Toolbox for Exploration of Energy-Efficient Event Processors for Human-Computer Interaction

Tayyar Rzayev[1], David H. Albonesi[1], François Guimbretière[2], Rajit Manohar[3], and Jaeyeon Kihm[2]

[1]Computer Systems Laboratory
Cornell University
Ithaca, NY, USA
{tr265, dha7}@cornell.edu

[2]Information Science Department
Cornell University
Ithaca, NY, USA
{fvg3, jk2443}@cornell.edu

[3]Computer Systems Laboratory,
Yale University
New Haven, CT, USA
{rajit.manohar}@yale.edu

***Abstract*—The advent of high speed input sensor and display technologies and the drive for faster interactive response suggests that human-computer interaction (HCI) task processing deadlines of a few milliseconds or less may be required in future handheld devices. At the same time, users will expect the same, if not better, battery life than today's devices under these more stringent response requirements.**

In this paper, we present a toolbox for exploring the design space of HCI event processors. We first describe the simulation platform for interactive environments that runs mobile user interface code with inputs recorded from human users. We validate it against a hardware platform from prior work. Given system-level constraints on latency, we demonstrate how this toolbox can be used to design a custom heterogeneous event processor that maximizes battery life. We show that our toolbox can pick design points that are 1.5-2.5x more energy-efficient than general-purpose big.LITTLE architectures.

I. Introduction

Computer systems have evolved from batch systems, to personal computers, to the present day proliferation of handheld devices such as cell phones, tablets, and eReaders. Mobile devices have evolved into high performance devices used for a wide range of everyday activities with compute components were once scaled-down versions of personal computers.

For these handheld systems, the "user experience," involving particular human-computer interaction (HCI) tasks, has been gaining increasing attention from designers, due to both the nature of the tasks being performed on these devices, (e.g., finger drawing), and increasing user expectations for rapid responsiveness. For these types of tasks, the time between the action (such as a key press or a screen touch) and the response (the appearance of the key or a marking) is expected to meet a specified (soft) real time deadline, the length of which depends on the particular task and the price point of the system. Response time has traditionally been dominated by the display and sensor interfaces, whose combined latency in the tens of milliseconds overshadowed the response time contribution of the HCI processing hardware. However, HCI task processing speeds are becoming increasingly important. The emergence of display technologies such as OLED [1] and bi-stable displays [2] and developing sensor interfaces [3] have dramatically lowered input and output device response time and power compared to previous generations, such that combined response times of less than 1ms are achievable today [4].

Moreover, studies that reported 100ms as the satisfactory response time for human-computer interaction are several decades old [5]. More recent studies [3, 6, 7] with high frame rate cameras and much faster sensors and processing technologies have shown that users can perceive orders of magnitude faster response times. These studies separate touch interactions into two types: tapping inputs and direct manipulation inputs. Tapping inputs are discrete events, where the user merely provides a one-time stimulus to the device and expects feedback. Direct manipulation, on the other hand, is an action (such as finger drawing) where the user continuously provides input and expects continuous real-time response. Response times of under 20ms may be satisfactory for tapping inputs, with some of that latency dedicated to waiting for the full contact and mapping the input to a point [8]. For direct manipulation inputs, users may perceive differences in response times over a millisecond in duration [3, 6, 7]. In this paper, we study both types of touch interactions, specifically common HCI tasks such as finger typing, keyboard typing, annotating, and reading. These tasks are used for activities such as text messaging (finger typing), drawing (annotating), email and Facebook (reading, page rendering, typing), and productivity apps (annotation, reading, page rendering, typing).

At the same time, users expect long battery life while performing these tasks with highly stringent latency requirements. During periods of relatively "quiet" activity, battery life may be expected to seemingly last forever, yet the system must provide high responsiveness during intensive activities. For these reasons, HCI task processing times of a few milliseconds or less will soon become the norm for handheld devices, yet with the expectation of the same, if not better, battery life as current, less responsive,

systems. This accentuates the need for the development of fast and energy-efficient HCI processing hardware.

However, not all interactive tasks require an aggressive processor design. Tasks such as typing can be adequately handled by a simpler core [9]. Handling typing tasks on a high power core necessary for intensive tasks such as page rendering wastes considerable energy, thereby degrading battery life.

While heterogeneous multicore architectures have been extensively studied for consumer, scientific, and commercial workloads, relatively little attention has been paid to interactive HCI tasks with real time constraints. Some prior work [10] has demonstrated that the use of a big.LITTLE architecture comprising ARM A9 and M3 cores can improve the energy efficiency of HCI tasks, but this study was limited to the two core types implemented in the hardware. Moreover, it is currently unclear whether big.LITTLE architectures developed for general-purpose computing are, in fact, energy efficient when processing HCI tasks. The reason is that the architecture community has to date lacked a flexible simulation infrastructure that permits microarchitecture design space exploration for HCI-specific hardware under real time constraints and using real user input traces.

In this paper, we present what we believe to be the first simulation-based platform for interactive environments that runs user interface code ported from a second generation experimental ARM-based tablet [10]. The simulator is based on the GEM5 ARM simulator [11], McPAT power models [12], and idle time power management models based on [13]. Our platform takes input traces from real users that are captured on the handheld hardware, processes them using the interactive application, and measures the response latency and energy. Thus, the simulator permits the exploration of HCI-dedicated hardware across user characteristics and real-time requirements.

The contributions of this paper include the following:

- We introduce a toolbox that models real-time human interactions with a mobile device, as well as architectural details such as processing latency, idle-time power management and a simple OS model.
- We validate the toolbox against a hardware platform from prior work.
- We present a design methodology for selecting a custom architecture to maximize energy efficiency based on UI latency constraints from the system design.
- We show experimental results for event processor designs that are 1.5-2.5x more energy efficient than general purpose big.LITTLE architectures.

The rest of this paper is organized as follows. In the next section, we discuss related work. Then in Section III we describe the simulation methodology that we have developed for studying architectures for interactive applications, and validate its performance and power results versus real hardware. We then examine energy-efficient event processor designs in Section IV and quantify their energy savings compared to general-purpose approaches. We conclude in Section V.

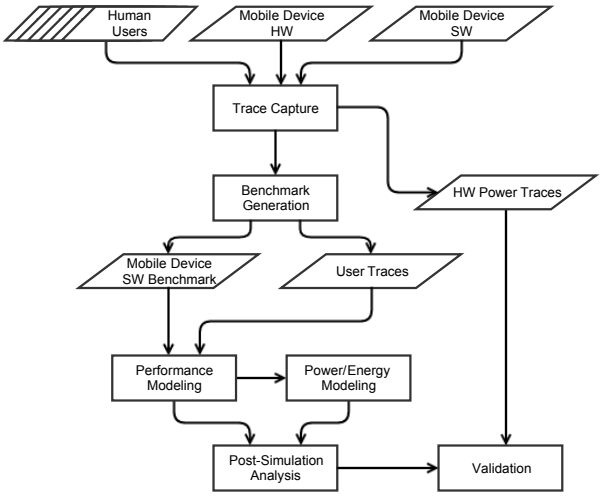

Fig. 1: Overall infrastructure organization.

II. Related Work

Recent work has observed that the development of better tools to study human computer interaction leads to new observations regarding the limits of human perception. Jota et al. [3] investigated the impact of latency on the performance of direct physical manipulation tasks. They studied user behavior when faced with systems of various latency and concluded that humans can perceive latency of tapping inputs below 25ms. They also state that others have concluded that humans can recognize touch latencies as low as 2.38ms [6]. Researchers from Microsoft further show that for direct manipulation inputs delays of over 1ms may be perceivable to users and degrade their experience [7]. This is very critical to our work because we consider tasks of both kinds: tapping inputs and direct manipulation inputs. For example, our inking task is direct manipulation and the keyboard tasks are tapping inputs. The reading task falls into both categories. When the user is just reading, it can be considered a tapping input, however, when the user is scrolling through book pages, it becomes a direct manipulation task.

Advanced input sensors [3] and displays have been developed with significantly lower latencies and power consumption. Commercial OLED displays [1] have response times in the range of μs. There has been a body of work [14, 15] developing power saving techniques utilizing OLED technology. For instance, Zhao et al. [14] study OLED in the context of mobile video streaming. They design dynamic techniques to limit the OLED portion of system power and energy. In other work, Anand et al. [15] investigate opportunities to apply display power saving techniques developed for LCD technology to OLED.

Task	Sub-tasks	Description
Physical Keyboard Typing (PK)	Physical Keypress, Blank Line Load	Typing on the physical keyboard. Blank line is loaded at bottom of page.
Virtual Keyboard Typing (VK)	Virtual Keypress, Shift/CAPS, Blank Paragraph Load	Typing on the virtual keyboard. Keyboard is updated on Shift/CAPS press. Blank paragraph is loaded at bottom of page.
Inking (INK)	Inking, Blank Page Load	Annotating with pen. Blank page is loaded when reach bottom of page.
Reading (RD)	Book Page Load, PDF Rendering	New book page is loaded when user presses *Next*, and another PDF page is rendered into the image cache.

TABLE I: Major user interface tasks and their sub-tasks.

Improvements in user interface technology have influenced the design of recent experimental mobile platforms. Guimbretiere et al. [9] present a prototype of an asymmetric dual core platform for interactive applications that can leverage the large core for high intensity tasks, while utilizing the smaller core to save energy during "quieter" periods in the interaction with the user. More recently, Kihm et al. [10] present a prototype system to show the energy savings using an OMAP4460, with the big.LITTLE A9 and the M3 cores splitting HCI tasks. They demonstrate energy efficiency improvements for HCI tasks while remaining responsive to the needs of the user. These works explore the benefits of using heterogeneous multicore architectures for running an essentially single-threaded application for the purpose of energy efficiency, but are limited to the core types in the hardware platform. In particular, while they demonstrate that the big.LITTLE approach is an improvement over the use of the A9 or the M3 alone, the limitations of their platform prevent them from exploring additional multicore architectures that may be more energy efficient, and quantifying that difference.

Another work that models mobile systems is GemDroid [16], which uses Android applications and focuses on analyzing the memory subsystem of an SoC, including improving core performance by modifying the memory controller design. Our work is fundamentally different as we focus on the HCI component of mobile devices by running HCI system software and interactive user workloads complete with user input traces, and we evaluate energy-efficient architectures specific to these workloads.

III. Modeling Infrastructure

In this section, we present what we believe to be the first simulation infrastructure for evaluating interactive applications, including real user traces. Our methodology, which is shown in Figure 1, comprises the following components:

- *Interactive user trace capture* performed with test subjects using custom-developed interactive eBook reader hardware and software.
- *Benchmark generation* in which the user traces are aggregated into a set of data traces representative of the user activity on the platform.
- *Performance modeling* on a GEM5-based simulator that runs the mobile user interface software.
- *Power modeling*, a post-processing step using McPAT to obtain static and dynamic power, as well as to model processor sleep states.

Each of these steps is now described in detail.

A. Interactive User Trace Capture

User interface traces are gathered on a second generation experimental tablet [10]. The setup uses a Pandaboard, which contains a Texas Instruments OMAP4460 SoC, connected to a Wacom digitizer and a mini projector projecting directly in the Wacom sensor to create a static, table top device. The Wacom module with a serial interface simplifies the interface and avoids the additional power of the USB stack that runs on the ARM Cortex-A9 core. This simplification is essential to isolate user interface power from other effects. The platform runs custom software implementing typical handheld functionality such as page turning, scrolling with inertia, inking, and text entry using either a virtual keyboard or a physical keyboard that can be connected to the system through a serial interface.

The simulator takes as input the reading, inking, and text extry input traces previously captured on the platform by Kihm et al. [10] and summarized in Table I. Each of the nine user participants in the study, all of whom had prior experience using mobile devices, read four text pages and typed and wrote short texts at their own pace. Typing tasks were performed with both physical and virtual keyboards. Each of the captured tasks comprises multiple sub-tasks. For instance, the physical keyboard typing task consists of keypress processing plus the addition of a blank line whenever the bottom of the screen is reached. (The virtual keyboard task loads a blank paragraph.)

More details on the platform and the input trace gathering methodology can be found in [10].

B. Benchmark Generation

From the user traces, we derive the average injection periods (average time between back-to-back packets, or the inverse of the injection rate) for the different sub-tasks

Sub-task	Injection Period [ms]
Phys Key Press	248.0
Blank Line Load	248.0
Virt Key Press	70.2
Shift/CAPS	70.2
Blank Paragraph Load	70.2
Inking	14.4
Blank Page Load	719.0
Book Page Load	35,365.8

TABLE II: Sub-task average injection periods.

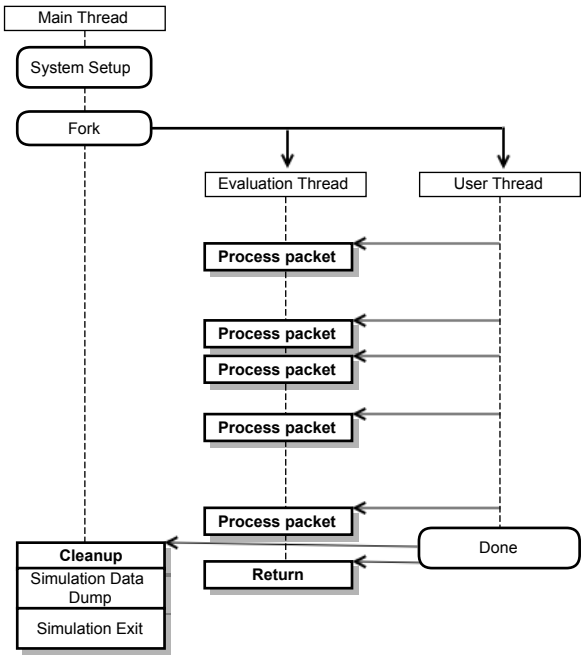

Fig. 2: Operation of the simulator, including *main*, *evaluation*, and *user* threads.

(Table II). The periods range from a low of 14.4ms (highest rate) for inking to over 35,000ms for Book Page Load, which only occurs when the user has finished reading a page. Since the minimum time between consecutive events in all the traces is much longer than the time to enter a low power idle mode, we adopt an average injection rate model, where the energy of each event includes the energy to process the event plus the average idle time until the next event. We then combine these per-event energies across sub-tasks to determine the energy cost of typing, inking, and reading.

C. Performance Modeling

The user interface application was ported to the GEM5 ARM-based performance model with integrated 45nm McPAT power models. The simulator uses Syscall Emulation mode, which necessitates static compilation of the binaries. This removes any dependence on target operating system shared library handling and resulting variability. Since the out-of-order processor models (described below) include SIMD units, we compiled the code for these models with the following flags to turn on auto-vectorization: –vectorize –cpu=Cortex-A9 -O3 -0time.

Frequency (MHz)	1200	1000	800	600	300
Voltage (V)	1.4	1.3	1.2	1.1	1.025

TABLE III: Core frequency and voltage settings.

The user trace hardware captures the precise timing of the occurrence of sub-task events for injection into the simulator. As shown in Figure 2, after system setup, the simulator spawns an *evaluation thread* that runs the performance simulator, and a *user thread* that runs on a separate core for the sole purpose of creating a packet injection stream of inputs (e.g., characters from a keyboard) that matches the relative timing of the real input trace in simulation. (A post-processing step removes the power generated by all but the evaluation thread.) The simulator then measures the real time response time between a packet injection and the required screen event response by the user interface software running on a particular hardware architecture. This capability permits the exploration of real time response versus energy tradeoffs of different hardware designs. So long as the real time delays do not exceed the user perception time [3, 6], as is the case in our system, these benchmarks can be run open loop, since they will not affect user behavior.

D. Power Modeling

The maximum and minimum processor core voltage and frequency points are taken from the Cortex-A9 core in the OMAP4430, which is implemented in 45nm technology, the same as the McPAT power models. A number of intermediate points, derived by geometric scaling from the end points, are also modeled as shown in Table III. McPAT pipeline and cache models are generated for the highest frequency and voltage for each core model. GEM5 performance results of each sub-task are then generated at each frequency point and core model and post-processed by the McPAT model. For the lower frequency and voltage points, the dynamic power values generated by McPAT are scaled by V^2f and the static values by V [17]. This approach is adopted as an alternative to having McPAT generate a different pipeline and cache model for each frequency and voltage point, which could significantly skew the results.

In conjunction with the McPAT power models, we also model transitions to and from power management sleep modes, which may be entered between input packet processing. In these modes, the application is in wait-for-interrupt (WFI) mode and aggressively attempts to save energy. The characteristics of the sleep mode are defined by several simulator input parameters:

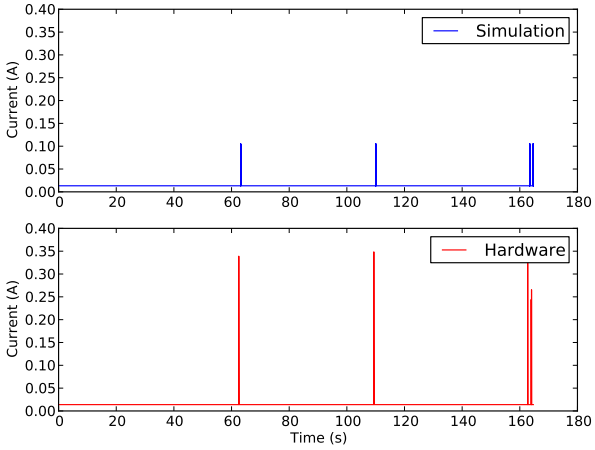

Fig. 3: Comparison of simulation (top) and hardware measurement (bottom) for reading task.

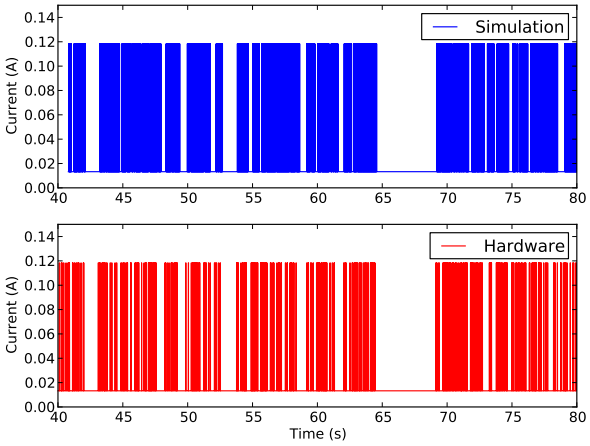

Fig. 4: Comparison of simulation (top) and hardware measurement (bottom) for inking task.

- Power gating wake-up/sleep delay
- Power gating slowdown due to the footer transistor
- Dynamic voltage scaling delay
- Clock gating delay
- Static power reduction

We explore three sleep state models in our experiments:

- Clock Gating / Dynamic Voltage and Frequency Scaling (CGDVFS), where the clock is gated and the voltage and frequency scaled to their minimum settings.
- Ideal Power Gating (IdealPG), an idealized, zero-overhead, power gating model.
- Realistic Power Gating (RealPG), which models the performance degradation from the footer transistor, and assumes non-ideal static power reduction in the idle state.

The framework is easily extendable to include other power management schemes.

E. Post-simulation Analysis

This is the final phase, which combines the performance and power modeling results into the final timing model constructed using the power management assumptions for idle states. Here, the user can turn various knobs to explore the design space, set deadlines for tasks, and compare different architectures.

F. Simulator Validation

The infrastructure leverages established computer architecture simulation tools–GEM5 and McPAT–for detailed system modeling. While these tools have been extensively validated against real hardware [11, 12, 18], we also validated our model against actual hardware using data available from Kihm et al. [10]. In that user study, the authors measured the current consumption of the Cortex-A9, Cortex-M3 cluster, and memory system power domains. The Cortex-M3 cluster power domain includes I/O devices and various busses, making it difficult to validate an M3-type processor core. The Cortex-A9 power domain is better suited for this comparison, since it doesn't include other SoC subsystems. Thus, we compare the Cortex-A9 measurements from [10] with ARM-based GEM5 and McPAT simulation results using the OOO2 model (Table VI).

Due to limitations of the hardware measurements [10], we only had access to instantaneous current data from the user studies and not the power data. The reason is that all measurements were conducted from the pads on the Pandaboard printed circuit board (PCB), and voltage could not be measured since it was controlled on-chip. Another limitation was the coarse grain level of the measurements (millisecond scale). Thus, the keyboard and virtual keyboard processing activity could not be accurately captured due to their extremely low activity factor and very short packet processing time.

However, the reading and inking tasks lended themselves well to measurement and validation. While the reading task has low activity, processing a reading trace packet takes significant processing time, resulting in a relatively wide and high pulse, which can be captured with high accuracy. While the inking task packets are processed quickly, they are plentiful so they can be grouped into a single pulse of activity that is easily observable on the measurement scale.

To compare with the hardware current readings, we divide the simulation power values by voltage to obtain instantaneous current. However, there are two differences between the hardware and simulator that require calibration of our model. First, we do not model in-rush currents from DVFS or power gating. Second, we do not model operating system (OS) effects such as extra work, scheduling, OS quantum effects, and delays in waking up from interrupt. To bridge this modeling gap, we assume a simple OS model that wakes up the processor upon interruption to process a task and then puts the processor

Metric:	Idle(s)	Total(s)	Error	Charge[A s]	Error
Sim 1	164.7	164.7	0.1%	2.2	-6.0%
HW 1	164.5			2.3	
Sim 2	221.5	221.6	0.1%	3.0	11.5%
HW 2	221.3			2.6	
Sim 3	242.5	242.6	0.0%	3.2	1.0%
HW 3	242.4			3.2	
Sim 4	191.5	191.6	0.1%	2.6	1.6%
HW 4	191.4			2.5	

TABLE IV: Percentage error in percent idle time and drawn battery charge for simulation versus hardware measurements [10] for four user traces of the reading task.

Metric:	Idle(s)	Total(s)	Error	Charge[As]	Error
Sim 1	122.9	129.2	2.4%	2.3	-14.8%
HW 1	120.0			2.7	
Sim 2	101.2	104.7	-0.2%	1.8	0.7%
HW 2	101.4			1.7	
Sim 3	162.5	170.4	-1.3%	3.0	5.8%
HW 3	164.8			2.9	
Sim 4	116.4	123.5	2.4%	2.7	-10.8%
HW 4	113.6			2.4	

TABLE V: Percentage error in percent idle time and drawn battery charge for simulation versus hardware measurements [10] for four user traces of the inking task.

to sleep a specified amount of time after activity ends. In order to pick the right value for this sleep parameter, we trained the model on a separate training data set, which we did not use for testing. We also trained the static and dynamic power levels since we only aimed to model relative power trends. For this we also picked training sets that we did not use in testing.

In order to make a comparison between the hardware measurements and the simulation, we quantized the raw hardware data using its own statistics. Due to the nature of the measurements (connecting probes directly to the PCB, and the inability to separate spurious current effects due to OS activity and switching states), the raw data contained significant noise. For tasks such as reading where the activity factor is low (<1%), we pick the mean value to represent the low quantization level as this will mostly represent the mean value of the idle phase, and the active phase will have minimal effect. For tasks such as inking with higher activity factors (5-10%), we chose a separate trace that had no activity and used its mean value for the low quantization level. For the high quantization level, we picked the portion of a trace where the user actively performs the task and used the mean value. We also computed the standard deviation of the noise in order to pick a threshold about which to quantize. We picked the threshold to be at low quantization level plus six sigma, since the traces that we analyzed are long (100-300 seconds) with millisecond resolution and we desired a low false positive expectation.

Figures 3 and 4 show the resulting comparisons of the simulation and hardware data after post-processing and training. For the reading task (Figure 3), the spikes represent the user flipping to the next page in the reading app. The hardware activity is closely matched by the simulation data. The amplitudes are mismatched because we trained our simulation on active current for an inking trace, while the reading trace active current is significantly higher (this contributes to some of the error shown later in Table IV). The inking task has much more activity than the reading task. Therefore, we visually compare the idle periods, which are shown to be relatively well matched in Figure 4.

In addition to visual comparison, we performed a quantitative analysis of the reading and inking tasks. In our analysis, we used several user traces and compared the simulation results to the data measured on the hardware platform. We used traces from four different users for each task and evaluated two metrics: the percentage of idle time (an important metric given that many HCI tasks are dominated by idle time) and the drawn battery charge. The results are shown in Tables IV and V. For both tasks, the idle time percentage is extremely well matched, with virtually no error for reading and less than 2.5% error for inking. Thus, for the reading and inking tasks, the combination of the previously shown close visual match and the low error in idle time percentage indicate good simulation accuracy with respect to the processor current profile, which is a proxy for the power profile.

In addition to the power profile, another important metric is the drawn battery energy, for which we use as a proxy the drawn battery charge, which we can obtain by integrating the current profile over time. The results are shown in the right columns of Tables IV and V. The result here is the accumulated error from mismatching low and high power levels as well as mismatching active and idle portions. While this results in a higher error than for percent idle time, the correlation between the hardware and simulator is still good, with the maximum error less than 15%.

IV. Design Exploration of Energy-Efficient HCI Event Processors

Unlike prior work [10], our modeling methodology permits the evaluation of general-purpose architectures for HCI processing, and the development of more energy-efficient designs for current interface technologies and those expected in the future. In this section, we apply our modeling infrastructure to the design of such specialized processors and their per-core power management strategies. We quantitatively compare these designs against general-purpose single core and big.LITTLE implementations and show significant gains in energy efficiency.

A. Design Parameters and Alternatives

We define at a range of core performance/power points that comprise both in-order and out-of-order designs with

Parameter	IO1-1K	IO1	OOO1	OOO2	OOO3	OOO8
Fetch/Dispatch/Issue width	1/1/1	1/1/1	1/1/1	2/2/2	3/3/3	3/6/8
Out-of-order?	No	No	Yes	Yes	Yes	Yes
Issue Queue entries	-	-	8	16	24	32
ROB entries	-	-	10	20	30	40
Int/FP ALUs	1/1	1/1	2/2	2/2	2/2	2/2
Int Mpy/Div	1	1	1	1	1	1
L1 I/D Caches (KB)	1/1	32/32	32/32	32/32	32/32	32/32
I/D MSHRs	1/1	1/1	2/2	2/4	2/6	2/6

TABLE VI: Summary of core model characteristics.

the parameters shown in Table VI. The models range from a simple scalar in-order model (*IO1*) to the most complex out-of-order superscalar ARM model provided with GEM5 (*OOO8*). For all models, we use 32KB two-way set associative instruction and data caches, except for IO1-1K, which has a smaller 1KB caches that may be suitable for the small instruction and data footprints of some sub-tasks. None of the models include an L2 cache. We examined the performance benefit of L2 caches of sizes up to 32MB and found they had little impact on response time. We use the GEM5 LPDDR2-S4 main memory system model.

For those HCI tasks that include long idle periods, idle time power management is critically important for energy efficiency. Therefore, we have built into our methodology three idle time management policies. The two main policies are Clock Gating with Dynamic Voltage and Frequency Scaling (CGDVFS), and Power Gating (PG). For the former, we stop the clocks and then transition to the lowest voltage level. For the latter, we explore two models: Ideal PG and Real PG. The Ideal model assumes perfect core and cache power gating (and thus no idle power) and no performance penalty for power gating. This shows the absolute limit of what power gating could theoretically provide. The Real PG model includes leakage power while power gated and the performance degradation resulting from the footer transistor. We assume 80% static power reduction during idle periods and 20% performance degradation during active times, based on the power gating studies and evaluation of various techniques in [13].

We assume transition times of 1 µs and 300 µs for the CGDVFS and PG models, respectively. Transition times are incurred at the arrival of a new event packet and at the conclusion of its processing. The larger transition time for power gating accounts for the loss of state and the need to control in-rush currents.

B. Analysis of HCI Sub-Task Latency and Energy

In order to propose practical designs, we first gain an understanding of the latency and energy of different HCI sub-tasks for the different core models, for both clock gating/DVFS and power gating. We then use this information to determine the number of cores and their types to incorporate into a multicore design. While our analysis is specific to our sub-tasks and particular hardware assumptions, the methodology is generally applicable to a range of sub-task types and hardware designs.

1) Sub-Task Characterization Under Idle Time Clock Gating/DVFS: Figure 5 shows the latency versus energy for processing events of each sub-task for the different core types, assuming idle time clock gating/DVFS. Each curve includes the five frequency and voltage settings; the top point is the lowest setting. The energy results account for both the processing of the event and the energy of the average idle time between back-to-back events. Thus, infrequent events will have a higher idle time energy component[1].

For the simpler and low event rate sub-tasks such as the external keyboard, the per-event energy is relatively independent of clock frequency. This is due to the fact that the idle time energy predominates due to the low event rate and low processing time. Thus, the greater dynamic power of the higher frequency cores has little impact on per-event energy.

The lower per-event energy of the virtual keyboard compared to the physical keyboard arises from its higher injection rate due to repeated pressure events. This leads to more events and thus on average less idle time between events.

For the PDF rendering sub-task, energy is dominated by the active component. As frequency increases so does dynamic energy but static energy reduces due to the shorter processing time. Thus, the lowest energy point lies at an intermediate frequency that best balances the two.

As expected, the keypress and inking sub-tasks have very low response latencies, at most tens of μs even with the simplest core. The remaining sub-tasks require manipulating a larger portion of the screen and thus have orders of magnitude higher latencies, for even the most complex cores. The PDF rendering sub-task has some design points with latencies in the hundreds of ms range, the highest of all sub-tasks.

[1]The PDF rendering sub-task ignores idle time since this is included in page loading, which together with PDF rendering encompasses the reading task.

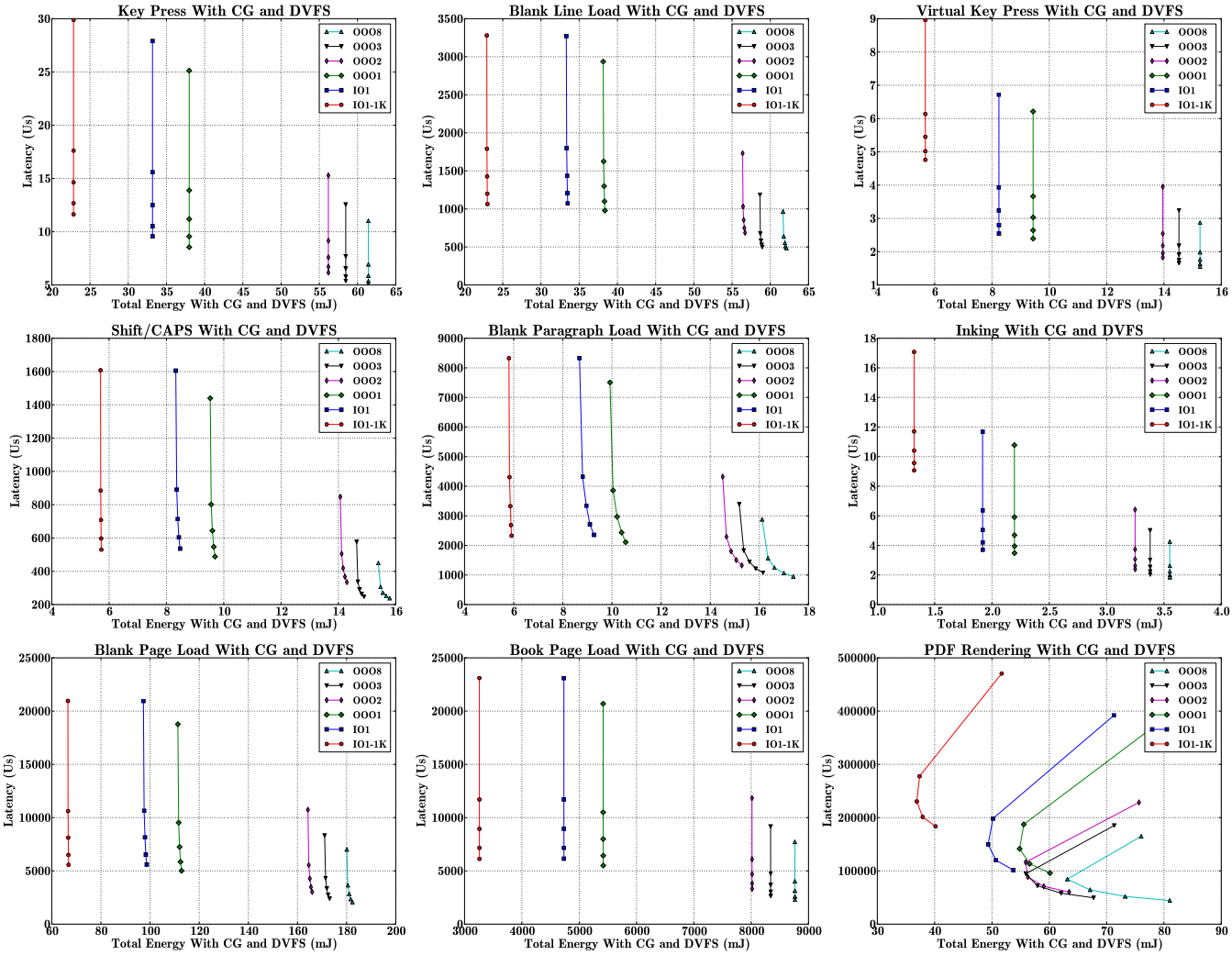

Fig. 5: Sub-task per-event latency and energy for clock gating/DVFS.

The large disparity between the simpler keypress and inking tasks and the display rendering tasks motivates the need for a customized heterogeneous multicore design. As shown later, the page load tasks require an OOO8 core for the stringent real time deadlines, while the simplest core suffices for the simpler tasks. The use of the OOO8 core for these simple tasks would more than double or triple the per-event energy over the simpler core. Coupling a very simple core with the complex one achieves large energy savings over either a single "do everything" core or a general-purpose big.LITTLE design, as we will demonstrate in Section IV-C.

2) Sub-Task Characterization Under Idle Time Power Gating: We also explore the impact on power and latency of realistic power gating (Figure 6). As expected, the per-event energy is dramatically reduced compared to idle time clock gating/DVFS. The effect is more pronounced for simple sub-tasks with long idle times. However, because the idle time power is eliminated, the lowest energy configuration is no longer the one with the lowest frequency (as with DVFS) but rather the one with the best balance between dynamic and static energy. Moreover, since per-event energy is dominated by active time energy, designs that process extra packets can consume an order of magnitude more energy. For example, using our infrastructure, we found this to be the case in platforms with proximity sensing (hovering), a detailed analysis of which we leave for future work.

The response latencies also increase with power gating due to the higher transition times. The effect is more pro-

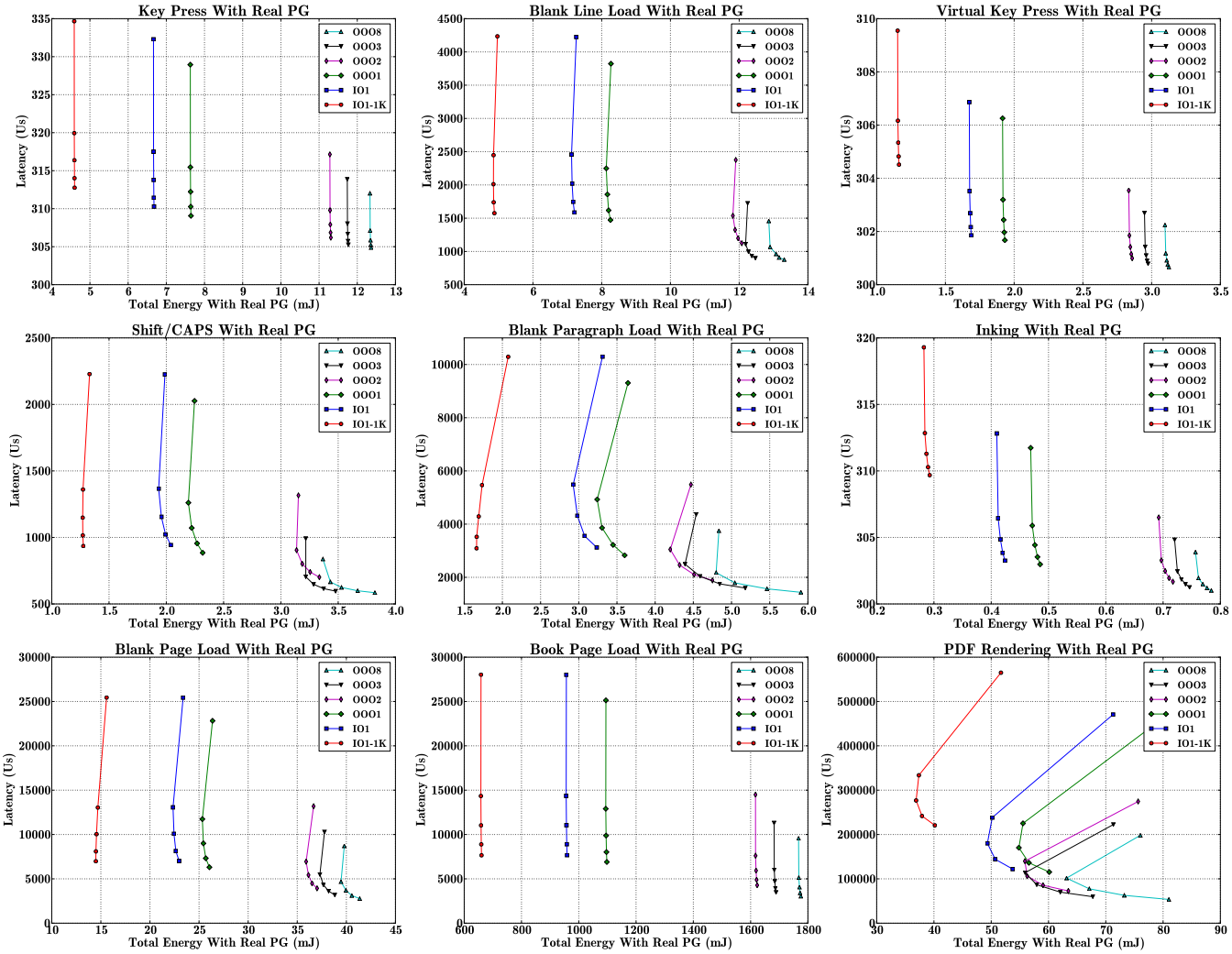

Fig. 6: Sub-task per-event latency and energy for realistic power gating.

nounced for the keypress and inking sub-tasks, although their latencies remain far below that of the other sub-tasks.

C. Energy-Efficient Customized Designs

In this section, we explore the event processor design space with implications on system design. We view the interactive system design problem as a constrained design problem, where we have L_{total} ms of budgeted latency and we need to pick a sensor, a display, and an event processor, where $L_{disp}+L_{sense}+L_{eventProc}<L_{total}$ with an objective to maximize battery life within reasonable cost constraints. Our proposed toolbox explores the design space for $L_{eventProc}$ with all the consequences on energy-efficiency/battery life, as we showed in the previous section. We analyzed CGDVFS and realistic power gating with three latency deadlines: 7ms, 4ms, and 2ms to explore progressively more aggressive architectures, each of which may be applicable for a different combination of input and display technologies.

For both clock gating/DVFS and power gating, the simplest IO1-1K core suffices for all sub-tasks for the slowest 7ms latency deadline, although this requires using almost the entire range of frequencies. As the deadline becomes more stringent, the more complex cores are required. The mid-range OOO2 and OOO3 cores are required to meet the deadline at 4ms, and the most complex OOO8 core at full frequency is needed to meet the 2ms deadline for several sub-tasks. The best single and dual core configurations also require the more complex core. However, the IO1-1K core at the lowest frequency always suffices for the keypress and inking sub-tasks.

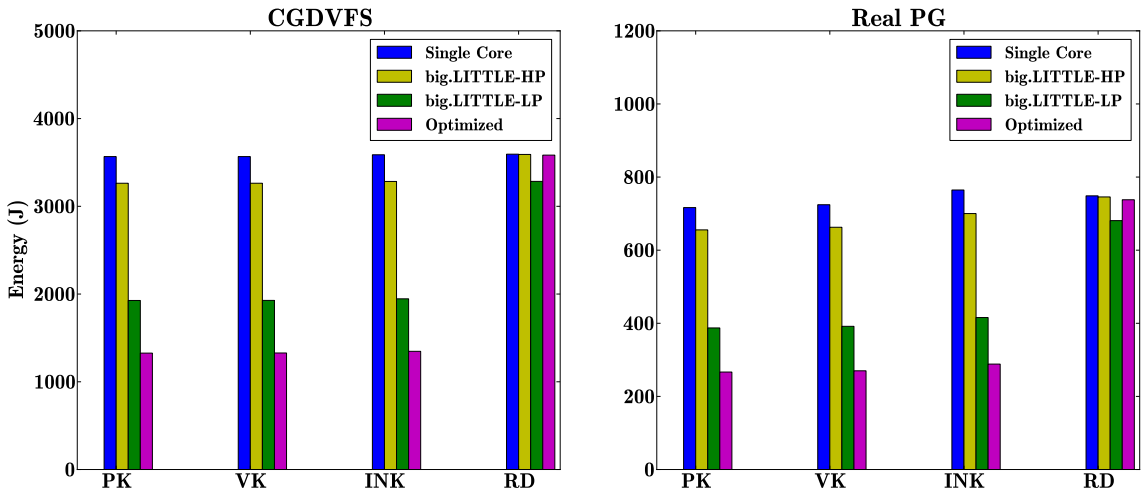

Fig. 7: Overall energy usage for the best single core (SC) and best dual core (DC) for clock gating with DVFS and Real PG running each task for 4 hours. PK = physical keyboard, VK = virtual keyboard, INK = inking, RD = reading.

Our methodology shows that, for our given subtasks, the IO1-1K core is an essential component as shown in Table VII. Moreover, we show that the most energy efficient frequency is not always the minimum one but that which best balances active-time dynamic and idle-time static power.

We use this insight into the processing requirements of the different sub-tasks to evaluate the energy usage of the Physical Keyboard Typing (*PK*), Virtual Keyboard Typing (*VK*), Inking (*INK*), and Reading (*RD*) tasks. We compare the customized configurations with the best single core design and our best effort attempt (within the constraints of GEM5) to model two big.LITTLE architectures: (1) big.LITTLE-LP, a low-power configuration comprising the OOO2 and IO1 cores (inspired by OMAP); and (2) big.LITTLE-HP, a higher performance configuration consisting of the OOO8 and OOO2 cores (inspired by Cortex-A57+A53). All configurations use identical idle time power management approaches.

We assume a system similar to [9, 10] in which sub-tasks are statically assigned to a specific core and do not migrate to the other core during their allotted time schedule. When a sub-task request arrives, the lowest power core wakes up and decides whether to perform the work or to wake up the other core to perform the sub-task.

Table VIII shows the lowest energy cores used by each of the two big.LITTLE configurations to meet the deadline, as well as the increased energy overhead compared to the customized dual core design for the 2ms deadline. For the LP configuration, the overheads are 40-50% for many sub-tasks, and in several cases it misses deadlines (indicated by a negative performance overhead). While the HP big.LITTLE meets the deadlines, its energy overheads with respect to the customized design approach 2.5X for a number of sub-tasks. For blank paragraph load, big.LITTLE-HP uses slightly less energy (-5.8%) than the customized dual core. This is because in this sub-task OOO2 happens to be optimal. However, for a dual core design, the best overall choice is to use OOO8 coupled with IO1-1K in order to meet the deadlines and use the least possible energy overall. For the 7ms and 4ms deadlines, big.LITTLE-LP expends as much as 51-53% and 76-85% more energy over our approach. For big.LITTLE-HP, the energy overhead increases to 149-150% and 153% for CG and PG, respectively.

Figure 7 shows the energy consumption with four hours of operation allotted to each of the four tasks (physical keyboard, virtual keyboard, inking, and reading) for the best single core, big.LITTLE designs, and customized dual core under the 2ms latency requirement. Overall, the heterogeneous dual core achieves significant energy savings over the single core and big.LITTLE designs. For reading, the LP design uses less energy since it misses the deadline.

For idle time clock gating/DVFS, while the savings for the reading task is less than 1%, for the physical keyboard, virtual keyboard, and inking tasks, the energy savings rise to 62-63%, or 2238-2240J in absolute terms. With respect to big.LITTLE those savings are 30-31% for the LP and 58-59% for the HP configurations. Assuming a 23.8Wh battery as in the iPad Mini 3 [19], four hours of performing these tasks using the single core and big.LITTLE-HP designs would drain the battery several percent more than the customized design, a significant difference.

Under the realistic power gating model, the heterogeneous dual core saves a significant amount of energy over the best single core design. Here, the improvement is 1.4% in energy savings for reading but 62-63% for the other three tasks, which translates into 447-495J in absolute energy. In terms of battery life over four hours of continuous operation, the single core design would drain the battery by 0.8-0.9%, but by only 0.35% for the heterogeneous dual-core architecture. The savings over big.LITTLE designs for

	CGDFVS			Power Gating		
	7ms	4ms	2ms	7ms	4ms	2ms
Best Single Core	IO1-1K	OOO2	OOO8	IO1-1K	OOO3	OOO8
Best Customized Dual Core	-	OOO2+IO1-1K	OOO8+IO1-1K	IO1+IO1-1K	OOO3+IO1-1K	OOO8+IO1-1K

TABLE VII: Best single core, and best dual core architecture for CGDVFS and realistic PG for 7, 4, and 2ms deadlines.

Task	CGDFVS				Power Gating			
	big.LITTLE-LP		big.LITTLE-HP		big.LITTLE-LP		big.LITTLE-HP	
	Core	Overhead	Core	Overhead	Core	Overhead	Core	Overhead
Key Press (*PK*)	IO1-300	45.2%	OOO2-300	145.9%	IO1-300	45.2%	OOO2-300	145.9%
Blank Line Load (*PK*)	IO1-600	45.5%	OOO2-300	145.9%	IO1-1008	47.6%	OOO2-600	143.3%
Virtual Key Press (*VK*)	IO1-300	45.2%	OOO2-300	145.9%	IO1-300	45.2%	OOO2-300	145.8%
Shift/CAPS (*VK*)	IO1-300	46.0%	OOO2-300	146.7%	IO1-600	52.2%	OOO2-600	147.0%
Paragraph Load (*VK*)	OOO2-800	-9.2%	OOO2-800	-9.2%	OOO2-1200	-5.8%	OOO2-1200	-5.8%
Inking (*INK*)	IO1-300	45.2%	OOO2-300	145.9%	IO1-300	45.0%	OOO2-300	145.1%
Blank Page Load (*INK*)	OOO2-1200	-9.0%	OOO8-1200	0.0%	OOO2-1200	-10.5%	OOO8-1200	0.0%
Book Page Load (*RD*)	OOO2-1200	-8.5%	OOO8-1200	0.0%	OOO2-1200	-8.6%	OOO8-1200	0.0%
PDF Rendering (*RD*)	IO1-800	33.7%	OOO2-600	51.5%	IO1-800	33.7%	OOO2-600	51.5%

TABLE VIII: Best core configuration per sub-task and per task for the big.LITTLE high performance and low power designs and their respective per-subtask energy overheads versus the customized dual-core design for 2ms deadline.

PG are similar to those using CGDVFS - ~30% over LP and ~60% over HP.

V. Conclusions

Recent improvements in input sensor and display technologies coupled with increasing demand for faster interactive response calls for rapid yet power efficient interactive task processing. We developed a toolbox that models real-time human interactions with a mobile device, as well as architectural details such as event processing latency and idle-time power management. We validated our models against a hardware platform from prior work. We presented a design methodology that customized event processing architectures to maximize energy efficiency based on UI latency constraints from the system design.

We used this methodology to explore energy-efficient HCI task processing hardware and quantified the energy savings compared to single core and big.LITTLE architectures. We showed savings approaching 2.5X (60% improvement) on some sub-tasks over the high performance big.LITTLE approach, and roughly 1.5X (33% improvement) over the low power big.LITTLE design that could not meet all of the deadlines. Overall, we showed how our methodology can be applied to the design of HCI-specific hardware, and how such designs can achieve significant energy efficiency improvements over general-purpose approaches.

References

[1] Sony Inc. OLED Sony Technical Report. Available at http://pro.sony.com/bbsccms/assets/files/micro/OLED/brochures/oled_techhndbk_di0248_f.pdf.

[2] Qualcomm. Interferometric Modulator (IMOD) Technology Overview. Available at "http://www.qualcomm.com/sites/default/files/uploads/imod_tech_overview-06-2009.pdf", 2009.

[3] Ricardo Jota, Albert Ng, Paul Dietz, and Daniel Wigdor. How Fast is Fast Enough?: A Study of the Effects of Latency in Direct-touch Pointing Tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2291–2300, New York, NY, USA, 2013. ACM.

[4] Ricardo Jota, Clifton Forlines, Darren Leigh, Steven Sanders, and Daniel Wigdor. Towards Zero-Latency User Experiences. Available at "http://www.tactuallabs.com/press/GDCTactual32014.pdf", 2014.

[5] Stuart K. Card, George G. Robertson, and Jock D. Mackinlay. The Information Visualizer, an Information Workspace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '91, pages 181–186, New York, NY, USA, 1991. ACM.

[6] Albert Ng, Julian Lepinski, Daniel Wigdor, Steven Sanders, and Paul Dietz. Designing for Low-latency Direct-touch Input. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 453–464, New York, NY, USA, 2012. ACM.

[7] Paul Ditz. Applied Sciences Group: High - Performance Touch. Available at http://research.microsoft.com/apps/video/default.aspx?id=160670.

[8] Christian Holz and Patrick Baudisch. Understanding Touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2501–2510, New York, NY, USA, 2011. ACM.

[9] François Guimbretiére, Shenwei Liu, Han Wang, and Rajit Manohar. An Asymmetric Dual-processor Architecture for Low-power Information Appliances. *ACM Trans. Embed. Comput. Syst.*, 13(4):98:1–98:19, March 2014.

[10] Jaeyeon Kihm, François V. Guimbretière, Julia Karl, and Rajit Manohar. Using Asymmetric Cores to Reduce Power Consumption for Interactive Devices with Bi-stable Displays. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1059–1062, New York, NY, USA, 2014. ACM.

[11] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. The Gem5 Simulator. *SIGARCH Comput. Archit. News*, 39(2):1–7, August 2011.

[12] Sheng Li, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. The McPAT Framework for Multicore and Manycore Architectures: Simultaneously Modeling Power, Area, and Timing. *ACM Trans. Archit. Code Optim.*, 10(1):5:1–5:29, April 2013.

[13] Carlos Ortega, Jonathan Tse, and Rajit Manohar. Static power reduction techniques for asynchronous circuits. In *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, pages 52–61. IEEE, 2010.

[14] Mengying Zhao, Hao Zhang, Xiang Chen, Yiran Chen, and Chun Jason Xue. Online OLED Dynamic Voltage Scaling for Video Streaming Applications on Mobile Devices. In *Proceedings of the Ninth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, CODES+ISSS '13, pages 9:1–9:10, Piscataway, NJ, USA, 2013. IEEE Press.

[15] Bhojan Anand, Karthik Thirugnanam, Jeena Sebastian, Pravein G. Kannan, Akhihebbal L. Ananda, Mun Choon Chan, and Rajesh Krishna Balan. Adaptive Display Power Management for Mobile Games. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 57–70, New York, NY, USA, 2011. ACM.

[16] Nachiappan Chidambaram Nachiappan, Praveen Yedlapalli, Niranjan Soundararajan, Mahmut Taylan Kandemir, Anand Sivasubramaniam, and Chita R. Das. GemDroid: A Framework to Evaluate Mobile Platforms. *SIGMETRICS Perform. Eval. Rev.*, 42(1):355–366, June 2014.

[17] J. Adam Butts and Gurindar S. Sohi. A Static Power Model for Architects. In *Proceedings of the 33rd Annual ACM/IEEE International Symposium on Microarchitecture*, MICRO 33, pages 191–201, New York, NY, USA, 2000. ACM.

[18] A. Gutierrez, J. Pusdesris, R.G. Dreslinski, T. Mudge, C. Sudanthi, C.D. Emmons, M. Hayenga, and N. Paver. Sources of error in full-system simulation. In *Performance Analysis of Systems and Software (ISPASS), 2014 IEEE International Symposium on*, pages 13–22, March 2014.

[19] Apple. iPad Mini 3 - Technical Specifications. Available at http://http://www.apple.com/ipad-mini-3/specs/.