Applying of Quality of Experience to System Optimisation

Sascha Bischoff[*,**], Andreas Hansson[**], and Bashir M. Al-Hashimi[*]

[*]Department of Electronics and Computer Science, University of Southampton, UK
[**]Research and Development, ARM, Cambridge, UK

***Abstract*—State-of-the-art mobile devices, such as smartphones and tablets, are highly versatile and must deliver high performance across a multitude of applications. The perceived performance and resulting user experience can make or break a design. It is therefore vital that device design and optimisation take into account the expectations and perceptions of the end user, as well as the types and requirements of the applications running on the device. Traditionally, device optimisation focuses on low-level metrics, such as CPU floating point performance or GPU frame rate, rather than on the aspects most important to the end user.**

In this work, we investigate the applicability of Quality of Experience (QoE) to system optimisation. We define a set of measurable QoE metrics, and run a set of experiments around a web browser and two graphics benchmarks. Using the results of these experiments, we show the advantages of using QoE as an optimisation metric by demonstrating the ability to optimally trade CPU performance for energy usage whilst taking into account the user experience. We investigate two GPU benchmarks to determine the ideal number of cores for energy efficiency, whilst ensuring a sufficiently high frame rate to maintain a high-quality user experience. We then look at the system as a whole, and the feasibility of using QoE to optimise performance and power consumption for the complete system, without sacrificing user experience. We achieve up to 60% savings in system energy usage with limited impact on the user experience.

I. Introduction

The processing capability of modern smartphones is increasing rapidly, as is the number and performance of the individual components [1], [2]. However, the power consumption is also rising, whilst battery capacity is not scaling at the same rate, and therefore there is ever-present pressure to deliver more power-efficient designs. Modern smartphones and tablets make use of a System-on-Chip (SoC) which combines the Central Processing Unit (CPU), Graphics Processing Unit (GPU), and memory system on a single piece of silicon. These SoCs are designed to be incorporated into many different types of system and hence are not tuned for any particular application. It is therefore vital that these can be optimised in situ in order to trade off device performance for energy usage.

There are numerous strategies to balance and trade off system performance and power consumption, such as Dynamic Voltage and Frequency Scaling (DVFS) and heterogeneous multi-processing [3], [4]. In these approaches, the system configuration is adjusted on the basis of low-level metrics such as the number of Floating-point Operations Per Second (FLOPS) or load on a particular core. However, these metrics give very little indication of how well the device performs for the end user who has little or no interest in the computational performance or memory bandwidth and is much more concerned with the responsiveness of the system for everyday activities [5]. Specifically, the performance perceived by the end user depends heavily on the applications running, how well the device is able to run these applications, and the user's expectations and preconceptions [6].

As we shall see, the low-level system performance is generally of little importance to the user. Thus it becomes possible to change the optimisation criteria to target low-power operation without affecting user experience. For example, it becomes possible to reduce the frame rate of the GPU as long as it remains above the minimum level required to perceive fluid motion [7]. Understanding the limits of human perception provides a basis for trading device performance for battery life without the user even being aware that this trade-off is taking place. However, the system must have knowledge of the limits of human perception and application requirements.

The notion of Quality of Service (QoS) is used to provide performance guarantees for a system. However, QoS focuses on temporal resource sharing, e.g. interconnect or memory bandwidth and latency. As it focuses on low-level performance metrics, it alone is insufficient to ensure user satisfaction. Quality of Experience (QoE), on the other hand, takes into account the perceptions of the end user and can be used as part of user-centric system optimisation. However, there are difficulties in determining which metrics to focus on and in modelling human perception [5], [8].

As the main contributions of this paper, we demonstrate the applicability of QoE to system level design, and highlight the range of trade-offs that can be made when the performance is measured in a user-centric way. The main contributions are:

1) Development of a set of metrics which can be used to measure the user experience for web browsing and 3D graphics.
2) Presentation and evaluation of QoE as an optimisation tool for system level design.
3) Establishing a link between system power consumption, low level performance indicators and QoE.

The rest of the paper is organised as follows. QoE is discussed in Section II. The simulation framework used to obtain empirical results is described in Section III, focusing on the CPU, GPU and memory models. The experimental results are presented in Section IV. Section V focuses on related work in this field and Section VI concludes the paper.

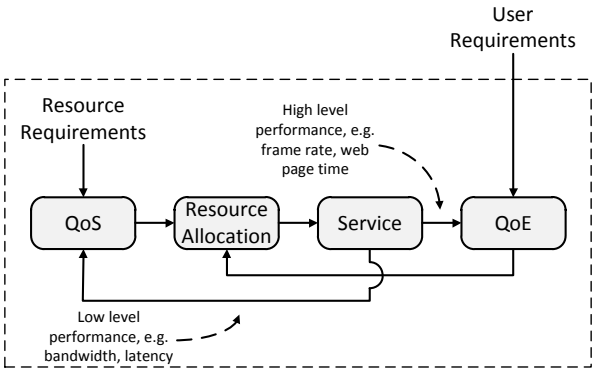


Fig. 1: Relationship between QoS, QoE, user requirements and system resources.

II. Quality of Experience

When consumers use a device, they have certain expectations such as the device's ability to play a particular game, long battery life, or simply being responsive. As long as the device is able to live up to these expectations, the user remains satisfied. If, however, the device is unable to satisfy the user's expectations he/she will become increasingly dissatisfied with it and will rate it very poorly for large disturbances in the delivered service. This has been highlighted by Fiedler et al. [6] who artificially generated QoS disturbances for web browsing and monitored user satisfaction. Therefore, great care needs to be taken when designing the system and corresponding QoS schemes that try to ensure a high quality service without disruption.

QoS schemes provide a mechanism for dividing up the available system resources to ensure that each component in the SoC, such as a CPU or a GPU, gets at least the minimum resources it requires, e.g., high bandwidth or low latency. As the load on the system increases beyond that which can be sustained, QoS sets the order of resource allocation failure, and the more important devices will receive their required resources first and at the expense of less important components. It is, however, no simple task to determine which components are the most important since this changes dynamically as the services delivered by the system change, and therefore require run-time adjustment.

CPU governors adjust the DVFS state based on the type of governor and the performance of the CPU [9], [10]. They are efficient at targeting high performance or extreme power efficiency. However, they are inadequate at preserving high user experience whilst operating at the minimum power required to do so. As we discuss in Section IV-B, by using the user experience to control the optimisation of the system, rather than lower-level metrics, it becomes easier to target the ideal operating point.

In Section IV we show that there is merit in explicitly using user experience for system optimisation. Hence, we need a set of measurable system level metrics which can be linked to the user experience. Specifically, a set of them needs to be defined which are able to quantify the user experience and provide an estimation of the experienced performance. We call this set of metrics Quality of Experience (QoE), which is a term used frequently by the telecommunications industry when referring to a delivered end-to-end service, such as streaming video [11]–[13]. More recently, applications are being viewed as delivering a service or set of services [14], and it is to the delivered services which we wish to apply QoE. QoE does not appear to have been explored in the context of system-level optimisation.

The relationship between QoS and QoE is presented in Figure 1 and illustrates how QoE can be used for system optimisation. The dashed box shows the system boundary and illustrates that the subjective requirements come from outside the system, i.e., from the user of the device. QoE is a measure of how the user perceives and reacts to the services provided by the device. QoE is used as a part of a feedback loop to measure the performance of the services provided by the system in a user-centric manner. It is then used to influence the resource allocation, which in turn alters the performance of the delivered services. A similar feedback loop is used for QoS. QoS is able to act upon objective requirements such as the distribution of bandwidth but is unaware of the subjective, user-specific requirements - QoE is designed to encompass precisely these. Understanding how the end user interprets device performance enables the system to make sensible, user-centric resource allocation decisions which can be combined with those taken by traditional QoS mechanisms to ensure proper operation, and high user satisfaction. Fiedler et al. [6] present a link between the time taken to load a web page and the Mean Opinion Score (MOS) for a group of users. This provides us with an established link between QoS and QoE for a real scenario and shows how a disturbance can have a pronounced effect on the user experience.

It is worth noting that there is no single, universal measure of user experience as this depends heavily on the types of applications running [15]. As an example, a GPU-intensive task, such as a 3D game, will require a sufficiently high frame rate to ensure that the user is satisfied with the performance. This is quite different to, say, a file retrieval task or the display of a still picture, where frame rate is an irrelevant concept. Therefore, whilst there is a set of measurable metrics which give an indication of the user's satisfaction and the quality of their experience, those of relevance vary as the applications running on the device change.

In order for systems to be able to optimise their performance for user experience, multiple requirements must be fulfilled:

1) As stated in Section I, a set of metrics which are correlated with the user experience but remain measurable from within the confines of the device must be determined. We measure Cycles-Per-Instruction (CPI) for the CPU when rendering web pages, the frame rate of the GPU, as well as energy usage for the CPU, GPU and memory controller in Section IV.
2) Metrics which are of relevance must be selected at run time based on either observed traffic or explicit notification from a higher level within the system. In our experiments we pre-determine the metrics as we control both the system and the workloads. Specifically, our QoE metrics are webpage render time and the ability of the GPU to render at a sufficiently high frame rate.

III. Experimental Setup

To demonstrate the feasibility of applying QoE to system level design, we concentrate on performance/power trade-offs. We use full-system simulation to explore these trade-offs as detailed simulation offers flexibility and observability, without sacrificing accuracy. The simulation framework models a realistic and representative system and is able to run full-system workloads on detailed and accurate models. This section describes and justifies the simulator choice and system design used in this work.

We use gem5 [16], an event-driven full-system simulator that facilitates architectural exploration. It features a realistic out-of-order CPU model, cache models, and a Dynamic Random-Access Memory (DRAM) controller model alongside a multitude of peripherals. For the purpose of this study it has also been augmented with a detailed GPU model which is capable of rendering real frames, thereby producing traffic patterns representative of a real device.

A. System Architecture Models

We use a detailed cycle-based out-of-order CPU model that has been configured to perform like an industry-leading embedded processor. The CPU model takes into account the internal resource contention of the CPU, and the performance varies accordingly. The CPU model is detailed enough to boot full operating systems, as well as to produce realistic interactions with the memory system.

The GPU is a detailed cycle-level model of a commercial GPU which has been verified against real hardware. The GPU model renders frames from memory dumps captured from a real device which contain all of the information required to render the frame, and therefore has no interaction with the CPU. As the GPU is functionally correct and renders real frames, the time it takes for the GPU to calculate the frame varies with frame complexity and interactions with the memory system. The GPU is connected to the rest of the system via two L2 caches which handle the coherency between the GPU and the rest of the system (see Figure 2).

The memory controller simulates the architecture of a DRAM controller, as well as the memory itself. For these experiments, the DRAM model has been tuned to resemble a dual-channel LPDDR3 controller and associated memory (see Table I) which is found in high-end consumer devices. The memory controller is configured to use First-Ready First-Come-First-Serve (FR-FCFS) when servicing the incoming requests as well as using an open-row policy. Therefore it will prioritise accesses to open rows, and thus attempt to maximise the row hit rate which results in higher memory efficiency.

An LCD controller reads the system frame buffer at 60 Hz and drives a display at a resolution of 1920x1080 (Full HD). All other peripherals needed to run the unmodified Android operating system are on a separate IO-bus which is connected to the rest of the system via a bridge.

B. System Architecture and Workloads

The system architecture simulated in gem5 is shown in Figure 2, and the settings used are summarised in Table II. The system uses a single core ARMv7-compatible out-of-order CPU based on the detailed CPU model discussed in Section III-A. This system is chosen to be representative of current SoCs such as the OMAP series by TI [2] and the Exynos series by Samsung [1], and also has similarities with the model system for ARM's CCI interconnect [17]. As the CPU and GPU models are scalable, the system is expandable and is able to cover future, multi-core workloads.

Channels	Dual
Bits per Channel	32
Clock Rate	800 MHz (1600 MT/s)
Banks per Rank	8
Ranks per Channel	1
Row Buffer Size	1024 bytes
tRCD, tCL, tRP	15 ns
tRFC	130 ns
Refresh Cycle Time	64 ms
Scheduling	FR-FCFS
Row Policy	Open
Total Bandwidth	12.8 GB/s

TABLE I: DRAM Configuration

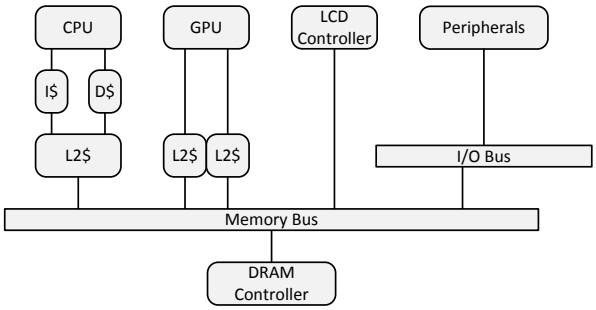


Fig. 2: System Platform.

We use BBench [18] as our main CPU workload. BBench is a browser-based benchmark which loads a subset of the most-visited websites using an operating system's native browser. System performance is evaluated by measuring the time taken to render each web page and scroll to the bottom of the page. A faster page load time is representative of a higher QoE [6]. The GPU frames used for this paper are a 3D game benchmark called Taiji and a navigation benchmark, Navi. Taiji is from RightWare's Basemark ES 2.0 [19] and Navi is from 3DMark Mobile 2.0 [20]. Both frames are rendered in Full HD and represent typical resolutions used for current high-end devices. These benchmarks have been selected as they represent a set of real use cases which cover both gaming and navigation and provide a varied set of bandwidth requirements.

We use SimPoints [21] for the CPU which break down the CPU workload into multiple smaller chunks based on the phases detected in the workload. Each SimPoint is assigned a weighting based on how frequently the CPU is in the phase. These SimPoints are simulated in parallel, and their

CPU Clock	200 MHz-1.7 GHz, 100 MHz steps
CPU Voltage	0.925V-1.3V
L1 Cache Size	32 kB Inst & 32 kB Data
L2 Cache Size	1 MB
System Memory	Total: 1536 MB OS: 1024 MB GPU: 512 MB
Memory Bus Clock	800 MHz
Operating System	Android 4.1.1
Kernel Version	3.5.0-rc7

TABLE II: System Configuration

results combined to give a prediction for the overall workload. This allows simulation speed to be vastly increased without a significant impact on the accuracy of the results, thereby allowing accurate profiling of larger applications.

C. Power Consumption Estimation

System dynamic power consumption is estimated for the CPU, GPU, and DRAM. We estimate the power using simplistic, ad-hoc techniques, as established tools, such as McPAT [22], do not provide representative data for our target system.

The CPU power is estimated using the legal DVFS operating points for the Samsung Exynos 5 Dual [1]. These frequencies and corresponding voltages have been taken from the Linux kernel [9]. We calculate $V^2 f$ for each DVFS point, which ensures that the differences between power points is representative of a real device. We include the power consumption in CPU caches as they are part of the same voltage and frequency domain.

The GPU power consumption is estimated on the basis of the number of active cores and the time taken to render the frame. Specifically, the power consumption of a GPU is relatively constant when active, and, therefore, we only need to measure the active time to determine the overall power consumption for a particular core configuration [23].

We assume that the average energy per bit accessed in the DRAM is constant and use it to estimate DRAM power consumption. We base the power consumption of the DRAM on power figures provided by Micron [24] which state that a 2-channel LPDDR3 memory will consume 9.2 pJ/bit at 800 MHz. We do not use CACTI-D [25] for calculating the DRAM power as we do not require an extremely detailed power analysis.

IV. Experimental Results

This section discusses the results of the experiments performed on the CPU and GPU individually. This leads onto the optimisation of the system based on the characteristics observed.

A. CPU and GPU Analysis

The SimPoints for the CPU are evaluated for a set of different voltage and frequency points, which allows us to observe the change in performance and energy consumption for the CPU. In Figure 3 we present the trade-off between benchmark run time and the energy consumed by the CPU, shown relative to a CPU frequency of 1.7 GHz, which is the maximum operating frequency for the Exynos 5 Dual [1]. The energy is estimated by multiplying $V^2 f$ for each of the DVFS operating points by the run time, assuming a roughly constant load for the duration of the benchmark. There is a non-linear relationship between the benchmark slowdown and the energy consumed, and the relationship is split into two key parts. For small reductions in CPU frequency there is a significant decrease in CPU energy usage across the benchmark for a modest increase in overall benchmark run time. For larger CPU frequency reductions, there are diminishing returns as the run time increases more rapidly than the power savings.

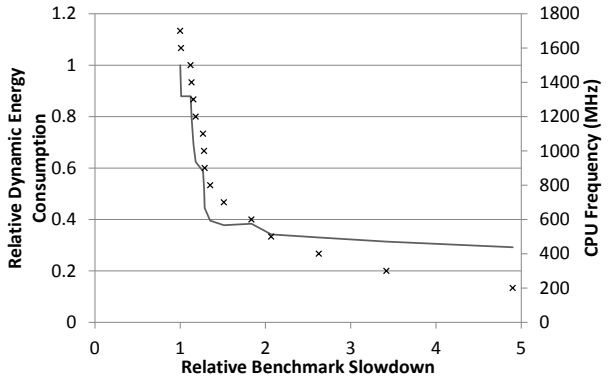


Fig. 3: Energy consumption and CPU frequency as a function of system slowdown.

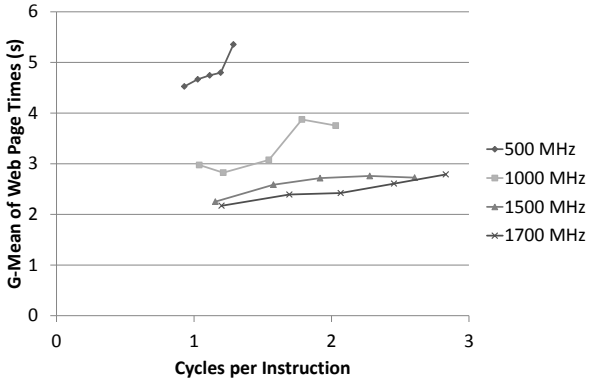


Fig. 4: CPI vs. geometric mean of web page times for BBench as CPU frequency varies.

Small reductions in frequency allow large energy savings, with minimal performance loss, and therefore it is possible to trade-off the performance of the CPU for energy consumption efficiently. The non-linear relationship exists because the average memory access latency does not vary significantly as the CPU frequency is adjusted. Therefore, the benchmark spends a large proportion of time waiting for data from main memory, and the CPU idles, wasting energy. Modest reductions in the CPU frequency result in fewer wasted cycles whilst still performing a similar amount of work in a given time period. Once the CPU frequency has been reduced significantly, the CPU is unable to process the data at the rate that the memory can supply it and the run time increases significantly, resulting in poor energy-efficiency.

Taking the energy-delay product, we determine that the ideal DVFS point in terms of total energy consumption is at a CPU frequency of 800 MHz. Operating at 800 MHz results in an increase in run time of 35% and a decrease in energy consumed of over 60% relative to 1.7 GHz. In order to relate this reduction in energy and performance to QoE, Figure 4 shows the geometric mean for web page render times in BBench as the memory latency seen by the CPU is varied, and shows the relationship between CPI and web page render time. As the memory latency increases, the time taken to render each web page increases, as does the CPI of the CPU for a given frequency. This not only demonstrates that the CPU is able to operate more efficiently at lower frequencies due to the relative decrease in memory latency, but also provides us

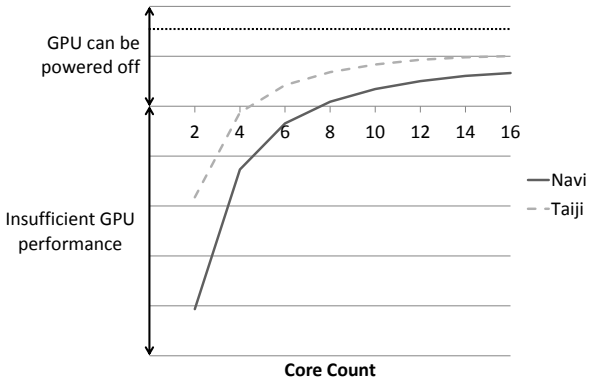


Fig. 5: Amount of time where GPU can be powered off for Navi and Taiji. Note that the y-axis labels have been deliberately removed.

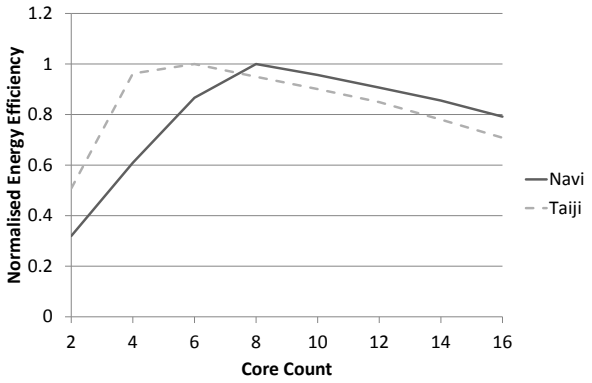


Fig. 6: Performance per Watt for Navi and Taiji as the number of cores is varied.

with a link between low-level system metrics and higher-level user experience. It thus provides a QoE-specific reference for the results presented in this paper (similar to the link shown between MOS and QoS disturbance in [6]) and allows QoE to be used as per Figure 1.

The GPU is dual-ported, and hence the number of GPU cores is varied from 2 through 16 in increments of 2, thereby ensuring that the load on each port is equal. Please note that there is no other traffic present in the system so these results apply specifically to the GPU in isolation. The GPU should render the frame before the deadline imposed by the screen refresh rate is reached. If the GPU fails to render the frame in time, then the frame will be skipped, thereby resulting in a decreased overall frame rate and a decreased user experience. On the other hand, if the GPU completes frame rendering significantly before the deadline, then it can be power-gated, saving energy [23], [26]. Therefore, in order to determine the ideal GPU configuration it is important to understand the relationship between the number of cores and the time taken to render different GPU frames.

Figure 5 shows, for varying core counts, the time for which the GPU can be powered off as it has completed rendering before the deadline. Please note that the frame times have been obfuscated. The results shown assume a target frame rate of 60 Frames Per Second (FPS), which is shown as a horizontal dashed line in order to demonstrate the maximum amount of time the GPU can be powered off. Values above the x-axis indicate that GPU can be powered off whilst values below the x-axis show GPU configurations where performance is insufficient to achieve 60 FPS. It can be seen that Navi requires at least 8 cores to render at 60 FPS, and that Taiji requires 6 as, otherwise, the frame takes longer to render than the allowed time per frame.

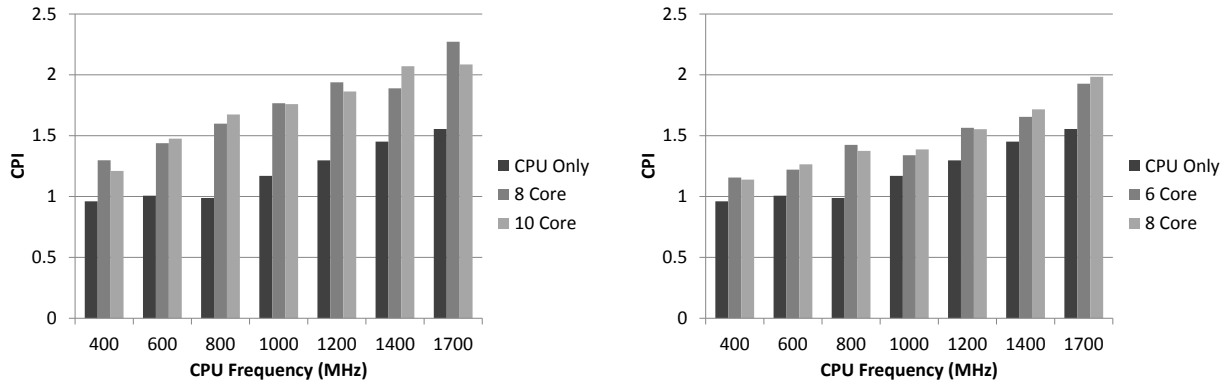
The run time has been used to calculate the energy consumption for the GPU for each of the core configurations when rendering each of the two frames. These results, which again assume a screen refresh rate of 60 Hz, are summarised in Figure 6, and are presented in terms of GPU efficiency, i.e. the performance in frames per second per Watt. These results demonstrate that it is important to minimise the number of active GPU cores in order to keep energy usage low.

As the GPU generally requires a large amount of data to render a frame and is able to process vast quantities in parallel, it has a large impact on the dynamic power consumption of the DRAM. However, once the GPU has reached its maximum allowed frame rate, such as 60 Hz, it will request roughly the same amount of data (this can be seen in Figure 9). Therefore, whilst the dynamic DRAM power consumption increases, the total energy consumed in the DRAM remains roughly constant per frame rendered.

B. System Optimisation

We investigate how the additional GPU traffic affects the performance of the CPU and compare to the results from Section IV-A. We then use this to determine what the ideal CPU frequency and GPU core count should be in order to minimise the overall system power consumption, whilst maintaining a sufficiently high QoE. Without the GPU rendering in the system, the most energy-efficient operating point for the CPU was 800 MHz. The GPU frames Navi and Taiji required 8 and 6 cores, respectively, in order to render at 60 Hz or more. In this section, we change the frequency of the CPU in 200 MHz steps from 400 MHz to 1400 MHz, as well as running at the maximum frequency of 1700 MHz. Simultaneously, we run the GPU in 6 and 8 core configurations for Taiji or the 8 and 10 core configurations for Navi. This allows us to observe which configuration is best for the complete system. Please note that the DRAM controller uses First-Ready First-Come-First-Serve (FR-FCFS) scheduling, and no attempt is made to prioritise the CPU traffic over the GPU traffic. All trade-offs are investigated statically and no adjustments occur at run-time.

Figure 7 shows the change in CPI for the CPU as the load placed on the shared memory changes with the GPU core count. The results clearly demonstrate that the addition of the GPU has significantly increased the latency to main memory, thus reducing the performance of the CPU. It can also be seen that Navi requests more bandwidth from the memory than Taiji, as it has a larger impact on the CPI of the CPU. If we compare the CPI measured here to the CPI shown in Figure 4, we see that we should expect roughly 4.2 seconds on average for a page to load and scroll for 8 core Navi, and 3.7 seconds for 6 core Taiji. For reference, using the CPI of the CPU-only run at 800 MHz - which is 0.99 - we estimate the mean time for a web page to be approximately 3.1 seconds. When running at the maximum clock frequency of 1.7 GHz without a GPU the average web page time is 2.4 seconds, and



(a) Navi

(b) Taiji

Fig. 7: CPI for BBench with varying GPU core count.

therefore reducing the CPU frequency to 800 MHz leads to a 30% increase in the average render time.

The increase in the CPI of the CPU manifests itself in an increased run time for the benchmark. The overall impact on user-perceived CPU performance is shown in Figure 8. It is worth noting that the impact of additional GPU cores has little influence on the run time for the CPU once the maximum frame rate has been reached. Therefore, there are similar run times for 8 and 10 core configurations for Navi, and 6 and 8 core configurations for Taiji.

The addition of a GPU has a significant impact on the overall memory bandwidth consumed, and hence increases the power consumption of the memory. This is illustrated in Figure 9. The memory power consumption is similar to the change seen in CPI, and this is to be expected as the contention between the CPU and the GPU is in the shared memory. However, the power consumption is averaged across the run time, and therefore the overall energy usage increases significantly as the run time of the benchmark rises due to resource contention.

Figure 10 shows the relationship between the slowdown of the benchmark (shown relative to 1.7 GHz) and the energy consumed in the CPU. The frames rendered on the GPU, the number of GPU cores, and the DVFS points of the CPU are changed. For reference, the CPU-only runs from Figure 3 are shown using a fine dotted line at the bottom of the diagram. First of all, it is apparent that for any scenario which includes a GPU, the CPU energy usage increases as the overall run time for the benchmark increases due to the intensified resource contention. The increased load has a secondary effect on the CPU performance as the CPU spends a larger proportion of time waiting for data from memory. Therefore, the higher the GPU-induced load on the shared memory, the smoother the trade-off between run time and energy usage. There is also a tight grouping between the 8 and 10 core GPU configurations for Navi, and the 6 and 8 core configurations for Taiji, which is to be expected given the previous figures.

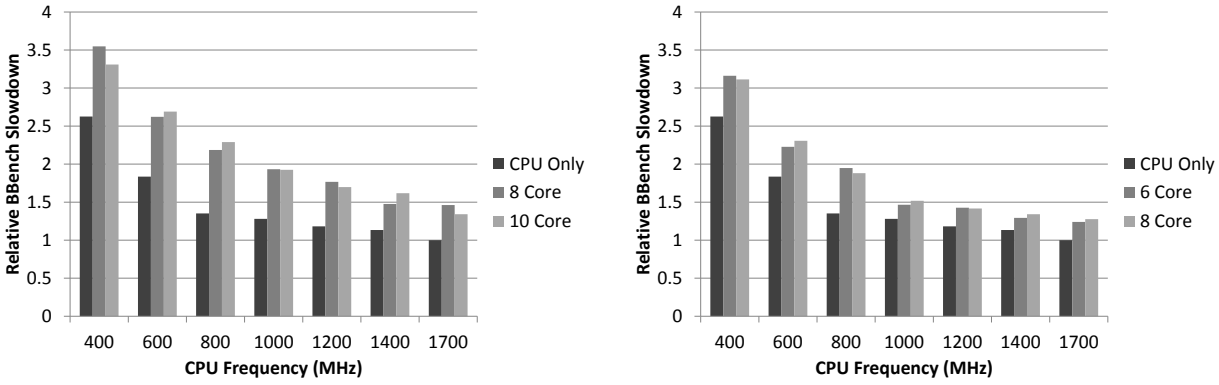
The addition of the GPU shifts the ideal point for certain GPU core configurations and frames. Specifically, for Taiji in the 6 and 8 core configurations the most energy-efficient CPU frequency (determined by minimising the energy-delay product) shifts from 800 MHz to 1 GHz, as does the 10 core configuration for Navi. The 8 core configuration for Navi remains most efficient at 800 MHz. Therefore, as the load induced by the GPU increases, the ideal operating point for the CPU increases. This motivates the need for dynamic run time adjustment of system configuration based on both QoS and QoE measurements (see Figure 1). Run-time adjustment based on QoE requires measurable metrics, such as the CPI of the CPU and the frame rate of the GPU. As previously stated, the metrics of interest change as the applications running on the device change, and therefore run time adjustment also is needed to keep track of the ever-changing requirements and demands placed on the system. Finally, a set of knobs which can be adjusted in order to tweak the system performance is required.

We compare standard Linux CPU governors to our results. The *Performance* [9] governor will choose a CPU frequency near the top left of Figure 10, whilst the *Powersave* governor will choose the bottom right [10]. The *OnDemand* governor will try to use the lowest DVFS point possible given the load on the CPU but does not take into account the user experience when switching the frequency. QoE provides a user-centric measure of performance which would allow a governor to choose an experience-optimised DVFS point.

V. Related Work

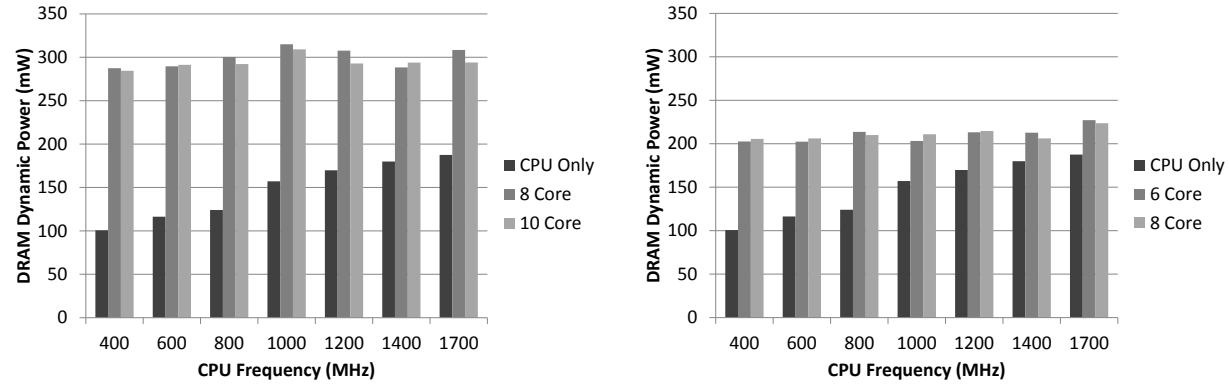
QoE has been investigated in the context of telecommunications and networks. Collange et al. [13] present a methodology which allows passive estimation of QoE for large scale networks, and demonstrate their approach with ADSL traffic traces. They use the packet loss for the ADSL traces to try and estimate the user experience. However, as their approach is designed to work with lossy networks, it is less suited for SoC interconnects as these are usually designed not to lose packets.

The recent work by Jagathessan et al. [27] presents the notion of Application Defined Computing. Specifically, they tune the performance of generic hardware to the characteristics of the applications running by adjusting the brokering used for the DRAM controller. This allows them to demonstrate both a decrease in overall system power consumption and an increase in performance. As we have shown in Section IV-B it



(a) Navi (b) Taiji

Fig. 8: Run time for BBench with varying GPU core count.



(a) Navi (b) Taiji

Fig. 9: Dynamic DRAM power for BBench with varying GPU core count.

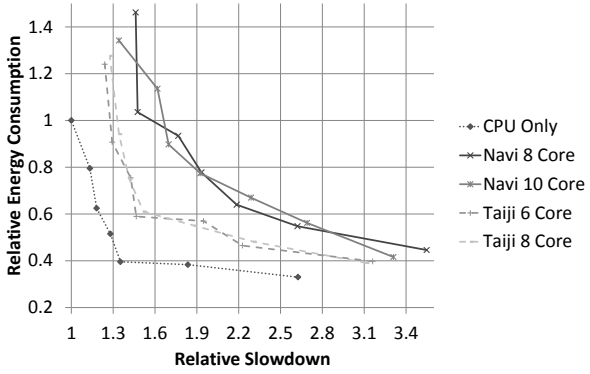


Fig. 10: Relative slowdown vs. relative energy consumed by the CPU as the load placed on the shared memory varies due to the GPU traffic. The points on the lines show the different DVFS points for the CPU.

is important to take into account the performance of the CPU and the GPU when optimising the system.

Other work underlines the need to observe, learn from and react to user behaviour. Falaki et al. [5] present the results from a survey of 255 smartphone users which illustrate the diversity in the way that different people use their smartphones. This clearly demonstrates that it is important to understand which applications are of the greatest importance to a particular user before optimising the device for any particular scenario. It supports our claims in Section II where we state that the metrics of importance vary with each application and user.

Carroll et al. [28] show that, in addition to the GSM and Wifi modules, the CPU and GPU are two of the biggest consumers of power in a mobile device. Therefore, it is important to focus on reducing the power consumption of these key components which are found in every smartphone. Bircher et al. [29] demonstrate that for certain benchmarks the memory and memory controller are able to consume more power than the CPU core. This work considers CPU, GPU as well as memory power as it can have a marked impact on overall system power consumption.

Shye et al. [30] present work which explores the activity of a set of users on a real device, and apply the findings to construct a power model. This power model is then used to demonstrate that the power consumption varies between users as each of them places different demands on the device. The authors then demonstrate that reducing both screen brightness and CPU frequency gradually over time, rather than making a step change, has less impact on the end user experience and allows them to save over 10% of the system power. This underlines the importance of taking into account user

behaviour and preferences.

Wijnants et al. [8] investigate the link between QoS, in the form of technical parameters, and QoE for a multi-player game which relies on location sensing and player communication over a 3g connection. They demonstrate a clear link between artificial distortions in the QoS aspects and the enjoyment and absorption reported by a group of players. They also note that it is important to minimise the communication latency as it showed the largest impact on the user satisfaction. They illustrates that it is important to consider both the services that are delivered by the device, and the context in which they are presented when optimising the performance of the system. Therefore, it is vital to consider the system as a whole as we do in our work.

VI. Conclusions

This paper demonstrates for the first time that Quality of Experience (QoE) can be used as an embedded system optimisation tool. We selected system-level metrics which allowed measurement of QoE from within the confines of the device. Using these, we demonstrated that a relatively small decrease in the frequency of the CPU, which results in a small decrease in the perceived performance, is able to offer significant power savings. It is important to understand the user-centric performance reduction as this gives a much clearer idea of how the performance affects the end user. When evaluating the performance of the system as a whole, we have shown that the optimal operating point for the CPU shifts as the load placed on the shared memory by the GPU changes. Whilst this shift was only by 200 MHz, it is worth noting that this effect was produced by having only one other master in the system. Therefore, it is important that the system configuration can be dynamically adjusted.

Our future work includes the characterisation of applications based on their demands and the type of service they provide to the end user. This will provide the basis for a framework which allows the system configuration to be adjusted at run time taking into account the types of applications running on the system, their demands, and the requirements of the end user.

References

[1] Samsung, "Enjoy the Ultimate WQXGA Solution with Exynos 5 Dual," *White Paper*, 2012.

[2] TI, "OMAP 5 mobile applications platform," *Texas Instruments Incorporated*, 2011.

[3] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proc. of the 8th Int. Symp. on High Performance Computer Architecture*. IEEE Computer. Soc, 2002, pp. 29–40.

[4] P. Greenhalgh, "Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7," *ARM White Paper*, no. September, pp. 1–8, 2011.

[5] H. Falaki, R. Mahajan, S. Kandula, D. Lymberopoulos, R. Govindan, and D. Estrin, "Diversity in smartphone usage," *Proc. of the 8th Int. Conf. on Mobile systems, applications, and services*, p. 179, 2010.

[6] M. Fiedler, T. Hossfeld, and P. Tran-Gia, "A Generic Quantitative Relationship between Quality of Experience and Quality of Service," *IEEE Network*, vol. 24, no. 2, pp. 36–41, Mar. 2010.

[7] A. A. Webster, C. T. Jones, M. H. Pinson, S. D. Voran, and S. Wolf, "An objective video quality assessment system based on human perception," in *Proc. SPIE Human Vision, Visual Processing, and Digital Display*, J. P. Allebach and B. E. Rogowitz, Eds., Sep. 1993, pp. 15–26.

[8] M. Wijnants, W. Vanmontfort, J. Dierckx, P. Quax, W. Lamotte, K. D. Moor, and J. Vanattenhoven, "Quality of Service and Quality of Experience Correlations in a Location-Based Mobile Multiplayer Role-Playing Game," in *Proc. of the 10th Int Conf. on Entertainment Computing*, 2011, pp. 101–112.

[9] The Linux Kernel Archives, "The Linux Kernel Archives, [online], http://www.kernel.org."

[10] V. Pallipadi and A. Starikovskiy, "The ondemand governor: Past, present, and future," *Proc. of the Linux Symp.*, pp. 215–230, 2006.

[11] ETSI TR 102 274, "Human Factors (HF); Quality of Experience (QoE) requirements for real-time communication services," pp. 1–37, 2010.

[12] G. Ghinea and J. P. Thomas, "Quality of perception: user quality of service in multimedia presentations," *IEEE Trans. on Multimedia*, vol. 7, no. 4, pp. 786–789, Aug. 2005.

[13] D. Collange, J.-l. Costeux, O. Labs, and S. Antipolis, "Passive Estimation of Quality of Experience," *Journal of Universal Computer Science*, vol. 14, no. 5, pp. 625–641, 2008.

[14] D. Hirsch, A. Lluch-Lafuente, and E. Tuosto, "A Logic for Application Level QoS," *Electronic Notes in Theoretical Computer Science*, vol. 153, no. 2, pp. 135–159, May 2006.

[15] P. Brooks and B. Hestnes, "User measures of quality of experience: why being objective and quantitative is important," *IEEE Netw.*, vol. 24, no. 2, pp. 8–13, Mar. 2010.

[16] N. Binkert, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, D. A. Wood, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, and T. Krishna, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, p. 1, Aug. 2011.

[17] A. Stevens, "Introduction to AMBA 4 ACE ," *ARM White Paper*, pp. 1–15, 2011.

[18] A. Gutierrez, R. G. Dreslinski, T. F. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver, "Full-system analysis and characterization of interactive smartphone applications," in *IEEE Int. Symp. on Workload Characterization*, 2011, pp. 81–90.

[19] Rightware, "Basemark ES 2.0," 2012.

[20] Futuremark, "Futuremark Releases 3DMarkMobile ES 2.0," 2012.

[21] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *10th Int Conf. architectural support for programming languages and operating systems*. New York, New York, USA: ACM Press, 2002, p. 45.

[22] S. Li, J. H. Ahn, R. Strong, J. Brockman, D. Tullsen, and N. Jouppi, "McPAT : An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," *42nd Annu. IEEE/ACM Int. Symp. on Microarchitecture*, pp. 469–480, 2009.

[23] S. Collange, D. Defour, and A. Tisserand, *Computational Science ICCS 2009*, ser. Lecture Notes in Computer Science, G. Allen, J. Nabrzyski, E. Seidel, G. D. Albada, J. Dongarra, and P. M. A. Sloot, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, vol. 5544.

[24] Micron, "Migrating to LPDDR3: An overview of LPDDR3 commands, operations, and functions," in *JEDEC LPDDR3 Symp.*, 2012.

[25] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A Comprehensive Memory Modeling Tool and Its Application to the Design and Analysis of Future Memory Hierarchies," *Int. Symp. on Computer Architecture*, pp. 51–62, Jun. 2008.

[26] P.-H. Wang, C.-L. Yang, Y.-M. Chen, and Y.-J. Cheng, "Power gating strategies on GPUs," *ACM Trans. on Architecture and Code Optimization*, vol. 8, no. 3, pp. 1–25, Oct. 2011.

[27] A. Jagatheesan, "Application Defined Computing in smartphones and consumer electronics," *IEEE 10th Consumer Communications and Networking Conf.*, pp. 857–858, Jan. 2013.

[28] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," in *Proc. of the 2010 USENIX Conf.*, ser. USENIXATC'10. Berkeley, CA, USA: USENIX Association, 2010, p. 21.

[29] W. L. Bircher and L. K. John, "Analysis of dynamic power management on multi-core processors," *Proc. of the 22nd Annu. Int. Conf. on Supercomputing*, p. 327, 2008.

[30] A. Shye, B. Scholbrock, and G. Memik, "Into the Wild: Studying Real User Activity Patterns to Guide Power Optimizations for Mobile Architectures," in *Proc. of the 42nd Annu. IEEE/ACM Int. Symp. on Microarchitecture*. New York, New York, USA: ACM Press, 2009, p. 168.