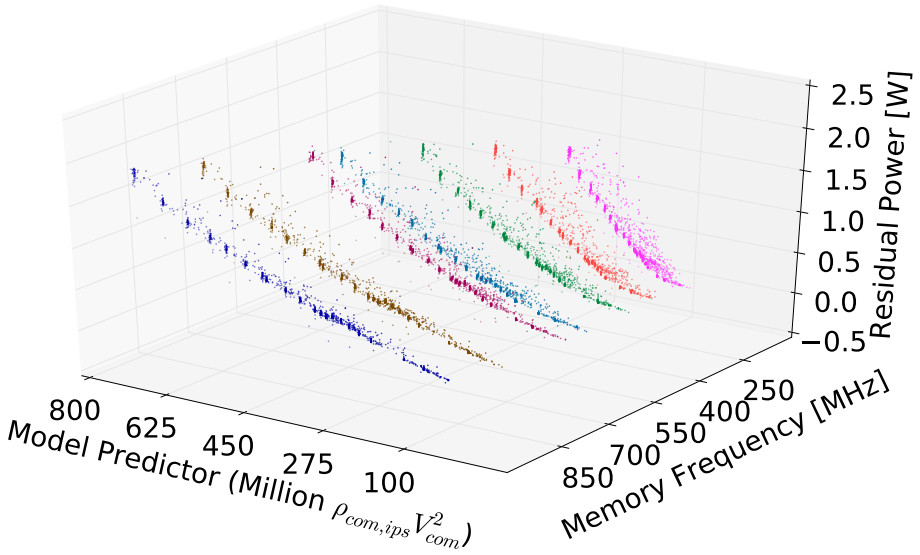
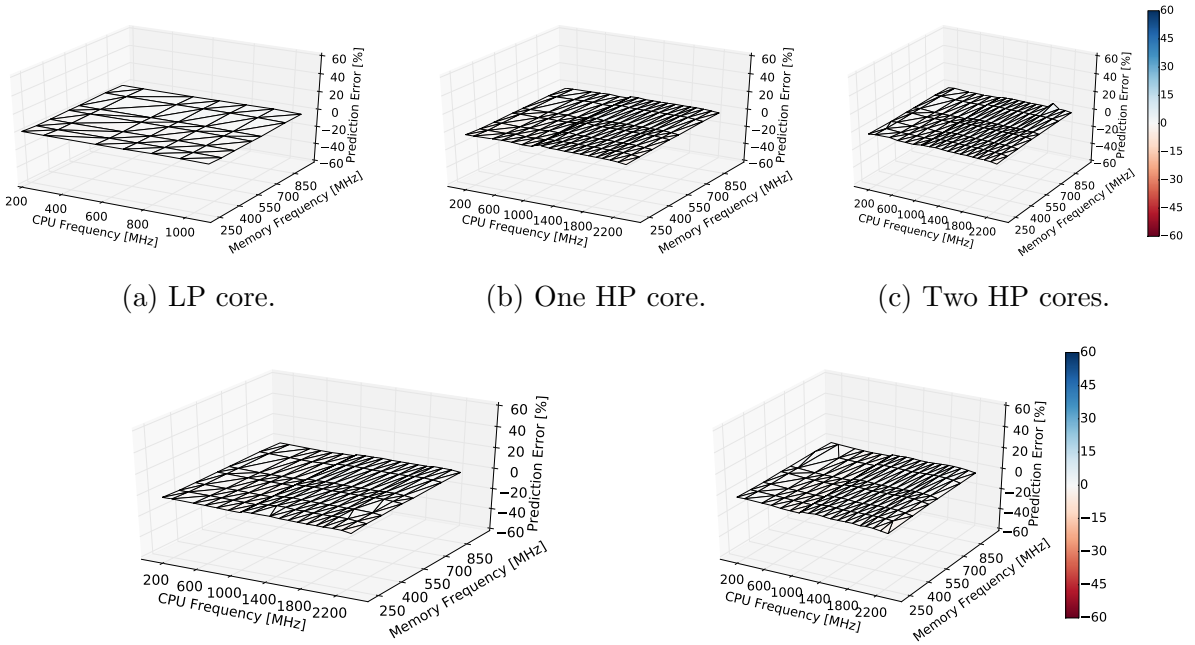


Figure 4.19: Residual power for the debarreling filter, plotted for V^2IPS and memory frequencies.

when all the CPU cores are of the same type and are running the same workload. The rate of instruction issue (and V^2IPS) increases as more cores are added, but the cost per instruction remains similar. The capacitive load per instruction for the MVS and DCT filters is generally between 270 and 325$\frac{pC}{V}$, respectively. For the Huffman filter, the capacitive load per instruction is between 212 and 281$\frac{pC}{V}$. Additionally, the regression lines' intercept (P_b in Equation 4.11) is very close to zero.

The capacitive load per cycle for the debarreling filter (see Figure 4.18d) appears to be more unstable than for the other filters. For any given core configuration, we can see that the residuals do not follow one *specific* trend line. Instead, there seems to be many trend lines where it is possible to fit individual regression lines. For example, for the four-core HP cluster variant (blue residuals in Figure 4.18d) there appears to be between five or six regression lines that individually yield better fits. Incidentally, this number is very close to the number of memory frequencies used in the experiment (seven). Additionally, as we suspect this filter to be the most memory-intensive benchmark, it is possible that the observed capacitive load per instruction varies depending on the memory frequency. Figure 4.19 shows the residual power for the debarreling filter in a three-dimensional plane, where the x- and y-axes correspond to the V^2IPS and memory frequency used. Note that the residual color represents the memory frequency used, and not the core configuration (as in Figure 4.18). This also means that the residuals for each memory frequency includes data from all five core configurations. From the figure, we can see that the capacitive load per instruction for this filter must be further classified depending on the memory operating frequency. The cost per instruction is highest at the lowest memory frequency (5.74$\frac{nC}{V}$ at 204 MHz). As memory frequency increases, the instruction cost for this filter decreases to 2.41$\frac{nC}{V}$ at 924 MHz memory frequency. This cost is very large compared to the relatively low capacitive loads per instruction of the other filters, at



(a) LP core. (b) One HP core. (c) Two HP cores.

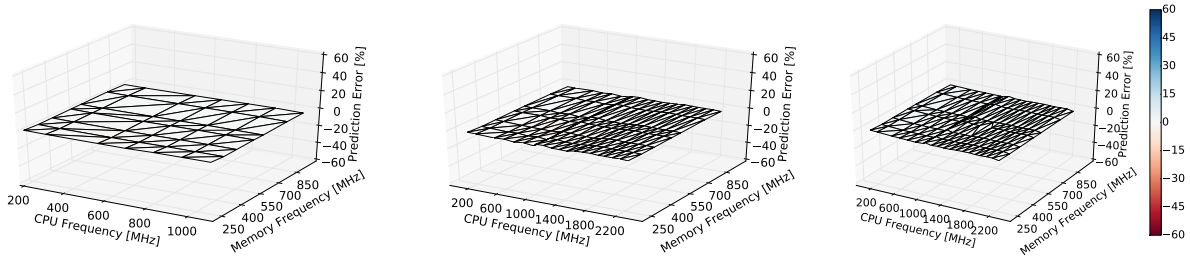
(d) Three HP cores. (e) Four HP cores.

Figure 4.20: Model error (MVS).

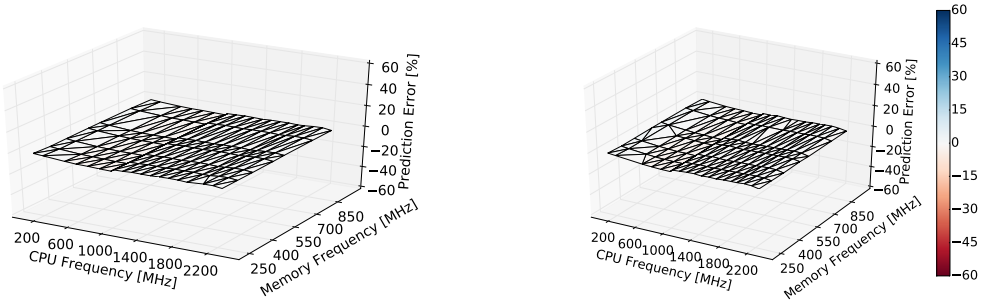
around $300\frac{pC}{V}$. However, V^2IPS is also very low due to the slow rate of instruction issue, which means that each instruction spends more time in the processor pipeline. This is likely due to memory access stalls and causes more switching activity to be attributed to each instruction. Another point, which is harder to distinguish from Figure 4.19, is the regression lines' intercept point P_b. For the 204 MHz memory frequency, this is -410 mW, which is an order of magnitude smaller than the intercept point found in the other filters. Ultimately, this means that the generic power model is *overpredicting* when the memory frequency is low for this memory-intensive benchmark. As memory frequency increases, however, the intercept increases and at 924 MHz it is only -144 mW.

With decent estimates for the capacitive instruction loads for all the video processing filters, it is now possible to add the workload-dependent power (see Equation 4.10) to compare the accuracy of the model with real measurements. The error percentage for all of the five core configurations over all four filters comprise the 20 error plots shown in Figure 4.20, 4.21, 4.22 and 4.23. The average absolute error is rarely above 1 % (1.18 and 1.08 % for two-core DCT and two-core Huffman). Additionally, worst case error is generally below 3 %. The exception is the debarreling filter, which has a worst case error of -4.57 % on three cores. This is better than for the idle system model, where we observed worst-case modelling errors of up to almost 10 %. This is because the frequency-voltage tables in our power estimation framework are more accurate when the Tegra K1's CPU is actively processing a filter (see Figure 4.17). The standard deviation is always below 3 %.

In this section, we have evaluated our CPU model for the Tegra K1. As for the GPU, this model was built using spesialised, synthetic benchmarks that stress various architectural units of the SoC. We have seen that our model successfully captures the

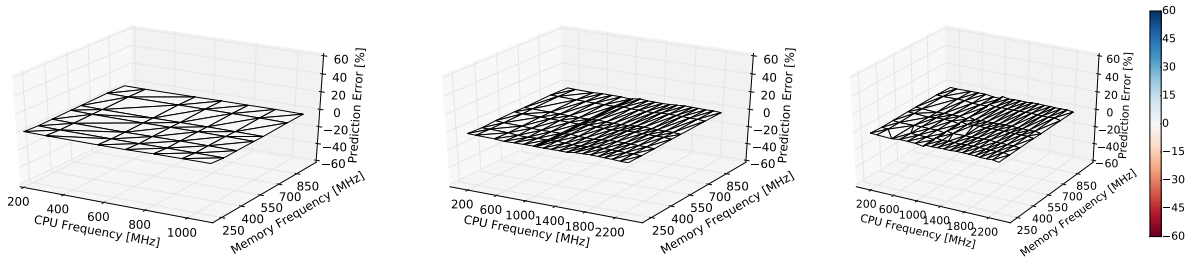


(a) LP core. (b) One HP core. (c) Two HP cores.

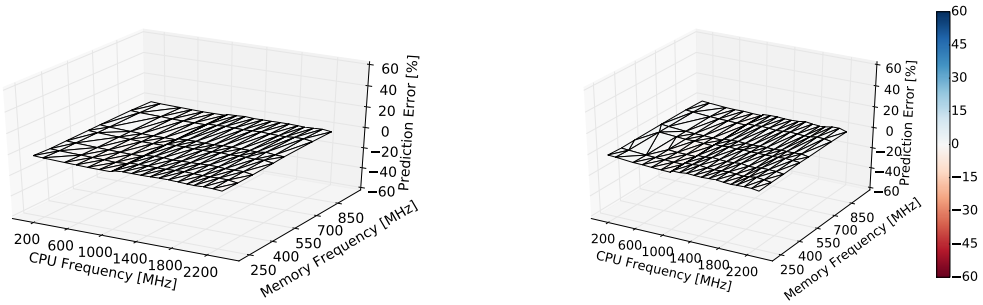


(d) Three HP cores. (e) Four HP cores.

Figure 4.21: Model error (DCT).

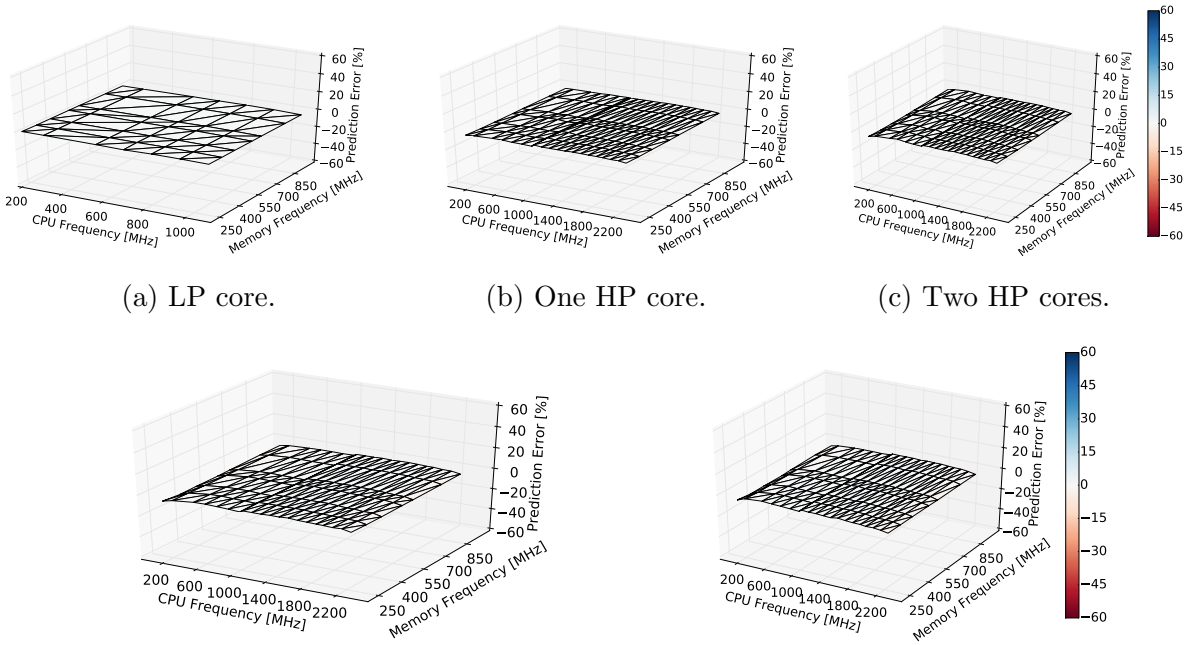


(a) LP core. (b) One HP core. (c) Two HP cores.



(d) Three HP cores. (e) Four HP cores.

Figure 4.22: Model error (Huffman).



(a) LP core. (b) One HP core. (c) Two HP cores.

(d) Three HP cores. (e) Four HP cores.

Figure 4.23: Model error (debarreling).

relation between power usage and our dynamic and static model predictors. However, we only had to consider stressing different levels of the CPU's cache hierarchies, and not specific instruction types. With respect to modelling, a challenge with the Tegra K1's CPU is that it does not have fine-grained instruction accounting, allowing us to measure the type and number of executed instructions. In this aspect, we are the first to develop a method to model instruction power based on individual applications. The method is based on removing the generic (workload-independent) power components from power estimation samples, and subsequently use regression to estimate the average capacitive load per instruction per benchmark. We have seen that we are able to observe a linear trend between the residual (instruction) power and the model predictor, which is the number of square-voltage instructions per second, V^2IPS. However, our debarreling filter showed that the average capacitive load may also depend on the memory frequency for memory-intensive benchmarks. Finally, investigating our model's prediction accuracy, we have demonstrated that our method is able to estimate power with 99 % accuracy for our video processing filters. The worst case error for most of the filters is below 3 %, but for the debarreling filter it is 5 %. The prediction error when the Tegra K1 is idle is below 0.34 % on average. However, the worst case error can be as large as almost 10 %. In general, the larger errors are found at higher CPU frequencies. This is because the idle-system CPU rail voltage exhibits considerable variation at higher frequencies, and therefore, our static frequency-voltage tables are not sufficient to capture the HP rail voltage accurately. A future topic for research is therefore whether model accuracy can be improved by measuring rail voltages throughout the model training and verification phases. In conclusion, these results show that we have successfully captured power usage for the video processing filters with very high precision. As a result, the accuracy is much

better than the traditional modelling methods (CMOS-, state- and rate-based models in Figure 3.3).

4.7 Full-Hybrid Models for Different Platforms

In the previous sections, we have verified and built power models for the Tegra K1's individual compute units (CPU and GPU). We have shown that our methodology to model power is successfull, with very close to 100 % accuracy over many different software workloads, over all CPU, GPU and memory frequencies and CPU core configurations. These models were built and verified for a Jetson-TK1 development kit called **Rey**. However, the general applicability of our method has not been demonstrated on several Tegra K1-enabled systems. Additionally, the GPU model presented in Section 4.5 did not include the more developed CPU model in Section 4.6. In this section, we therefore build the CPU and GPU model simultaneously on Rey and on two additional platforms: another Jetson-TK1 development kit called **Ren**, and the **T-17**, which is a customised development platform containing only the Tegra K1 SoC. We constrain ourselves to build models for these devices, and do not verify them using software workloads (to the extent presented in Section 4.5 and 4.6). The full-hybrid models for Rey and Ren, however, are used and verified in Chapter 5 as the "final" models built for these systems.

An introduction to the T-17 platform is necessary before we proceed. The T-17 is a heavily simplified version of the Jetson-TK1 development kit presented in Figure 2.1. It contains only the Tegra K1 SoC and the regulators powering the HP, Core, GPU, memory and USB (clock) rails. With the exception of USB, there are no external peripherals. Instead of DDR3 RAM operating at 1.35 V, it is equipped with LPDDR3 RAM operating at 1.2 V. The capacitances between the main buck regulators and the Tegra K1 SoC are much smaller than on the Jetson-TK1 kits. These capacitances form a low-pass filter into the SoC, effectively "smoothing out" the rail voltage. Because the "rail capacitances" are much smaller than on the Jetson-TK1 kits, the rail voltages on the T-17 exhibit considerable variation (noise) and are very challenging to control using hard-coded frequency-voltage tables. The platform is not able to sustain the current throughput at maximum frequencies. For this reason the maximum CPU (HP) frequency used is 2.0 GHz, while for the Jetson-TK1, it is 2.3 GHz.

The T-17 was only available to us for a brief period of time. Unfortunately, the problems with PERF and our kernel tracing framework to measure clock gating in the CPU cores (see Section 4.4) is present in this model's training data. Due to these issues, along with the difficulty of measuring rail voltages, we had to adjust the model predictors as follows:

- The NEON CPU training benchmark was removed, because the instruction cost was estimated to be negative.
- The cache is only modelled based on the number of L1 refills, whereas on the Jetson-TK1 kits, it is modelled as a function of both L1 and L2 refills.
- The HP rail leakage is assumed to be the leakage of one core. We were unable to model the HP rail leakage independently.

Rail	Predictor	Description	Coefficient	Ren	Rey	T-17
HP	V_{hp}	HP rail voltage (when powered)	$I_{hp,leak}$	53.39 mA	106.29 mA	27.86 mA
	$\rho_{hp,clk1}$	Active cycles (one core)	$C_{hp,clk1}$	$411.70\frac{pC}{V}$	$402.63\frac{pC}{V}$	$430.62\frac{pC}{V}$
	$\rho_{hp,clk2}$	Active cycles (two cores)	$C_{hp,clk2}$	$279.75\frac{pC}{V}$	$251.07\frac{pC}{V}$	$437.66\frac{pC}{V}$
	$\rho_{hp,clk3}$	Active cycles (three cores)	$C_{hp,clk3}$	$246.01\frac{pC}{V}$	$221.78\frac{pC}{V}$	$333.90\frac{pC}{V}$
	$\rho_{hp,clk4}$	Active cycles (four cores)	$C_{hp,clk4}$	$248.97\frac{pC}{V}$	$238.39\frac{pC}{V}$	$353.49\frac{pC}{V}$
Core	V_{core}	Core rail voltage + core clocks †	$I_{core,acc}$	342.56 mA	379.31 mA	499.31 mA
	$\rho_{core,clk}$	Active cycles (LP core)	$C_{core,clk}$	$242.76\frac{pC}{V}$	$202.91\frac{pC}{V}$	$311.55\frac{pC}{V}$
Common	$V_{com,online}$	CPU Core leakage	$I_{cpu,leak}$	15.63 mA	35.78 mA	27.86 mA
	$\rho_{com,l1l2}$	L1 data / instruction cache refills	$C_{com,l1rf}$	$16.11\frac{nC}{V}$	$27.11\frac{nC}{V}$	$8.27\frac{nC}{V}$
	$\rho_{com,l2m}$	L2 data cache refills	$C_{com,l2rf}$	$6.77\frac{nC}{V}$	$27.06\frac{nC}{V}$	
	$\rho_{com,ips}$	Instructions (workload-specific)	$C_{com,l1tlb}$	See Figure 4.25c		
GPU	V_{gpu}	GPU voltage	$I_{gpu,leak}$	156.41 mA	158.95 mA	24.93 mA
	$\rho_{gpu,clock}$	Total clock cycles per second	$C_{gpu,clock}$	$2.22\frac{nC}{V}$	$2.12\frac{nC}{V}$	$2.14\frac{nC}{V}$
	$\rho_{gpu,L2R}$	L2 cache 32B reads per second	$C_{gpu,L2R}$	$9.54\frac{nC}{V}$	$8.73\frac{nC}{V}$	$10.59\frac{nC}{V}$
	$\rho_{gpu,L1R}$	L1 cache 4B reads per second	$C_{gpu,L1R}$	$7.14\frac{nC}{V}$	$5.27\frac{nC}{V}$	$9.03\frac{nC}{V}$
	$\rho_{gpu,L1W}$	L1 cache 4B writes per second	$C_{gpu,L1W}$	$8.76\frac{nC}{V}$	$8.01\frac{nC}{V}$	$10.70\frac{nC}{V}$
	$\rho_{gpu,INT}$	Integer instructions per second	$C_{gpu,INT}$	$38.89\frac{pC}{V}$	$36.32\frac{pC}{V}$	$43.11\frac{pC}{V}$
	$\rho_{gpu,F32}$	Float (32-bit) instructions per second	$C_{gpu,F32}$	$37.67\frac{pC}{V}$	$11.29\frac{pC}{V}$	$39.05\frac{pC}{V}$
	$\rho_{gpu,F64}$	Float (64-bit) instructions per second	$C_{gpu,F64}$	$104.54\frac{pC}{V}$	$73.77\frac{pC}{V}$	$112.41\frac{pC}{V}$
	$\rho_{gpu,CNV}$	Conversion instructions per second	$C_{gpu,CNV}$	$33.17\frac{pC}{V}$	$509.19\frac{pC}{V}$	$13.52\frac{pC}{V}$
	$\rho_{gpu,MSC}$	Miscellaneous instructions per second	$C_{gpu,MSC}$	$31.16\frac{pC}{V}$	$28.70\frac{pC}{V}$	$84.97\frac{pC}{V}$
Memory	$\rho_{mem,clk}$	Clock cycles	$C_{mem,clk}$	$233.57\frac{pC}{V}$	$227.04\frac{pC}{V}$	$184.94\frac{pC}{V}$
	$\rho_{mem,CPU}$	CPU memory cycles	$C_{mem,cpu}$	$3.23\frac{nC}{V}$	$2.82\frac{nC}{V}$	$7.15\frac{nC}{V}$
	$\rho_{mem,CPU}$	Other memory cycles	$C_{mem,cpu}$	$1.98\frac{nC}{V}$	$1.80\frac{nC}{V}$	$2.02\frac{nC}{V}$
	$\beta_{mem,204}$	Power offset at 204 MHz	$P_{mem,204}$	-2.25 mW	-2.74 mW	N/A
	$\beta_{mem,300}$	Power offset at 300 MHz	$P_{mem,300}$	59.06 mW	55.68 mW	
Other	P_{base}	Base power	1	1211.94 mW	1173.69 mW	67.18 mW

†Leakage current modelled with core clocks.

Table 4.5: Full-hybrid models for three different Tegra K1 platforms.

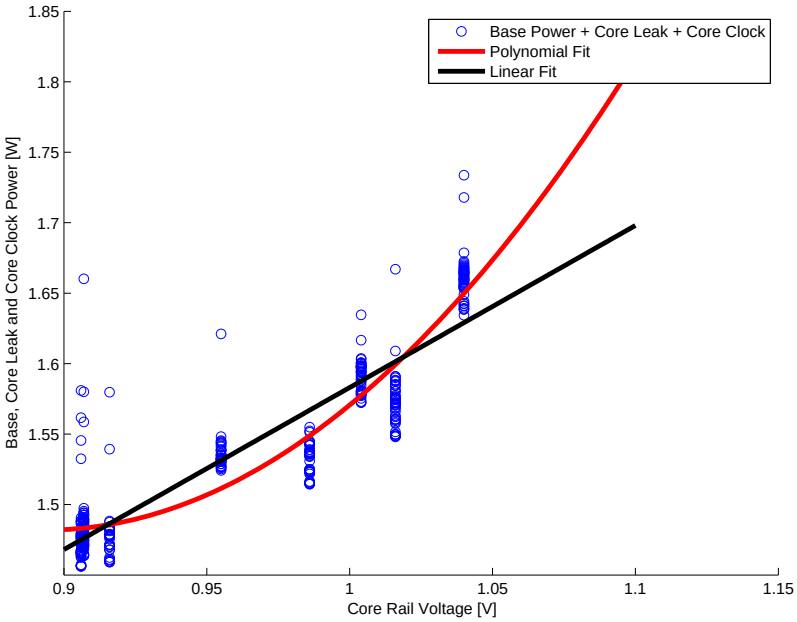


Figure 4.24: Base, core leak and core clock power residuals plotted over core rail voltage.

All these issues, however, are fixed in the full-hybrid models for Rey and Ren.

The method used to build these models is a combination of the methods used to build the CPU and GPU models in Sections 4.5 and 4.6. The main difference is that the CPU model has been improved. The intention is that this will help to build the GPU model more accurately. However, during our initial development of the full-hybrid model on the T-17, we had problems estimating a positive base power (P_{base} in Table 4.5). The T-17's base power was always estimated to be around -300 mW, even when all the other coefficients were positive and similar to those estimated on Rey and Ren. We believe that the reason behind this is that core rail leakage is over-estimated, causing the base power to be underestimated. This occurs because we have forgotten to model some of the core clocks (`sbus`, `mselect`, `pciex` and `host1x`). Consider Figure 4.24, where the residual power ($P_{res,core}$) stemming from base power, core leak and core clocks are plotted over core rail voltages on Ren. All other power components have been removed (based on a model for Ren). Under these (incorrect) assumptions the residual power is described as:

$$P_{res,core} = P_{base} + V_{core} I_{core,leak} \tag{4.12}$$

According to Equation 4.12, the residuals in Figure 4.24 should show a linear trend with core rail voltage. However, it can be argued that this is not the case. The second-degree polynomial curve-fit in Figure 4.24 shows a better fit with the residuals. This may be hard to realise from the figure, where the other power components have been removed based on this (faulty) model. However, we provide this figure for demonstration purposes. The fact that the polynomial fit is a better approximation to these residuals is supported by the fact that we have not modelled the core clocks on the core rail (see Table 2.1). In reality, the *accessory* power (residuals in Figure 4.24) should rather be described as a sum of base power, core leakage and core clocks $\mathbb{CLK}$:

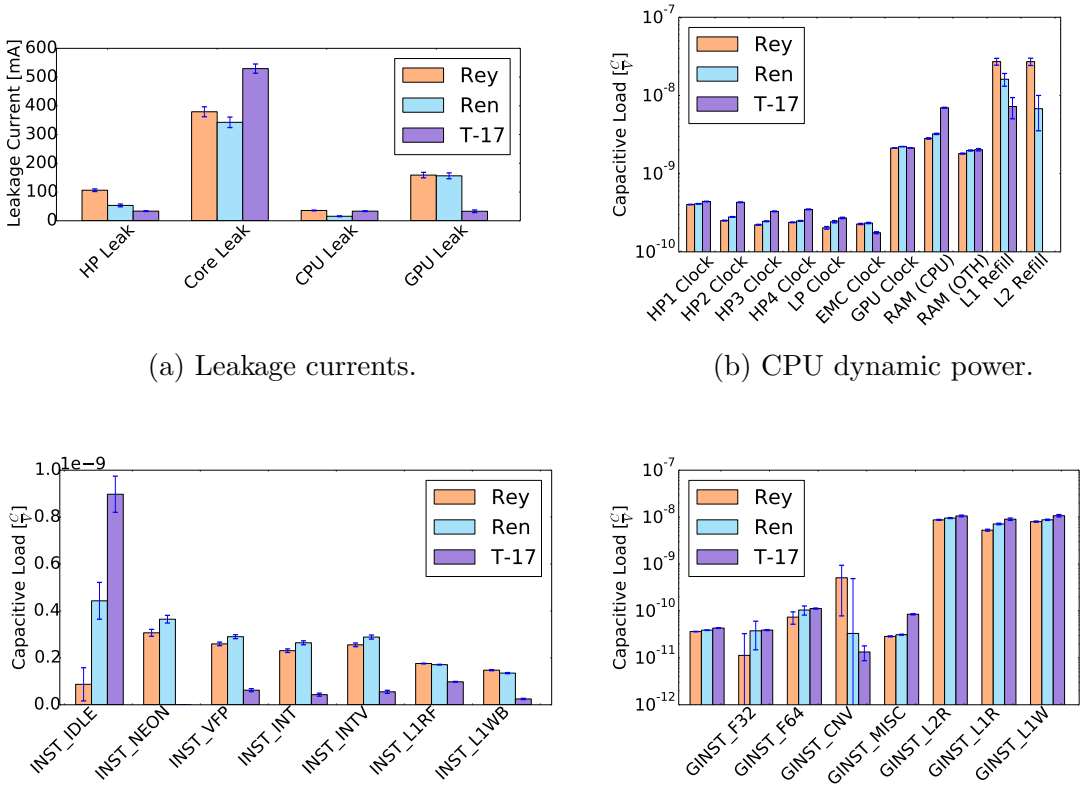
$$P_{core,acc} = P_{base} + P_{core,stat} + \sum^{clk \in \mathbb{CLK}} C_{clk} f_{clk} V_{core}^2 \tag{4.13}$$

The core clock frequencies are set to fixed values and do not vary over execution time. However, as rail voltage increases, the dynamic power contribution from these clocks will increase. Consequently, the leakage current (slope of the linear fit) is overestimated. To overcome this issue, we model accessory power as a simple polynomial:

$$P_{core,acc} = P_{base} + V_{core}^2 I_{core,acc} \tag{4.14}$$

where $I_{core,acc}$ is a coefficient with units of ampere per volt. Note that we also attempted to include a first order variable ($V_{core} I_{core,leak}$). However, we removed this term because $I_{core,leak}$ was always estimated to be zero.

When the core clocks have been considered in the model, the resulting models for Ren, Rey and T-17 show more promising results (see Table 4.5). It should first be noted that, in general, the three models are similar. The models for Ren and Rey, which are Jetson-TK1 kits, are most similar to each other. These also have fixes for the issues with PERF and clock gating measurements. The T-17 does not have these fixes, and as a result, the model for this system is more different compared to the Jetson-TK1 development kits. The area where the models are *most* different is in terms of base power. The T-17 now has a positive base power of 67.18 mW, whereas the Jetson-TK1 development kits have



(a) Leakage currents.

(b) CPU dynamic power.

(c) Instruction coefficients.

(d) GPU dynamic power.

Figure 4.25: Full-hybrid model coefficients and standard deviations for three different Tegra K1 platforms.

more similar values of 1211.94 and 1173.69 mW. This is reasonable given that the T-17 does not have any external peripherals. The Jetson-TK1 kits implement a plethora of external periphery, such as USB host controllers, network interfaces, other PCI-e devices, serial interface, HDMI ports and more. The power usage of these units are interated into a constant power term, the base power, for the Jetson-TK1 kits. Consequently, the estimated base power for the Jetson-TK1 kits is higher.

A graphical comparison of the model coefficients for the three Tegra K1-based systems are shown in Figure 4.25. The core rail leakages are shown in Figure 4.25a. Generally, the estimates are good with relatively low 95 % confidence intervals. We can see that the leakage currents for Ren and Rey agree, except for the HP rail leakage. Here, the leakage is estimated to be over twice as high on Rey as for Ren. The model differences are larger between the Jetson-TK1 kits and the T-17. For example, core leakage is estimated to be at about 540 mA on the T-17, but only some 370 mA on the Jetson-TK1 kits. Additionally, the GPU leakage is too low (only some 20 mA compared to 170 mA on the Jetson-TK1 kits). However, it is possible to manually model the GPU leakage on the T-17 according to our own method [63]. By only considering the increase in dynamic and static power as GPU frequency increases in an idle system, the leakage is much closer to those observed on the Jetson-TK1 kits. We believe that the large differences in leakage currents between the T-17 and the Jetson-TK1 kits can be caused by the problems with

PERF, measurement of CPU clock gating and rail voltages at high GPU frequencies.

Figure 4.25b shows the dynamic power coefficients on the CPU. We can see that the estimated coefficients between Rey and Ren are very close to each other. The only coefficients where there is a discrepancy is for the L1 and L2 cache refill costs. These are estimated to be much higher on Rey than on Ren. L2 refills, for example, are estimated to cost an order of magnitude more. However, it is worth noting that the 95 % confidence interval is very large on Ren for the L1 and L2 refill cost. It is possible that something has occurred during this experiment that caused this coefficient to be mis-predicted. For example, it is possible that the low cost of L2 refills on Ren can be attributed to instructions, where we in Figure 4.25c can observe that instruction cost is slightly higher on Ren than on Rey for the different CPU benchmarks. If we study the clock power, we can see that the T-17 has a higher clock cost than the Jetson-TK1 kits. Additionally, the cost per clock cycle gradually decreases as more CPU cores are added to the system (from $430\frac{pC}{V}$ at one HP core to $437\frac{pC}{V}$, $333\frac{pC}{V}$ and $353\frac{pC}{V}$ for each of the remaining cores, respectively). This is very different from the Jetson-TK1 kits, where the cycle cost on the first core is larger ($400\frac{pC}{V}$) than for the second, third and fourth CPU core, which have the same cost of around $250\frac{pC}{V}$. Finally, the T-17 and the Jetson-TK1 kits have different memory dynamic power costs. This is reasonable to expect given that the T-17 is equipped with LPDDR3 RAM, whereas the Jetson-TK1 kits have DDR3. We can see that the LPDDR3 cycle cost is lower at $194\frac{pC}{V}$ instead of $233\frac{pC}{V}$ and $227\frac{pC}{V}$ on Ren and Rey, respectively. Interestingly, the cost per active CPU memory cycle is higher on T-17 at $7.15\frac{nC}{V}$ instead of $2.23\frac{nC}{V}$ and $2.82\frac{pC}{V}$. The active memory cycle cost from other (GPU) hardware units is similar at around $2.0\frac{nC}{V}$.

Studying the CPU instruction costs in Figure 4.25c, we can make several interesting observations. First, the idle cost is much lower, and it can be comparable to the other instruction costs. In the CPU-only model, this cost was estimated to be much higher (around $1.0\frac{nC}{V}$ in Figure 4.14), but now it is only $0.08\frac{nC}{V}$ and $0.44\frac{nC}{V}$ per idle instruction on Rey and Ren, respectively. Furthermore, we see that the per-benchmark instruction costs remain very similar between Rey and Ren, leading us to the conclusion that the model performs well between these devices. The instruction costs for the T-17 are estimated to be much lower than on the Jetson-TK1 kits. We believe that this is caused by the problems measuring instruction throughput with PERF. Additionally, it is possible that the instruction power has been absorbed by the CPU cores' cycle cost in Figure 4.25b. These are estimated to be higher than on the Jetson-TK1 kits.

Finally, Figure 4.25d shows the estimated GPU instruction costs. This is the group of model coefficients that agree the most between Rey, Ren and the T-17 development kits. The integer, double-precision floating point and miscallaneous instructions, as well as the L1 and L2 cache utilisation costs, are almost in complete unison with very small 95 % confidence intervals. However, there are a few things to note. On Rey, the single-precision floating point instructions have a 95 % confidence interval which is very large, and indeed, goes below zero. This is not the case on Ren and the T-17. Furthermore, conversion instructions show very large 95 % confidence intervals across the three devices. This leads us to believe that for some reason, it is very hard to get a good estimate for this type of instruction. However, it is challenging to make an educated guess as to why this occurs.

In this section, we have built our full-hybrid models for three different platforms:

Two Jetson-TK1 development kits, Rey and Ren, as well as a special development kit containing only the Tegra K1 SoC, the T-17. We have seen that the model coefficients between Ren and Rey are similar. The coefficients for the T-17 are more different to the two due to implementation problems with PERF. Additionally, the rail voltages on the T-17 are hard to capture using static frequency-voltage tables in our code. However, the resulting model for the T-17 was good enough to discover a modelling error, where we in Sections 4.5 and 4.6 had forgotten to consider some of the clocks running on the core rail. This caused the estimated base power to be negative. By also considering these clocks in the model, the resulting base power on the T-17 was estimated to be 67.18 mW, which is reasonable given that the platform does not have any external peripherals around the SoC. The improved models on Rey and Ren show that the coefficients are similar between the platforms, but it is hard to state whether modelling is necessary on every SoC. Dynamic power components are for example slightly higher on Ren than on Rey, but static power components are smaller. Considering the coefficients in Table 4.5, we would argue that only one model is necessary for Rey and Ren. However, this may not be true for other Tegra K1 SoCs. The PMIC, for example, and its DCDC and LDO regulators powering the power rails, will waste power differently depending on the circuit layout and hardware components. Additionally, one cannot neglect the possibility of manufacturing differences of the SoC.

4.8 Summary

There is a lack of power modelling methodologies that can successfully capture the complex relations between software activity and the energy consumption of processors, memory and other hardware units. State-, rate- and CMOS-based methods all have their own strengts and weaknesses, but none of them are able to accurately model power usage of the Tegra K1's CPU and GPU over different frequency ranges. In this chapter, we have introduced a high-precision power modelling method that is able to predict power usage of several multimedia workloads running on the Tegra K1 with "worst-case" errors of only 3 %, with an average accuracy very close to 100 %. Our method achieves higher precision than state-of-the-art approaches by combining their different insights to model both static and dynamic power of the system. By expressing dynamic and static power in terms of measureable hardware activity, such as instructions for workloads (CPU) and architectural units (GPU), L1 and L2 cache usage, clock-, power- and rail-gating, and subsequently combining these metrics with measured rail voltages, our method achieves fine-grained insight into a closed system. We have also built our full-hybrid model on both Rey and Ren (Jetson-TK1 development kits) and a customised platform, the T-17, which is a minimal Tegra K1 implementation with very few hardware peripherals. Through these experiments we have shown that our model is applicable to different devices, where the estimated model coefficients are comparable. This also made us aware that we had forgotten to model the constant system clocks on the core rail. We believe that the best way to improve the current model is to also measure the rail voltages of the platform, instead of resorting to hard-coded frequency-voltage tables that are, especially in the case of an idle system, inaccurate. Ending our discussion on high-precision power modelling, we now finally focus on *using* these models to achieve insight into the power usage of running workloads.

Chapter 5

Energy-Efficient Multimedia Processing

In the previous chapters, we have motivated, developed and evaluated our high-precision power modelling method for the Tegra K1. We have shown that our model is able to capture a complete picture of the energy consumption of the platform, broken down into processor elements, instructions, cache and off-chip memory usage. In this chapter, we show through several preliminary case studies the insight that our model can give into the energy consumption of a complex, mobile SoC such as the Tegra K1. Through the use of our high-precision power model, we show how energy is consumed on the Tegra K1 under idle and active processing scenarios, how the Tegra K1 is energy-inefficient, and how heterogeneous cores can be used to conserve energy. These experiments should be considered to be preliminary studies that show the type of insight our power model can provide. We leave it to future work to continue investigating these scenarios.

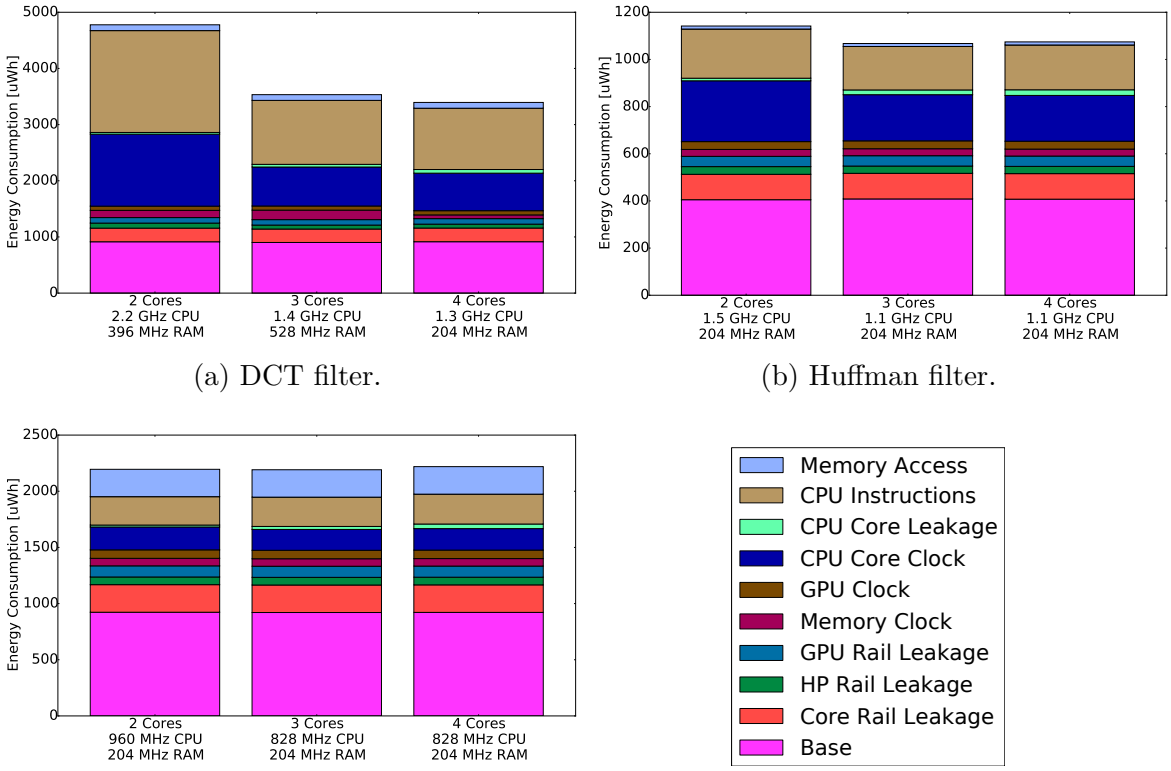
The remainder of this section is organised as follows. In Section 5.1, we focus on our video processing filters from Section 2.5 and their per-frame energy usage. Using our power model, we investigate the reasons behind energy-inefficiency at higher platform frequencies. We find that the increased rail voltage at high operating frequencies, as well as the number of processor cycles to complete each instruction, are important factors for energy-efficiency. In Section 5.2, we exploit the fact that the Tegra K1's processors are inefficient at higher operating frequencies, and show that workloads can be split between the Tegra K1's heterogeneous compute cores to save 5 % energy under our DCT filter. In Section 5.3, we investigate idle system power breakdown. We show through our model the main power consuming entities of the Tegra K1 SoC, such as the core rail leakage and memory clock. We demonstrate how kernel modules can be analysed and optimised for better energy-efficiency. The effects of different GPU and CPU instruction types and cache access modifiers, in terms of performance and energy consumption, is investigated in Section 5.4. We demonstrate 3.2 % energy saving by exploiting different GPU instructions and cache hierarchies, and 50 % saving by using NEON CPU instructions, for the DCT filter.

5.1 Processing Live Video

In Section 2.5, we studied the effects that DVFS has on continuous video filter processing. We discovered that a good heuristic to save energy is to minimise processor and memory frequencies such that the QoS requirements, which in our case is a framerate, is met. This heuristic consistently outperforms the standard DVFS, cluster and core algorithms operating on the Tegra K1 by between 7 to 40 %. However, it is hard to argue why this occurs, because there is no way to measure the individual power draw of the Tegra K1's compute components. In this section, we investigate the sources of energy-inefficiency by using our model (Section 4.7) to gain insight into-per component energy consumption. We focus our study on the DCT, MVS and Huffman filters, which are common components of a video processing chain. We first study how energy consumption varies depending on how many CPU cores are active, where we let the Tegra K1 process these filters at two, three and four CPU cores. Here, we only consider the processor and memory frequency combinations that reach the lowest EPF. We do not evaluate one core, because most of the filters cannot reach 25 FPS in this core configuration. Subsequently, we evaluate how frequency scaling affects energy consumption of the the filters. Only three processor and memory frequency combinations that reach the 25 FPS requirement for each filter are evaluated, where we first consider the maximum processor and memory frequencies, before we reduce the memory and processor frequencies to the lowest that reach 25 FPS, respectively. Therefore, a possible extension to these results is to consider all the power components of our model over all possible processor and memory frequencies. This may uncover additional sources of energy-inefficiency in the Tegra K1 SoC.

Figure 5.1 shows the energy breakdown for the DCT (Figure 5.1a), Huffman (Figure 5.1b) and MVS (Figure 5.1c) filters processing on two, three and four CPU cores. The energy displayed is the total consumption processing 70 frames. The breakdowns represent the tests (processor and memory) frequencies that reached the lowest total energy consumption. The figures therefore show the effects of adding and removing cores. If we study the DCT filter in Figure 5.1a, we see that there is a large difference in processing between two and three cores. The total energy consumption is reduced from $4750\mu Wh$ to $3530\mu Wh$. The breakdown shows that the large reduction in energy consumption is due to reductions in CPU clock and instruction costs. This is due to reduced CPU voltage: at two HP cores, the CPU must operate at 2.2 GHz (1.08 V), whereas on three cores, it can be lowered to 1.4 GHz (0.88 V). Otherwise, most of the energy components are similar or the change is negligible. For example, adding a CPU core adds some leakage current on the HP rail, and the memory frequency is increased to 528 MHz, effectively increasing the memory clock energy slightly. However, the extra consumption is much lower than the savings from the CPU clock and instruction energy. Finally, adding the fourth HP core to process the DCT filter further increases the energy-efficiency. There is some (negligible) increase in HP rail leakage, but adding the fourth core enables us to drastically reduce the memory frequency to only 204 MHz. The CPU frequency is also reduced to 1.3 GHz (from 1.4 GHz at three cores), but this has little effect on CPU cycle- and instruction-energy.

Similar observations can be made for the Huffman and MVS filters in Figures 5.1b and 5.1c. Note that the CPU instruction power for the DCT filter is significant at some 1000 mWh compared to the relatively low instruction costs of between 200-250 mWh for



(a) DCT filter.

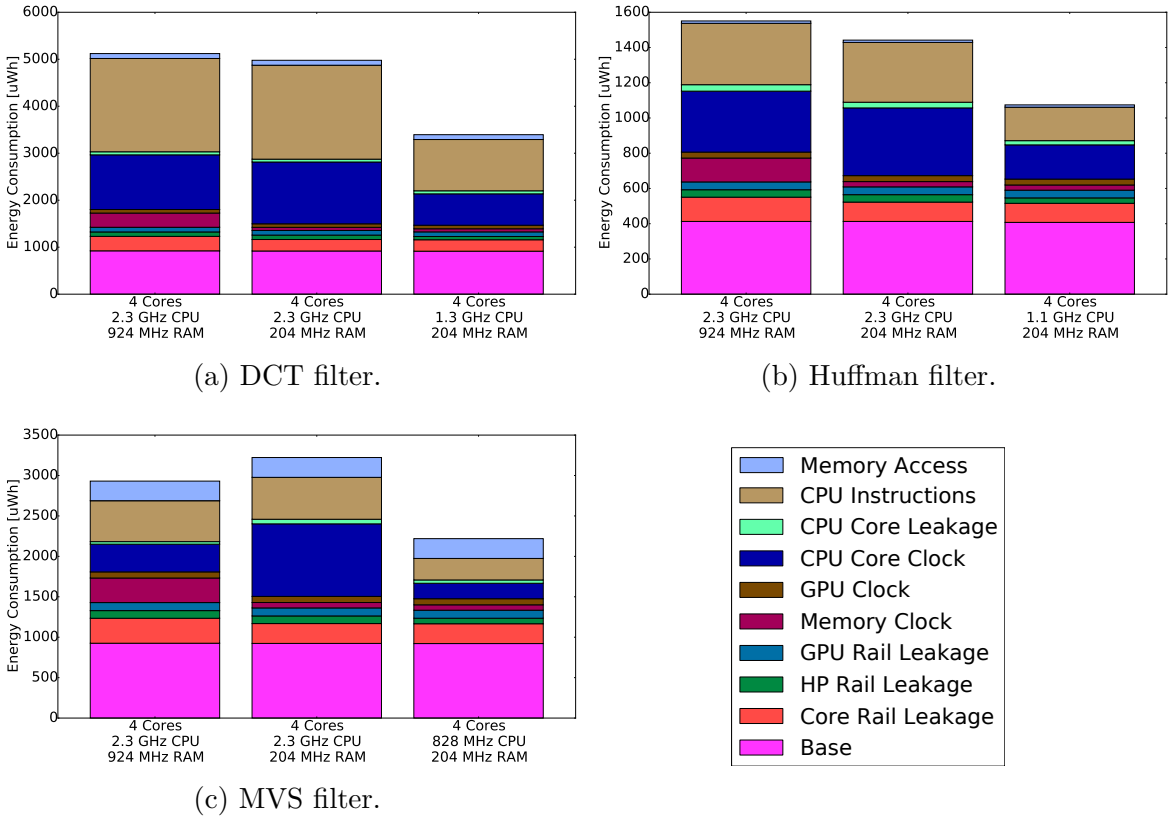
(b) Huffman filter.

(c) MVS filter.

Figure 5.1: Energy breakdown for the CPU filters. The barplots show the best CPU-memory frequency combination for two, three and four active cores.

the other filters. For the Huffman filter in Figure 5.1b, we can see that moving from two to three cores enables us to reduce CPU frequency from 1.5 GHz to 1.1 GHz. Similarly to the DCT filter, this helps reduce CPU clock and instruction power. However, adding the fourth core here has a negative effect on the total energy consumption. This result is consistent with what we can measure. The energy per frame increases from $33.14\frac{\mu Wh}{frame}$ to $33.21\frac{\mu Wh}{frame}$. The increase stems from the increase in HP rail leakage as the fourth core is added. There is no positive effect, such as further reduction in processor frequency, that can balance out the increase in static power usage on the HP rail. For the MVS filter in Figure 5.1c, the number of active CPU cores has negligible impact on energy consumption. 25 FPS is reached with only two cores on at 960 MHz, and adding cores has no other positive effect than reducing this to 828 MHz. This frequency reduction does *not* reduce HP rail voltage, and therefore, there is no reduction in CPU clock and instruction power. By adding cores, the same amount of work (instructions and cycles) is done, but it is distributed over more cores. This indicates that the main source of energy-inefficiency is purely the increase in HP rail voltage, and not a result of increased hardware utilisation (more cycles or instructions).

In Section 2.5, we saw that lowering processor and memory frequencies had a positive effect on the EPF of our filters. However, we could not state exactly why this occurred because it is not possible to measure the power usage of individual units on the Tegra K1.



(a) DCT filter.

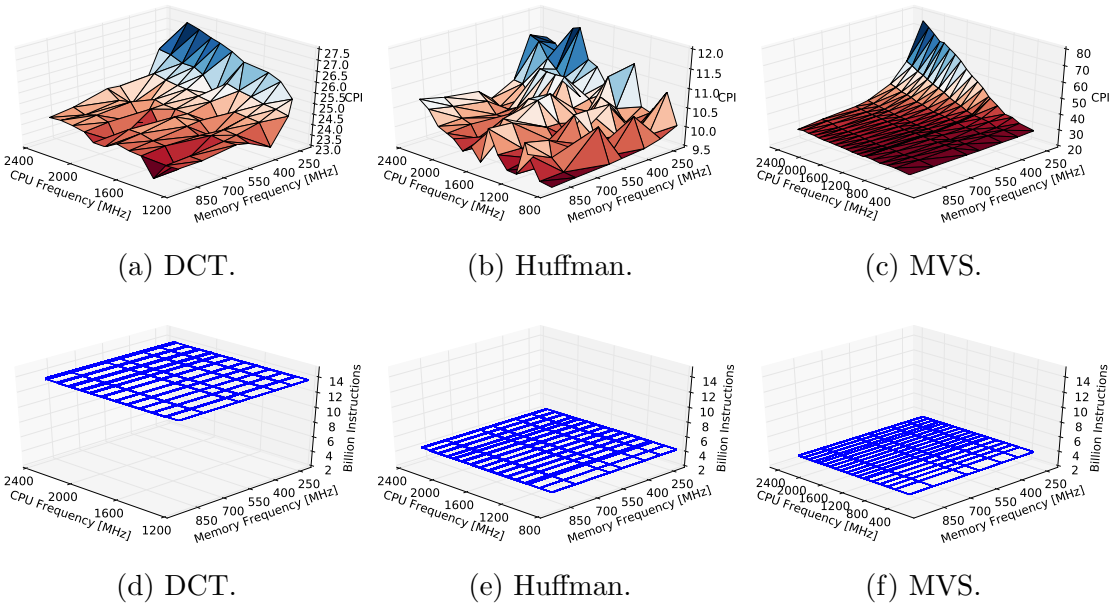
(b) Huffman filter.

(c) MVS filter.

Figure 5.2: Energy breakdown for the CPU filters actively processing on four cores.

Our model is able to show how frequency scaling influences the power usage on the Tegra K1. Figure 5.2 shows the energy consumption of our filters on four cores. We now study the total energy consumption as the processor and memory frequencies are reduced from the maximum processor and memory frequencies (left bar in each plot). The memory and processor frequencies, respectively, are set to the lowest that reach the required framerate in the middle and right bars. For each of the filters (DCT in Figure 5.2a, Huffman in Figure 5.2b and MVS in Figure 5.2c), we can observe that, reducing memory frequency has many positive effects. In all of the filters, the memory frequency can be lowered to 204 MHz while still reaching 25 FPS. The Tegra K1 supports memory frequencies down to 12.5 MHz, but we were unable to test these because the Tegra K1 suffers from occasional hangs when using these frequencies. The core leakage energy is reduced. This is because processing is restricted to the HP cluster, and therefore, the core rail voltage only depends on the memory frequency. Additionally, the memory clock energy is reduced. There is, however, no change in the cost of memory accesses (active memory cycles). This is because the memory rail voltage is always 1.35 V, and the benchmarks all consume the same number of active memory cycles, regardless of the frequency settings.

If we study the middle bars in Figures 5.2a, 5.2b and 5.2c, we can see that lowering the memory frequency to 204 MHz has an interesting side-effect: the CPU core clock energy increases. For the DCT and Huffman filter, the total energy usage is lower than



(a) DCT. (b) Huffman. (c) MVS.

(d) DCT. (e) Huffman. (f) MVS.

Figure 5.3: Cycles per instruction (top row) and total number of instructions (bottom row) for the video processing filters.

when the memory frequency is at 924 MHz, but for the MVS filter, reducing memory frequency to 204 MHz *increases* the total energy usage. These results are consistent with actual measurement of the platform energy usage, which is $3281\mu Wh$ at 204 MHz memory frequency and $2858\mu Wh$ at 924 MHz. Because the CPU frequency remains unchanged, the increase in CPU core clock energy must be caused by an increase in the number of cycles. This is likely to be an effect of reduced instruction pipeline inefficiency, where each instruction consumes more processor cycles to complete. This effect is hard to capture without our power model. Interestingly, when we additionally reduce the processor frequency to 1.3, 1.1 and 0.8 GHz (right bar), the HP rail leakage and CPU core clock energy is reduced. This reduction in core clock energy is caused by the drop in HP rail voltage as the CPU frequency is reduced, but it is also possible that each instruction consumes less CPU cycles (increased pipeline efficiency).

Figure 5.3 illustrates the effects that frequency scaling has on the number of Cycles per Instruction (CPI). We can see that, for all the filters, the CPI count is higher at low memory and high CPU frequencies. The effect is most prominent for the MVS filter in Figure 5.3c. Here, the processor efficiency is at best around 27 CPI, but when the memory and processor frequencies are set to 204 MHz and 2.3 GHz, respectively, the CPI can be worse at around 72 CPI. If we study the measurements in Figure 2.12d, we can see that the EPF increases substantially in this same frequency region. The total number of instructions executed in each benchmark is always the same (see Figures 5.3d, 5.3e and 5.3f), but because each instruction consumes more cycles to complete, the energy per frame increases for frequency combinations where the CPI increases. The same observation can be made for the DCT and Huffman filters in Figures 5.3a and 5.3b, but the effect is not as detrimental as for the MVS. At low memory and high processor frequencies, the processor efficiency decreases to 27 CPI for the DCT filter. This is an increase from the "best case"

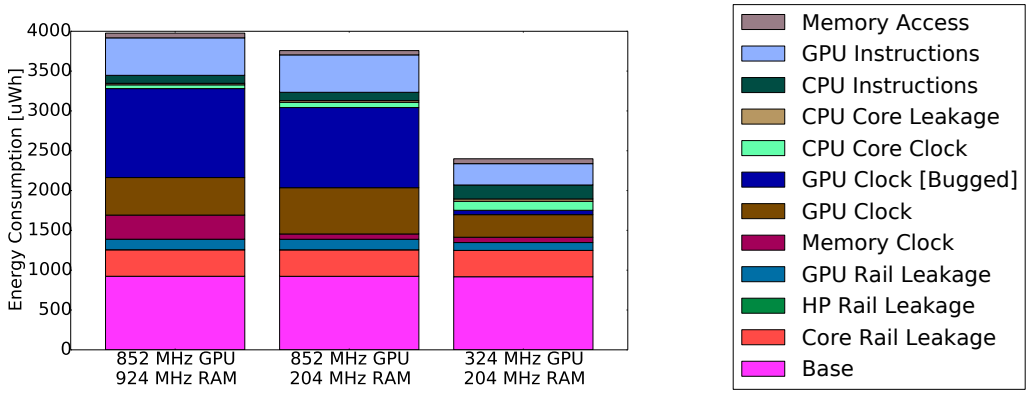
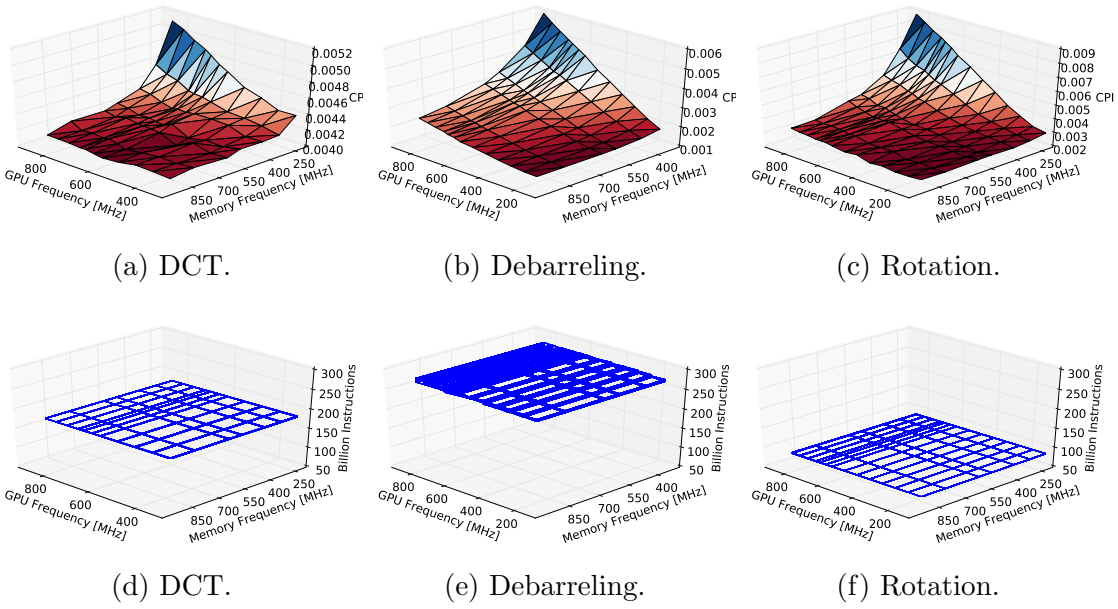


Figure 5.4: Energy breakdown for the DCT filter running on the GPU. The barplots show the best GPU-memory frequency combination, where the CPU is idle and restricted to the LP core.

CPI value of around 24. The change in CPI value is negligible for the Huffman filter. As a result, these findings indicate that only *some* workloads are severely affected by poor CPI scaling at certain processor and memory frequency combinations. The effect of the increasing CPI value for the DCT filter, for example, is not easily distinguishable in the measurements in Figure 2.10c.

We now study the factors that lead to energy-inefficiency on the GPU. The Tegra K1's GPU is not able to process the MVS and Huffman filters at 25 FPS. However, it reaches 25 FPS for the DCT filter, where the energy breakdown over three GPU and memory frequency combinations is shown in Figure 5.4. In this energy breakdown, we differ between "bugged" and "normal" GPU clock cycles. The bugged cycles are caused by the CUPTI power management bug, and are there because the GPU is not able to clock-gate itself when idle. The normal cycles are those caused by the actual processing of the filter, and is measured as normal using CUPTI HPCs. Note that, under normal processing (not bugged) circumstances, the GPU clock is gated after a pre-defined number of idle milliseconds. This number can be set manually in sysfs. This means that there will always be some excess waste when the GPU is idle, caused by the clock being active for some set amount of time, before it is gated. The displayed CPU instruction energy is only caused by cache activity, where we neglect the cost of instructions.

From Figure 5.4, we can make similar observations as for the CPU filters. Reducing memory frequency to 204 MHz also reduces the memory clock energy, but core leakage is not reduced. This is because the LP core frequency is fixed at 1.0 GHz, forcing the rail voltage to stay at above 1.0 V. Interestingly, we can also here observe that the number of clock cycles directly related to the processing of the filter (non-bugged cycles) increases. This means that the GPU may also be affected by inefficiency in the processor's pipeline. The "bugged" cycle energy is reduced, because more of these "always-on" GPU cycles are used to process the actual filter. The net amount of GPU cycle energy, however, remains the same. Reducing the GPU frequency to 324 MHz, which is the optimal GPU frequency to process this filter, we can see that GPU leakage energy decreases as a result of reduced GPU rail voltage. GPU instruction cost also decreases because of this, and additionally,



(a) DCT. (b) Debarreling. (c) Rotation.

(d) DCT. (e) Debarreling. (f) Rotation.

Figure 5.5: Cycles per instruction (top row) and total number of instructions (bottom row) for the video processing filters running on the GPU.

the GPU clock power is greatly reduced. There is no visible change in the memory access costs.

We have observed that, similarly to the CPU, there is an inefficiency in the GPU's processor pipeline. At low memory and high GPU frequencies, more processor cycles are consumed to process the same number of instructions. To show this effect in detail, consider the DCT, debarreling and rotation filters in Figure 5.5. The GPU can reach 25 FPS for all of these filters. The results show that the CPI is heavily dependent on the frequency settings. At 204 MHz memory frequency, the number of cycles required to complete an instruction increases with GPU frequency. For examle, the CPI for the DCT in Figure 5.5a goes from 0.004 to 0.005, the debarreling filter in Figure 5.5b from 0.002 to 0.006, and for the rotation filter in Figure 5.5c from 0.003 to 0.009. Compared to the CPU, the number of cycles required to complete an instruction is four orders of magnitude smaller (between 9 and 27 CPI for the CPU). This is because the GPU is able to sustain a higher number of instructions due to its large number of compute cores. However, the large number of cores increases the pressure on RAM. In essence, the GPU's 192 CUDA cores can simultaneously request 128 unique 64-bit data values from memory. We believe this is the reason for the GPU's susceptibility to low memory frequencies. The processor is effectively stalling when waiting for unsatisified memory read and write requests, consuming additional cycles. The effect is destructive for energy-efficiency. The clock energy required to process the same number of instructions (see Figure 5.5d, 5.5e and 5.5f) increases by between 20 %(DCT) to 600 % (debarreling). Note that this effect is not distinguishable from the measurements in Figure 2.10d, 2.8 and 2.9d in Section 2.5. This is because the GPU clock is operating in a bugged state where it is never turned off when idle. Therefore, any cycles saved by operating at frequency levels with lower CPI will be consumed as "bugged" GPU cycles, and it will not be possible to observe this

effect with physical measurement.

In this section, we have studied the effects that adding and removing CPU cores, as well as adjusting processor and memory frequencies, has on the energy-efficiency of our video processing filters. We have shown that adding CPU cores generally has a positve effect on energy consumption for all our filters. This is because the time taken to process each frame is reduced, allowing us to reduce both processor and memory frequency. This contributes to energy-efficiency by lowering rail voltages, reducing both static and dynamic power usage of the Tegra K1. This result means that multi-threaded applications that scale well with the number of CPU cores active have positive effects on power usage. The cost of adding a core, in terms of additional leakage current, is negligible compared to the saving of reducing the workload per core, and by extension, processor and memory frequency. However, if the performance benefit of adding a core is not enough to reduce operating frequencies, such as when adding the fourth CPU core for the Huffman and MVS filters, we observed that the energy consumption increases slightly due to increase core leakage currents. However, this extra cost is negligible.

Considering changes in frequencies, the main cause of increased energy-efficiency at lower frequency levels is reduced operating voltage. It is possible to minimise processor and memory frequencies, such that the required framerate is met, and therefore save both static and dynamic power. Otherwise, the same number of processor instructions are executed. However, we also revealed an additional effect that is hard to observe without our fine-grained power model. At certain processor and memory frequency combinations, the number of cycles required to complete each instruction increases. This means that, at these frequencies, more processor cycles are consumed to process the same number of instructions. We believe that this is caused by reduced pipeline-inefficiency due to off-chip memory access stalls, and that additional memory access contention from contending processes may further worsen the pipeline inefficiency. An interesting topic for DVFS algorithms is that, since the CPI count can be easily measured and is a source of energy-inefficiency, DVFS algorithms should monitor it and attempt to configure platform frequencies to avoid high CPI values for different processes. However, this topic is left for future work.

5.2 Energy-Efficient Load Balancing on Heterogeneous Cores

Many researchers attempt to increase the energy-efficiency of heterogeneous architectures, by dividing workloads between heterogeneous processors [12, 22, 35, 54, 73]. These target mobile SoCs such as the Tegra 2 [12, 73], Samsung S4, Samsung Note II, Google Nexus 7 and Tegra 250 [54], Tegra 3 [22] and Texas Instruments' OMAP 3530 platform [35]. The tested application areas are for example the scale-invariant feature transform [22, 54], where Huang and Lai [22] also experiment with BLAS benchmarks, mobile face recognition [12, 73] and face tracking [35].

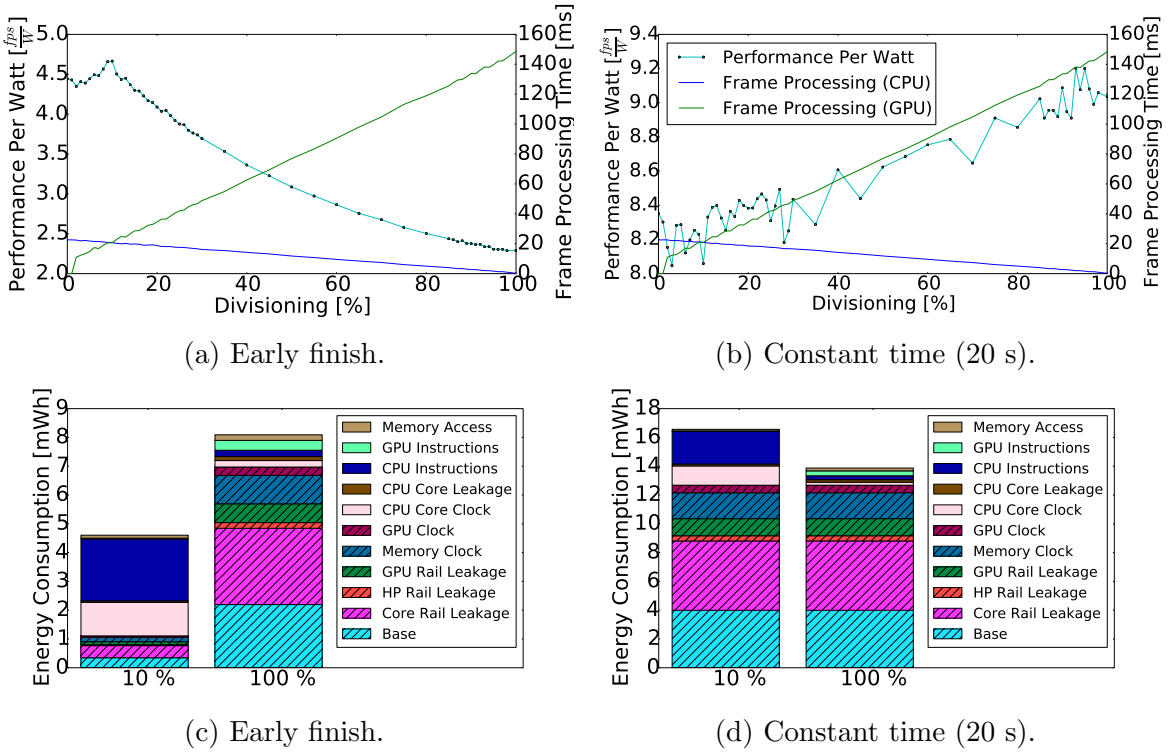
The common approach in these studies is to offload certain computation blocks entirely to the SoC's GPU using the OpenGL ES graphics library. General purpose computing frameworks such as CUDA were not available until the Tegra K1. With regards to what computation blocks are offloaded, this is rarely discussed or argumented for in detail [54].

GPU-offloaded computational blocks for the scale-invariant feature transform, is for example gaussian scale space [54] or convolution, difference of gaussians, octave gradient and key description generation [22, 73]. In their face recognition system for the Tegra 2, Cheng and Wang [12] offload the gabor wavelet to the mobile GPU. Lopez et. al. [35] 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.

Recent research into energy-efficient, mobile computing often implicitly or explicitly make the assumption that processors are more energy-efficient at lower frequencies. We have also observed this phenomenon on the Tegra K1 in the preceding chapters of this thesis. As we concluded in Section 5.1, an 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 [30]. We have also observed that the GPU and CPU CPI increases at certain processor and memory frequencies. In effect, additional cycles and dynamic power are required to process the same number of instructions. However, this raises an interesting question. 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? In this section, we first investigate whether this is possible when the Tegra K1's GPU is processing only the DCT filter, before we let the CPU process the MVS and Huffman filters in additon. A possible extension to this work is to investigate the impact of offloading under scenarios where there is additional contention for processing time on the Tegra K1's CPU and GPU, for example from other processes.

5.2.1 Measuring Energy-Efficiency

Authors who offload work to GPUs [12,21,22,35,54,73] report performance speedups and increased energy-efficiency when both processors (CPU and GPU) are used. However, as explicitly noted by Wang and Cheng [73], the increase in energy-efficiency is probably a result of *reduced execution time*. All electrical platforms have idle power 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 means that authors who *quantitatively* measure energy-efficiency, for example in total runtime energy [12,22,35,73] or Performance Per Watt (PPW) [19,54], observe increases in power-efficiency under offloading 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. 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 processor, 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.



(a) Early finish.

(b) Constant time (20 s).

(c) Early finish.

(d) Constant time (20 s).

Figure 5.6: Evaluating energy-efficiency with different strategies for test runlength.

To illustrate how different conclusions we can make when not considering idle power, consider the following experiment. Consider having our DCT filter process 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 5.6a shows the PPW (in frames per second per watt) as measured over varying offloading degrees from 0 to 100 % GPU offloading. For example, at 40 % offloading, the GPU is processing 40 % of each incoming frame, and the CPU is processing 60 %. 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. 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 processors. 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 visible in Figure 5.6a, where the highest performance (framerate) 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 [64, 65]. The per-frame energy breakdown can be seen in Figure 5.6c, where the lined blocks correspond to energy consumption from idle components. The

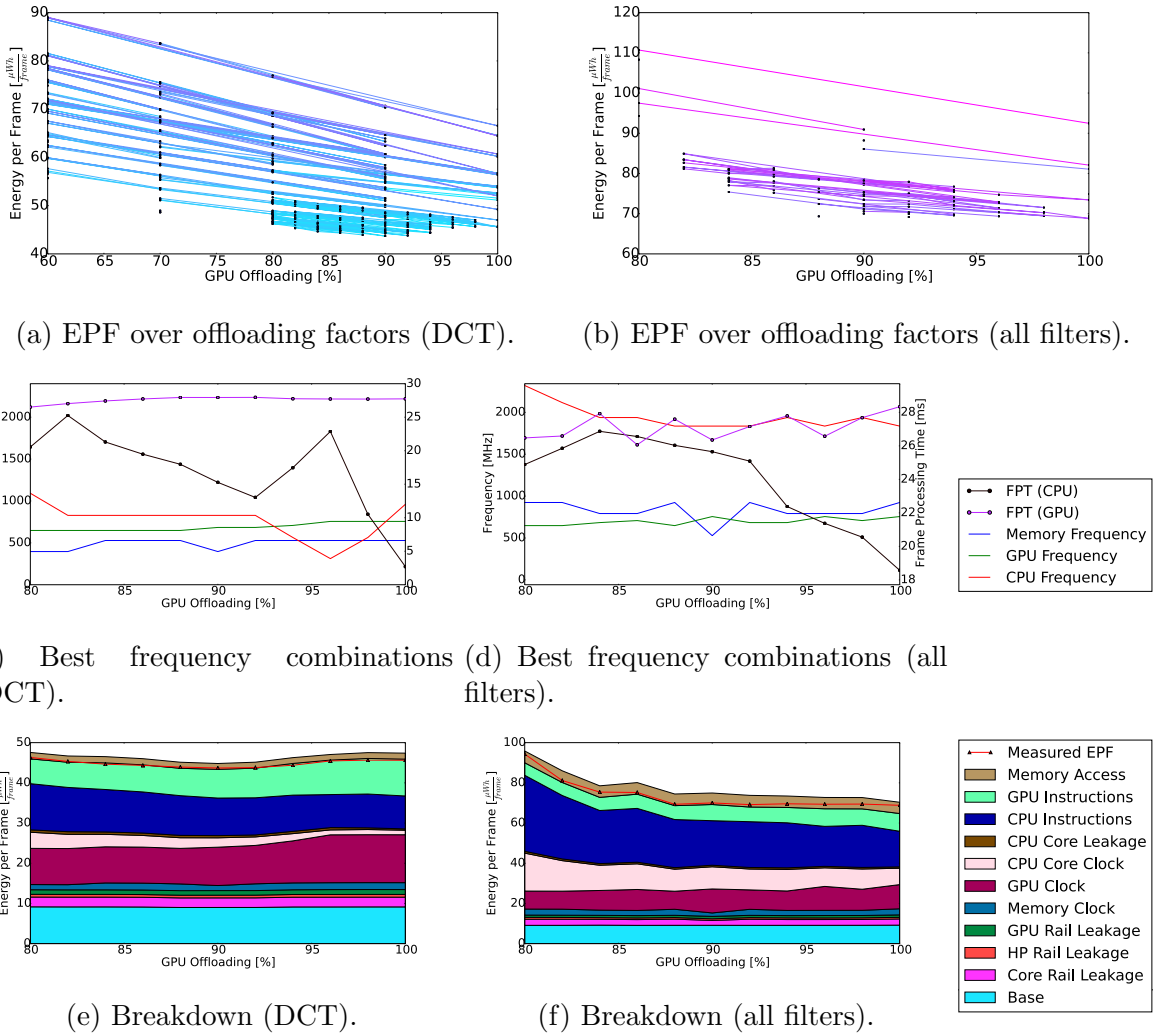
rest (non-lined) components are directly related to only the DCT's hardware utilisation on each processor. Comparing the offloading factors, we see that at 10 % there is a substantial reduction in idle energy usage compared to 100 % offloading. This is caused by the short runtime of the test. However, 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 5.6a.

Many hardware manufacturers use PPW as a metric when evaluating energy-efficiency of their products, but it is usually not clear how their experiments were run. Therefore, it is impossible to say whether the actual power usage of the benchmarks on their hardware is reduced. It is for example possible that the increase in PPW is due to increased floating-point performance, allowing benchmarks to finish faster and thereby save more base power. Interestingly, both approaches have merits. Consider for example a server farm, where the service provider charges customers for power usage. From the perspective of the customer, it is in their best interest to have a high PPW value for their workloads, to avoid unnecessary expenses. However, as we have seen in this section, this can increase the overall energy consumption, and the power expenses of the service provider. The problems of measuring energy-efficiency using quantitative metrics can be overcome without detailed power breakdowns and models. All that is necessary 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 5.6b, we see that the PPW is best at high GPU offloading percentages. This is again confirmed with a detailed breakdown in Figure 5.6d, 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.

5.2.2 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, MVS 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 MVS filter scales much better, in terms of both performance and energy, on the CPU than the GPU. For the video encoder to meet a



(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) Breakdown (DCT).

(f) Breakdown (all filters).

Figure 5.7: Offloading experiments.

35 FPS full-HD QoS requirement, the HP cluster therefore *must* be active with three or four cores to process the MVS 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 [61] and other previous work [48, 74], that more energy efficient configurations are found there.

5.2.3 Offloading a Single Filter

We now focus on the DCT filter according to our methodology in the previous section. Figure 5.7a shows the measured DCT EPF 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 frequencies are used. Only the data points that reach the target frame rate of 35 FPS 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 [48, 61, 74]. In the coarse test run, the frequency combination that achieved the lowest EPF (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 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. 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 EPF is also lower at $43.69\frac{\mu Wh}{frame}$.

Figure 5.7c shows, for any offloading percentage, the *best* frequency combination (in terms of EPF) 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, if we further lower the CPU frequency.

Further studying Figure 5.7c, 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, the 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 frequency 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 unit 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 use our power model to understand what is happening on each rail in terms of static and dynamic power. Figure 5.7e shows, for the best frequency combination at each offloading factor, EPF broken 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. We can also 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 offloaded to 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 %.

5.2.4 Offloading Under Heavy Processing

In the previous section, we observed that it is possible to save 5 % EPF by offloading 10 % of the DCT workload from the GPU to the CPU, under a QoS requirement of 35 FPS. 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 MVS and Huffman encoding. If the CPU is busy processing other tasks, the extra cost of offloading 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 the MVS as well as the Huffman filters. 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 5.7b, where we see that while the 100 % offloading configuration is best, there are several configurations, such as at 86 and 92 % GPU offloading, that are very close to achieve the same EPF. Studying Figure 5.7d, 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 5.7f shows that 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 MVS 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 $67.25\frac{\mu Wh}{frame}$.

In this section, motivated by the energy-inefficiency of the Tegra K1's processors at

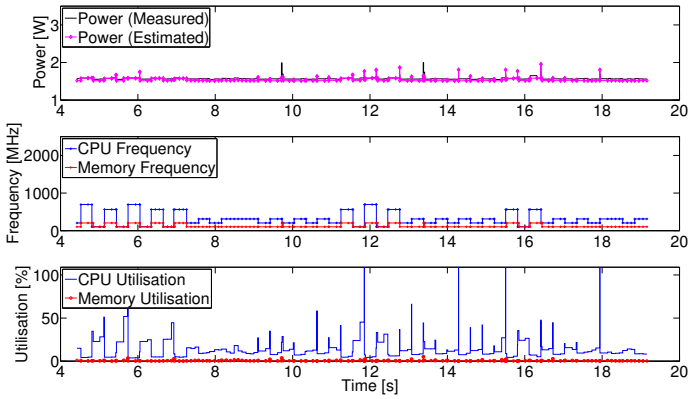


Figure 5.8: Platform state under normal (idle) operating conditions.

high frequencies, we have performed several preliminary experiments to save energy by off-loading processing between heterogeneous processors. We have seen that it is possible to save 5 % EPF by offloading 10 % of the per-frame DCT workload to the CPU, where the remaining 90 % is processed on the GPU. By offloading some processing to the CPU, we can reduce the frame processing time on the GPU, additionally reducing GPU frequency and rail voltage. The benefit of offloading, in terms of reduced GPU static and dynamic power, is larger than the additional cost of processing 10 % on the CPU. However, if the CPU is already constrained and working on other workloads, such as when performing MVS and Huffman coding to file at the same framerate, we have observed that the additional cost of processing on the CPU is at least equal to the saving of reducing GPU frequency. This occurs because the CPU is already processing at a frequency close to 2.0 GHz, where the energy-efficiency of CPU instructions is decreased due to increased CPU rail voltage.

5.3 Tegra K1 System-Level Energy Analysis

Many mobile devices such as smart phones typically spend large amounts of time in an idle state, where most components are left unused and the power usage should be minimal. An important challenge for system-level energy optimisation is therefore to understand how energy is consumed at a fine-grained level. For example, if we know how much energy is consumed for cache maintenance, instruction execution and off-chip memory access, it is easier to locate the important sources of energy loss. If for example a driver is performing excessive device communication over an expensive communication bus, it may be possible to further optimise power usage by doing changes to the code. In this section we perform an idle state analysis of the Tegra K1, investigating how the platform manages its resources, how system services contribute to power and how these services can be optimised for energy-efficiency.

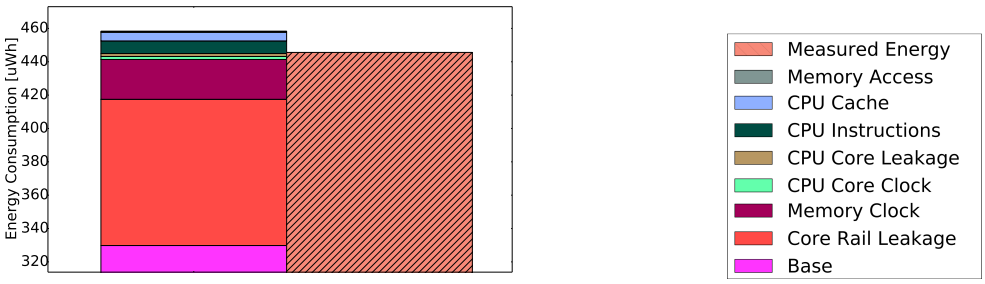


Figure 5.9: Idle power component breakdown. The right bar is the measured energy consumption, and the left bar is the estimated.

5.3.1 Component-Level Breakdown

Under normal operating conditions, the Tegra K1 is idling on the LP core at low memory and processor frequencies. Figure 5.8 shows how frequencies are tuned in response to changes in processor utilisation on Rey. The only significant process running in the system is our profiler, which periodically samples and logs the platform state. We can see that the CPU frequency varies between 200 to 800 MHz. The memory frequency remains close to 200 MHz. The lower part of the plot shows the CPU and memory utilisation. These are defined as the number of active cycles (CPU or memory) relative to the current frequency operating point of that component. The graph shows that the CPU DVFS algorithm reacts quickly to changes in processor utilisation. As the CPU utilisation is already well below 100 %, it is possible that these increases in CPU operating frequency are unnecessary.

We now use the model coefficients from Table 4.5 for Rey to build a more complete picture of the power usage of individual units of the Tegra K1. When we attempted to do this while the Tegra K1's default power management mechanisms were active, the HP cluster was activated and the CPU frequency was set to values well above 1 GHz. We therefore force the Tegra K1 to operate at 204 MHz CPU andn 204 MHz memory frequency, and restrict processing to the LP core. This has no detrimental effects on our profiler's ability to predict and log power usage in time. To model instruction cost, we let the Tegra K1 idle for one second at every possible CPU and memory frequency. Using regression, we calculate the cost of idle instructions to be $1475\frac{pC}{V}$, with a "constant base instruction power" of 10.07 mW. With an V^2IPS of around 10^6, this corresponds to an instruction power of around 15 mW. The prediction can be seen in Figure 5.8 and is 97.7 % accurate over time.

The component-level breakdown can be seen in Figure 5.9, where the total energy consumption for an idle run of 1 s is shown. With $329.8\mu Wh$, the base cost is the most substantial component. However, this energy represents electrical components external to the Tegra K1 SoC. The most substantial component on the SoC is the core rail leakage with $87.6\mu Wh$. This rail is impossible to turn off without settling the Tegra K1 in more low-powered sleep modes. The memory clock is the second most energy-consuming component, consuming $23.9\mu Wh$. The memory clock cannot be clock-gated because it must always be on to maintain data consistency. It is possible to lower the memory frequency below 204 MHz, but as already stated in the preceding section, this tends to

hang the Tegra K1. The rest of the components, meaning the LP core clock and leakage, cache and instructions as well as external memory accesses comprise $12.3\mu Wh$. Relative to the rest of the SoC, this is only 10 % of the idle system energy (not considering base energy). Of these components, the estimated CPU cache and instruction energy components are larger and estimated to be 5.3 and $2.8\mu Wh$, respectively.

5.3.2 Optimising the ACTMON Kernel Driver

Following the discussion of the previous section, the rest of the idle power components (LP core leakage, instruction cost, cache maintenance and off-chip memory accesses) cost $12.3\mu Wh$ over one second. Much of this energy consumption is directly related to system software, but is only 10 % of the Tegra K1's idle power. Although this cost is small, we now conduct an analysis to investigate and optimise the power usage of individual system services.

There are many system processes running on the Tegra K1. Of these, only the activity monitor drivers tracking off-chip memory utilisation are the most active. Because our profiler process is normally not present, the largest optimisation potential is therefore in the activity monitor. We now run the Linux tool `perf` to collect HPCs for 30 s, which shows the percentage of instructions executed in specific function calls. The following points show the five most significant entries:

1. 8.52 % avahi-daemon, unnamed function call.
2. **7.48 %** activity-irq, software interrupt callback.
    - **4.40 %** activity-irq, get clock pointer from string.
        - **4.39 %** activity-irq, string compare.
3. 4.83 % swapper, spin unlock and enable interrupt.

It is clear from the above points that 4.40 % of the platform instructions are executed in the activity monitors' interrupt handler, with 4.39 % of in a string comparison function. This function is in turn called from a function which gets a pointer to a clock from a clock name. Upon inspection of the source code, we see that this function call is redundant for every invocation of the interrupt handler. This means that it can be safely removed as long as a single reference to the clock is stored at the first access to that clock pointer. Additionally, the interrupt intervals for the activity monitor is set to 12 ms. This can safely be increased to lower the rate at which these interrupts occur, and consequently, reduce the instruction throughput from these drivers.

Figure 5.10 shows the effect that these optimisation strategies have on the estimated and measured idle energy consumption over five seconds. The y-axis starts at 2180 μWh to increase the visibility of the instruction and cache power components. The left bar shows the standard scenario where the interrupts are occurring every 12 ms. The clock-call is functioning as normal. The middle bar shows the effect of only calling this function once (to retrieve a clock data structure) and subsequently re-using it in subsequent kernel interrupts. We can see that the estimated energy consumption is lowered by $2\mu Wh$. However, the measured reduction is only $0.2\mu Wh$. This corresponds to a measured reduction

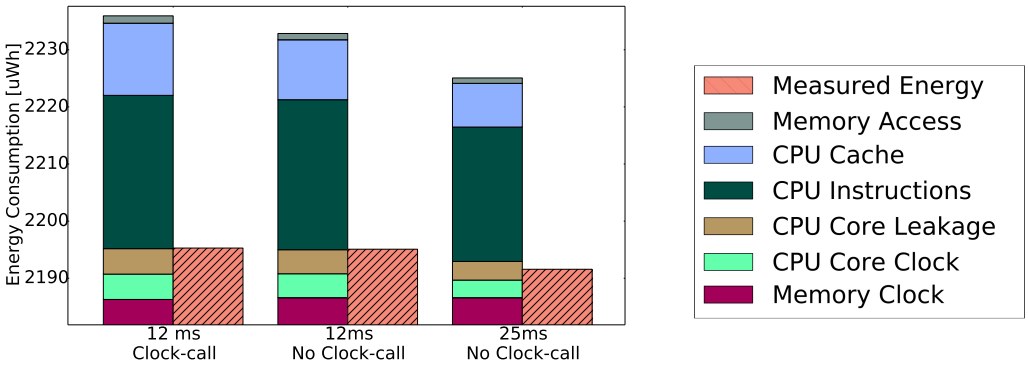


Figure 5.10: Idle system-level energy breakdown for the Tegra K1 under various optimisation strategies for the activity monitor drivers. The right bars represent the measured energy consumption, and the left bars represent estimated.

in average power of 0.7 mW. The larger estimated reduction in power is mainly caused by a reduction in the cache energy component. The right bar in Figure 5.10 shows the effect of doubling the interrupt interval from 12 to 25 ms. This halves the number of interrupts per second. The estimated reduction from the original source code (left bar) is $11\mu Wh$, with a measured change of $4\mu Wh$. This corresponds to a reduced average power of 14.4 mW. In addition to further reduced cache and instruction utilisation, the LP core leakage current is reduced. This is because the LP core is being power gated more often due to higher idling intervals between the interrupts.

The estimates in Figure 5.10 overpredict the reduction in power usage. However, we regard it as a positive result where we can observe reductions in the estimated and measured values. It is also important to be aware that we are handling changes in energy that are extremely small when compared to the rest of the system. Our method of attributing a constant, average capacitive load per idle instruction may also be improved. For example, the OS operates based on events that occur in random intervals, such as hardware interrupts and applications' interactions with system calls. It is therefore not given that the same capacitive load can be observed over time, because the OS does not always execute the same code paths over time. Also, many of the services in the OS have a constant throughput of instructions that do not vary in terms of for example frequencies. The ACTMON drivers is a good example, where the rate of instruction throughput rather depends on the interrupt frequency. Therefore, frequency scaling cannot affect the instruction throughput sufficiently to trigger changes in instruction throughput. An interesting topic for future work is to further analyse the power usage of instructions, by attributing a capacitive load per instruction executed in individual code-paths in the kernel. This could also be done for software applications.

In this section, we have investigated the idle-system power usage of the Tegra K1. We have identified the largest consumers of energy, which is the core rail leakage and memory clock. Such an analysis can be useful for system architects, for example to implement further stages of power gating on the core rail or to use more modern types of memory that can operate on lower voltages. For example, LPDDR3 can operate at 1.2 V, whereas conventional DDR3 memory equipped with the Jetson-TK1 operates at 1.35 V. Furthermore, we have considered idle system instruction power on the Tegra K1. We

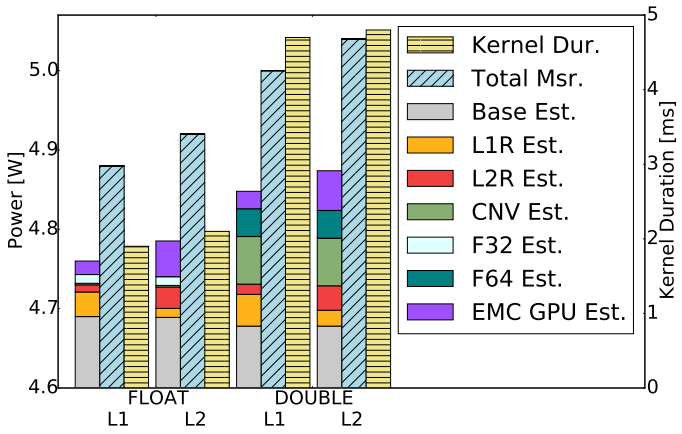


Figure 5.11: Effects of different instructions and cache usage on GPU power usage. The left and middle bars show the estimated and measured power usage, respectively. The right bars show the kernel duration.

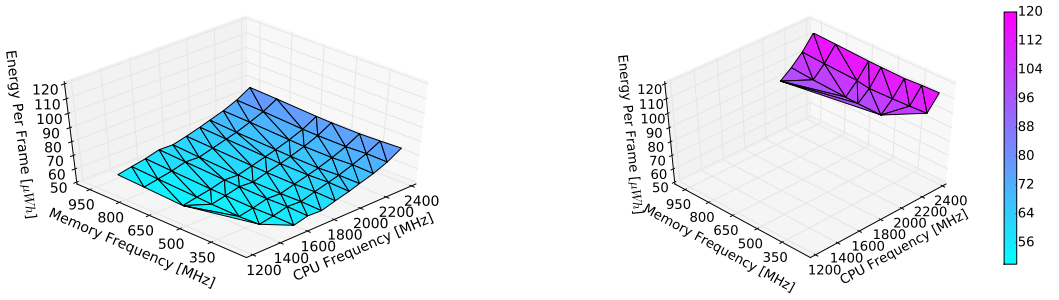
analysed the power usage of the ACTMON drivers and detected several potential areas of optimisation, such as removing expensive function calls and increasing the interval at which interrupts occur. However, while we observed reductions in energy usage, our power model here overestimated the reduction. We believe that more fine-grained instruction power estimation is needed, and that this can be a topic for future work. For example, an interesting question can be if it is possible to attribute instruction costs through individual code-paths in applications and drivers, and thereby increase the power estimation accuracy.

5.4 Instructions' Effect on Energy Consumption

Our discussion so far has revolved mostly around frequency settings and how these impact performance and energy consumption of various multimedia workloads. This type of optimisation is "macroscopic" in that the actual code is not augmented. A form of "microscopic" optimisation would be to study the effects that for example different instructions, or caching in different cache hierarchies, have on energy consumption. In this section, we show the effects that these effects have on performance and energy consumption on the Tegra K1's GPU and CPU.

5.4.1 GPU Instructions and Cache Modifiers

Figure 5.11 shows the measured total and estimated breakdown of power for the DCT benchmark running on the Tegra K1 at maximum operating frequencies. The right bar shows the default implementation using double-precision floating point data types and caching in the GPU's L2 cache. We now instruct the compiler to cache data in only L1 cache and not L2 (first and third bar from the left in Figure 5.11). Compared to default caching in L2 (second and fourth bar) there is no change in the power usage of the floating point and conversion instructions. However, L2 read power has bee reduced,



(a) With NEON instructions.

(b) Without NEON instructions.

Figure 5.12: Energy per frame for the DCT filter with and without NEON instructions.

while L1 read power has increased by the same amount. An interesting side-effect is that the GPU's memory power usage (EMC GPU) also decreases, which also causes the L1 caching strategy to be more energy-efficient than L2 caching. This surprising result can be explained if we consider the fact that the GPU's L1 cache is not cache coherent with memory. This reduces communication with off-chip memory to maintain consistency of the data. The estimation error with respect to measured power is large because these results were based on the GPU model in Section 4.5, which did not have our improved CPU model. However, if we study the relative reduction in measured and predicted power with L2 and L1 caching, we can see that the reduction in power usage is the same. Additionally, caching in L1 over L2 has a small positive effect on the kernel's performance.

Using single- over double-precision floating point instructions can also increase energy consumption and performance. Considering L1 caching with single- and double-precision instructions (first and third bar in Figure 5.11), we can see that the double-precision floating point power has been replaced with a smaller single-precision component. Additionally, conversion instruction power has been removed entirely. We also see that the relative predicted and measured power is similar, with an improved power usage of 3.2 %. However, the actual saving is larger if we isolate the GPU from other power components such as the CPU and base power. Considering only the power components in Figure 5.11, for example, we have more than halved GPU cache and instruction power. These effects would be impossible to see without our power model. Additionally, using single-precision floating-point values has a positive effect on performance, where the kernel duration is reduced by around 200 ms.

5.4.2 NEON Acceleration

The Tegra K1's CPU does not have support for choosing where to cache data. However, it does support NEON instructions. These are SIMD instructions that can perform arithmetic operations in parallel on several data values using only one instruction vector. Our DCT benchmark has been written both with and without NEON. To test the difference in these implementations, we run the DCT with and without NEON instructions over all CPU and memory frequencies with a framerate target of 25 FPS. Four HP cores are active. In these experiments, we observed that the heaviest level of compiler-time optimisations, `-O3` in `gcc`, is able to accelerate the no-NEON implementation with NEON

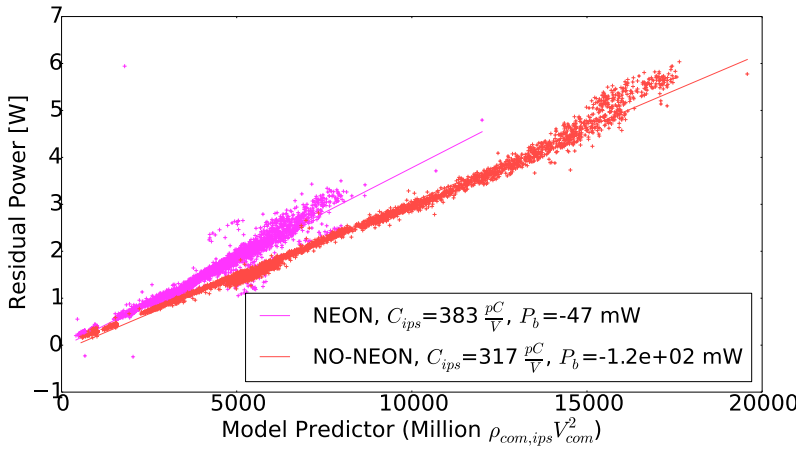


Figure 5.13: Capacitive load per instruction for the DCT filter with and without NEON instructions.

instructions, and the performance is similar to our own, hand-written NEON implementation. Therefore, for demonstration purposes to show the differences, the code in this test was compiled with `-O1` instead, which does some optimisation but avoids the use of NEON instructions. The results can be seen in Figure 5.12, where Figure 5.12a shows the frequency combinations that reached the 25 FPS requirement with NEON instructions enabled. Figure 5.12b shows the combinations that reached 25 FPS without NEON acceleration enabled. From the figure it is clear that this type of instructions have critical effects on both performance and power usage. Without NEON, the lowest energy per frame is $96.10\mu Wh$ at $f_{cpu} = 2GHz, f_{mem} = 924MHz$. The NEON-accelerated version reaches 25 FPS at $f_{cpu} = 1.3GHz, f_{mem} = 600MHz$ with an energy per frame at $51.92\mu Wh$. This is an improvement of almost 50 % when considering the whole Tegra K1.

Our modeling methodologies can reveal what is happening within the SoC. For example, if we study the capacitive load in Figure 5.13, we can make two important observations. First, the capacitive load per instruction is higher for the NEON DCT than the one without NEON acceleration. This is reasonable given that NEON instructions operate on more data elements at a time, exersising more of the CPU's circuitry in terms of dynamic power. Second, the NEON-accelerated version reaches a lower model predictor, V^2IPS. For example, the maximum V^2IPS for the NEON-accelerated version is around 7.5 billion V^2IPS, while it is much larger at 17 billion V^2IPS for the normal version. This occurs because a fewer number of instructions are executed in total, when multiple arithmetic operations can be done with a single instruction. An energy-breakdown for the best frequency combinations with both DCT versions is shown in Figure 5.14. Here, we can clearly see that the most substantial reduction comes from reduced instruction energy consumption. Some of this reduction can be attributed to reduced HP rail voltage, but as already stated the number of instructions required to complete processing the frames are also reduced. This contributes to lowering instruction power, despite the fact that the average capacitive load per instruction is higher. Cache energy is also reduced. In addition to these effects, the traditional gains of lowering frequencies has positive effects for both leakage currents as well as the CPU and memory clock.

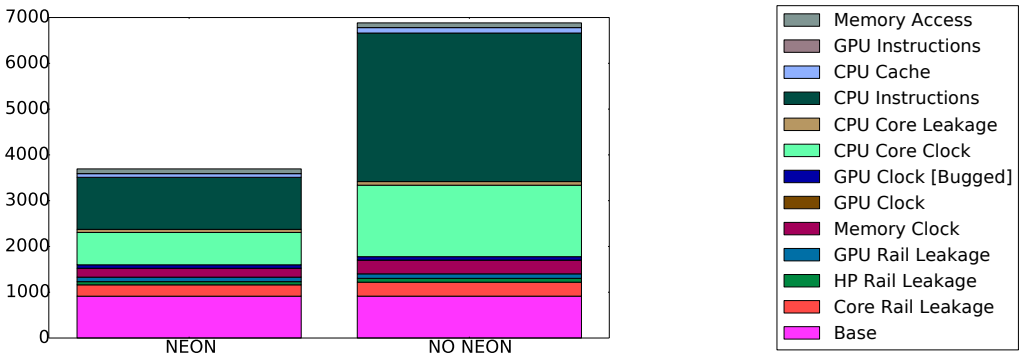


Figure 5.14: Comparison between NEON and non-NEON DCT filter energy breakdowns.

In this section, we have shown the effects in terms of power and performance of programming using different types of instructions and cache modifiers on the Tegra K1. Using shorter floating point datatypes and caching in the GPU's non-coherent L1 cache, for example, can save 3.2 % energy for our DCT filter. We have here experimented with 32-bit versus 64-bit floating point datatypes, but newer Tegra SoCs support half-precision (16-bit) datatypes as well. Based on our observations that shorter datatypes consume less energy per instructions, as well as less processor cycles, we consider this to be good for energy-efficiency. Caching in L1 over L2 cache additionally saves energy, because that cache is not cache-coherent and as a result, there is less off-chip memory communication to maintain the cache contents. This detail is hard to discover without our fine-grained power model. Additionally, we saw that NEON instructions on the Tegra K1's CPU radically impacts energy-efficiency for the DCT filter. This is because they increase application performance, allowing a significant reduction in processor and memory frequencies compared to programming without them. However, in our experience with the DCT filter, this type of optimisation is easy to achieve because the compiler is already very effective at utilising NEON instructions to boost performance.

5.5 Summary

In this chapter, we have demonstrated the type of insight our power model for the Tegra K1 can yield under different processing scenarios. We showed that increased voltage and CPI are two main factors that contribute to energy-inefficiency at high processor and memory frequencies. Furthermore, we considered the case of an idle system and demonstrated how our model can show the main power consuming entities of the Tegra K1 SoC. Such information is useful to system designers and architects, for example to identify and optimise the power usage of the platform. We also demonstrated how our methodology can be used to identify power usage of individual system services, and to optimise the energy consumption of kernel drivers. The effect of different instructions, such as NEON instructions on the CPU, or single- and double-precision instructions on the GPU, as well as GPU cache modifiers, was also shown to have significant effect on power usage. Smaller floating point data types and NEON instructions, for example, reduces the energy consumption of our DCT filter. Surprisingly, caching in the GPU's L1 cache saved energy

because that cache is non-coherent with respect to memory, and consequently, energy was reduced in off-chip memory activity. Finally, we exploited the fact that the Tegra K1's processors are energy-inefficient at higher frequencies to save 5 % EPF by moving 10 % of the DCT per-frame workload from the Tegra K1's GPU to the CPU. This saving is possible because the reduction in the GPU's per-frame workload allows us to reduce the GPU's operating frequency. The additional cost of processing on the CPU is lower than the saving of reducing the GPU's operating frequency. However, we also showed that if the CPU is already busy processing the MVS and Huffman filters, it was impossible to save energy by offloading because the CPU is more energy-inefficient due to increased CPU rail voltage.

Chapter 6

Conclusion

Devices such as smart phones, laptops and drones provide mobility and utility at unprecedented levels when compared to traditional, stationary compute systems. However, with mobility also comes the issues of energy storage. Modern mobile devices integrate SoCs to run mobile OSs and applications. These are heavily integrated chips providing various functions such as DSPs for hardware-accelerated video encoding and decoding, peripherals such as USB host controllers and PCI-e, as well as general purpose processors. The Tegra K1, for example, provides a rich compute environment with a low-power CPU, a high-performance, quad-core CPU as well as a CUDA-capable GPU. SoCs like the Tegra K1, however, provide no insight into the energy consumption of these hardware components and there is typically limited support for direct measurement of these. In this thesis, our main contribution is a power modelling methodology that is able to predict power usage of the Tegra K1 with close to 100 % accuracy. By only measuring the total power usage of the SoC, our developed model for the Tegra K1 provides detailed insight into dynamic and static power components of heterogeneous processors at an unprecedented level. The model exposes detailed insight into a closed system, and ties software activity and power management mechanisms to the energy consumption of hardware components. In this chapter, we conclude our work in Section 6.1, and outline open issues and future research topics in Section 6.2.

6.1 Summary and Contributions

Many researchers have attempted to build power models for SoCs such as the Tegra K1. These have the potential to expose how a platform consumes energy, for example in terms of different software workloads, to a system architect or a programmer. However, as stated in Section 3.5, we found that even CMOS-based models can mispredict power usage with up to 13 % depending on frequency levels. Most previously developed power models do not evaluate model accuracy over frequency levels. This questions the reliability of these models under power management mechanisms such as for example frequency scaling. Our initial problem statement in Section 1.2 was therefore to show that existing modelling methods are not able to capture an accurate picture of static and dynamic power of the Tegra K1's different processors. In this aspect, we have studied state-, rate- and CMOS-based power modelling methodologies, and found that they all have the potential to mispredict power usage substantially on the Tegra K1. The misprediction

varies depending on operating frequencies of the SoC.

The three main power modelling methods in use today have several weaknesses that cause them to mispredict power on the Tegra K1. State-based models, for example, associate a fixed power usage with hardware states, for example depending on whether the CPU is actively processing or sleeping. As such, they capture constant, static power usage of computational elements, but fail to capture dynamic power usage. They also ignore changes in voltage. On the Tegra K1, for example, the rail voltages vary significantly depending on operating frequencies. Rate-based models correlate power usage with hardware activity, such as elapsed processor cycles or cache misses. This resonates with dynamic power usage (transistor switching activity) in the circuits, but factors such as variations in rail voltage levels nor static power dissipation are not considered in these models. CMOS-based models compensate for rail voltages, correlating power with the clock frequency of the various hardware components, such as the CPU or memory. They also model leakage currents in transistors. However, the capacitive load per clock cycle is not constant over different software workloads, because all programs have different ways of excersising the underlying processor through different instruction mixes. Also, CMOS-based models implicitly assume independency between frequency and capacitive load per clock cycle in different domains.

In addition to the limitations of the three main power modelling methodologies, we have also seen that the methods that are used to build them are not detailed enough to capture the complex hardware mechanisms of modern SoCs, such as rail-, core- and clock-gating. Dynamic power is also rarely reflected accurately through extensive hardware activity measurements. There are for example no examples in the literature that take into account memory utilisation or cache maintenance costs. Motivated by the fundamental weaknesses of state-of-the-art power modelling methods, the second problem statement in Section 1.2 reflects the main hypothesis of this thesis:

> **Hypothesis 1:** To achieve a high degree of accuracy, a power model for the Tegra K1 SoC must combine chip-, hardware- and software-level knowledge into dynamic and static power terms for all of the SoC's architectural units.

In terms of this hypothesis, we have introduced a *high-precision power modelling method* which is able to capture the power usage of the Tegra K1's individual computational elements with close to 100 % accuracy.

Our high-precision power model for the Tegra K1 is a combination of rate- and CMOS-based models, where we correlate dynamic power usage with hardware utilisation, while simultaneously compensating for variations in rail voltage levels in the various parts of the SoC. Dynamic power is reflected though a range of hardware activity measurements at a granularity which has never been done before, where we consider clock-power and clock-gating mechanisms, cache maintenance costs, the cost of active memory clock cycles serving both CPU and GPU memory requests, and the cost of different instructions executed on the GPU. In addition to dynamic power, we also model the static power dissipation of the various power rails of the Tegra K1, in addition to CPU core-gating, through our own kernel tracing frameworks. In contrast to other power models, we have shown that our high-precision power modelling method is sufficiently accurate, when compared to real measurements of the SoC, over all platform frequencies. This is necessary to trigger changes in rail voltages and prove the generality of the method. This detailed

evaluation method is rarely done by related modelling efforts. However, we have built the model on two Jetson-TK1 development kits, as well as a customised board, the T-17. The T-17 is a customised board essentially containing only the Tegra K1 SoC. We have shown that the model coefficients remain similar across these devices.

Through the work in this thesis, we could not build our model for SoCs such as for example the Tegra X1, Tegra X2 or other SoCs used in related research. Therefore, it was impossible to continue researching the generality of our approach on different types of systems. However, we believe that the challenge in terms of power modelling on other SoCs is in the availability of hardware activity sensors, and not in our model's theoretical foundations (see Section 4.1) or the method used to build it (see Section 4.3). To model dynamic and static power accurately, it is necessary to correlate these power components with predictors that cover transistor switching activity and power management mechanisms, to the highest degree possible. Our model categorises many such predictors into static and dynamic power predictors, and ties these to the theory of dynamic and static power dissipation in transistors. In this aspect, we argue that our model is well-founded for any electrical (transistor) implementation of a processor.

Some of the predictors used in our model can be termed "classically used" in that they surface as parts of simpler, rate-based models across different hardware implementations. In this sense, an argument can be made for our model that we use many classical predictors that, based on past experience, are likely to be good predictors in the future. However, these may not be available. A good example is the predictors for the power model developed by Pricopi et. al. [51]. They had access to ARM cores with many HPCs that could measure, in detail, the type and number of instructions executed by the individual cores. However, we could only measure the total number of executed instructions on the platform. A lack of ways to measure hardware activity in a hardware unit does not mean that modelling power in that unit is impossible. It means that the model designer must develop different metrics, possibly at higher abstraction levels, to reflect hardware activity in that unit. If, for example, the activity monitor was not implemented on the Tegra K1, we could resort to using CPU and GPU HPCs to estimate the number of off-chip memory accesses. In this sense, the choice of predictors used in our model should be regarded as guidelines, but not rules, as to what predictors should be used when modelling power on different SoCs.

The lack of available HPCs to measure the type and number of instructions executed on the Tegra K1's CPU complicated our efforts. With only one generic instruction counter per CPU core, we were restricted to measuring the total number of instructions executed by applications. This led us to our second main hypothesis from Section 1.2:

> **Hypothesis 2:** The average capacitive load per CPU instruction remains the same over time for repetitive workloads.

We have shown that this hypothesis is true for our video processing filters, given the linear trend between the V^2IPS and power usage residuals in for example Figure 4.18. This is because repetitive workloads, for example multimedia processing filters that do the same work on consecutive frames, will have the same average cost per instruction over time. However, the debarreling filter also showed us that instruction cost can also depend on the memory frequency if the filter is memory-intensive. This enables us to get an accurate view of the different computational elements of the SoC, bridging the

gap between software activity, power management mechanisms and the actual energy consumption of the Tegra K1's CPU clusters, GPU and memory.

Our power model for the Tegra K1 provides a fine-grained view of the SoC's energy consumption, and reveals what is happening in hardware under the influence of software and power management mechanisms such as DVFS. Lowering platform frequencies, for example, is often explicitly or implicitly assumed to reduce energy consumption of software workloads that must adhere to QoS restrictions, such as a framerate. We found this assumption to be valid in our case studies of several multimedia processing operations that can be applied as *filters* to a raw video stream. It is easy to achieve reduced energy consumption compared to the Tegra K1's standard DVFS algorithms by minimising platform frequencies such that a QoS requirement (a framerate) is met. However, it was impossible to state exactly why this occurred, which led us to our fourth problem statement in Section 1.2. Why are the Tegra K1's processors more energy-efficient at high frequencies? Using our power model, we identified reduced voltage as the main cause of energy-inefficiency. The hardware utilisation, for example in terms of the number of instructions and active memory cycles, remain the same independently of the processor and memory operating frequency. However, the increase in voltage as the standard DVFS algorithms tend to stay at unnecessarily high frequencies has a drastic effect on dynamic power. We have seen this occur under our video processing filters as well as in an idle system. Additionally, static power dissipation increases with voltage. It therefore becomes important to design or tune DVFS algorithms such that the required performance can be delivered, while at the same time using as little energy as possible.

In addition to increased rail voltages, we also discovered another interesting effect that contributes to energy-inefficiency. At some frequency combinations, the number of cycles required to complete an instruction increases. For example, the number of cycles, CPI, required to complete each MVS instruction on low memory and high processor frequencies is above 70 CPI. This is much higher than the "best case" CPI value of 27. This means that more cycles are required to complete the same number of instructions. This is probably an effect of reduced processor pipeline efficiency. While this effect is restricted to a small number of processor and memory frequency combinations, the effect is detrimental to energy-efficiency in scenarios where the DVFS algorithms employ those frequencies. Additionally, the results indicate that the effect is tied to the use of low memory frequencies, which is important to keep the core rail voltage and static power usage low.

We have seen that the Tegra K1's processors and memory are more energy-efficient at low frequencies. This is mainly due to increased voltage, but also due to an increased number of processor cycles required to execute each instruction. Given that a processor is more energy-inefficient at higher frequencies, is it possible to save energy by offloading some work to another processor? The idea behind this is that, if some workload can be offloaded from a processor that is working at high and energy-inefficient frequencies, its frequency can be reduced while still meeting the framerate requirement. The saving caused by the reduction in frequency must be higher than the additional energy cost of processing some of the workload on another core. For the DCT filter operating at 35 FPS, we showed that it is possible to save 5 % energy by offloading 10 % of the per-frame workload to the Tegra K1's CPU. However, if we also let the Tegra K1's CPU process the MVS and Huffman filters, we were unable to achieve a saving although the

margins were very small. This is because the CPU has to work at 1.8 GHz to reach 35 FPS for its filters. At this frequency, CPU rail voltage also increases, and therefore the additional cost of processing the 10 % DCT workload becomes too high to achieve a saving.

Whether offloading can save energy arguably depends on three factors, although we have not looked into all of them. First, it is the voltage state of the processors. The higher the voltage, the higher the potential for a saving if processor frequency is reduced. However, it also reduces the potential of a core to relieve other processors of their workloads. Second, the performance of the filter is important. The faster a filter finishes processing on a core, the fewer clock cycles it will need to completion. Of course, other factors such as instruction and memory usage costs also matter. Most of our filters exhibit very different performance on the Tegra K1's CPU and GPU. The MVS and Huffman filters are good examples, where we have never been able to achieve a saving by offloading to the GPU. As such, the DCT is the best example because both the CPU and the GPU reach 35 FPS at a broad range of operating frequencies. A third factor is the impact of competition for resources. For example, if the GPU in our single-filter offloading scenario was busy processing other workloads, its frequency must also be set higher to reach QoS requirements, thereby reducing energy-efficiency. This could again have a positive effect on the effects of offloading. An important last remark on the usefulness of offloading is that it is an additional burden on the programmer. In addition to implementing an algorithm on two heterogeneous cores, the workload would have to be divisible across those cores. However, based on our observations throughout this thesis, we believe that not all workloads have to be divisible. It is enough with one filter that has similar performance and energy consumption characteristics across the cores. A load balancer can then divide that single filter in response to changing processor demand to help keep the frequencies down between the cores.

Our power model is also useful to investigate idle system power usage. This type of analysis can be useful to system architects and designers, in order to identify important areas of improvement. For example, on the Jetson-TK1, base power is the most substantial power usage component with approximately 1.2 W. However, most of the components integrated on the Jetson-TK1, with the exception of the network controller, is not used and would be cut off in an actual implementation. On the T-17, which was a minimalistic development board essentially containing only the Tegra K1 SoC, the base power component was estimated to be only 67 mW. Not considering base power, the most substantial power component is the core rail leakage at 340 mW. Little can be done about this cost, however, as the core rail must always be on and powered. There is support for a deep-sleep mode, where the core rail is powered down and only the RTC rail is powered, but we have not tested this on the Jetson-TK1. From a systems perspective, the deep-sleep mode should be enabled to conserve additional energy when the platform is entirely idle. However, entry and exit latencies to and from the deep-sleep mode must also be considered. Other important details that are exposed through our model is for example the memory clock, which is the second largest consumer of energy. At 204 MHz, the memory clock is estimated to draw 85 mW. This cost is impossible to avoid, because every memory cycle consumes energy whether the memory is in use or not. It is, however, possible to equip the platform with improved memory technology. LPDDR3 RAM, for example, can operate at lower supply voltages, which will positively affect idle clock power as well as

off-chip memory access costs. It is also possible to reduce the Tegra K1 memory clock to 12 MHz, but at this rate, the platform easily became unstable and crashed. We also investigated idle instruction cost. While this cost is very small, and estimated to be only 40 mW, we identified and optimised a critical code segment in the Tegra K1's activity monitor kernel module.

While frequency scaling is an important tool to tune system performance and energy consumption, some additional flexibility is given to programmers in terms of types of instructions and cache modifiers. On the Tegra K1's GPU, for example, single- and double-precision instructions are available, and newer Tegra SoCs also support half-precision (16-bit) floating point instructions. We have shown that using lower precision has significant gains in terms of both performance and energy consumption. Additionally, the Tegra K1's GPU allows the programmer to specify what cache hierarchy should be used to cache data accesses. Surprisingly, we found that caching in the GPU's L1 cache saved energy because of reduced off-chip memory traffic. This occurs because the GPU's L1 cache is not coherent with off-chip memory, and therefore, some signalling overhead is saved. These two optimisations reduced the energy-consumption of our DCT filter by 3.2 %. However, it is important to note that the actual achievable saving can be larger due to the high base power component and the high memory and GPU clock frequency in this experiment. Similarly, for the CPU, we found that NEON instructions have significant effects on both performance and energy consumption. NEON instructions accelerate workloads, such as the DCT, in terms of performance. Therefore, the QoS requirement of for example 25 FPS can be reached at much lower processor and memory frequencies. However, as we observed in these experiments, the compiler is already good at optimising CPU code with NEON instructions and consequently, this is an optimisation that may be easily achievable without writing the assembly code manually.

6.2 Open Issues and Future Work

There are many candidates for future research that can extend and augment the results presented here:

- *Building the model on new devices.* The power models presented in this thesis were all built on Tegra K1-enabled devices. While this was necessary to achieve a fine-grained, accurate and high-quality model, it limited us to this system. A definite next step is to build power models for other types of systems. The Tegra K1's successor, Tegra X1, for example, is a good candidate. This is because the architecture is similar to the Tegra K1, and therefore it is possible that some or all of the model predictors can be re-used. Extending the method to other types of systems, such as the Samsung Exynos and Snapdragon SoCs, or even stationary compute environments such as server farms, can be more challenging depending on the availability to measure hardware activity, as well as openness of the platform in terms of for example power management and power distribution.

- *Accelerating Model Training Time.* Our power model takes less than 24 hours to generate. In future work, and especially for different SoCs, a relevant topic is to reduce this training time and study the effects that reduced training time has

on model accuracy. This can for example be achieved by reducing the amount of frequencies in the training runs, where it is essentially only needed to trigger variation in rail voltages. Also, it may not be necessary to test all combinations of online CPU cores.

- *Self-constructive, high-precision modelling.* Research has shown that all devices are different in terms of physical energy consumption characteristics. Models that are built by the same device that they are for, can solve this issue. Future research should attempt to use our method to build power models for Tegra K1-enabled systems. Production devices with power measurement capability are desirable. Some platform-specific information, such as rail voltages, is here a challenge, but it may be possible to use hard-coded frequency-voltage tables in the Linux kernel as reference voltages.

- *Temperature-aware power modelling.* In the course of the work in this thesis, we were unfortunately not able to include temperature in our model. As a result the models presented here have been built under constant cooling with chip temperatures between 20 to 30 degrees. Temperature is, nonetheless, an important factor that impacts both static and dynamic power usage. The Tegra K1 is equipped with temperature sensors on the core rail, HP cluster, GPU, memory controller and clock generators. A research challenge in this aspect is to include temperature in the equations for power in all the various rails, as well as controlling the temperature through our model training benchmarks and methodology.

- *Code-path power modelling.* The Tegra K1's CPU does not have HPCs to measure the type and number of instructions executed on the various cores, such as the GPU. This complicates modelling because the average capacitive load per instruction must be re-estimated per workload. However, there are also advantages of using a single (global) instruction counter. There is only a fixed number of HPC slots on each CPU core. This means that the HPCs must be sampled when there are not enough slots to count all the HPCs. In future work, an interesting research topic would be whether average capacitive instruction loads can be estimated in individual code paths, for example between branching points in kernel and application code.

- *Effects of different instructions.* Newer Tegra SoCs, such as the Tegra X1, include half-precision (16-bit) floating point data types. In Section 5.4.1 we saw that using shorter datatypes has positive effects on the Tegra K1 GPU's power usage and performance. A possible topic for future research is to investigate the effects of this new datatype.

- *Automatic Load Balancing.* In a complete video processing chain, a number of filters will be active at any point in time, processing a stream of video. A challenge with these filters is that they must be distributed to the appropriate processor to provide an adequate framerate, while consuming as little energy as possible. As we have seen in this thesis, the heterogeneous cores on the Tegra K1 excel at processing different workloads. For example, the CPU is best at processing the MVS and Huffman filters, and the GPU is most suitable for the computationally heavy DCT. A possible topic for future work is to design, implement and evaluate a system that can detect which

parts of a video processing chain should be offloaded to which processors, for example in response to changing demands for system resources. Furthermore, we have seen that filters with good performance across heterogeneous processors, such as the DCT filter, can be balanced across the Tegra K1's processors to reduce operating frequency and energy consumption of the filters. A load balancer can potentially help mitigate the negative effects of high processor frequencies, by offloading filters such as the DCT between heterogeneous cores.

- *Investigating the CPI value.* In this thesis, we have seen that the Tegra K1's processors are more energy-inefficient at some frequencies due to increased CPI value. The CPI value is a measure of how many processor cycles are necessary to execute each instruction. Therefore, the higher the CPI value, the more cycle energy is consumed to perform the same number of instructions. We believe that the CPI value is a result of reduced processor pipeline inefficiency when the memory bandwidth is too low. As such, an interesting future work is to investigate whether the CPI value increases if more video filtering operations are operating concurrently, increasing the memory pressure. In the context of DVFS algorithms, the CPI value is also an important metric because the memory and processor frequency areas where CPI increases, should be avoided. In fact, for the Tegra K1, this is an important observation that can potentially be used to increase the effectiveness of DVFS algorithms.

Bibliography

[1] ARM. Cortex-A15 Technical Reference Manual. Revision r4p0. `http://infocenter.arm.com/help/topic/com.arm.doc.ddi0438i/DDI0438I_cortex_a15_r4p0_trm.pdf` [Online. Last accessed: August 2016], 2013.

[2] ARM. ARM v7-AR Architecture Reference Manual. `https://silver.arm.com/download/download.tm?pv=1603196` [Online. Last accessed: August 2016], 2014.

[3] Todd Austin, Eric Larson, and Dan Ernest. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer*, 35(2):59–67, 2002.

[4] Robert Basmadjian and Hermann de Meer. Evaluating and Modeling Power Consumption of Multi-Core Processors. In *Proceedings of Future Energy Systems: Where Energy, Computing and Communication Meet (e-Energy)*, pages 1–10. IEEE, 2012.

[5] Luca Benini, Robin Hodgson, and Polly Siegel. System-level Power Estimation and Optimization. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 173–178. ACM, 1998.

[6] Daniele Bortolotti, Andrea Bartolini, Mauro Mangia, Riccardo Rovatti, Gianluca Setti, and Luca Benini. Energy-aware bio-signal compressed sensing reconstruction: Focuss on the wbsn-gateway. In *Proceedings of the International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*, pages 120–126, Sept 2015.

[7] Carlo Brandolese, William Fornaciari, Fabio Salice, and Donatella Sciuto. Energy Estimation for 32-bit Microprocessors. In *Proceedings of the International Workshop on Hardware/Software Codesign (CODES)*, pages 24–28. ACM, 2000.

[8] David Brooks, Vivek Tiwari, and Margaret Martonosi. Wattch: A Framework for Architectural-Level Power Analysis and Optimizations. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 83–94. ACM, 2000.

[9] Juan-Carlos Cano, José-Manuel Cano, Eva González, Carlos Calafate, and Pietro Manzoni. Evaluation of the Energetic Impact of Bluetooth Low-Power Modes for Ubiquitous Computing Applications. In *Proceedings of the International Workshop on Performance Evaluation of Wireless Ad-Hoc, Sensor and Ubiquitous Networks (PE-WASUN)*, pages 1–8. ACM, 2006.

[10] Aaron Carroll and Gernot Heiser. An Analysis of Power Consumption in a Smartphone. In *Proceedings of the Annual Technical Conference (ATC)*. USENIX, 2010.

[11] Andrea Castagnetti, Cécile Belleudy, Sebastien Bilavarn, and Michel Auguin. Power Consumption Modeling for DVFS Exploitation. In *Proceedings of the Euromicro Conference on Digital System Design: Architectures, Methods and Tools (DSD)*, pages 579–586. IEEE, 2010.

[12] Kwang Ting Cheng and Yi Chu Wang. Using Mobile GPU for General-Purpose Computing a Case Study of Face Recognition on Smartphones. In *Proceedings of the International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 54–57. IEEE, 2011.

[13] Cisco. White paper: Cisco VNI Forecast and Methodology, 2015–2020. `http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/complete-white-paper-c11-481360.html` [Online. Last accessed: August 2016], 2016.

[14] Peter J. Denning, Douglas E. Comer, David Gries, Michael C. Mulder, Allen Tucker, A. Joe Turner, and Paul R. Young. Computing as a Discipline. *Communications of the ACM*, 32(I):1–11, 1989.

[15] Mian Dong and Lin Zhong. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *Proceedings of the International Conference on Mobile Systems, Applications and Services (MobiSys)*, pages 335–348. ACM, 2011.

[16] Prabal Dutta, Mark Feldmeier, Joseph Paradiso, and David Culler. Energy Metering for Free: Augmenting Switching Regulators for Real-Time Monitoring. In *Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN)*, pages 283–294. IEEE, 2008.

[17] Laura Marie Feeney and Martin Nilsson. Investigating the Energy Consumption of a Wireless Network Interface in an Ad Hoc Networking Environment. In *Proceedings of the Conference of the IEEE Computer and Communication Societies (INFOCOM)*, volume 3, pages 1548–1557. IEEE, 2001.

[18] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. Time and Energy Profiling in Production Sensor Networks with Quanto. In *Proceedings of the Conference on Operating Systems Design and Implementation (OSDI)*, pages 323–338. USENIX, 2008.

[19] Rong Ge, Ryan Vogt, Jahangir Majumder, Arif Alam, Martin Burtscher, and Ziliang Zong. Effects of Dynamic Voltage and Frequency Scaling on a K20 GPU. In *Proceedings of the International Conference on Parallel Processing (ICPP)*, pages 826–833. IEEE, 2013.

[20] Anton Hergenroder and Jochen Furthmuller. On Energy Measurement Methods in Wireless Networks. In *Proceedings of the International Conference on Communications (ICC)*, pages 6268–6272. IEEE, 2012.

[21] Sunpyo Hong and Hyesoon Kim. An Integrated GPU Power and Performance Model. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, pages 280–289. ACM, 2010.

[22] Miaoqing Huang and Chenggang Lai. Accelerating Applications Using GPUs on Embedded Systems and Mobile Devices. In *Proceedings of the International Conference on High Performance Computing and Communications (HPCC) and the International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 1031–1038. IEEE, 2013.

[23] D. A. Huffman. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers (IRE)*, 40(9):1098–1101, 1952.

[24] Ikhwan Lee, Hyunsuk Kim, Peng Yang, Sungjoo Yoo, Eui-Young Chung, Kyu-Myung Choi, Jeong-Taek Kong, and Soo-Kwan Eo. PowerViP: SoC Power Estimation Framework at Transaction Level. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 551–558, 2006.

[25] Canturk Isci and Margaret Martonosi. Runtime Power Monitoring in High-End Processors: Methodology and Empirical Data. In *Proceedings of the International Symposium on Microarchitecture (MICRO)*. IEEE, 2003.

[26] Y. Jiao, H. Lin, P. Balaji, and W. Feng. Power and Performance Characterization of Computational Kernels on the GPU. In *Proceedings of Green Computing and Communications (GreenCom) & the International Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 221–228. IEEE, 2010.

[27] Russ Joseph and Margaret Martonosi. Run-Time Power Estimation in High Performance Microprocessors. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 135–140. ACM, 2001.

[28] Wonwoo Jung, Chulkoo Kang, Chanmin Yoon, Donwon Kim, and Hojung Cha. DevScope: a Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 353–362. ACM, 2012.

[29] Praveen Kalla, Jörg Henkel, and Xiaobo Sharon Hu. SEA: Fast Power Estimation for Micro-Architectures. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 600–605. ACM, 2003.

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

[31] Sheayun Lee, Andreas Ermedahl, Sang Lyul Min, and Naehyuck Chang. An Accurate Instruction-Level Energy Consumption Model for Embedded RISC Processors. In *Procedings of the SIGPLAN Workshop on Languages, Compilers and Tools for Embedded Systems (LCTES)*, pages 1–10. ACM, 2001.

[32] Woojoo Lee, Yanzhi Wang, Donghwa Shin, Naehyuck Chang, and Massoud Pedram. Power Conversion Efficiency Characterization and Optimization for Smartphones. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 103–108. ACM, 2012.

[33] Jingwen Leng, Tayler Hetherington, Ahmed Eltantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. GPUWattch: Enabling Energy Optimizations in GPGPUs. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*, volume 41, pages 487–498. ACM, 2013.

[34] Niu Limin, Tan Xiaobin, and Yin Baoqun. Estimation of System Power Consumption on Mobile Computing Devices. In *Proceedings of the International Conference on Computational Intelligence and Security (CIS)*, pages 1058–1061. IEEE, 2007.

[35] Miguel Bordallo López, Henri Nykänen, Jari Hannuksela, Olli Silvén, and Markku Vehviläinen. Accelerating Image Recognition on Mobile Devices Using GPGPU. In *Proceedings of Parallel Processing for Imaging Applications (PPIA)*. SPIE, 2011.

[36] John C McCullough, Yuvraj Agarwal, Jaideep Chandrashekar, Sathyanarayan Kuppuswamy, Alex C Snoeren, and Rajesh K Gupta. Evaluating the Effectiveness of Model-Based Power Characterization. In *Proceedings of the Annual Technical Conference (ATC)*. USENIX, 2011.

[37] Gordon E. Moore. Cramming more components onto integrated circuits. *IEEE Electronics*, 38(8):114–117, 1965.

[38] Luca Negri, Jan Beutel, and Matthias Dyer. The Power Consumption of Bluetooth Scatternets. In *Proceedings of the Consumer Communications and Networking Conference (CCNC)*, volume 1, pages 519–523. IEEE, 2006.

[39] Luca Negri, Mariagiovanna Sami, David Macii, and Alessandra Terranegra. FSM-Based Power Modeling of Wireless Protocols: the Case of Bluetooth. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 369–374. ACM, 2004.

[40] Luca Negri, Mariagiovanna Sami, Que Dung Tran, and Davide Zanetti. Flexible Power Modeling for Wireless Systems: Power Modeling and Optimization of Two Bluetooth Implementations. In *Proceedings of the International Symposium on a World of Wireless Mobile and Multimedia Networks (WoWMoM)*, pages 408–416. IEEE, 2005.

[41] NVIDIA. CUDA Binary Utilities. `http://docs.nvidia.com/cuda/cuda-binary-utilities/` [Online. Last accessed: August 2016].

[42] NVIDIA. Next Generation CUDA Compute Architecture: Kepler GK110. `http://international.download.nvidia.com/pdf/kepler/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf` [Online. Last accessed: August 2016], 2012.

[43] NVIDIA. Tegra K1 - A New Era in Mobile Computing. `http://www.nvidia.com/content/PDF/tegra_white_papers/tegra-K1-whitepaper.pdf` [Online. Last accessed: August 2016], 2014.

[44] NVIDIA. Tegra K1 Technical Reference Manual. `https://developer.nvidia.com/embedded/downloads` [Online. Last accessed: August 2016], 2014.

[45] Venkatesh Pallipadi and Alexey Starikovskiy. The Ondemand Governor: Past, Present and Future. In *Proc of the Linux Symposium*, pages 215–230, 2006.

[46] Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-Grained Power Modeling for Smartphones Using System Call Tracing. In *Proceedings of the Conference on Computer Systems (EuroSys)*, pages 153–168. ACM, 2011.

[47] Anuj Pathania, Alexandru Eugen Irimiea, Alok Prakash, and Tulika Mitra. Power-Performance Modelling of Mobile Gaming Workloads on Heterogeneous MPSoCs. In *Proceedings of the Annual Design Automation Conference (DAC)*. ACM, 2015.

[48] Anuj Pathania, Qing Jiao, Alok Prakash, and Tulika Mitra. Integrated CPU-GPU Power Management for 3D Mobile Games. In *Proceedings of the Design Automation Conference (DAC)*, pages 1–6. IEEE, 2014.

[49] Martin Peres. Reverse Engineering Power Management on NVIDIA GPUs - A Detailed Overview. Technical report.

[50] Gian Paolo Perrucci, Frank H P Fitzek, and Jörg Widmer. Survey on Energy Consumption Entities on the Smartphone Platform. In *Proceedings of the Vehicular Technology Conference (VTC)*, pages 1–6. IEEE, 2011.

[51] Mihai Pricopi, Thannirmalai Somu Muthukaruppan, Vanchinathan Venkataramani, Tulika Mitra, and Sanjay Vishin. Power-Performance Modeling on Asymmetric Multi-Cores. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*. IEEE, 2013.

[52] Enrico Rantala, Arto Karppanen, Seppo Granlund, and Pasi Sarolahti. Modeling Energy Efficiency in Wireless Internet Communication. In *Proceedings of the Workshop on Networking, Systems and Applications for Mobile Handhelds (MobiHeld)*, pages 67–68. ACM, 2009.

[53] A. Rice and S. Hay. Decomposing Power Measurements for Mobile Devices. In *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*, pages 70–78. IEEE, 2010.

[54] Blaine Rister, Guohui Wang, Michael Wu, and Joseph R. Cavallaro. A Fast and Efficient SIFT Detector Using the Mobile GPU. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2674–2678. IEEE, 2013.

[55] James F Rohan, Maksudul Hasan, Sanjay Patil, Declan P Casey, and Tomás Clancy. Energy Storage : Battery Materials and Architectures at the Nanoscale. In *ICT - Energy - Concepts Towards Zero - Power Information and Communication Technology*, pages 107–138. 2014.

[56] Mariagiovanna Sami, Donatella Sciuto, Cristina Silvano, and Vittorio Zaccaria. Instruction-Level Power Estimation for Embedded VLIW Cores. In *Proceedings of the International Workshop on Hardware/Software Codesign (CODES)*, pages 34–38. ACM, 2000.

[57] Øyvind Skjøld. Investigating the Impact of Processor Gating Techniques on Energy Consumption Modelling. Master's thesis, University of Oslo, 2015.

[58] Kristoffer Robin Stokke. Energy and Performance Optimisation of a Simple Video Encoder on the Jetson TK1, 2015. Poster presentation at the GPU Technology Conference (GTC), NVIDIA.

[59] Kristoffer Robin Stokke. A High-Precision Power Model for the Tegra K1 CPU, GPU and RAM, 2016. Poster presentation at the GPU Technology Conference (GTC), NVIDIA.

[60] Kristoffer Robin Stokke. A High-Precision Power Model for the Tegra K1 CPU, GPU and RAM, 2016. Talk and presentation at the GPU Technology Conference (GTC), NVIDIA.

[61] Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC. In *Proceedings of the International Workshop on Mobile Video (MoViD)*, pages 15–16. ACM, 2015.

[62] Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, and Pål Halvorsen. Energy Efficient Video Encoding Using the Tegra K1 Mobile Processor. In *Proceedings of the Multimedia Systems Conference (MMSys)*, pages 20–23, 2015.

[63] Kristoffer Robin Stokke, Håkon Kvale Stensland, Pål Halvorsen, and Carsten Griwodz. Why Race-to-Finish is Energy-Inefficient for Continuous Multimedia Workloads. In *Proceedings of the International Symposium on Embedded Multicore/Manycore Systems-on-Chip (MCSoC)*. IEEE, 2015.

[64] Kristoffer Robin Stokke, Håkon Kvale Stensland, Pål Halvorsen, and Carsten Griwodz. A High-Precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads. In *Proceedings of the International Conference on Multimedia Systems (MMSys)*. ACM, 2016.

[65] Kristoffer Robin Stokke, Håkon Kvale Stensland, Pål Halvorsen, and Carsten Griwodz. High-Precision Power Modelling of the Tegra K1 Variable SMP Processor Architecture. In *Proceedings of the International Symposium on Embedded Multicore/Manycore Systems-on-Chip (MCSoC)*. IEEE, 2016.

[66] Texas Instruments. INA219 Datasheet. `http://www.ti.com/lit/ds/symlink/ina219.pdf` [Online. Last accessed: August 2016], 2015.

[67] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: a first step towards software power minimization. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2(4):437–445, Dec 1994.

[68] Gergely Vass and Tamás Perlaki. Applying and Removing Lens Distortion in Post Production. In *Proceedings of the Hungarian Conference on Computer Graphics and Geometry*, pages 9–16, 2009.

[69] Jarkko M. Vatjus-Anttila, Timo Koskela, and Seamus Hickey. Power Consumption Model of a Mobile GPU Based on Rendering Complexity. In *Proceedings of the International Conference on Next Generation Mobile Apps, Services and Technologies (NGMAST)*, pages 210–215. IEEE, 2013.

[70] Yu Xiao, Rijubrata Bhaumik, Zhirong Yang, Matti Siekkinen, Petri Savolainen, and Antti Yla-Jaaski. A System-Level Model for Runtime Power Estimation on Mobile Devices. In *Proceedings of the International Conference on Green Computing and Communications (GreenCom) & the International Conference on Cyber, Physical and Social Computing (CPSCom)*, pages 27–34. IEEE, 2010.

[71] Fengyuan Xu, Yunxin Liu, Qun Li, and Yongguang Zhang. V-edge: Fast Self-Constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics. In *Proceedings of the Symposium on Networked Systems Design and Implementation (NDSI)*, pages 43–55. USENIX, 2013.

[72] W. Ye, N. Vijaykrishnan, M. Kandemir, and M.J. Irwin. The Design and Use of SimplePower: A Cycle-Accurate Energy Estimation Tool. In *Proceedings of the Annual Design Automation Conference (DAC)*, pages 340–345. ACM, 2000.

[73] Yi-Chu Wang and Kwang-Ting Cheng. Energy-Optimized Mapping of Application to Smartphone Platform - A Case Study of Mobile Face Recognition. In *Proceedings of the IEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR)*, pages 84–89. IEEE, 2011.

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

[75] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proceedings of the International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 105–114. ACM, 2010.

[76] Shan Zhu and Kai-Kuang Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2):287–290, Feb 2000.

Part II

Research Papers

Paper I: Energy Efficient Video Encoding Using the Tegra K1 Mobile Processor

Title: Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC [62].

Authors: K. R. Stokke, H. K. Stensland, C. Griwodz and P. Halvorsen.

Abstract: Energy consumption is an important concern for mobile devices, where the evolution in battery storage capacity has not followed the power usage requirements of modern hardware. However, innovative and flexible hardware platforms give developers better means of optimising the energy consumption of their software. For example, the Tegra K1 System-on-Chip (SoC) offers two CPU clusters, GPU offloading, frequency scaling and other mechanisms to control the power and performance of applications. In this demonstration, the scenario is live video encoding, and participants can experiment with power usage and performance using the Tegra K1s hardware capabilities. A popular power-saving approach is a race to sleep strategy where the highest CPU frequency is used while the CPU has work to do, and then the CPU is put to sleep. Our own experiments indicate that an energy reduction of 28 % can be achieved by running the video encoder on the lowest CPU frequency at which the platform achieves an encoding framerate equal to the minimum framerate of 25 Frames Per Second (FPS).

Lessons learned: In this paper, we presented a live video encoding demonstration system on the Tegra K1. The encoder performed many common video processing operations, such as the DCT, iDCT, MVS and Huffman encoding. Attending participants could fine-tune CPU and GPU clock frequencies, CPU cluster and the number of active CPU cores on the Tegra K1. The encoder could also be partly offloaded to the GPU, and the current framerate and energy consumption per frame were displayed live to the participant. Through our demonstration, we learned that a good heuristic to minimise the energy consumption per frame is to reduce the processor frequencies, such that application requirements are met. This factor was one contributing factor to the later frequency variation experiments.

Author's contibutions: Stokke designed and implemented the video encoding system, and built power measurement circuitry using the INA219 rail sensor. He wrote

NEON-accelerated versions of the DCT, iDCT and MVS encoding blocks, a user-space webcam driver, an improved I2C driver for the INA219, as well as a communication protocol with a dedicated decoding and playback computer. Stokke was also the main author of this paper.

Published: Proceedings of the 6th Multimedia Systems Conference (MMSys), pages 81-84, Portland (Oregon). ACM, 2015.

Energy Efficient Video Encoding Using the Tegra K1 Mobile Processor

Kristoffer Robin Stokke, Håkon Kvale Stensland, Carsten Griwodz, Pål Halvorsen

Simula Research Laboratory & University of Oslo, Norway
{krisrst, haakonks, griff, paalh}@ifi.uio.no

ABSTRACT

Energy consumption is an important concern for mobile devices, where the evolution in battery storage capacity has not followed the power usage requirements of modern hardware. However, innovative and flexible hardware platforms give developers better means of optimising the energy consumption of their software. For example, the Tegra K1 System-on-Chip (SoC) offers two CPU clusters, GPU offloading, frequency scaling and other mechanisms to control the power and performance of applications. In this demonstration, the scenario is live video encoding, and participants can experiment with power usage and performance using the Tegra K1's hardware capabilities. A popular power-saving approach is a "race to sleep" strategy where the highest CPU frequency is used while the CPU has work to do, and then the CPU is put to sleep. Our own experiments indicate that an energy reduction of 28 % can be achieved by running the video encoder on the lowest CPU frequency at which the platform achieves an encoding frame rate equal to the minimum frame rate of 25 Frames Per Second (FPS).

Categories and Subject Descriptors

C.1.3 [**Other Architecture Styles**]: Heterogeneous (hybrid) systems;
C.1.4 [**Parallel Architectures**]: Mobile processors;
I.4.0 [**General**]: Image processing software;
J.2.0 [**Physical Sciences and Engineering**]: Electronics

General Terms

Experimentation, Measurement

Keywords

Demonstration, video encoding, real-time, energy, power

MMSys'15, Mar 18-20, 2015, Portland, OR, USA
ACM 978-1-4503-3351-1/15/03.
http://dx.doi.org/10.1145/2713168.2713186.

1. INTRODUCTION

Energy consumption is an important aspect for the usability of mobile devices, where the evolution in battery technology has not kept pace with the increasing power usage of the devices [4]. This has not gone unnoticed by hardware producers, who are developing more power-efficient and flexible SoC architectures. For example, NVIDIA's Tegra K1 [6] is a highly flexible mobile multicore SoC equipped with one low-performance, low-power CPU cluster, one high-performance, high power CPU cluster as well as a GPU with 192 CUDA cores. The platform gives full control of these capabilities to the developer. For example, heavy, parallel computing operations can be offloaded to the GPU while executing on the power efficient CPU cluster to save energy. The challenge is to understand how these choices impact runtime performance and energy consumption, and whether any energy can actually be saved depending on the choices of the programmer.

In this demonstration, we let the participant tweak the hardware settings of a Tegra K1 to minimise the energy consumption of live video encoding. The challenge is to meet a performance requirement of 25 FPS. The participant is free to adjust the CPU and GPU core frequency, select the active CPU clusters and control the number of active cores (see Table 1). The encoder, called Codec 63 (C63), is a highly simplified MPEG-inspired codec created for teaching purposes (see Section 2.2 for further details). The effects of the participant's hardware settings in terms of the achieved frame rate, power usage and CPU/GPU frequency are presented over time on a dedicated laptop. The laptop also handles live video decoding (see Figure 1). Although C63 cannot compete with the performance of a dedicated hardware encoder, the point is here to study the effects of the

Name	Description
CPU Frequency	CPU operating frequency.
CPU Cluster	Encode using the high-power or low-power CPU cluster.
CPU Cores	Controls the number of active cores to be used while actively encoding.
GPU Frequency	GPU operating frequency.
GPU Offloading	Offload nothing, some or all frames to the GPU.

Table 1: Parameters that can be set by the participant. See Section 2.2 for further description.

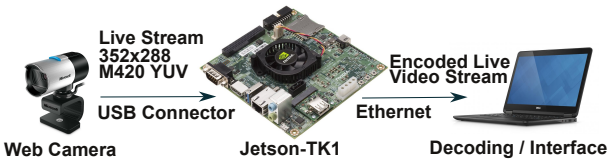


Figure 1: Demonstration set-up.

Tegra K1's hardware capabilities, and not the efficiency of the workload.

C63 can easily encode a frame in less than 40 ms, achieving a frame rate of at least 25 FPS. Although research shows that a frame rate of 12 FPS may be satisfactory with current mobile phones' screen size, we use 25 FPS as the minimum for the sake of display quality on the decoding computer. The challenge is for the participant to minimise the consumption of energy while meeting this frame rate requirement. For example, when we encoded 300 frames at 25 FPS in real-time, almost 30% of the energy was saved by minimising CPU frequency instead of running the processor on full speed to maximise the sleep period before the next frame (a "race to sleep" heuristic, see Figure 2).

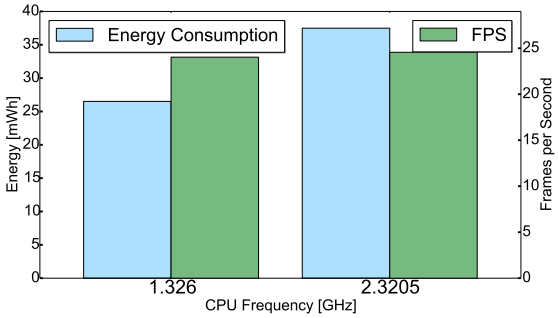


Figure 2: Our own experiments indicate almost 30% energy saving when minimising CPU frequency such that a target frame rate of 25 FPS is achieved.

2. SYSTEM OVERVIEW

Our demonstration is composed of three units (see Figure 1). A web camera is continuously transmitting a raw video stream to the encoder platform over a USB interface. The encoder platform is a Jetson-TK1 mobile development kit [5] featuring a Tegra K1 SoC. The Tegra K1 is continuously encoding the live-stream using the C63 encoder. As seen in Figure 3, we have also added a power measurement sensor that can log the energy consumption of the Jetson-TK1. Finally, the encoded stream is sent to a dedicated decoding laptop, which fulfils two tasks. First, it decodes the stream and plays it back live. Second, it displays the achieved performance- and power-related parameters such as the power usage, energy per frame and frame rate over time. The participant may use the decoding laptop to control the parameters in Table 1. Throughout the rest of this section, we will explain the system set-up in more detail.

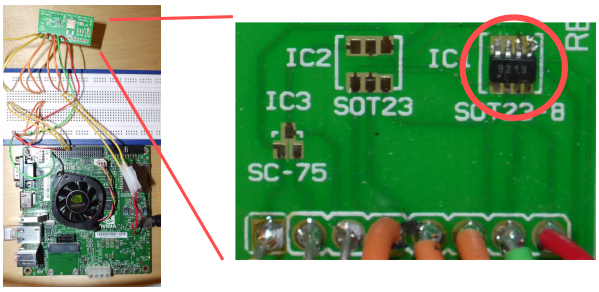


Figure 3: The figure shows our power measurement extension (INA219, circled) on the Jetson-TK1.

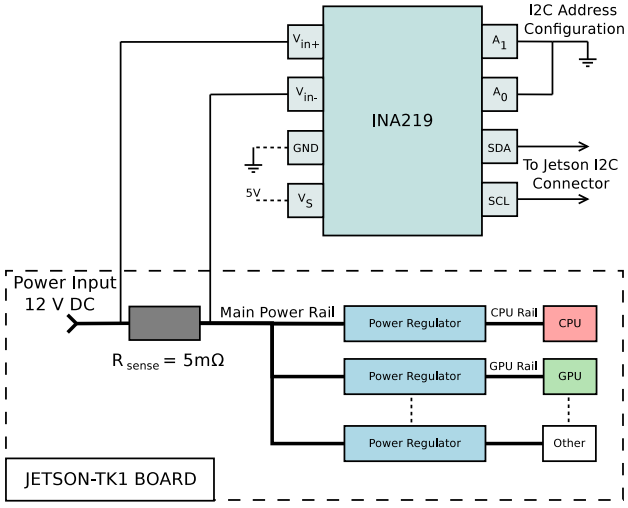


Figure 4: The INA219 measures total platform power usage over the Jetson's main power rail.

2.1 Video Encoding Platform

We use a Tegra K1 multiprocessor SoC as our encoding platform (see Figure 3). This processor is especially interesting in terms of energy consumption, because it provides many hardware capabilities that can be used for power optimisation. The Tegra K1, as well as SoCs with similar capabilities, have been implemented in real devices such as NVIDIA's SHIELD tablet. The Tegra K1's capabilities are as follows:

- **CPU**: Two clusters in "4+1" core configuration[7, 8].
  - One high performance, high power (HP) cluster with four Cortex-A15 ARM cores. Three of the cores can be individually shut down.
  - One low performance, low power (LP) cluster with a single Cortex-A15 ARM core.
  - Hardware-supported migration of OS and applications between the clusters.
  - A 128-bit NEON single instruction, multiple data (SIMD) instruction set tailored for multimedia workloads.

  - 22 configurable CPU frequencies (between 51 MHz and 2.32 GHz).

- **GPU**: 192 programmable CUDA-cores based on the Kepler-architecture on-chip.
  - 15 configurable GPU frequencies (between 72 and 852 MHz).

The participant is free to change the CPU and GPU frequencies, as well as to enable and disable cores and switch CPU clusters. The relationship between the core frequency and power is given by the Dynamic Voltage and Frequency Scaling (DVFS) formula and is especially useful here [2]:

$$P_{core} = \alpha C V_{core}^2 f_{core} \quad (1)$$

In Equation 1, α is the core utilisation level, C is the core switching capacitance, V_{core} is the core voltage and f_{core} is the core frequency. The DVFS formula shows that higher performance in terms of processor frequency can be traded for increased power consumption, allowing fine-grained tuning of power and performance characteristics depending on application requirements.

The Jetson-TK1 platform is not pre-equipped with any power measurement sensors. Consequently, we have equipped our platform with an INA219 power measurement sensor [9] (see Figures 3 and 4) and developed a Linux kernel module that provides access to the device through the sysfs filesystem. The INA219 works by measuring the voltage drop, U_{sense}, over a sense resistor connected in series with the main power rail of the Jetson-TK1. The electrical resistance of the sense resistor, R_{sense}, must be very small to avoid affecting the main circuitry ($R_{sense} << R_{jetson}$). The INA219 amplifies the voltage drop U_{sense}, converts it to the digital domain, and calculates the board's current drain I_{jetson} as follows:

$$I_{jetson} = \frac{U_{sense}}{R_{sense}} \quad (2)$$

The power consumption, P_{jetson}, at any time, is given by:

$$P_{jetson} = I_{jetson} U_{jetson} \quad (3)$$

where U_{jetson} is the main rail bus voltage potential. The set-up is similar to that found in PowerScope [3], but more complex because the INA219 is a small, surface mounted device that requires a high degree of manual configuration. There is also no separate logging machine; power logging is done by the Tegra K1 itself, eliminating as much latency as possible between the sensor and the Jetson-TK1. The INA219 itself interfaces over an I2C bus adapter. Reading directly from the CPU, we achieve a rate of 4000 power measurement samples per second using this set-up.

2.2 Video Encoding Framework

As workload for our demo we use our own implementation of the C63 video encoder, which is suitable for parallelisation using the Jetson-TK1's hardware capabilities. C63 is capable of encoding raw YUV 4:2:0, and the encoding operations are similar to H.264 or Google's VP8. The operations are as follows (see Figure 5):

- *Motion vector search:* The encoder divides the current frame into a set of macroblocks, and attempts to estimate their displacement between frames.
- *Motion compensation:* The encoder compensates the current frame by removing information for each macroblock where a motion vector could be found. This reduces the amount of information that must be stored (only the vector needs to be stored).
- *Discrete Cosine Transform (DCT) and inverse DCT (iDCT):*. The DCT transforms each macroblock to the frequency domain. The inverse DCT transforms it back. We use the "Fast-DCT" algorithm [1] on the CPU.
- *Quantisation and de-quantisation:* Quantisation reduces the value of the highest frequency components of each macroblock, to which the human eye is less sensitive to. De-quantisation transforms it back.
- *Variable-length coding (VLC):* The last step writes each frame to persistent storage using variable length huffman encoding.

The demo implementation attempts to reach a frame rate of 25 FPS by offloading none, some or all parts of the video processing to the GPU ("CPU-only", "GPU-only" or "hybrid"). For example, for the hybrid scheme, Y and U frames are transmitted to and encoded on the GPU, while the V frame is encoded on the CPU (see Figure 5). Frame writing is interleaved, such that the next frame starts processing while the last is currently being written to network. If the single frame encoding time is below the requirement (for example 40 ms for 25 FPS), the frame rate is met, and the CPU (and GPU) idles. In this case, the encoder sleeps for the remaining duration. Figure 6 illustrates an example:

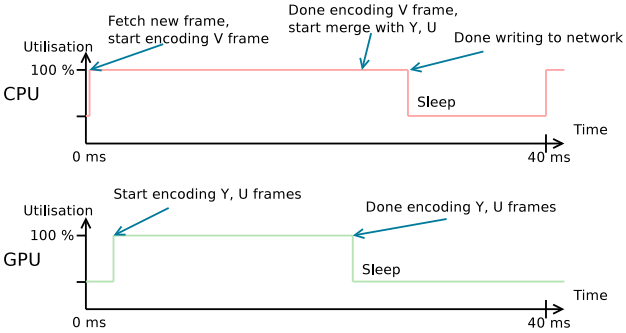


Figure 6: An illusation of single-frame encoding under the "hybrid" processing scheme.

1. The CPU fetches a raw YUV frame from the web camera.
2. The CPU distributes the Y and U frames to the GPU, starts processing the V frame, and starts writing the previously encoded frame to the network.
3. The GPU finishes processing before the CPU is done, and idles.
4. The CPU finishes its processing before the 40 ms deadline. The CPU now sleeps for the remaining duration.
5. The cycle continues for the next frame.

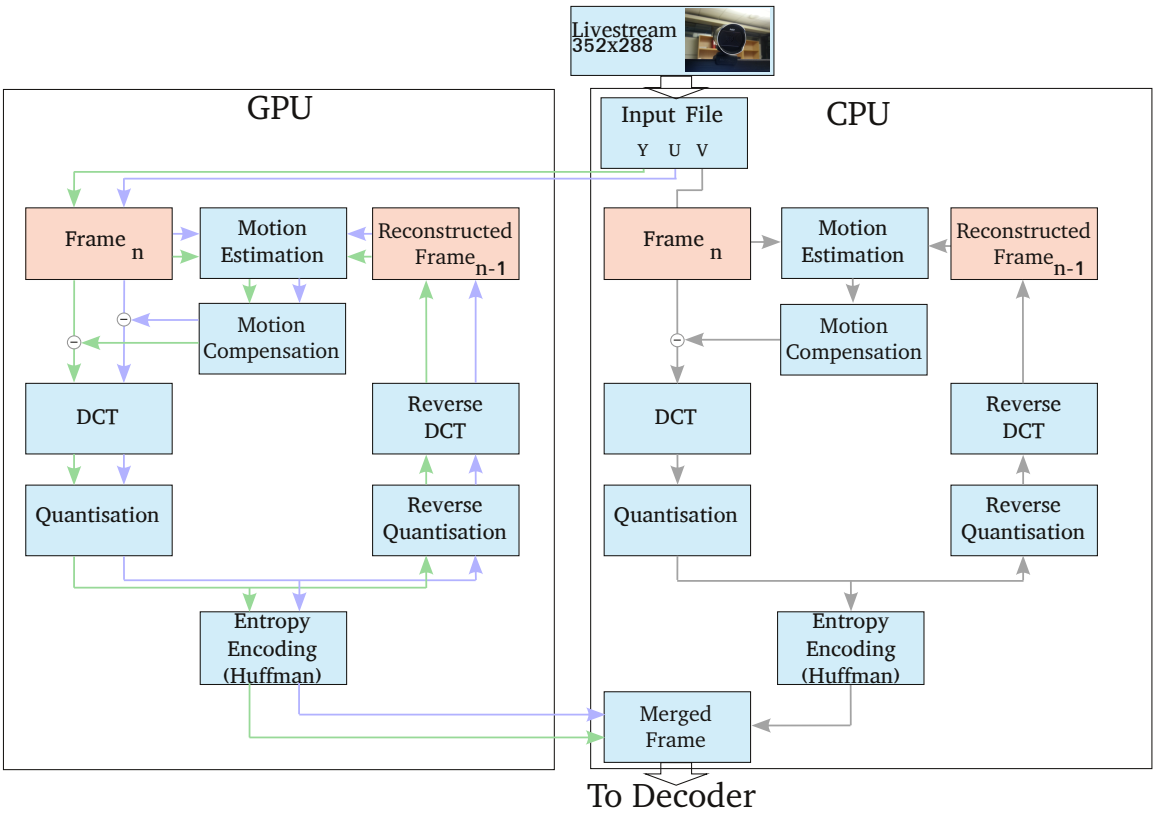


Figure 5: Our implementation of the C63 encoder operating in the "hybrid" processing scheme.

3. DEMONSTRATION

In this demo, we show how the energy consumption and performance of a video encoder is impacted by hardware and software configuration. Our demo supports runtime configuration of CPU and GPU core frequency, active CPU cluster, core shutdown and varying degrees of GPU offloading. The impact of these reconfigurations, that is power usage, frame rate, frequency and the decoded live video is displayed live to the participant as plots over time. Thus, the question is how energy-efficiently can you live-encode video?

4. REFERENCES

[1] L. V. Agostini, I. S. Silva, and S. Bampi. Pipelined fast 2d dct architecture for jpeg image compression. In *Integrated Circuits and Systems Design, 2001, 14th Symposium on.*, pages 226–231. IEEE, 2001.

[2] A. Castagnetti, C. Belleudy, S. Bilavarn, and M. Auguin. Power consumption modeling for dvfs exploitation. In *Digital System Design: Architectures, Methods and Tools (DSD), 2010 13th Euromicro Conference on*, pages 579–586. IEEE, 2010.

[3] J. Flinn and M. Satyanarayanan. Powerscope: A tool for profiling the energy usage of mobile applications. In *Mobile Computing Systems and Applications, 1999. Proceedings. WMCSA'99. Second IEEE Workshop on*, pages 2–10. IEEE, 1999.

[4] K. Lahiri, S. Dey, D. Panigrahi, and A. Raghunathan. Battery-driven system design: A new frontier in low power design. In *Proceedings of the 2002 Asia and South Pacific Design Automation Conference*, page 261. IEEE Computer Society, 2002.

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

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

[7] NVIDIA. Tegra K1 Whitepaper., 2014. www.nvidia.com/content/PDF/tegra_white_-papers/Tegra-K1-whitepaper-v1.0.pdf.

[8] NVIDIA. Variable SMP., 2014. www.nvidia.com/content/PDF/tegra_white_-papers/Variable-SMP-A-Multi-Core-CPU-Architecture-for-Low-Power-and-High-Performance.pdf.

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

Paper II: Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC

Title: Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC [61].

Authors: K. R. Stokke, H. K. Stensland, C. Griwodz and P. Halvorsen.

Abstract: Energy consumption is an important issue for mobile devices, as the technological development in battery technology has not kept pace with the power requirements of mobile hardware. In this paper, we use a video rotation filter to study the effects of CPU and GPU frequency scaling in terms of performance and energy. Our platform is the Tegra K1 mobile processor with a quad-core CPU and a CUDA-capable GPU. We find that most energy can be saved by minimising CPU frequency while meeting the filter's framerate requirement. Interestingly, the frequency scaling affects GPUs differently, where the best frequency is always moderately higher than the minimum which meets the framerate requirement. Using these heuristics, it is possible to save up to 10 % energy compared to the standard Linux frequency scaling algorithms, which use processor utilisation to adjust processor frequency.

Lessons learned: This paper was an extension to our demonstration paper, where we investigated the impact of frequency scaling on energy consumption and performance of a rotation filter. Comparing with the standard frequency scaling algorithms on the Tegra K1, we learned that we could reduce energy consumption with up to 11 % by minimising GPU and CPU frequencies, such that framerates are still met. However, in concluding this work, we realised that it is hard to argue why frequency minimisation is an effective strategy to reduce energy consumption. This is because it is essentially impossible to measure the individual power draw of the Tegra K1's processors.

Author's contributions: Stokke designed and implemented the video processing filters, ran and evaluated the experiments, and was the main author of this paper.

Published: Proceedings of the 7th Workshop on Mobile Video (MoVid), pages 15-16, Portland (Oregon). ACM, 2015.

Energy Efficient Continuous Multimedia Processing Using the Tegra K1 Mobile SoC

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

ABSTRACT

Energy consumption is an important issue for mobile devices, as the technological development in battery technology has not kept pace with the power requirements of mobile hardware. In this paper, we use a video rotation filter to study the effects of CPU and GPU frequency scaling in terms of performance and energy. Our platform is the Tegra K1 mobile processor with a quad-core CPU and a CUDA-capable GPU. We find that most energy can be saved by minimising CPU frequency while meeting the filter's framerate requirement. Interestingly, the frequency scaling affects GPUs differently, where the best frequency is always moderately higher than the minimum which meets the framerate requirement. Using these heuristics, it is possible to save up to 10 % energy compared to the standard Linux frequency scaling algorithms, which use processor utilisation to adjust processor frequency.

Categories and Subject Descriptors

C.1.3 [**Other Architecture Styles**]: Heterogeneous (hybrid) systems;
C.1.4 [**Parallel Architectures**]: Mobile processors;
J.2.0 [**Physical Sciences and Engineering**]: Electronics

Keywords

Multimedia, Tegra K1, CUDA, energy, performance, CPU-GPU frequency scaling

1. INTRODUCTION

Battery capacity is a severe limitation of modern mobile devices. Dynamic Voltage and Frequency Scaling (DVFS) [1] algorithms minimise power usage of CPUs and GPUs by lowering operating frequency, and is an effective way to save energy when full performance is not needed. There have been many proposals from the research community on how DVFS can be improved for mobile devices. However, these consider GPUs that do not support general purpose computing, such as CUDA [5, 4]. Those that consider DVFS for CUDA-capable GPUs [2] typically target high-performance applications found in data centres. We argue that common mobile applications are lighter processes where the goal is not to finish processing quickly, but to meet application-specific QoS constraints such as a specific framerate. In this paper, we consider a video processing filter implemented for both CPU and GPU, where the goal is to provide an acceptable video quality of 25 FPS. We study the impact that CPU and GPU frequency settings have on application performance and energy using a Tegra K1 mobile SoC, and find that 10 % energy can be saved by using workload specific frequency settings compared to the standard Linux DVFS algorithms.

MoVid'15, March 18-20 2015, Portland, OR, USA

ACM 978-1-4503-3353-5/15/03...$15.00
http://dx.doi.org/10.1145/2727040.2727044 ...$15.00.

2. SYSTEM SETUP

Our system is based on a Jetson-TK1 mobile development kit equipped with a custom power measurement sensor. The workload used in our study is an image rotation filter. The filter rotates each frame of a 25-FPS video stream by a continuously increasing angle. Its operations resemble that of a video stabiliser. The filter has been implemented for both the CPU and the GPU using CUDA. To study the effects of frequency scaling we disable the Linux DVFS algorithms and set CPU and GPU operating frequency manually while processing frames as follows:

1. While the CPU or the GPU is busy processing a frame, its respective frequency is set to a higher frequency which we vary throughout our experiments.

2. If the CPU or GPU processing ends before the frame deadline (40 ms for 25 FPS), CPU frequency is lowered to 204 MHz for the CPU and 72 MHz for the GPU. The CPU sleeps for the remaining time.

It is important to note that the performance of the filter never exceeds 25 FPS, but that it can be lower than this if the manually set processing frequencies are too low.

3. EXPERIMENTAL RESULTS

We run our experiments using our setup with three input resolutions (see Table 1). Each test is run ten times for each frequency, stopping if the encoding time is longer than 12 s. This is to equalise the energy consumption of idle

Resolution	Userspace P-SAV			Linux Governor ("ondemand")		Improvement
	Core	Max Frequency	Energy	Core	Energy	
352x288	CPU	204 MHz (CPU)	9.22 mWh	CPU	9.02 mWh	-2.2 %
640x480	CPU	304 MHz (CPU)	11.33 mWh	CPU	12.86 mWh	11.9 %
1920x1080	GPU	804 MHz (GPU)	26.00 mWh	GPU	29.16 mWh	10.8 %

Table 1: The most energy efficient frequency configurations compared with the best Linux DVFS algorithm.

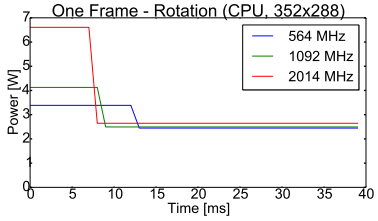


Figure 1: A single CPU frame encoding snapshot for different operating frequencies.

components from each run. The framerate and total energy usage for CPU- and GPU-execution can be seen in Figure 2. The best frequencies are marked with red. As expected, we see that higher frequencies increase the framerate. For the CPU, the best frequency is very close to the point where the 25 FPS requirement is reached. After this, increasing the operating frequency further only increases the total energy usage on the CPU. A likely reason for this observation is that a doubling in frequency effectively doubles the power usage of the processor [1], but does not reduce the frame encoding time by a corresponding amount, which can be seen in Figure 1.

The GPU experiments show a different trend than that of the CPU. For the mid-resolution video, 25 FPS is reached at 72 MHz. The best frequency is four frequency steps above, at 324 MHz, reducing the energy usage by 23 %. In other words, the frequency should not be minimised as for the CPU. The effect is similar, but not as clear, for the high- and low-resolution videos. However, we have run other types of video filters which confirm this observation. Unfortunately, we could not include them here due to space restrictions.

We also run the experiments with the standard Linux DVFS algorithms [3], where the CPU and GPU operating frequencies are automatically adjusted in response to changes in processor utilisation. There is only one GPU DVFS algorithm, but of the four CPU algorithms, the "ondemand" algorithm was consistently better. Compared to these, up to 10 % energy can be saved by using workload-specific frequencies (see Table 1).

4. CONCLUSION

In this paper, we study the impact of CPU and GPU frequency settings on a mobile processor. Our workload is a video rotation filter implemented for both the CPU and the GPU using CUDA. We find that up to 10 % energy can be saved by minimising CPU frequency such that a framerate of 25 FPS is met generally saves energy over the standard Linux DVFS algorithms. It is clear that the standard Linux DVFS governors, which change processor frequency in response to changes in utilisation [3], can be improved if QoS requirements such as framerate were considered. The effect of DVFS is different for the GPU, where the best frequency

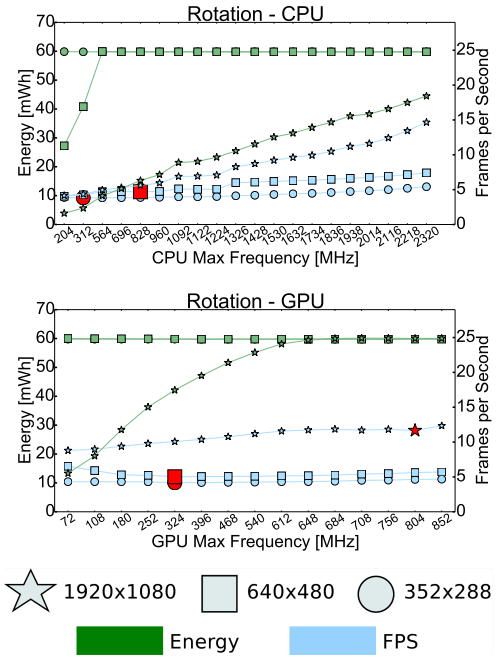


Figure 2: Frequency scaling experiments. The frequencies shown correspond to the one used while the CPU or GPU is actively processing a frame.

tends to lie moderately above the minimum which achieves 25 FPS. As this is work-in-progress, we do not yet know the exact reasons behind this observation. For future work, it would be interesting to see if existing GPU DVFS algorithm proposals [4, 5] can be improved for our platform.

5. REFERENCES

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

[2] R. Ge, R. Vogt, J. Majumder, A. Alam, M. Burtscher, and Z. Zong. Effects of dynamic voltage and frequency scaling on a k20 gpu. In *Proc. of ICPP*, pages 826–833, 2013.

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

[4] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated cpu-gpu power management for 3d mobile games. In *Proc. of the 51st Annual Design Automation Conference*, pages 1–6, 2014.

[5] D. You and K. Chung. Quality of service-aware dynamic voltage and frequency scaling for embedded gpus. *IEEE Computer Architecture Letters*, 2013.

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

Title: Why Race-to-Finish is Energy-Inefficient for Continuous Multimedia Workloads [63].

Authors: K. R. Stokke, H. K. Stensland, C. Griwodz and P. Halvorsen.

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.

Lessons learned: Motivated by the lack of insight into the power usage of our video processing filters under different processor frequencies, this paper was our first power modelling attempt. We built a CMOS-based model for the Tegra K1 to investigate the energy efficiency of different frequency scaling strategies. We learned many details that were important for modelling. For example, we found that the rail voltages on the Tegra K1 automatically increased to 1.0 V if the SoC was sufficiently cold. Also, we learned that the driver-reported rail voltages were not accurate enough. We therefore measured the rail voltages to build more accurate models. Our most important observation in the course of this work was that our model's prediction error varied with different CPU and memory frequencies. This phenomenon is rarely shown in related literature, where only average error is reported. Furthermore, we realised that, even with our careful method of attributing capacitive processor and memory cycle loads to applications, we could not build an accurate view of the power usage of the SoC. We believe this is because CMOS-based models implicitly assume that the increase in power usage as for example memory frequency is increased, is only due to increased dynamic power of that component.

Author's contributions: Stokke designed and evaluated the CMOS-based power model for the Tegra K1, as well as the experiments. He also implemented the different frequency scheduling strategies and was the main author of this paper.

Published: Proceedings of the 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), pages 57-64, Turin (Italy). IEEE, 2015.

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.

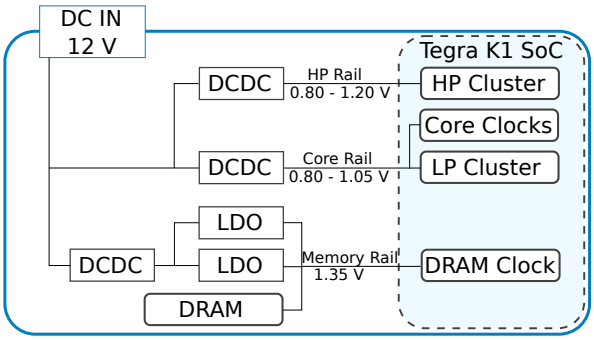


Fig. 1: Critical components of the Jetson-TK1 architecture.

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

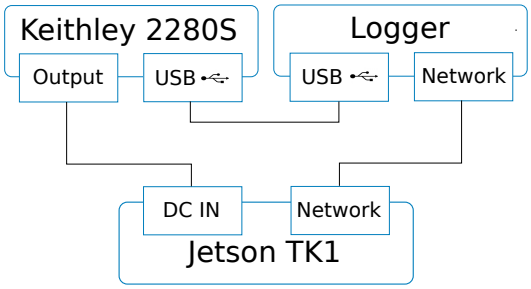


Fig. 2: Measurement Setup.

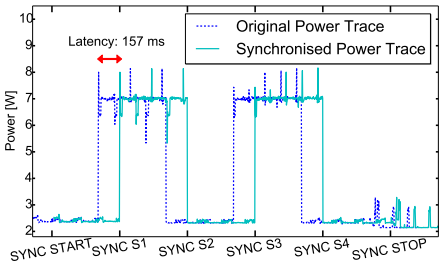


Fig. 3: A plot of a single synchronisation trace.

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

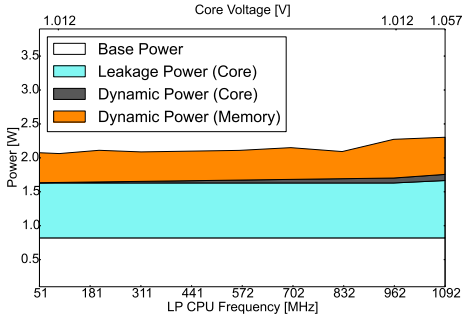
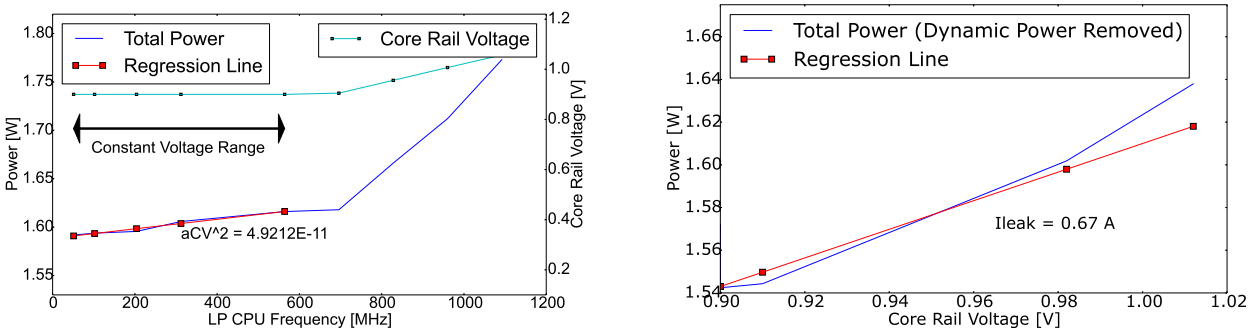
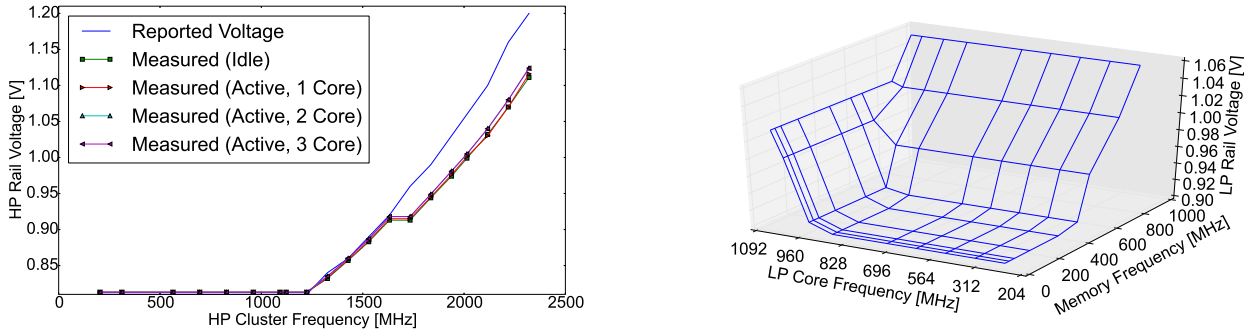


Fig. 4: Breakdown of different power components running the LP core (Tegra K1 idle, and HP rail off).



(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

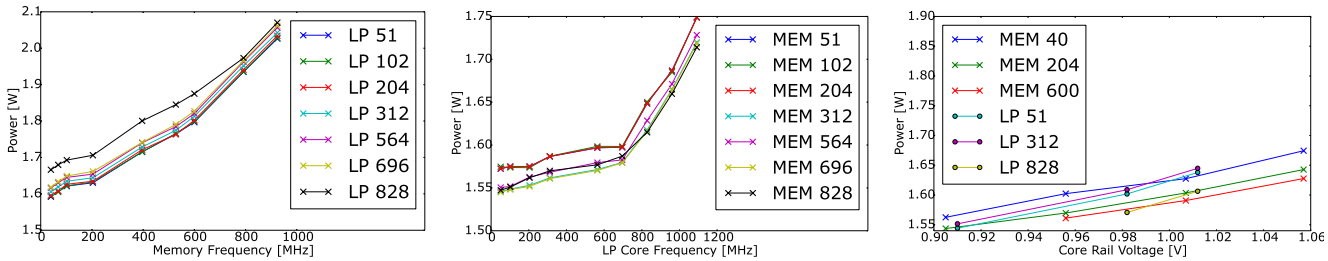
Static power P_{stat} on a rail is always present as long as the rail is powered. Static power can be described as [3]:

$$P_{stat} = I_{leak} V_{rail} \quad (3)$$

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

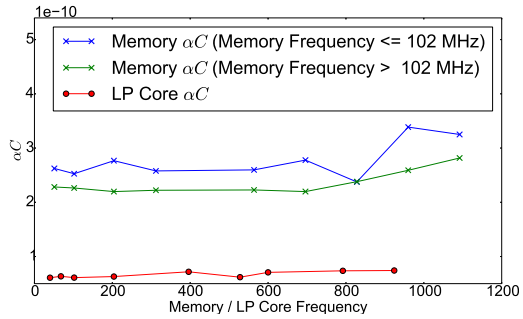
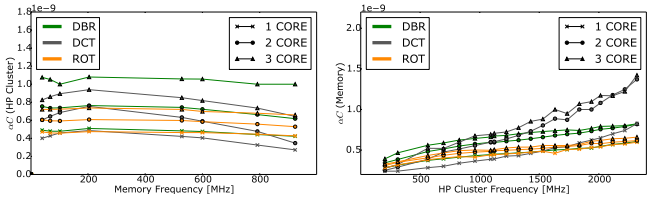


Fig. 8: DPC for Memory and LP Core (idle workload).



(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.

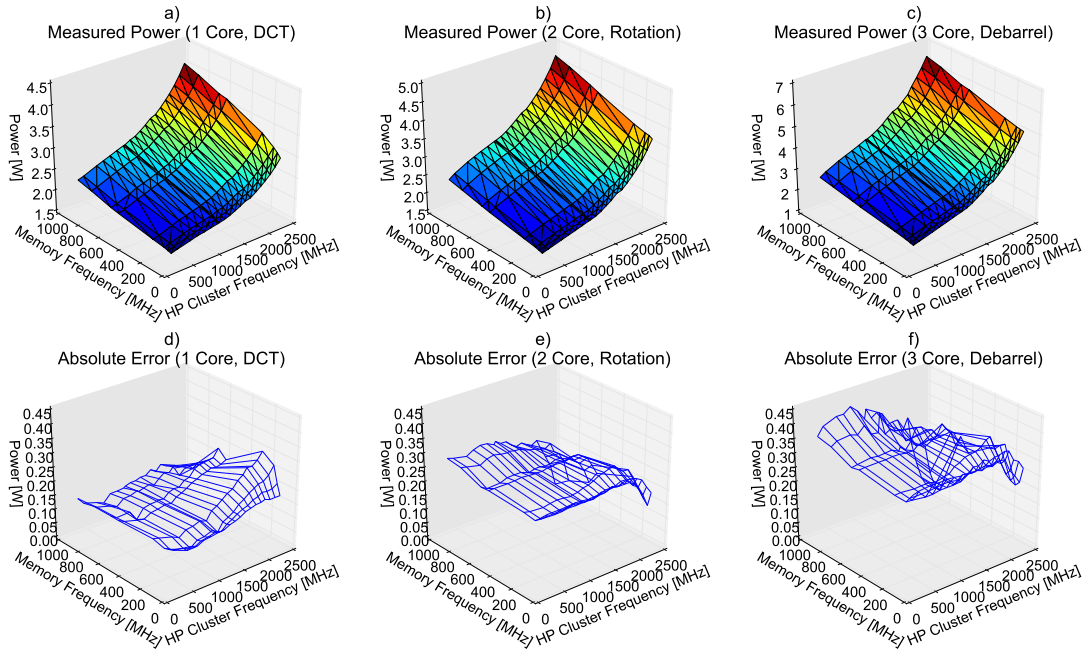


Fig. 10: Comparison of measured (top row) versus model error for some of the filters.

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).

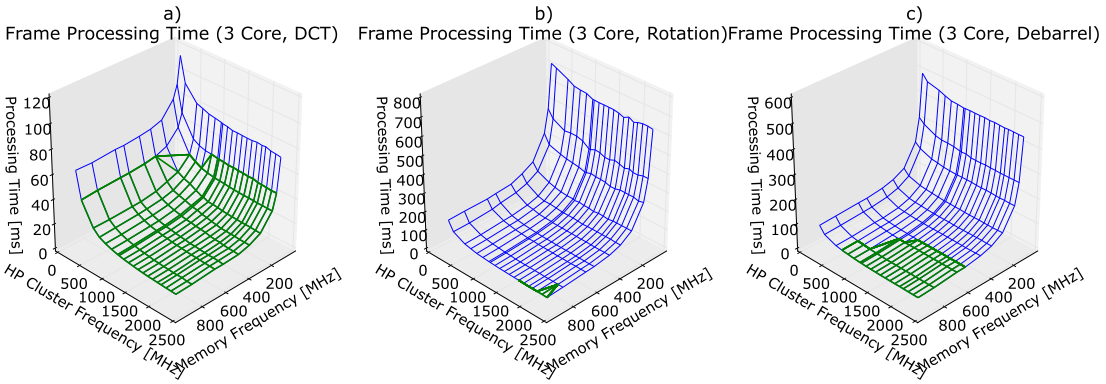


Fig. 11: Workload performance over memory and processor frequencies. The green lines mark configurations that achieve a frame processing time of 50 ms or below.

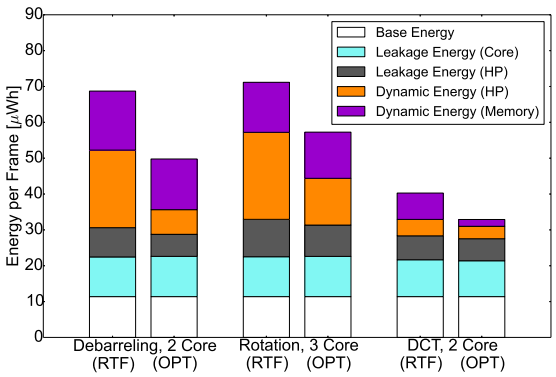


Fig. 12: Detailed breakdown of static, dynamic and base power.

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.

Paper IV: A High-Precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads

Title: A High-Precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads [64].

Authors: K. R. Stokke, H. K. Stensland, C. Griwodz and P. Halvorsen.

Abstract: Energy efficiency of multimedia processing is a hot topic in modern, mobile computing where the lifetime of battery-powered devices is low. Authors often use power models as tools to evaluate the energy-efficiency of multimedia workloads and processing schemes. A challenge with these models is that they are built without sufficiently deep hardware knowledge and as a result they have the potential to mispredict substantially depending on hardware configuration. Typical rate-based power models can for example mispredict up to 70 % on the Tegra K1 SoC. Inspired by multimedia workloads, we introduce a modelling methodology which can be used to build a generic, high-precision power model for the Tegra K1's GPU and memory. By considering hardware utilisation, rail voltages, leakage currents and clocks, the model achieves an average accuracy above 99 % over all operating frequencies, and has been rigorously tested on several multimedia workloads. Our method exposes detailed insight into hardware and how it consumes energy. This knowledge is not only useful for researchers to understand how power models should be built, but also helps to understand what developers can do to minimise power usage. For example, experiments show that for a DCT benchmark, 3 % power can be saved by utilising non-coherent caches and smaller datatypes.

Lesson's learned: In this paper, we investigated the accuracy of state-of-the-art power modelling methodologies for the Tegra K1's GPU and LP CPU core. Learning that rate- and CMOS-based power modelling methodologies can mispredict power usage substantially depending on operating frequencies, we motivated and developed our high-precision power modelling methodology. We demonstrated that the accuracy of our method is close to 100 % over all GPU and memory operating frequencies. In the course of this work, we learned that model training benchmarks that are designed specifically to stress various architectural units on the Tegra K1, are needed to build an accurate model. Furthermore, as the focus of this work was on modelling the

Tegra K1's GPU rather than the LP CPU core, we found that a more accurate CPU power model was needed to further increase the accuracy of our method.

Author's contribution: Stokke designed and implemented the power measurement setup, model training benchmarks, the modelling method and the video filter operations that were used to verify the approach. He also deisgned the experiments and is the main author of this paper.

Published: Proceedings of the 7th Conference of Multimedia Systems (MMSys), Klagenfurt (Austria). ACM, 2016.

A High-Precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads

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

ABSTRACT

Energy efficiency of multimedia processing is a hot topic in modern, mobile computing where the lifetime of battery-powered devices is low. Authors often use power models as tools to evaluate the energy-efficiency of multimedia workloads and processing schemes. A challenge with these models is that they are built without sufficiently deep hardware knowledge and as a result they have the potential to mispredict substantially depending on hardware configuration. Typical rate-based power models can for example mispredict up to 70 % on the Tegra K1 SoC. Inspired by multimedia workloads, we introduce a modelling methodology which can be used to build a generic, high-precision power model for the Tegra K1's GPU and memory. By considering hardware utilisation, rail voltages, leakage currents and clocks, the model achieves an average accuracy above 99 % over all operating frequencies, and has been rigorously tested on several multimedia workloads. Our method exposes detailed insight into hardware and how it consumes energy. This knowledge is not only useful for researchers to understand how power models should be built, but also helps to understand what developers can do to minimise power usage. For example, experiments show that for a DCT benchmark, 3 % power can be saved by utilising non-coherent caches and smaller datatypes.

CCS Concepts

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

Keywords

Multimedia, Tegra K1, CUDA, energy, performance, CPU-GPU frequency scaling

MMSys'16, May 10 - 13, 2016, Klagenfurt, Austria

ISBN 978-1-4503-4297-1/16/05…$15.00

DOI: http://dx.doi.org/10.1145/2910017.2910591

1. INTRODUCTION

Energy consumption is an important topic in mobile computing. Today's battery-powered mobile devices such as smart phones, tablets, laptops or even drones have limited uptime due to a combination of small battery capacity and power-intensive hardware. As such, research in multimedia systems have in the past decade focused on trying to understand the power requirements of hardware to aid energy optimisation efforts. For example, Hosseini et al. [6] attempt to save energy in a game streaming scenario by selectively downloading and processing more important textures over less important ones, and Sharrab et al. [16] propose a power model for MPEG-4, MJPEG and H.264 and shows the effect in terms of power usage under various H.264 compression parameters.

Unfortunately, researchers' view of how modern mobile devices consume energy is limited. To evaluate energy usage of for example multimedia systems, they use power models which are too simple to justify the complex mechanisms that govern the power usage of hardware. Hosseini et al. [6] employ a smart phone power estimation tool called *PowerTutor* [21] to estimate the power of an HTC 3D evo phone under their game streaming scenario. However, inspecting the source code from PowerTutor we can see that the estimates are actually based on a power model from the HTC Dream. This leads to inaccurate estimates that give a false impression of how hardware consumes energy.

Powertutor [21] and other authors [6, 4, 20] implement *rate-based* power models, where for example power of a processor is assumed to grow linearly with the processor's utilisation level. Figure 1 shows prediction error for three types of power models running a motion estimation workload on the Tegra K1's GPU, where 1a is the commonly-used rate-based model. While the accuracy of this model type can be close to 100 %, the misprediction can also be substantial up to 70 % depending on the GPU and memory operating frequency. Thus the accuracy of such models is entirely dependent on the operating frequencies selected by the Dynamic Voltage and Frequency Scaling (DVFS) algorithms running on the platform at the time of the verification. The inaccuracy of rate-based models is not only limited to frequency settings. They also ignore many other mechanisms that have non-negligible effects on power usage such as core and rail power gating, frequency scaling and variations in rail voltage levels [15], variable cost of executing different instructions (see Figure 12), different software workloads and contention of hardware resources such as caches [1].

Some authors, such as Sharrab et al. [16] attempt to model

more accurately the effects of power saving mechanisms such as frequency scaling. Their processor power model is based on the work by He et al. [22] and suggests that processor power is proportional to the cube of the number of cycles required to finish processing. However, this assumes that performance is inversely proportional to processor frequency. This has been shown to be an incorrect assumption for the Tegra K1, where performance scales sublinearly with frequency due to contention for resources [15]. The model will therefore mispredict on our platform.

In this paper, we propose a methodology to build high-precision, generic power models for the Tegra K1 SoC with special focus on its CUDA-capable GPU and main memory. We use several multimedia workloads intended for post-processing a live, raw video feed as a case study and demonstrate an average accuracy above 99 %. The model has been more rigorously tested than related work by running all experimental benchmarks over all possible GPU and memory frequencies. The accuracy is never below 96 %. Our method is based on estimating swithing capacitance of captureable hardware events, such as integer, floating point and conversion instructions, clock cycles and leakage currents by running a set of synthetic workloads stressing the various hardware components [1]. It also takes into account measured voltage levels on the rails supplying the SoC. Compared to related modelling efforts, the increased accuracy of our model does not compromise performance in power estimation. Our contributions are as follows:

- An accurate power model which gives good insight into power usage of a modern platform and can be used for evaluating energy efficiency of generic multimedia workloads.
- Our methodology shows other authors in the field how accurate power models should be built and validated for modern SoCs.
- By estimating the switching capacitance of various hardware events, our model shows how developers can be more energy-efficient by utilising local cache better or using smaller datatypes.
- We demonstrate a 3 % power saving for the DCT multimedia workload without compromising performance.

Our outline is as follows. Section 2 gives a short introduction to two common types of power models, and as a motivation compares the estimation accuracy of these and our model. Section 3 introduces our GPU multimedia workloads used to verify our model. In Section 4 and 5 we introduce the GPU, memory and CPU model predictor, and we derive the power model and the methodology to build it. The model is verified and discussed in Section 6, where we also discuss some preliminary energy-optimisation approaches. Finally, we conclude our work in Section 7.

2. BACKGROUND AND RELATED WORK

There is a plethora of work that attempts to model the power usage of various types of computing devices, such as smart phones, embedded development systems, laptops and stationary computers. These generally describe power or energy as linear systems where the cost of executing various types of instructions, accessing different cache hierarchies, disks or network interfaces is found using different methodologies. While some authors have used neural networks to estimate these costs [7, 10] the vast majority use multivariable, linear regression. The typical way of describing for example the power usage of GPUs [5, 9, 18] and CPUs [14, 19] is of the form:

$$P_{tot} = \beta_0 + \sum_{i=1}^{N_\rho} \rho_i \beta_i \quad (1)$$

In Equation 1, β_0 is the power of idle components with constant power draw, ρ_i is a predictor with units of accesses per second, β_i is the cost in Watt-seconds per access and N_ρ is the number of predictors. Note that some works [5, 9] have slightly different interpretations of Equation 1. For example, β_i can be replaced with the maximum power of a hardware component $P_{i,max}$, and ρ_i can be replaced with the utilisation of that component ($\rho_i \in [0, 1]$). The general form of the expression remains the same and viable to solution with regression. During the past five years several researchers have extended these modelling efforts out of the laboratory, where such models are built on-line on smart phones using various types of power measurement sensors [4, 20, 21].

A liability with modelling power using Equation 1 is that it does not consider the physical rules that govern energy consumption. Modern SoCs such as the Tegra K1 (see Figure 6) are complex platforms featuring several *rails* powering the various components within the SoC. Figure 6 shows four of these: the GPU, memory and CPU cluster rails. Kim et al. [8] describes the rate of energy consumption for logic CMOS circuits as the sum of dynamic and static power. These equations can be readily applied to individual power rails [3, 15]. Static power is the product of the circuit's leakage current $I_{R,leak}$ and rail voltage V_R:

$$P_{R,stat} = I_{R,leak} V_R \quad (2)$$

Dynamic power is caused by switching activity and is governed by both hardware and software. For example, as a processor executes instructions, loads data from cache, or the memory chip spends cycles serving memory read and write requests, dynamic power is being used:

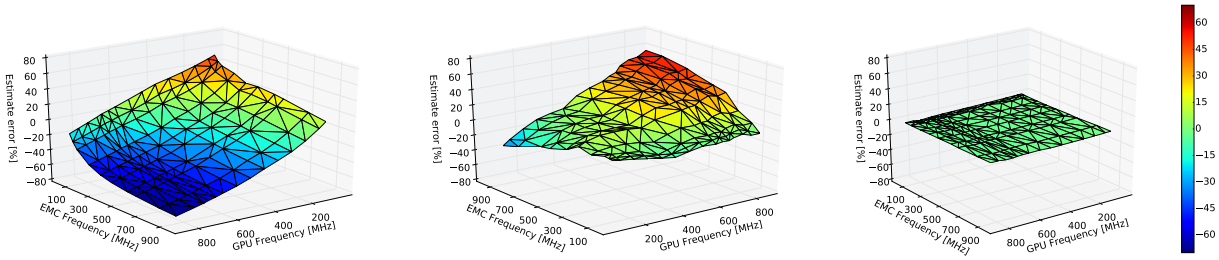
$$P_{R,dyn} = \alpha_R C_R V_R^2 f_R \quad (3)$$

In Equation 3, C_R is a potential maximum switching capacitance per cycle (with a unit of coloumbs per volt per cycle) and $\alpha_R \in [0, 1]$ is a workload-specific factor which decides how much switching capacitance C_R is being consumed per cycle. f_R is the operating frequency in cycles per second.

The take-away point from Equations 2 and 3 is that the cost of executing instructions (dynamic power) and using hardware (static power) varies with voltage. The voltage on a rail V_R further depends on the operating frequency on that rail f_R [8], and is regulated by DVFS algorithms (see Figure 2). When building rate-based power and energy models using Equation 1, it is therefore reasonable to expect that, depending on the frequency operating point where the model was built, the estimated costs β_i will vary.

We now conduct an experiment to find out how inaccurate power estimations β_i can be if rail voltage is not taken

[1]Source code and trace files are available at http://folk.uio.no/krisrst/mmsys16/



(a) Rate-based model. (b) Modelling switching capacitance αC. (c) Our hybrid model.

Figure 1: Prediction error for a motion estimation kernel.

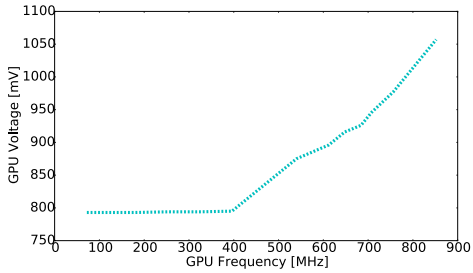


Figure 2: GPU voltage versus frequency.

into account. The details around this experiment is similar to the method presented in Section 5.2. We use Equation 1 and multivariable, linear regression to build a simple power model based on our synthetic benchmarks listed in Table 2 and our predictors in Table 1 (excluding memory, clock and leakage predictors which are not typically used but rather a part of our contributions). We verify the model with a motion estimation filter for videos running on the GPU (see Section 3.2). The result can be seen in Figure 1a, where model accuracy is plotted versus memory and GPU frequencies. As expected, we see that accuracy is very variadic depending on operating frequency because Equation 1 does not consider rail voltages. The resulting planes show a gradual increase in accuracy which for some frequency combinations is very close to 100 % (green and blue area). Our hypothesis is that some of these areas show better accuracy because they reflect better the frequency levels that were set in the model training phase by the memory and GPU DVFS algorithms. At high frequency levels the model underestimates up to 60 %. In the opposite case (low frequencies), the model overestimates by up to about 60 %. This is a direct effect of the fact that rail voltage is not considered.

There are few works which attempt to incorporate rail voltages into power models. Hong and Kim [5] model static power considering both rail voltage and temperature, but have a rate-based dynamic power model. Castagnetti et al. [3] model the power of an Intel XScale CPU. Pathania et al. [13] model power of a 4+4 ARM CPU in big. Little configuration as well as a PowerVR GPU for gaming workloads. Both take into account rail voltages as shown in Equations 2 and 3. However, they are based on finding the dynamic power coefficient $\alpha_R C_R$ either directly or through the processor hardware utilisation level, similarly

Figure 3: Video stream rotation.

to the work by Stokke et al. [15]. The error of such a model built for the Tegra K1 can be seen in Figure 1b. While the error rate of such a model is better than a pure rate-based one it can still be very high up to 50 %. The methodology also has two disadvantages. First, $\alpha_R C_R$ changes depending on the workload characteristics and must be subsequently re-estimated for any workload combination. Secondly, increasing frequencies in various domains (GPU or CPU) inevitably increases hardware utilisation α_R in other parts of the platform. In this aspect, we provide an entirely generic model which takes as inputs the various utilisation levels of different components in the Tegra K1 SoC as well as rail voltages, and achieve a much better accuracy close to 100 %, as shown in Figure 9c.

3. WORKLOADS

Our workload represents video processing operations intended for post-processing raw video streams stored in the YUV format. These have been implemented in CUDA for the Tegra K1 GPU. In our benchmarks, these workloads process 80 full-HD video frames.

3.1 Image Rotation

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

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} cos - \theta & -sin - \theta \\ sin - \theta & cos - \theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (4)$$

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

3.2 Motion Vector Search

In the second test, we apply motion vector search (MVS) on the raw video stream. MVS is a common technique in

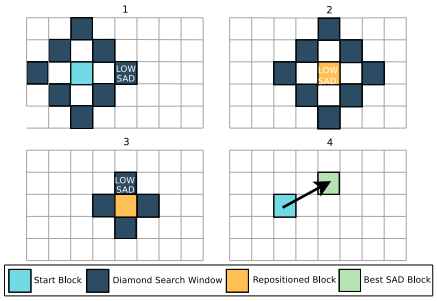


Figure 4: An illustration of the operation of the diamond search algorithm.

video encoding to reduce the amount of information that has to be stored with each frame. In our case, it works by dividing each frame into a set of macroblocks of 8x8 pixels, and then attempting to estimate each block's displacement (the vector) relative to the previous frame.

We have implemented the diamond search algorithm [23]. Diamond search estimates the displacement of each macroblock by computing the sum of absolute difference (SAD) of the current macroblock and the eight surrounding macroblocks in the previous frame (as shown in Figure 4). At every step, as long as the macroblock with the lowest SAD is not in the center of the "search window", the window will be re-centered at the macroblock with the lowest SAD. After three iterations, the pattern changes to a smaller diamond with only four surrounding macroblocks, where the one block with the lowest SAD is estimated to be correct.

3.3 Compression

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

$$M_{u,v} = \gamma(u)\gamma(v)\sum_{x=0}^{7}\sum_{y=0}^{7} m_{x,y} cos[\frac{\pi}{8}(x+\frac{1}{2})u] cos[\frac{\pi}{8}(y+\frac{1}{2})v] \quad (5)$$

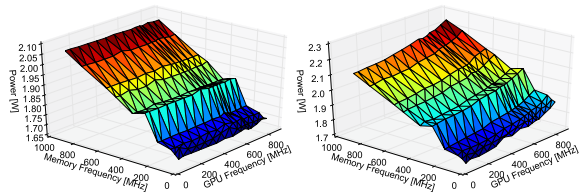
In Equation 5, $u, v \in [0, 7]$ are the DCT output coordinates, $M_{u,v}$ are the frequency components, $m_{x,y}$ are the original pixel values in the macroblock, and $\gamma(w)$ is a normalising function.

3.4 Debarreling

Barrel distortion is an effect that occurs with different lenses [17]. Our "debarreling" workload computes a constant debarreling map for one type of lens. This map only needs to be calculated once and is subsequently applied to each frame. The debarreling filter is the least compute-intensive filter we consider.

4. ENERGY CONSUMING ENTITIES ON THE TEGRA K1 SOC

To build a power model for the Tegra K1, it is important to have solid understanding of the relevant parts of the architecture (see Figure 6) which consumes energy under our workloads. In this section, we describe these components, the Hardware Performance Counters (HPCs) they expose to our power model as predictors, and why we use them (see Table 1 for an overview of the predictors).



(a) GPU is off (b) GPU is on and idle

Figure 5: Power usage depending on whether the GPU is off or not.

4.1 Memory

The Jetson-TK1 is equipped with off-chip DDR3 Random Access Memory (RAM) and two Embedded Memory Controllers (EMC) [11]. The EMCs arbitrate memory accesses from the CPU complexes (EMC_{cpu}, 32-bit accesses) as well as the GPU (EMC_{gpu}, 64 bit accesses). The Tegra K1 is thus capable of executing both CPU and GPU memory-intensive programs at the same time. Carroll and Heiser [2] show in their analysis of the OpenMoko Freerunner smartphone that RAM can account for a substantial amount of total power depending on the type of workload, even without an active GPU. Despite this, RAM power is usually not directly accounted for in literature.

4.1.1 *Dynamic Power for RAM*

The possibility to monitor memory activity is not trivial. Not accounting for two CPU HPCs that can count L2 cache misses and writebacks there are few predictors which can be used to trace memory utilisation. The CPU HPC implementation only allows for collection of a maximum of six counters simultaneously, so these should be avoided. The GPU is also equipped with set of HPCs which can be read during program execution, but these do not provide access to the total number of bytes read or written to memory by the GPU.

The Tegra K1 instead implements a hardware activity monitor (ACTMON) which is intended to guide the memory DVFS algorithm [11]. The ACTMON is capable of counting the following:

- The total number of memory cycles spent serving CPU memory requests.
- The total number of memory cycles spent serving any memory requests (including GPU and other sources).

We wrote a modified kernel driver for the ACTMON, enabling us to count these variables in any time interval above one ms. While this solution is not able to distinguish between memory reads and writes, it has several advantages. For example, it provides a fine-grained measure of hardware activity created by the GPU and the CPU. The method is also able to count *all* accesses to memory, for example caused by GPU driver overhead and memory accesses from other sources, and avoid occupying HPC space.

4.1.2 *Clock Power*

The memory bank is continuously spending energy to maintain its consistency. In Figure 5a), the total power of the

Jetson-TK1 is plotted versus memory and GPU frequency. In this example, the GPU rail is off. Therefore, there is no change in power as GPU frequency increases. However, the total power usage increases linearly with memory frequency. We assume that this is due to increased self-refresh frequency. The power model must take this factor into account.

At memory frequencies 204 and 300 MHz, we see that there is an inconsistency in our assumption. Despite the linear trend at other frequencies, power drops slightly at 204 MHz and increases at 300 MHz. We do not know the reason for this, but some PLL clocks are activated in the core domain at these frequencies. We take the drop and increase of power usage (relative to the "linear" increase in power at the other frequencies) into consideration in our model as simple offsets.

4.2 GPU

The Kepler-based GK20A GPU on the Tegra K1 is a more parallel architecture than the CPU, capable of running 128 threads in parallel. The GK20A contains a single SMX. An SMX implements four warp schedulers, each of which is capable of concurrently running groups of 32 threads called *warps*. Two independent instructions per warp can be executed at the same time [12]. The threads utilise 192 CUDA cores which provide arithmetic and floating point functionality and various other units such as code, data and texture caches (see Figure 6). The Special Function Units (SFUs) implement special functions such as sin and cosine. They are out of scope for this paper.

Due to the high number of concurrently active threads, memory pressure (and power usage) is substantial. The SMX features a complex memory hierarchy to improve memory access latency and bandwidth. Read accesses from memory are performed through the Tegra K1's 64 bit EMC2 interface. Accesses are cached in a L2 cache with size 128 kB, which is global to all threads running on the SMX. Memory read accesses can also be stored in a warp-local 64 kB L1 cache, either implicitly on RAM loads or directly through the use of shared memory (in which case it can also be written).

When considering the power usage of the Tegra K1's GPU it is important to have the best control over hardware utilisation of the different parts of the Kepler architecture. In this section, we outline the HPCs used to collect this information using NVIDIA's profiling tool, `nvprof`.

4.2.1 Memory Hierarchy

Any memory access can result in hardware utilisation on three different components of the Tegra K1. In the worst case, the memory is fetched or stored off-chip in RAM. Data can also be cached in L2 and L1.

An easy misconception is that the global memory (RAM) throughput HPCs, `gst_throughput` and `gld_throughput` reflect actual amount of data stored and loaded from RAM. However, we found in our experiments that these counters do not separate between actual RAM accesses and L2/L1 cached accesses. Instead, we use the ACTMON activity monitor to track the GPU's utilisation of RAM.

The `l2_subp0_total_read_sector_queries` HPC counts the number of 32 B read accesses to the L2 cache. In our attempts to train the model, we also used the `l2_subp0_total_write_sector_queries`, but we were unable to estimate a meaningful coefficient to it. We suspect this is because L2 writes are directly written to RAM, and that the power usage related to such events is instead captured by our RAM activity counter.

The L1 cache utilisation is more difficult to trace accurately because there is no single counter (as for the L2 cache) which enables us to trace the raw number of read and write transactions. L1 is used for caching thread-local memory accesses, global memory accesses, and finally, it can be configured for shared memory access between threads. To accurately trace L1 cache utilisation, we need to trace all these types of accesses.

Thread-local data consists of for example function parameters. This memory is allocated in RAM and automatically cached in L1. Accesses to local memory via the L1 cache is traced by the `l1_local_{store/load}_hit` HPCs, which increment by one for each 128-bit transaction. Memory reads from RAM may also be stored in L1, and are traced with the `l1_global_load_hit` HPC. Memory stores to RAM are not cached in L1 because L1 caches are not coherent, but are instead directly written to L2 cache or RAM.

L1 utilisation that occurs as a result of shared memory usage is harder to trace. These transactions are counted by the HPCs `l1_shared_{load/store}_transactions`. However, shared memory accesses are often broadcasted to all or several of the threads running in a warp, because all threads access the same memory location. In this case, the number of transactions is still counted for each thread-initiated share memory load or store. Therefore, the number of transactions reported by these HPCs is much higher than what is *actually* read or written in hardware. To estimate the actual utilisation, we found that it is possible to use the `shared_efficiency` HPC. This counter gives the ratio of requested (actual) shared memory throughput to the required (thread-initiated total) throughput. The actual L1 shared access utilisation can be estimated as follows:

$$L1_{shr\_act\_\{l/s\}} = L1_{\_shr\_\{l/s\}} \times shared\_efficiency \quad (6)$$

Note that the estimate can be inaccurate because `shared_efficiency` does not separate between reads and writes.

4.2.2 Functional Units

Running GPU threads utilise various hardware units to perform additions, control operations, register loads and stores etc. NVIDIA provides fine-grained accounting over which types of instructions have been executed over time, which are specified in several groups. The grouping allows for more accurate tracking of hardware utilisation. For example, it is to be expected that floating point operations cost more power compared to pure integer operations. The groupings of HPCs are as follows:

- `inst_fp_32 / inst_fp_64` counts floating point operations on different datatypes (32-bit or 64-bit operations).
- `inst_integer` counts integer operations.
- `inst_bit_convert` counts bit conversion instructions.
- `inst_control` counts control flow instructions, such as branching and jumps.
- `inst_misc` counts miscellaneous instructions, such as register moves and NOP instructions.

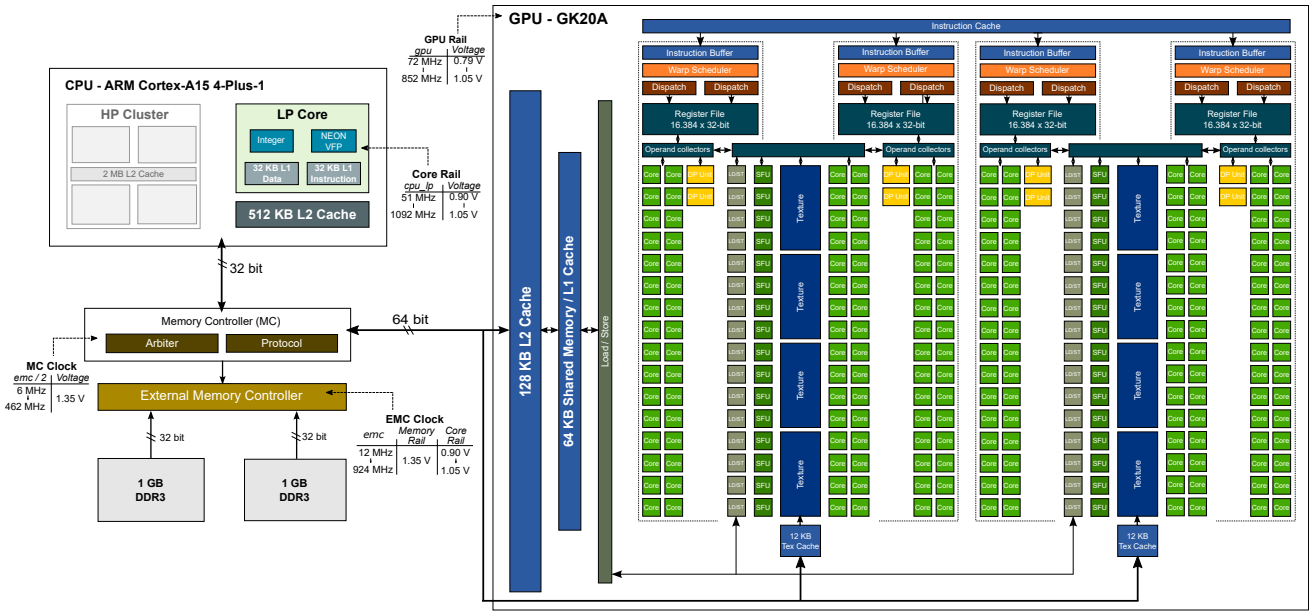


Figure 6: Overview of the Tegra K1 and the Kepler SMX architecture.

4.2.3 *Clock Power*

In addition to dynamic power caused by application utilisation in the GK20A's memory hierarchies and functional units, there is some power usage which is related to the GPU's own clock (clock power). This is visible when the GPU is on (GPU rail is powered), but idle. Consider for example Figure 5b. Here, we have set the GPU's inactivity timer (the time before the GPU rail is powered off after for example a kernel launch) to a few seconds, and logged the average power usage of the Jetson-TK1 in this period over all GPU and memory frequencies. Comparing with Figure 5a, where the GPU is off, we can clearly see that there is non-negligible additional switching activity when the GPU is powered and idle. Furthermore, we can see that the additional overhead scales linearly with frequency until GPU rail voltage starts to increase (at 400 MHz, see Figure 2). From these point, power scales linearly with $f_{gpu}V_{gpu}^2$. This confirms that the additional overhead is an effect of dynamic power on the GPU rail.

4.2.4 *Static Power*

Static power usage is caused by leakage currents in the GPU's circuitry (for example transistors). For the GK20A, we have found that the estimated leakage current on the GPU rail has higher standard deviation than the other predictors. This is likely due to power gating inside the circuitry.

4.3 LP CPU Core

The Tegra K1's CPU is not the main target of this study, but its power usage is non-negligible for several reasons. It is for example running the operating system and all its system services and drivers, as well as launching CUDA kernels through our benchmark programs. It also utilises RAM through its own EMC, in particular to fetch and store processed frames. For these reasons, one cannot assume that the power usage caused by CPU activity is negligible.

While the Tegra K1 has a complex, dual-cluster CPU with a High-Performance (HP) quad-core cluster in addition to the LP companion core, we restrict processing to the LP core because the main target of this study is the GPU. Ideally, it should only be necessary to model dynamic power of the LP core by disabling the HP cluster and fixing processor frequency to 1092 MHz. This causes static power to be estimated as part of the base power, because the core rail voltage (and static power contribution) will not vary. However, in our experiments, we observed that the estimated dynamic CPU power coefficients give more precise values if the static power usage is also modelled.

The Tegra K1's CPU has an HPC implementation which can be queried with the `perf` Linux framework. Compared to the GPU HPC implementation, the CPU has less fine-grained instruction counting, with only one global (total) instruction counter. It does, however, have better accounting for L1 and L2 cache accesses, which are able to separate between hits and misses for both data and instruction cache reads and writes. Only seven counters can be monitored simultaneously, and one of these is always occupied by the *active cycle counter*. Because we do not expect the dynamic CPU power to be substantial, we ignore the CPU cache hierarchy and variation in costs of different types of instructions. We define a new metric, $\rho_{cpu,ipc}$, which is defined as the ratio between the instruction count over the active cycle count which accounts for CPU dynamic power caused indirectly by software execution:

$$\rho_{cpu,ipc} = \frac{N_{cpu,inst}}{N_{cpu,cycles}} \tag{7}$$

Intuitively, $\rho_{cpu,ipc}$ will decrease estimated CPU dynamic power when the CPU is stalling more. We let this counter represent the software workload.

GPU and RAM have a clock-dependent dynamic power cost which is basically an estimated cost per clock cycle. This is also the case for the CPU, but the number of cy-

cles per second is *not* the raw frequency point for the CPU (1092 MHz), as for RAM and the GPU. This is evident because attempting to use the raw frequency point to estimate the dynamic clock power completely breaks the regression results, severely reducing the accuracy of the model and causing several negative coefficient estimates. We believe that the CPU is power gating the clock aggressively when it is idle, and therefore, the actual cycles per second is the active cycle counter itself.

5. A HIGH-PRECISION POWER MODEL

In this section, we derive the power model used to estimate power usage of our GPU multimedia workloads. The main idea behind our methodology is that dynamic power is modelled in terms of measurable hardware activity, and that we compensate for variations in rail voltages. We describe in detail the methodology used to build the model, where we have implemented specialised benchmarks to stress the relevant parts of the Tegra K1 in such a way that the regression results become accurate.

5.1 Derivation

The total power usage of the Jetson-TK1 is the sum of power usage on each rail P_R and base power P_{base}. The Tegra K1 has 21 power rails, but only three are being used in our scenario: the *core*, *memory* and *GPU* rails (see Figure 6). All other rails are assumed to be idle with constant power usage as a part of P_{base}. The total power usage becomes:

$$P_{total} = P_{core} + P_{mem} + P_{gpu} + P_{base} \tag{8}$$

The total power usage on a rail is the sum of static and dynamic power (see Equation 2 and 3). However, the dynamic power equation only gives a crude average of the switching activity (instructions executed, memory requests, caching operations etc). As mentioned in Section 2, the dynamic power coefficient $\alpha_R C_R$ can be estimated using regression [15], but the process is prone to error because α_R is not always constant over all frequency combinations. For example, when increasing memory frequency, switching activity (α_R) in other architectural units such as the CPU or GPU can also increase, leading to misprediction. The core improvement of our model is that we express dynamic power in terms of measurable hardware activity while also accounting for changes in rail voltage. By doing this, our hypothesis is that power usage can be more accurately estimated for any workload on any operating frequency point. We model dynamic power on a rail R as:

$$P_{R,dyn} = \sum_{i=1}^{N_R} C_{R,i}\rho_{R,i}V_R^2 \tag{9}$$

In the equation above, for rail R, $\rho_{R,i}$ is a hardware activity predictor in occurrences per second, N_R is the number of predictors, and $C_{R,i}$ is the switching capacitance (coloumbs per volt) per occurence of event $\rho_{R,i}$. The total power usage of the Jetson-TK1 becomes the sum of power usage on each of the three active rails $R \in \mathbb{R}$ and base power P_{base}:

$$P_{jetson} = \sum^{R\in\mathbb{R}} (P_{R,dyn} + P_{R,stat}) + P_{base} \tag{10}$$

Note that static power of the memory rail is not possible to model because the voltage on that rail does not vary, as per our discussion in the next section.

5.2 Methodology

By extension of Equation 10, the unknown variables are the switching capacitances $C_{R,i}$, leakage currents $I_{R,leak}$ and base power P_{base}. Our methodology to find these terms is based on multivariable linear regression and is summarised as follows. We create twelve spesialised benchmarks designed to stress different hardware blocks of the Tegra K1 (see Table 2). Each of these are run over all possible combinations of processor and memory frequencies (see Table 3) for a total of 1830 samples. In each run, we log the predictors $\rho_{R,i}$, voltages V_R and operating frequencies. The predictors must now be processed to account for the rail voltage:

- All dynamic power predictors $\rho_{R,i}$ must be multiplied by V_R^2 (see Equation 9).
- The static power predictor is just the rail voltage V_R (see Equation 2).

The resulting "final" predictors are then passed to the regression solver and produces the coefficients seen in Table 1. Note that while related works often use multivariable regression with non-negative coefficients [19] we simply use normal regression without ending up with negative coefficients[2].

Running each benchmark over all possible frequency combinations has several advantages.

1. It creates natural variation between the model predictors (access rates). When memory frequency is low and GPU frequency is high, the memory access rate is lower while the GPU hardware utilisation is higher, and vice-versa. This is also helps test the full range of model predictors (rates), which is extremely important for regression to estimate accurately.
2. Rail voltages vary depending on operating frequency. This also increases diversity in the training dataset's predictors.
3. Increasing frequency like this also helps estimate clock power and leakage currents (which would be impossible otherwise) because higher rail voltage is higher at high GPU frequencies.
4. Increased dataset size.

Note that our method consists of a high number of runs (1830) to complete the model training phase because each of the benchmarks listen in Table 2 is run over all possible GPU and memory frequency levels. The number of runs can be reduced by including fewer frequency levels and should not affect the accuracy of the model, but care must be taken to force variation in rail voltages. For example, most of the GPU frequencies below 400 MHz are not needed because rail voltage does not vary (see Figure 2). We leave it to future work to investigate the impact in terms of model accuracy resulting from a reduced number of training frequencies.

In our initial attempts to train our model, we found that trivially using example code (for example CUDA examples)

[2]The only exception is our memory offset "tweak" at 204 Mhz, which is expected.

Rail	Number	Predictor	Description	Coefficient	Value
GPU	0	V_{gpu}	GPU voltage	$I_{gpu,leak}$	$0.27A$
	1	$\rho_{gpu,clock}$	Total clock cycles per second	$C_{gpu,clock}$	$2.10\frac{nC}{V}$
	2	$\rho_{gpu,L2R}$	L2 cache 32B reads per second	$C_{gpu,L2R}$	$10.79\frac{nC}{V}$
	3	$\rho_{gpu,L1R}$	L1 cache 4B reads per second	$C_{gpu,L1R}$	$8.90\frac{nC}{V}$
	4	$\rho_{gpu,L1W}$	L1 cache 4B writes per second	$C_{gpu,L1W}$	$8.43\frac{nC}{V}$
	5	$\rho_{gpu,INT}$	Integer instructions per second	$C_{gpu,INT}$	$41.11\frac{pC}{V}$
	6	$\rho_{gpu,F32}$	Float (32-bit) instructions per second	$C_{gpu,F32}$	$38.15\frac{pC}{V}$
	7	$\rho_{gpu,F64}$	Float (64-bit) instructions per second	$C_{gpu,F64}$	$115.33\frac{pC}{V}$
	8	$\rho_{gpu,CNV}$	Conversion instructions per second	$C_{gpu,CNV}$	$72.42\frac{pC}{V}$
	9	$\rho_{gpu,MSC}$	Miscellaneous instructions per second	$C_{gpu,MSC}$	$28.36\frac{pC}{V}$
Memory	0	$\rho_{mem,clock}$	Total clock cycles per second	$C_{mem,clock}$	$258.66\frac{pC}{V}$
	1	$\beta_{mem,204}$	Power offset at 204 MHz	$P_{mem,204}$	$-0.03W$
	2	$\beta_{mem,300}$	Power offset at 300 MHz	$P_{mem,300}$	$0.05W$
	3	$\rho_{mem,CPU}$	CPU busy memory cycles per second	$C_{mem,cpu}$	$2.25\frac{nC}{V}$
	4	$\rho_{mem,OTH}$	Other (GPU) busy memory cycles per second	$C_{mem,oth}$	$2.17\frac{nC}{V}$
Core	0	V_{cpu}	CPU voltage	$I_{cpu,leak}$	$0.79A$
	1	$\rho_{cpu,cpi}$	CPU instructions per cycle	$C_{cpu,cpi}$	$3.72\frac{mC}{V_s}$
	2	$\rho_{cpu,acl}$	CPU active cycles per second	$C_{cpu,acl}$	$166.62\frac{pC}{V}$
Other	P_{base}		Base power	-	$0.78W$

Table 1: Overview of energy model predictors and coefficients.

Benchmark	Description	Components / instructions under explicit stress										
		CPU	RAM (CPU)	GPU	RAM (GPU)	L2	L1	INT	F32	F64	Conv.	Misc.
Idle CPU	GPU off, CPU in idle state.	✓										
CPU-workload	GPU off, CPU processing.	✓	✓									
Idle GPU	GPU on and idle, CPU in idle state.	✓		✓								
L2 Read	Stresses L2 cache reads only.	✓		✓		✓						
L1 Read	Stresses L1 cache reads.	✓	✓	✓			✓					
L1 Write	Stresses L1 cache writes.	✓	✓	✓			✓					
RAM	Stresses RAM activity (GPU EMC).	✓		✓	✓							
Integer	Stresses integer arithmetic unit.	✓		✓		✓		✓				
Float32	Stresses floating point unit.	✓		✓		✓		✓	✓			
Float64	Stresses floating point unit.	✓		✓		✓		✓		✓		
Control	Stresses conversion instructions.	✓		✓		✓		✓			✓	
Misc	Stresses miscellaneous instructions.	✓		✓		✓		✓				✓

Table 2: Overview of benchmarks and components under stress.

Clock	Rail	Description	Frequency		Voltage
			Steps	Range $[MHz]$	
cpu_lp	Core	LP core	9	$[51, 1092]$	$[0.80, 1.05]$
emc	Core	Memory	10	$[40, 924]$	$[0.80, 1.01]$
gpu	GPU	LP core	15	$[72, 852]$	$[0.79, 1.05]$

Table 3: The Tegra K1 clocks, voltage and frequency ranges.

has two major drawbacks. First, the measured model predictors are not diverse enough for the regression to estimate meaningful coefficients. For example, L1 and L2 read throughput will remain virtually the same independently of operating frequencies and benchmarks, as data passes both stages anyway. Secondly, only using benchmarks which are actively processing result in poor estimations. This is because some coefficients, such as leakage currents, clock and base power are independent of the workload. Much better estimations can be achieved by also profiling the power for an idle system.

Ideally, each benchmark should stress just one architectural unit of the system (memory writes, L2 reads, etc.). This is hard, and in some cases impossible, to achieve in practice. For example, attempting to *only* read 100 MB of data from memory to L2 cache results in no data being read at all, because the CUDA driver is very good at optimising code and detects that the data is not actually used. This optimisation happens at a driver level and can not be disabled. Furthermore, as mentioned above, at some points it is impossible to stress just one architectural unit. An example is stressing the L1 cache. Stressing the L1 cache inevitably results in stressing L2 cache *and* memory, because of cache eviction policies and the internal workings of the GPU.

We have written twelve spesialised different benchmarks, which can be seen in Table 2. Due to the limitations described above, these are written in a "pyramid" fashion where we stress the (possible) small groups of hardware units first, before adding units on top. For example, it is possible to stress L2 cache reads only, without stressing memory or any other hardware units, by reading the same data from memory over and over without L1 caching, and then conditionally writing it back in an if-condition which never evaluates to true. Then, the process can be done again, but this time forcing the GPU to cache the global reads in L1 as well. The regression will now have diversity in both predictors to accurately estimate their coefficients.

The results of the regression can be seen in Table 1. Note that there is no coefficient for L2 writes. The reason for this is that the estimated coefficient is very small compared to the others, and has very high standard deviation. We

believe this is because L2 writes are not actually cached there, but immediately written back to memory. The power usage related to this event is thus instead part of the active memory cycles.

6. EXPERIMENTS AND DISCUSSION

In this section, we perform extensive verication of our power model using generic multimedia workloads. We show that our model is able to predict power with high accuracy, discuss the model coefficients and finally look at possible ways to save power by exploiting local caches and shorter datatypes.

6.1 Setup

To verify our model, we have developed four different GPU benchmarks for video processing (debarreling, DCT, motion estimation and rotation). We let these process a full-HD video stream for 80 frames, over all possible GPU and memory frequency combinations. Figure 7 shows a simplified flow diagram for a single test run. Before power can be estimated, the GPU HPCs must be collected. This is challenging for a number of reasons:

- HPC collection for GPU kernels takes a substantial amount of time, slowing down the kernels. We must therefore log HPCs on a per-kernel basis to be re-used.
- HPC values vary not only based on the kernel executed, but also on launch configuration and (possibly) function parameters.
- Depending on the set of HPCs to collect, several runs over one kernel is needed.

To solve these issues, we implement callback functions to track kernel launches and kernel exits. On entry to these functions, a hash is computed based on the kernel's symbol name and execution configuration. We ignore variance in HPCs that result as changes in function parameters, which is only relevant for the rotation workload. On kernel launches, we initiate HPC collection as needed (in total three runs per hashed kernel). On kernel exits, if HPC collection is complete, we estimate power of the GPU, CPU and memory using the counters. GPU execution time is measured by reading the total elapsed GPU cycles for a hashed kernel (the `elapsed_cycles_sm` HPC, e_c) and dividing it with the frequency of the GPU:

$$T_{kern} = \frac{e_c}{f_{GPU}} \tag{11}$$

The experimental setup is based on a Keithley K2280S power source and monitoring unit using an external machine to avoid overhead on the Tegra K1 [15]. The power estimation phase is low-overhead which involves straightforward calculation of power using model coefficients and predictors collected with PERF and CUPTI APIs and equations from Section 5.

6.2 Accuracy

Figure 8 shows an example the measured and estimated power components in a single DCT frame encoding interval.

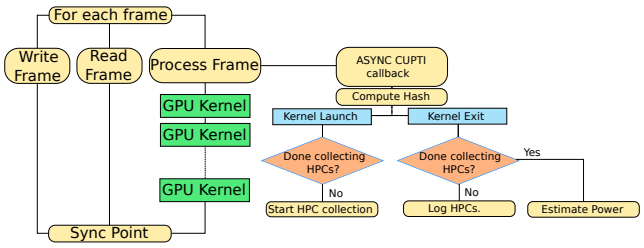


Figure 7: Per-Frame Benchmark flow.

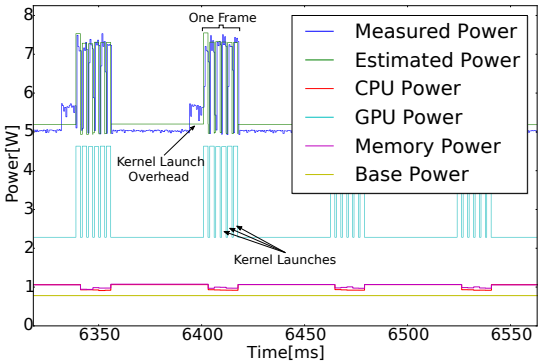
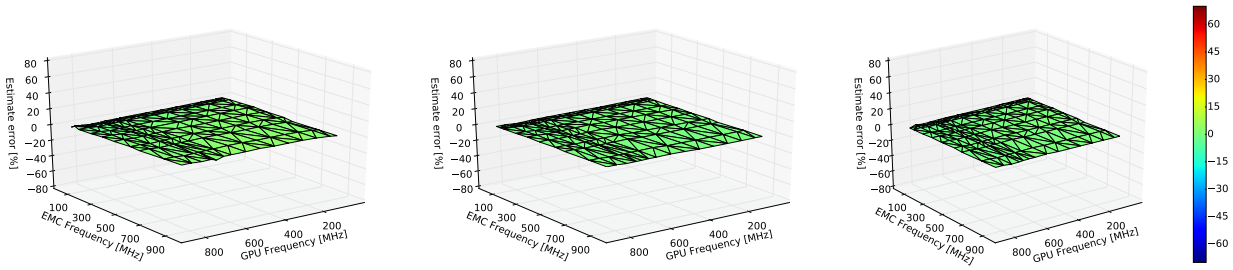


Figure 8: Power plot over time for the DCT kernel.

The peaks in the plot represent kernel launches. There are six kernel launches per DCT frame because the Y frame is divided into four subregions with the same size as a UV frame. We see that the estimation is very accurate over time, close to 100 %. This represents a substantial improvement from state-of-the-art methods that do not consider rail voltages (see Section 2). The total power estimate appears less precise between frames. This is because power estimation only occurs after each kernel launch, and there is a non-negligible kernel launch overhead before the first kernel launch at the beginning of each frame. This overhead is represented as an average since the exit of the last kernel of the previous frame.

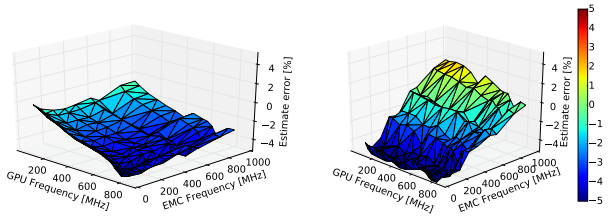
Figure 9 shows the model error in % for a total of 70 processed frames over all GPU and memory frequency combinations, for the debarreling, DCT and rotation workloads. The estimation is very accurate. Not considering the motion estimation filter, the accuracy of our model is over 99 % on average and always above 96 %. This demonstrates that our model, which is built using entirely synthetic benchmarks, is able to consisely capture the power usage of kernels comprised of a more complex mix of instructions and branches.

We now study Figure 10a, which shows a closed-up plot of prediction error for the motion estimation filter. We see that, in general, estimation is good (between 99 to 100 %) at low GPU frequencies. Moving towards the lowest memory and GPU frequency, estimation accuracy tends to degrade towards its lowest point. This is true for all the filters, but hard to see in Figure 9. Furthermore, we see that at 756 MHz GPU frequency, estimation accuracy shows a significant drop across all benchmarks. It is here important



(a) Debarreling. (b) DCT. (c) Rotation.

Figure 9: Prediction error for test benchmarks over all GPU and memory frequency combinations.



(a) Close-up view. (b) No GPU execution.

Figure 10: Motion estimation error over all frequencies.

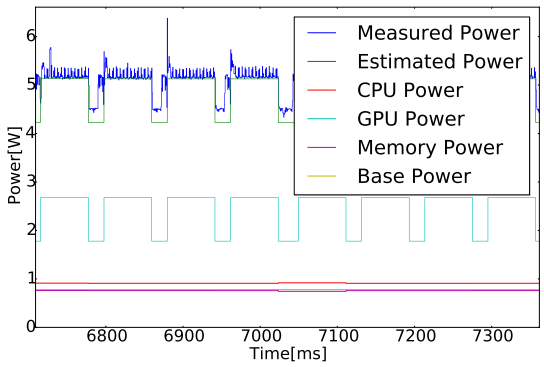


Figure 11: Power over time for a single motion estimation Y-frame processing at 756 MHz (GPU) and 924 MHz (memory).

to note that there is a substantial amount of time where the GPU is *not* actively processing, but the CPU is busy reading the next frame or writing results in memory (see Figure 8). When error starts increasing, we see that the power estimation during this period tends to be larger than when the GPU is active. Consider for example Figure 11, where a single peak (Y-frame motion estimation kernel) is shown. For this test, the error was 4 %. It is clear that the estimation during the kernel launch is very accurate. However, in the period between frames, the estimation is worse. This indicates that either the CPU or memory estimate is inaccurate.

To test the accuracy of the CPU and memory model, we run a differential test on the motion estimation benchmark. No GPU kernels are executed, but frames are read and results written as normal. To avoid compiler optimisations, we conditionally execute the motion estimation kernel in an if-statement which never evaluates to true. The result can be seen in Figure 10b. We see that the error over CPU-only execution can vary about the same amount as when the GPU kernels are also being run. We believe that a better power model for the LP core is needed to further increase the accuracy of the model.

The motion estimation filter has a lower accuracy than the rest of the filters. On average, it is 97 %, and the estimation accuracy decreases more with GPU frequency than for the other filters (at the lowest 95 % accurate). The motion estimation kernels stresses shared memory more, and each kernel runs for 5 to 26 times longer than for the other filters. We found that the CUPTI counters for shared memory transactions do not appear to be reliable, which can cause our model to mispredict more than the other filters. For the U and V frames, CUPTI is for example not able to count shared memory transactions at all. Only shared memory transactions for the Y frames are actually counted. Furthermore, the implementation used to run the motion estimation kernels is slightly different than for the filters. It is therefore also possible that the CPU model is causing the misprediction.

6.3 Model Coefficients and Power Breakdown

Table 1 shows all estimated model coefficients. Looking at the leakage currents and base power, we see that the CPU (LP core) leakage is estimated to be 0.79 A, and the base power 0.82 W. This is very similar to our previous work [15] where a different methodology is used to find dynamic, static and base power. GPU leakage is estimated to 0.27 A, which is much smaller than the CPU. Possible reasons for this is that the GPU is implemented using a smaller number of and/or another type of transistors. Related work has shown that the leakage of the HP cluster varies between 0.38 to 0.80 A [15] depending on the number of online HP cores. Given a choice between running a workload on any of these processors, the GPU therefore seems like a better choice, given that its static power dissipation will be much lower. However, dynamic power must also be taken into account to find the most energy-efficient alternative.

Figure 12 shows most of the dynamic power coefficients from Table 1 in coloumbs per volt. $C_{cpu,cpi}$ is not shown because it has units of coloumbs per volt per second. Considering clock power ($C_{*,clock}$), we can see that the memory

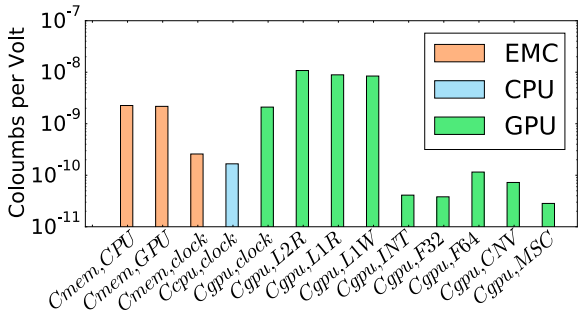


Figure 12: Model coefficients.

Source	Energy Cost			
	per Transaction		per Byte	
	LV	HV	LV	HV
Memory	$3.96nWs$		$0.49nWs$	
L2 Read	$6.73nWs$	$11.89nWs$	$0.42nWs$	$0.74nWs$
L1 Read	$5.55nWs$	$9.81nWs$	$1.38nWs$	$2.45nWs$

Table 4: Energy cost for reading from various cache and memory hierarchies (excluding static and clock energy).

and LP core clock coefficients are similar, i.e., the capacitance load per active clock cycle are the same. For example, at the highest operating frequencies (see Table 3), the LP core and memory clock power is estimated to be 0.20 and 0.43 W, respectively. The GPU clock capacitance is an order of magnitude higher than for the CPU and memory. At the highest frequency, the GPU clock power is estimated to be 1.97 W. From Section 4.2.3, we argue that clock power is independent of the workload. As a processor, the GPU therefore incurs a substantial overhead in terms of clock power compared to the LP core.

The capacitance load per active memory cycle, $C_{mem,*}$, is the same independently of the source. For example, if the CPU initiates a single memory active cycle, it will cost the same as one initiated by the GPU. Since the GPU EMC has a higher bandwidth than the CPU EMC (64-bit vs. 32-bit), it may be possible to save energy in this way. We wrote some simple programs to read and write memory from the GPU, and found that regardless of operation (read or write) the CPU EMC is capable of delivering 4 B per active memory cycle, and the GPU EMC is capable of delivering 8 B per active cycle. This means that reading and writing through the GPU EMC is twice as energy-efficient than the CPU EMC. Other factors must of course also be taken into account, such as the processing done on the data and cache hierarchies.

Studying the workload-relevant GPU coefficients in Figure 12, we see that the capacitance load per transaction on the cache hierarchies ($C_{gpu,L*}$) is larger than the cost for active memory cycles ($C_{mem,GPU}$). However, the actual energy cost depends also on the GPUs rail voltage (see Equation 9), as well as the number of bytes read or written per transaction (see Table 4). Interestingly, memory transactions appear to be more energy-efficient than reading from L1 and L2 GPU cache in most cases (except for the lowest GPU operating voltage). However, leakage and clock power also play important factors. If memory is only read through memory, memory fetches will take longer time, and so the contribution from leakage and clock power will be larger.

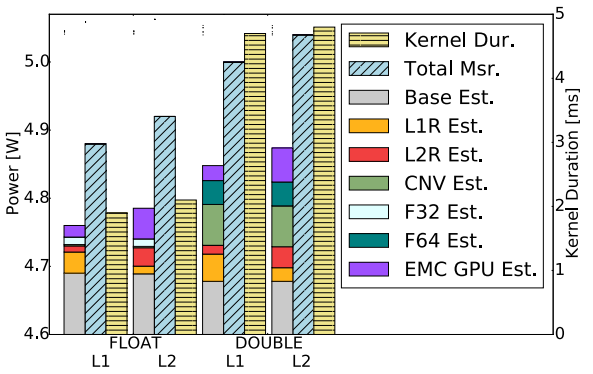


Figure 13: Difference in power usage when using different floating-point datatypes and cache strategies for the DCT.

6.4 Energy-Efficient Programming

In Section 6.3, we have seen that the cost of various instructions differ. For example, the model coefficients hint that loading memory through the L1 cache is cheaper than loading it through the L2 cache. 32-bit floating point instructions are also estimated to have a smaller switching capacitance (see Figure 12). We now attempt to energy-optimise some of our workloads using this information as a preliminary experiment.

We run the DCT benchmark at the highest GPU and memory operating frequencies where we study the impact in terms of performance and power when caching in L1 over L2 and using 32- and 64-bit floating point datatypes (see Figure 13). In these tests, we target a rate of five FPS. This is not because five FPS is very good, but only to equalise the contribution to power usage from the EMCs, CPU and other components. The variation in power usage will reflect only a change in the GPU power usage. At this framerate, the processing finishes within a window of 200 ms, and sleeps for the remaining time before starting processing of the next frame.

Studying Figure 13, we see through our model that using 64-bit floating point operations consumes above three times as much power as using 32-bit operations. Furthermore, a substantial amount of power is saved in bit conversion instructions. With L2 caching, changing the datatype to 32-bit saves 2.4 % power on average. Additionally caching in L1, we see that we are able to save 3.2 %. Our model indicates two main reasons behind this. Although the combined L1 and L2 read energy remains approximately the same, caching in L1 can reduce energy usage because it reduces GPU EMC hardware activity. We believe that this is because the L2 cache is coherent, and maintaining this coherency involves some activity in memory. Note that this is a system-wide measure with much idling between frames, and that the actual saving will vary depending on workload and framerate.

7. CONCLUSION

Power models commonly used for evaluation of multimedia systems in literature are very basic and have the potential for serious misprediction. This is because they do not take into account the physical phenomena which gov-

ern energy usage on modern platforms. In this paper, we have shown that the evaluation of such systems mandates more hardware insight than what is typical. Our method to model power usage achieves an average accuracy above 99 % for several GPU multimedia workloads on the Tegra K1. Our method achieves better accuracy than traditional power models by estimating the power costs of hardware utilisation on the Tegra K1's GPU, memory and CPU (for example instructions per second or active memory cycles) as well as clock and leakage power using measured rail voltages. The estimates are based on entirely synthetic benchmarks designed to stress specific hardware blocks. The model has also been more rigorously tested than normal, over all possible operating frequencies and with different, complex multimedia workloads that excersise more hardware components at the same time. Our model shows not only that it is possible to estimate power of workloads in a generic way (for any GPU multimedia workload) using available HPCs on the Tegra K1, but also helps us to understand better how software can optimise energy usage of multimedia workloads from a hardware point of view. Preliminary experiments show that 3 % power can be saved by using shorter datatypes (for example 32-bit floating point over 64-bit) and by exploiting local, non-coherent caches which do not consume power to maintain consistency with other memory hierarchies.

Acknowledgments

This work has been performed in the context of the BIA project *Unified PCIe IO* (#235530), with contributions from the FRINATEK project *EONS* (#231687) both funded by the Research Council of Norway (RCN).

8. REFERENCES

[1] R. Basmadjian and H. de Meer. Evaluating and Modeling Power Consumption of Multi-Core Processors. In *Proc of e-Energy*, pages 1–10, 2012.

[2] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIX Technical Conference*, volume 14, 2010.

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

[4] M. Dong and L. Zhong. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *Proc of MobiSys*, pages 335–348, 2011.

[5] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In *Proc of ISCA*, pages 280–289, 2010.

[6] M. Hosseini, J. Peters, and S. Shirmohammadi. Energy-Budget-Compliant Adaptive 3D Texture Streaming in Mobile Games. In *Proc of MMSys*, pages 1–11, 2013.

[7] Q. Jiao, M. Lu, H. P. Huynh, and T. Mitra. Improving GPGPU Energy-Efficiency Through Concurrent Kernel Execution and DVFS. In *Proc of CGO*, pages 1–11, 2015.

[8] 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.

[9] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi. GPUWattch: Enabling energy optimizations in GPGPUs. *SIGARCH Computer Architecture News*, 41(3):487–498, 2013.

[10] N. Limin, T. Xiaobin, and Y. Baoqun. Estimation of System Power Consumption on Mobile Computing Devices. In *Proc of CIS*, pages 1058–1061, 2007.

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

[12] Nvidia. Kepler GK110. *NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110*, 2012.

[13] A. Pathania, A. E. Irimiea, A. Prakash, and T. Mitra. Power-Performance Modelling of Mobile Gaming Workloads on Heterogeneous MPSoCs. In *Proc of DAC*, 2015.

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

[15] K. R. Stokke, H. K. Stensland, P. Halvorsen, C. Griwodz. Why Race-to-Finish is Energy-Inefficient for Continuous Multimedia Workloads. In *Proc of MCSoC*, pages 57–64, 2015.

[16] Y. O. Sharrab and N. J. Sarhan. Aggregate Power Consumption Modeling of Live Video Streaming Systems. In *Proc of MMSys*, pages 60–71, 2013.

[17] G. Vass and T. Perlaki. Applying and Removing Lens Distortion in Post Production. In *Proc. of Hungarian Conference on Computer Graphics and Geometry*, pages 9–16, 2003.

[18] J. M. Vatjus-Anttila, T. Koskela, and S. Hickey. Power Consumption Model of a Mobile GPU Based on Rendering Complexity. In *Proc of NGMAST*, pages 210–215, 2013.

[19] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. A System-Level Model for Runtime Power Estimation on Mobile Devices. In *Proc of GreenCom & CPSCom*, pages 27–34, 2010.

[20] F. Xu, Y. Liu, Q. Li, and Y. Zhang. V-edge: Fast Self-Constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics. In *Proc. of USENIX NSDI*, pages 43–55, 2013.

[21] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proc of CODES/ISSS*, pages 105–114, 2010.

[22] Zhihai He, Yongfang Liang, Lulin Chen, I. Ahmad, and Dapeng Wu. Power-Rate-Distortion Analysis for Wireless Video Communication Under Energy Constraints. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(5):645–658, 2005.

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

Paper V: High-Precision Power Modelling of the Tegra K1 Variable SMP Processor Architecture

Title: High-Precision Power Modelling of the Tegra K1 big.Little Processor Architecture [65].

Authors: K. R. Stokke, H. K. Stensland, P. Halvorsen and C. Griwodz.

Abstract: Energy efficiency is an important issue for many embedded systems, where limited battery lifetime and power-hungry hardware constrain the usefulness of such devices. Modern Systems-on-Chip (SoCs) such as the Tegra K1 employ advanced power management capabilities such as two CPU clusters, clock-gating, power-gating and dynamic frequency tuning to meet application demands. At design or runtime phases, it is challenging for system architects and software developers to understand the effects that these mechanisms have in terms of power and performance in all parts of the system. This is because it is impossible to measure directly the power usage of cores, caches, memory and other hardware components. Rate-based power models are often proposed as a solution for this, unfortunately these can mispredict substantially on the Tegra K1 up to 30 %. In this paper, we propose a power modelling method for the Tegra K1 CPU which overcomes the limitations of the most common types of models found in literature, but still only requires power measurement of the board. Through extensive empirical validation, we demonstrate an accuracy which is close to 100 %. Preliminary experiments show that our methodology is able to capture instruction power of individual system processes and applications and produce detailed power breakdowns of all components in the system.

Lesson's learned: In this paper, we complete the GPU model in Paper IV with an improved CPU model. Both CPU clusters are included, and we additionally modelled the CPU's cache hierarchy. In terms of the modelling effort, a challenge with the Tegra K1's CPU clusters is that they do not support fine-grained accounting of the type and number of instructions executed. However, through our experiments we learned that the instruction power of our workloads can be modelled with only a generic instruction counter. After the work in this paper, we learned several critical insights that we have used in this thesis to improve the CPU model. First, in the

purpose of modelling CPU cycle cost, we use a kernel tracing framework to track clock-gating in the CPU cores. However, we learned that the CPU cores are additionally clock-gating the cores in hardware. Therefore, the number of active cycles must be measured with a PERF HPC. Second, our model relies on PERF to measure hardware activity. However, we discovered that PERF sometimes multiplexes event counting through the CPU's available HPC slots, even if there are in principle enough HPC slots to track all the required HPCs. Therefore, we had to avoid tracking our PERF HPCs on a per-process basis, and instead only measure the total event counts over all processes on the CPU. Third, we learned that our hard-coded voltage-frequency tables do not capture variations in the HP rail voltages accurately enough. Therefore, the model accuracy could be improved by also associating the HP rail voltage with the number of currently active CPU cores.

Author's contributions: Stokke designed and implemented the model training benchmarks for the CPU and the video filter operations that were used to verify the approach. He also designed the experiments, and is the main author of this paper.

To appear in: Proceedings of the 10th International Symposium on Embedded Multicore/Many.core Systems-on-Chip (MCSoC), Lyon (France). IEEE, 2016.

High-Precision Power Modelling of the Tegra K1 Variable SMP Processor Architecture

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

***Abstract*—Energy efficiency is an important issue for many embedded systems, where limited battery lifetime and power-hungry hardware constrain the usefulness of such devices. Modern Systems-on-Chip (SoCs) such as the Tegra K1 employ advanced power management capabilities such as two CPU clusters, clock-gating, power-gating and dynamic frequency tuning to meet application demands. At design or runtime phases, it is challenging for system architects and software developers to understand the effects that these mechanisms have in terms of power and performance in all parts of the system. This is because it is impossible to measure directly the power usage of cores, caches, memory and other hardware components. Rate-based power models are often proposed as a solution for this, unfortunately these can mispredict substantially on the Tegra K1 up to 30 %. In this paper, we propose a power modelling method for the Tegra K1 CPU which overcomes the limitations of the most common types of models found in literature, but still only requires power measurement of the board. Through extensive empirical validation, we demonstrate an accuracy which is close to 100 %. Preliminary experiments show that our methodology is able to capture instruction power of individual system processes and applications and produce detailed power breakdowns of all components in the system.**

I. Introduction

Power usage is a timely topic in multicore embedded systems. Devices such as smart phones, drones and laptops are limited by a combination of power-intensive hardware and limited battery capacity. For such devices, it is important that they stay alive for as long as possible while at the same time providing acceptable quality of service. One example is video processing. According to Cisco, by 2019, the sum of all forms of video will be in the range of 80 to 90 percent of global consumer traffic, and of this, 14 percent will be mobile data traffic [4]. When recording live events such as sports using your smart phone or even a drone, it is important that the device can manage the limited energy resource.

It is well known that state-of-the-art power management generally reacts too quickly to changes in hardware utilisation, and consequently end up staying at unnecessarily high CPU and RAM operating frequencies [13], [15], [17], [22]. For system architects it is therefore important to fine-tune software algorithms, gating techniques, platform frequencies and workloads for the different parts of the system at early design stages, or even design intelligent power management schemes at runtime. However, this facilitates the need to understand power usage of individual units in the Tegra K1 Variable SMP (vSMP) [12] processor architecture, such as the Tegra K1's Low Power (LP) core, High Performance (HP) cluster, caches, RAM banks and clocks under the influence of software, which is often unfeasible due to circuit-layout limitations.

It is not trivial to understand power usage of embedded SoCs. Simply measuring power is a serious challenge due to a lack of power measurement sensors. Research has also shown that such sensors are inaccurate [5] and often only capable of measuring total power usage. This includes individual power draw from processor elements, buses, caches, graphics processing units, hardware accelerators, regulators [3] and other components [2]. Researchers often attempt to *model* power usage of individual components by correlating total power usage with hardware activity and/or hardware states, and three model types are dominant in literature: *state-*, *rate-* and *cmos-based* power models.

Aside from their individual strengths and limitations, the three different modelling methodologies share a common deficiency: they have the potential for serious misprediction of power on the Tegra K1. This flaw has not been apparent before because the model implementations are not tested extensively enough over *all CPU and memory frequency combinations*. Rate-based power models, which are by far the most commonly found in literature [5], [8], [21], [23], can for example mispredict up to 30 % on the Tegra K1 when the CPU frequency is above 1 GHz. This occurs for several reasons. Rate-based models correlate total power usage with hardware activity, such as the number of instructions, cycles or cache misses per second. However, many mechanisms that have non-negligible impact on power usage, such as frequency scaling, variable cost of instructions, resource contention [1] and clock, core and rail gating are usually ignored. Additionally, rate-based models do not consider that rail voltages vary with operating frequencies, having an adverse affect on power usage [3], [9], [15]. For this reason, the accuracy of rate-based models is entirely dependent of the frequency levels selected by Dynamic Voltage and Frequency Scaling (DVFS) algorithms at the time of the verification.

Models are necessary to understand the power usage of small hardware components where direct measurement is otherwise physically impossible or unsupported. They are also necessary to attribute energy consumption to individual software applications, determine and reduce the power usage of components and to evaluate how systems consume power. However, existing modeling methods fail to predict power accurately on the Tegra K1.

In our previous work [18], we developed a high precision power model for the GPU on the Tegra K1. This model included a very simplified model for the Tegra K1's LP core. In this paper, we extend this model to predict power usage of the Tegra K1's quad-core CPU, different CPU caches and memory. The accuracy is close to 100 % for multimedia

workloads such as the Discrete Cosine Transform (DCT), huffman encoding, motion vector search and rotation. The main contribution compared to the GPU model is that the CPU cannot be modelled *generally*. This is because there is a lack of Hardware Performance Counters (HPCs) that can measure individual integer, floating point, data movement and other types of instructions. Instead, our method for the CPU estimates the average capacitive load per instruction on a per-process basis. By taking into account measured rail voltages and fine-grained hardware activity predictors, our model exposes detailed insight into the power usage of individual components, such as rail and core leakage currents, RAM and CPU clock, RAM reads and writes, application-specific workload power and cache hierarchies. Using the model, it is possible to identify power-intensive hardware components and software workloads. This can for example be used to optimise the power usage of individual system services, reducing idle-system workload power by 21 %. Combined, our GPU and CPU models comprise a *complete* heterogeneous power model for the Tegra K1, covering all general purpose compute aspects of the SoC from its dual-cluster CPU, RAM and GPU.

The rest of this paper is organised as follows. We survey related work and background on power modelling in Section II. Section III describes the Tegra K1 platform and the details that are important to model the platform accurately, in particular, why it is impossible to build an entirely generic power model for the Tegra K1's CPU. The modeling methodology, model training benchmarks as well as the coefficients are introduced in Section IV. Furthermore, we extensively verify our model over all platform frequencies using several intense compute workloads in Section V, where we also investigate workload-dependent power components and some preliminary use cases of our model. We conclude our work in Section VI.

II. Background and Motivation

There is an abundance of work which attempts to model power usage of embedded systems. These generally fall into three categories of models: state-, rate- and cmos-based models. Each type has its own advantages and disadvantages; but as mentioned in the previous section, they can mispredict badly depending on operating frequencies. Surprisingly, all of them contribute with important insight into power, and they all hold some merit which is key to building accurate models. In this section, we take a brief look at these model types and study their accuracy on the Tegra K1. More extensive details about model predictors and estimated coefficients for these are available for download here[1].

Modern SoCs are based on CMOS technology and are built with *voltage-islands*. This means that the components within the SoC (clock generators, caches, buses, compute cores etc) are divided into regions supplied by individual *power rails* (see Figure 1). The power on a rail P_R is the sum of static $P_{R,stat}$ and dynamic $P_{R,dyn}$ power [9], [3] and make up the foundation for cmos-based models:

$$P_R = I_{R,leak} V_R + \alpha_R C_R V_R^2 f_R \quad (1)$$

In this equation, rail voltage V_R and leakage current $I_{R,leak}$ defines the static power component. Dynamic power is caused

[1] folk.uio.no/krisrst/papers/mcsoc/2016/model.ods

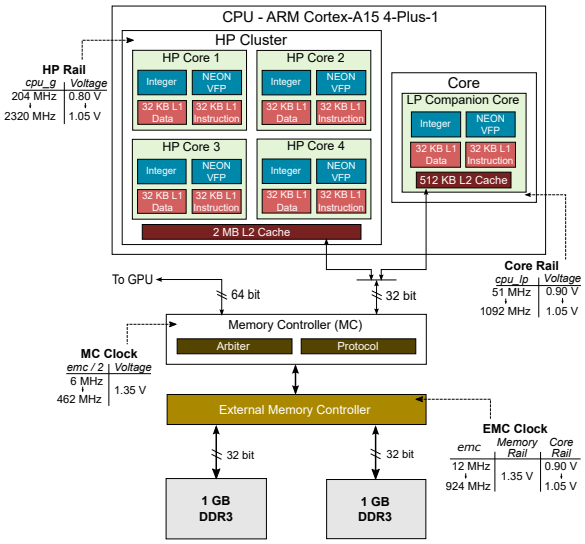
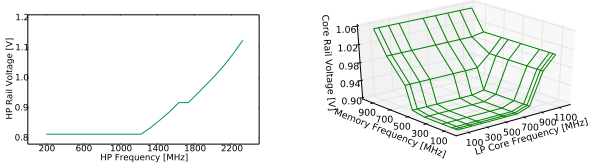


Fig. 1: Overview of the Tegra K1 SoC architecture [11].



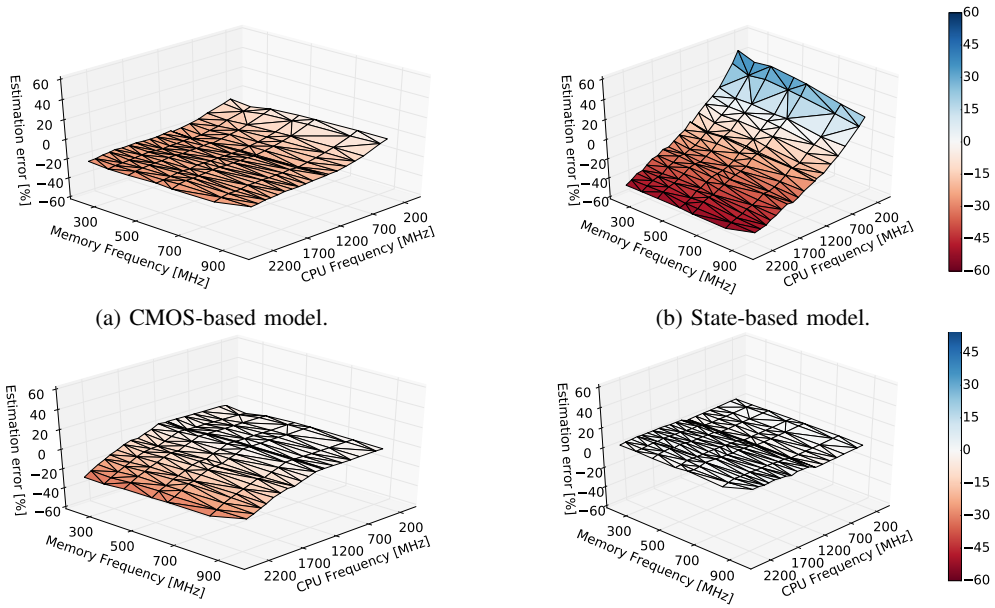
(a) The HP rail. (b) The core rail.

Fig. 2: Measured rail voltages.

by transistor switching activity, where f_R is the operating frequency in cycles per second on that rail, C_R is the potential maximum switching capacitance per cycle (in coloumbs per volt), and $\alpha_R \in [0, 1]$ is best viewed as a workload-specific factor which decides how much of C_R is being switched through the circuitry on that rail, per cycle. A key point from Equation 1 is that rail power is strongly dependent on rail voltage, which again depends on operating frequencies in that domain (see Figure 2).

Several authors [3], [15] have attempted to estimate leakage currents $I_{R,leak}$ and switching capacitance $\alpha_R C_R$ for rails directly using regression. However, despite being theoretically well-founded, cmos-based models mostly mispredicts between 20 to 25 % (see Figure 3a). This is a trend we also discovered in our cmos-model [15] which occurs because the model assumes an independent relationship between switching activity and frequency levels in different domains. It is implicitly assumed that the increase in power as for example CPU frequency increases, is caused only by increased switching activity on that rail. However, it is easy to imagine cases where this is not true, for example when executing a memory intensive program.

State-based models are perhaps the simplest model type found in literature [16]. Models of this type abstract hardware components into states and associate each with a constant power draw and transition cost between these [19]. For example, the Tegra K1's CPU can be abstracted into two states that reflect the currently active cluster, where either the Low Power (LP) core or High Performance (HP) cluster is active. It is further possible to add the state cost of having individual cores active. State-based models are relatable to the change in



(a) CMOS-based model.

(b) State-based model.

(c) Rate-based model.

(d) Our hybrid model.

Fig. 3: Power estimation error for common model types running a DCT filter (four cores).

leakage current $I_{R,leak}$ on power rails which is directly tied to the activation and deactivation of hardware components such as cores and rails. However, they have bad accuracy on the Tegra K1 because they ignore changes in rail voltages and dynamic power completely (see Figure 3b).

Rate-based models are the most intensively used models in the literature [5], [8], [14], [20], [21], [23] and has seen widespread adoption since 1999 [6] (which is the first rate-based model we have found). Rate-based models attempt to correlate power usage with the rate at which hardware events occur according to the following formula:

$$P_{tot} = \beta_0 + \sum_{i=1}^{N_\rho} \rho_i \beta_i \tag{2}$$

In Equation 2, β_0 is the power of idle components with constant power draw, ρ_i is a predictor (hardware event) with units of accesses per second, β_i is the cost in Watt-seconds per access and N_ρ is the number of predictors. Example of hardware events can be instruction execution, elapsed cycles, cache hits and misses and branching. These events naturally reflect transistor switching activity, tying it to the *dynamic power* component of Equation 1. With a prediction error between 0 to 30 %, this model predicts better than the state-based one (see Figure 3c). However, rate-based models ignore rail voltages and static power. The only known exception is Hong, S. and Kim, H. [7] who presented a standard rate-based model, but it considers voltage as a part of the leakage current. In defence of rate-based models, ignoring voltage levels might not have a negative effect if power management is poorly designed. For example, it is much easier to stick the Tegra K1 HP frequency at 1.15 V (see Figure 2a) independently of frequency. It does however have an adverse effect on systems where this is not the case, such as the Tegra K1.

In summary, state-based models reflect static power, but do not capture dynamic power. Rate-based models are the opposite. These reflect hardware utilisation and dynamic power but not static power. Both ignore changes in voltage as a result of changes in frequencies, and is a key source of inaccuracy in these model types. Cmos-based models incorporate both static and dynamic power as well as rail voltages. In terms of accuracy, it shows less error variation, but has an inherent weakness in that they assume independency in switching activity and operating frequencies between rails. Our hypothesis is therefore that we can achieve better accuracy if hardware utilisation and switching capacitance can be captured more accurately and independently on each rail. In the following sections, we will outline important hardware activity predictors on the Tegra K1's CPU and how these can be combined with voltage measurements to achieve close to 100 % accuracy (see Figure 3d).

III. Tegra K1 Mobile SoC

In order to successfully build a high-precision power model, it is important to have a solid understanding of the platform (see Figure 1). We have chosen the Tegra K1 as a case study due to its similarity with modern island-style SoCs. In this section, we introduce the most important power rails and clocks, and we describe how it is possible to monitor hardware activity in the various units. Power measurement and synchronisation is done as in our previous work [15], where the Jetson-TK1 retrieves power measurement readings from a Keithley 2280S power source via a dedicated measurement machine. We measure individual rail voltages using the Keithley 2110 high-precision voltage measurement unit.

A. Tegra K1 Architecture: Rails and Clocks

The Tegra K1 is a complete SoC featuring several *rails* powering various functional units on the processor [10]. The rails which are relevant for our investigation (see Figure 1) are the *core*, *HP* and *memory* rails. These rails power the LP (core) and HP clusters, as well as the External Memory Controller (EMC) and memory banks. The power usage on all other rails are assumed to be constant with no hardware activity.

Clocks drive switching activity inside the Tegra K1's hardware components. Every clock cycle consumes energy, and CPU clock cycles normally correlate very strongly with

Clock	Affects	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	$[51, 1092]$	$[0.90, 1.05]$
EMC	Core Rail	Memory	6	$[204, 924]$	$[0.90, 1.01]$

TABLE I: Tegra K1 cores, clock and voltage ranges.

power in rate-based models [20]. Clocks are also provided as a power management mechanism where the operating frequency of a component can be reduced to mitigate power dissipation (while reducing performance). The Tegra K1 features a range of different clocks driving different functional blocks (see Table I). When building power models, it is important to be aware that the voltages on different rails change, and that they change with clock frequency [3], [9]. For example, depending on the HP cluster clock (cpu_g), HP rail voltage V_{hp} varies between 0.81 and 1.13 V (see Figure 2a). The core rail voltage is more complex. Its voltage V_{core} depends on the LP core frequency (cpu_lp) as well as the EMC bus frequency. Unless the HP cluster is active, the voltage set on this rail is always the maximum required by any of these two clocks at any point in time (see Figure 2b). Finally, memory rail voltage is statically set to 1.35 V and does not change.

B. Power Saving Mechanisms

The Tegra K1 features several mechanisms to limit power usage depending on the current demand for resources and hardware utilisation. Generally, there are two types of optimisation approaches to achieve this goal, which have never been taken directly into account in previous models:

- *Clock gating* disables clock distribution to various entities of the SoC, which disables their function and reduces dynamic power. State is retained.
- *Power gating* disables supply voltage to various parts of the circuity. This completely removes static power draw and normally implies clock gating. Power gating causes loss of state (for example in caches).

Exactly when and for how long cores are gated is decided by both software and hardware. On the Jetson-TK1, the CPU-idle kernel drivers monitor idle CPU cores and make decisions to clock- or power-gate individual cores based on estimated idle intervals and state transition overheads. Additonally, the CPU cores perform clock-gating in hardware without the intervention of software.

Application demand for processing time is satisfied by automatic cluster and core selection as well as CPU and memory frequency tuning. All of these mechanisms impact power usage. Typically, when the Tegra K1 is idle, processing is restricted to the LP core, in which case the entire HP rail is power gated by disabling the HP rail voltage regulator. Kernel drivers monitor resource usage. These may decide to switch between the application clusters, activate additional CPU cores, clock- and power-gate individual cores in short idle periods and adjust EMC and CPU frequency. The challenge with regards to power modelling is to track when and for how long processors or other parts of the circuitry remains gated. To solve this, we modified the Tegra K1 kernel sources to count the time spent in the gated states and expose this information to running applications.

C. Instruction Cost (Workload-Dependent Power)

One of the main challenges of power modelling the Tegra K1 is that it does not have fine-grained accounting for the types and number of instructions executed. Pricopi et. al. [14] built a rate-based power model for a big.Little CPU, which is the same cluster technology the Tegra K1 uses. They show that it is possible to attribute an instruction cost to different instructions such as branching, integer and floating point operations, as well as RAM loads and stores. However, the CPU implementation they used had dedicated HPCs to count these instructions. This is also in accordance with our experiences on the Tegra K1's GPU [18], where we build an fully generic power model using fine-grained instruction accounting. Unfortunately, the Tegra K1 does not implement these HPCs on its CPU.

When there is no possibility to trace what types of instructions have been executed, loss of generality in terms of power modelling is unavoidable. Every application and system service has its own way of exersising the processing pipeline using different instructions. The only instruction counter we can use is the `instruction executed` HPC, which counts the number of instructions executed. Since the number of CPU instructions can be counted on a per-process basis, we choose to track these and estimate an average capacitance load per instruction for each workload. The assumption behind this is that each application will have roughly the same average capacitive load over time. ARM CPUs are not required to implement HPCs for fine-grained instruction accoutning, which means that for CPUs it will be unavoidable to model workload power in this way.

D. Cache Maintenance and Off-Chip Memory Access

The Tegra K1 implements a two-level on-chip cache hierarchy with last-recently used eviction policy. Cache costs usually surface as a part of rate-based power models, where authors attempt to attribute power costs to cache accesses and misses directly [20]. In our model training, we found that these events generally do not correlate well with power, often breaking the estimation such that coefficients are negative. This is not easy to understand, but we believe that the cost of *accessing* caches is not constant. A cache access may end up in other data being evicted elsewhere or trigger refills from other cache levels or memory: the cost is not *generic* in itself and depends on many other factors.

Because of the problems of modelling cache accesses, we decided to attempt to model instead the power which is spent *maintaining* the cache. The intuition behind this is that, if we can model the way the cache maintains its own consistency, cache accesses will instead just be an integral part of the average instruction cost (see Section III-C). The ARM HPC implementation has a range of cache performance counters. We use the following HPCs:

- **L1D_CACHE_REFILL.** Counts data cache refill events from external, off-chip memory.
- **L1D_CACHE_WB.** Counts data cache writebacks to external, off-chip memory.
- **L2D_CACHE_REFILL.** Counts data cache refill events from L1 cache or external, off-chip memory.

- **L2D_CACHE_WB.** Counts data cache writebacks to L1 cache or external, off-chip memory.

We then defined two new counters that trace cache maintenance operations between the cache hierarchies:

$$\rho_{l1l2} = (N_{l1,refill} - N_{l2,refill}) + (N_{l1,wb} - N_{l2,wb}) \quad (3)$$

$$\rho_{l2ram} = N_{l2,refill} + N_{l2,wb} \quad (4)$$

where ρ_{l1l2} is an indication of on-chip cache maintenance between L1 and L2 data caches, and ρ_{l2ram} is an indication of cache traffic between L2 and off-chip memory. Note that the Tegra K1 also implements other types of caches, for example L1 instruction cache. Since only seven HPCs can be occupied concurrently, and one of these always is the `active cycle` HPC, there is not enough HPC space to trace all the hierarchies.

Unique memory accesses ultimately end up in off-chip memory accesses at least once. Exactly how often this happens is very hard to trace. CPU HPCs exist, but there may be many other components in the system which also access RAM, such as PCI devices and peripherals. Fortunately, the Tegra K1 implements an activity monitor which can provide the number of active memory cycles spent serving the CPU or other peripherals. We modifiy the kernel driver for the activity monitor to provide us with these statistics, which represent our measure of memory utilisation.

IV. High-Precision Power Modelling

We outline our methodology to model the Tegra K1 power usage with high precision. The method is an extension to our previous work on the Tegra K1's GPU [18]. The main challenge with modelling power on the Tegra K1's CPU compared to the GPU is that it is impossible to build an entirely generic model. This is because the CPU does not have dedicated HPCs for the type and number of instructions executed (see Section III-C). Therefore, the resulting model is a combination of *generic* power (most of the predictors seen in Table II) and *workload-specifc* power, which varies depending on the various types of instructions executed on each core.

A. Derivation

Following our discussion in Section II, we now describe the dynamic power component of Equation 1 in terms of measurable hardware activity predictors as outlined in Section III. The dynamic part of Equation 1 can be re-interpreted as follows:

$$P_{R,dyn} = \sum_{i=1}^{N_R} C_{R,i}\rho_{R,i}V_R^2 \quad (5)$$

In Equation 5, N_R is the number of hardware predictors, $\rho_{R,i}$ is a hardware activity predictor for rail R in occurrences per second (for example instructions per second), and $C_{R,i}$ is the unknown switching capacitance per occurrence of event $\rho_{R,i}$. Static power on a rail R is the product of that rail's voltage V_R and total leakage current $I_{R,ltot}$ [9]:

$$P_{R,stat} = V_R I_{R,ltot} \quad (6)$$

The Tegra K1 continuously performs power gating of processors on the HP and core rails. When power gating a core, the leakage current $I_{cpu,leak}$ from that core is also removed. The total leakage current on the HP and core rail, for N_c active cores, is:

$$I_{R,ltot} = I_{R,leak} + \sum_{i=1}^{N_c} I_{cpu,leak} \quad (7)$$

The total power usage of the Jetson-TK1 becomes:

$$P_{jetson} = \sum^{R\in\mathbb{R}} (P_{R,dyn} + P_{R,stat}) + P_{base} \quad (8)$$

Noting that the static power on the memory rail cannot be modelled, because the rail voltage on this rail does not change. It is instead a part of base power P_{base}, which also includes the constant power draw of all other rails and idle components.

B. Methodology

The total power usage of the Jetson-TK1 is shown in Equation 8, where the unknown variables are the capacitive loads, rail and core leakage coefficients ($C_{R,i}$, $I_{R,leak}$ and $I_{cpu,leak}$). Base power P_{base} is also unknown. Our methodology to find these terms is based on multi-variable, linear regression. We create seven benchmarks spesialised to stress different architectural units of the Tegra K1's processor (see Table III):

- We have several versions of a simple matrix-multiply program, where the element type, for example integer, floating point (VFP) or vectorised floating point (NEON), is being varied.
- An idle benchmark, where only the Linux kernel and system services are running, is useful to trigger power- and clock-gating mechanisms.
- We found it necessary to also implement specialised benchmarks to stress L1 cache refills and write-backs more than the other benchmarks did.

Each benchmark is run over all possible memory and processor frequency combination (see Table I), and over the five possible core combinations (LP core or any number of the four HP cores active). During the benchmarks, power is being logged while the HPCs are being collected at regular, short intervals of 100 ms. This is to avoid counter overflow, which occurs relatively easily for some of the HPCs (under loads, the active cycle counter overflows within two seconds at the maximum operating frequency). The final dataset size is of 30912 entries containing the necessary predictors and power measurement samples. The coefficient estimates can be seen in Table II.

As argued in Section III-C, it is not possible to generalise the cost of instruction execution because different instruction types (integer, floating point, data movement) have different costs. The Tegra K1 only has one instruction counter, and the estimated cost per instruction will vary depending on workload. We make a single instruction cost estimate for each of the benchmarks in Table III. The estimates are shown in Table III. Not considering the IDLE test, which has a significantly larger instruction cost, the cost is similar across the rest of the benchmarks. Higher instruction cost does not automatically imply a higher rate of energy consumption. For example, the IDLE test consumes less power on average than

Rail	Number	Predictor	Description	Coefficient	Value
HP	0	V_{hp}	HP rail voltage (when powered)	$I_{hp,leak}$	$59.8mA$
	1	$\rho_{hp,clk1}$	Active clock cycles per second (first core)	$C_{hp,clk1}$	$395.65\frac{pC}{V}$
	2	$\rho_{hp,clk2}$	Active clock cycles per second (second core)	$C_{hp,clk2}$	$270.40\frac{pC}{V}$
	3	$\rho_{hp,clk3}$	Active clock cycles per second (third core)	$C_{hp,clk3}$	$261.89\frac{pC}{V}$
	4	$\rho_{hp,clk4}$	Active clock cycles per second (fourth core)	$C_{hp,clk4}$	$213.95\frac{pC}{V}$
Core	0	V_{core}	Core rail voltage (always powered)	$I_{core,leak}$	$633.7mA$
	1	$\rho_{core,clk}$	Active clock cycles per second (LP core)	$C_{core,clk}$	$301.49\frac{pC}{V}$
Common	0	$V_{com,online}$	Rail voltage when any core is online (not gated)	$I_{cpu,leak}$	$24.00mA$
	1	$\rho_{com,l1l2}$	Cache maintenance, L1 and L2	$C_{com,l1l2}$	$2.35\frac{nC}{V}$
	2	$\rho_{com,l2ram}$	Cache maintenance, L2 and RAM	$C_{com,l2ram}$	$2.29\frac{nC}{V}$
	3	$\rho_{com,ips}$	Instructions per second (workload-specific)	$C_{com,ips}$	See Table III
Memory	0	$\rho_{mem,clk}$	Total clock cycles per second	$C_{mem,clk}$	$238.21\frac{pC}{V}$
	1	$\beta_{mem,204}$	Power offset at 204 MHz	$P_{mem,204}$	$11.80mW$
	2	$\beta_{mem,300}$	Power offset at 300 MHz	$P_{mem,300}$	$69.00mW$
	3	$\rho_{mem,CPU}$	CPU busy memory (EMC) cycles per second	$C_{mem,cpu}$	$2.15\frac{nC}{V}$
	4	$\rho_{mem,OTH}$	Other busy memory (EMC) cycles per second	$C_{mem,oth}$	$2.76\frac{nC}{V}$
Other	P_{base}		Base power	-	$0.87W$

TABLE II: Overview of generic energy model predictors and coefficients.

Benchmark	Description	Components under explicit stress						Instruction Cost
		RAM (CPU)	L1 - L2	L2 - RAM	INT	FPU	NEON	
Idle CPU	CPU idle.		✓					$2.50\frac{nC}{V}$
L1-WB	L1 writeback stress.		✓					$0.15\frac{nC}{V}$
L1-RF	L1 refill stress.		✓					$0.18\frac{nC}{V}$
MMUL-INT	Matrix multiply (integer operations).	✓	✓	✓	✓			$0.45\frac{nC}{V}$
MMUL-INT-VOL	Same as above, volatile memory.	✓	✓	✓	✓			$0.27\frac{nC}{V}$
MMUL-F32	Matrix multiply (floating point operations).	✓	✓	✓		✓		$0.36\frac{nC}{V}$
MMUL-NEON	Matrix multiply (NEON floating point operations).	✓	✓	✓			✓	$0.44\frac{nC}{V}$

TABLE III: Benchmarks and components under stress.

the other benchmarks, despite the high estimated instruction cost. This is because so few instructions are executed compared to the other benchmarks. Similar conclusions can be made to the other benchmarks. Comparing for example floating point matrix multiply with VFP or NEON instructions, the NEON variant has a higher estimated cost per instruction. However, NEON instructions process four floating point values at a time whereas the VFP variant only process one at a time. Our training data therefore shows that the NEON variant draws less average power than the VFP variant.

V. Experiments

In this section, we verify the accuracy of our model by extensive validation over all operating frequencies. The workloads we use are common video processing operations for the DCT, Huffman coding, motion vector search and rotation. Due to space limitations, we are unable to describe these here, but interested readers can refer to our previous work for details [18]. We also argue that our method to model workload-specific instruction power is correct, and finally present some preliminary case studies of potential use cases for the model.

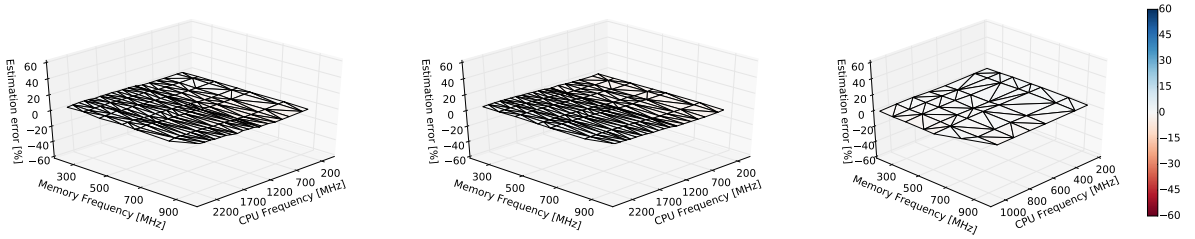
A. Verification

To verify our model, we let our video processing filters process an HD stream at 25 FPS over all possible operating frequencies. This process is done once for each core configuration (five possible combinations where either the LP cluster or any number of the four HP cores are active). While running, our implementation estimates generic platform power every 100 ms (see Section III-C) using the predictors in Table II and additionally logs measured total power usage. The workload-dependent power usage is now missing, and must be estimated per workload. When each filter is done processing the HD stream at all frequency combinations, per-instruction switching capacitance is estimated by subtracting predicted generic platform power from total measured power. The residual power is always positive and is used in a linear regression solver to estimate the switching capacitance per instruction, which is in turn used to estimate the power component directly related to instruction execution.

The resulting prediction error can be seen in Figures 3d and 4 for all of our filters. Due to space restrictions, we can not show all the error plots for each core configuration, but we include plots across all of them for at least one filter. The plots show that our model performs well, with a high accuracy across all frequency domains which is very close to 100 %. In general, the average accuracy over all frequencies is never below 98 % for any filter, with worst-case prediction errors that are never worse than $\pm$ 4 %. This is much better than the related modelling methods in Figure 3, where our model not only successfully captures hardware utilisation (dynamic power), but also takes into account increased power from higher rail voltages and leakage currents (static power).

B. Residual (Workload/Instruction-Dependent) Power

Our verification methodology has an unavoidable weak point in the way workload-dependent power is estimated. Per-instruction capacitance loss is estimated based on *residual* power *after* generic power components are removed from real measurements. The estimated workload power is subsequently *added* to the power prediction. It is possible to claim that this estimated workload-dependent power component "simply happens to nicely cover" most of the residual power which would otherwise be the root cause of a large prediction error. It is difficult to conclusively disprove this claim. Per-instruction capacitance loss is tied to software implementation; it can not be re-used for other pieces of code and it cannot be guaranteed to be the same if the code is changed by recompilation or by changing input data (for example, reducing video resolution or changing algorithmic parameters). It is, however, possible to show in some more detail how residual workload power behaves across frequencies, how diverse per-instruction capacitance loss estimates are and how they can be used in practice.



(a) Rotation filter (three cores). (b) Huffman filter (one HP core). (c) Motion vector search (LP core).

Fig. 4: Power estimation error for idle and video processing scenarios.

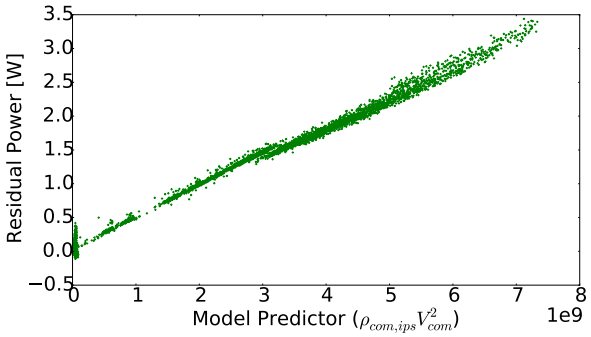


Fig. 5: Residual (workload-specific) power plotted for the model predictor (square-voltage instructions per second).

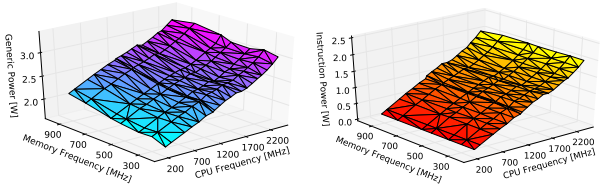
When generic power has been removed from measured power, the residual power P_{res} should reflect only the cost of executing instructions (see Equation 5):

$$P_{res} = C_{com,ips}\rho_{com,ips}V_R^2 \tag{9}$$

In Equation 9, the unknown variable is the per-instruction capacitance loss $C_{com,ips}$. Figure 5 shows the residual power P_{res} for the DCT filter running on four cores. Each residual, which represents a unique sample at an operating frequency point, is plotted for its corresponding predictor ($\rho_{com,ips}V_R^2$ in Equation 9). From the figure, we can see that residual power follows a linear trend with its predictor, which is as expected from Equation 9 and confirms our theory that instruction power can be modelled this way. At the data points around $3E9$, we can see that the residuals "drop" by some 50-60 mW. This is because the rail voltage reading, which is stored as a frequency-dependent static table in our code, is slightly incorrect: buck regulator output voltage is unpredictive and varies with output current. This is also visible in Figure 3d at around 1 GHz CPU frequency.

The gradient of the line in Figure 5 is the estimate of switching capacitance $C_{com,ips}$. This is easily found using regression, which yields a capacitive load of $439\frac{pC}{V}$ per instruction. We also use the estimated constant offset in the model, which is rarely above 30-40 mW. For the rest of the filters, estimated capacitive load is never above $607\frac{pC}{V}$ (HP, single-core DCT) and never below $197\frac{pC}{V}$ (LP core, huffman). In general, for all the filters, the more cores are active the higher the capacitive instruction load.

The capacitive load itself does not offer any insight into the actual power usage of instructions. This depends on how many of these are executed per second. To investigate this, we plot the generic and workload-specific power in Figures 6a and 6b. We see that instructions can account for a substantial fraction of the total estimated power. Not considering base power P_{base}, instruction power can account for up to 50 % of



(a) Generic components. (b) Instructions.

Fig. 6: Estimated power.

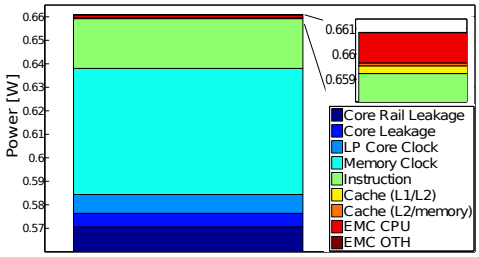


Fig. 7: Idle system power breakdown.

the total power and varies between 0.4 to 2.0 W. This means that instruction power is an important power component, and that care must be taken to estimate it correctly to achieve a high prediction accuracy. However, this may not always be easy to achieve in practice. Our workloads are *cyclic* in that they continuously perform the same work (and instructions) on new incoming frames. It is easy to imagine scenarios where it may not be so easy to estimate average instruction cost, for example for workloads which are not cyclic or incorporate heavy branching (not always following the same code paths). It is for example tempting to hypothesise that instruction cost could be estimate along code branches. SoCs with fine-grained instruction accounting (as for the Tegra K1's GPU) can potentially solve this problem by enabling fully generic models, but in practice, it is likely that there will always be SoCs which only implement HPCs for the total number of executed instructions.

C. Use Cases

We now conduct a set of experiments to show how high-precision power modelling can be useful to developers and system architects. A basic use case is to produce power breakdown of the system, in order to analyse the power consuming components and identifying important candidates to minimise power usage. For example, Figure 7 shows the contribution to power from each of the generic power components. The system is idle, and under the control of the standard power management algorithms where the CPU and memory are running generally underutilised at the same frequency (204 MHz). We can see that cache maintenance and off-chip RAM accesses

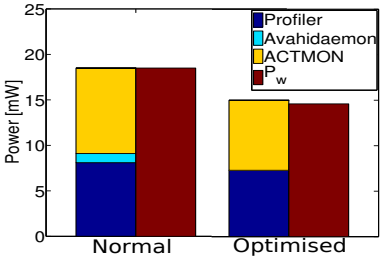


Fig. 8: Instruction power breakdown of the three dominating idle system services.

are virtually non-present. Workload power is negligible, only accounting for about 20 mW. The memory clock accounts for the second most substantial power component. This is interesting as the capacitance loss per cycle is comparative with the CPU and they are both running at the same frequency. However, the CPU is better at handling underutilisation with clock- and power-gating and is consequently better at hiding the unnecessary power overhead. The memory, however, is not clock-gated and therefore ends up wasting cycles which are not used. Among the static power contributors, the core rail leakage current is very large at 570 mW, but little can be done to this component other than ensuring that the core rail voltage is as small as possible.

Our methodology is also fine-grained enough to estimate instruction power of individual system services. To show that it is able to handle very small changes in instruction power, we focus on the three most dominating system services for the idle system. The residual workload power from these is only 20 mW and the breakdown can be seen to the left in Figure 8 for the profiler (our modelling and estimation tool), the avahidaemon and an activity monitor which logs RAM utilisation. Studying the source code for the activity monitor, we found several optimisation points where we were able to remove expensive string comparison functions. We also disabled the avahidaemon, which is unnecessary. The result shows that we are able to reduce residual power by 20 % (P_w). While the actual impact of the system is small (some 5 mW), this proves that the model is indeed fine-grained and is able to pick up on very small changes in the system.

VI. Conclusion

Power models are important tools to diagnose modern embedded multicore SoCs at early design stages or for runtime power analysis and management. This is because it is not trivial to measure and attribute power of individual computational units, such as cores, caches, clocks and memory to applications. In this paper, we have shown that state-of-the-art power modelling methods can mispredict substantially on the Tegra K1. Our main contribution is a methodology which, by considering rail voltages and fine-grained hardware activity measurements, is able to predict power with an accuracy close to 100 %. It is not possible to build a fully generic model for the Tegra K1 because of a lack of fine-grained CPU instruction accounting. Instead, we extend our method from previous work [18] and show how instruction power can be measured on a per-process basis using only a single HPC for the total number of instructions executed. It has been extensively verified over all frequencies using several video processing workloads.

References

[1] R. Basmadjian and H. de Meer. Evaluating and Modeling Power Consumption of Multi-Core Processors. In *Proc of e-Energy*, pages 1–10, 2012.

[2] A. Carroll and G. Heiser. An Analysis of Power Consumption in a Smartphone. In *Proc of ATC*, 2010.

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

[4] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019, White Paper. http://www.cisco.com/c/en/us/solutions/collateral/service-provider/ip-ngn-ip-next-generation-network/white_paper_c11-481360.html.

[5] M. Dong and L. Zhong. Self-Constructive High-Rate System Energy Modeling for Battery-Powered Mobile Systems. In *Proc of MobiSys*, pages 335–348, 2011.

[6] L. M. Feeney. An Energy Consumption Model for Performance Analysis of Routing Protocols for Mobile Ad Hoc Networks. *Ad Hoc Networks*, pages 1–13, 1999.

[7] S. Hong and H. Kim. An Integrated GPU Power and Performance Model. In *Proc of ISCA*, volume 38, pages 280–289, 2010.

[8] W. Jung, C. Kang, C. Yoon, D. Kim, and H. Cha. DevScope: a Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components. In *Proc of CODES+ISSS*, pages 353–362, 2012.

[9] 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.

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

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

[12] NVIDIA. Variable SMP – A Multi-Core CPU Architecture for Low Power and High Performance. Technical report, 2011.

[13] A. Pathania, Q. Jiao, A. Prakash, and T. Mitra. Integrated CPU-GPU Power Management for 3D Mobile Games. In *Proc of Design Automation Conference*, pages 1–6, 2014.

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

[15] K. R. Stokke, H. K. Stensland, P. Halvorsen, C. Griwodz. Why Race-to-Finish is Energy-Inefficient for Continuous Multimedia Workloads. In *Proc of MCSoC*, pages 57–64, 2015.

[16] T. Simunic, L. Benini, P. Glynn, and G. De Micheli. Dynamic power management for portable systems. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 11–19, 2000.

[17] 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, 2015.

[18] K. R. Stokke, H. K. Stensland, P. Halvorsen, and C. Griwodz. A High-Precision, Hybrid GPU, CPU and RAM Power Model for Generic Multimedia Workloads. In *Proc of MMSys*, 2016.

[19] Y. Xiao. *Modeling and Managing Energy Consumption of Mobile Devices*. PhD thesis, 2011.

[20] Y. Xiao, R. Bhaumik, Z. Yang, M. Siekkinen, P. Savolainen, and A. Yla-Jaaski. A System-Level Model for Runtime Power Estimation on Mobile Devices. In *Proc of GCC and CPS*, pages 27–34, 2010.

[21] F. Xu, Y. Liu, Q. Li, and Y. Zhang. V-edge: Fast Self-Constructive Power Modeling of Smartphones Based on Battery Voltage Dynamics. In *Proc of NDSI*, pages 43–55, 2013.

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

[23] L. Zhang, B. Tiwana, Z. Qian, Z. Wang, R. P. Dick, Z. M. Mao, and L. Yang. Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones. In *Proc of CODES+ISSS*, pages 105–114, 2010.