GemDroid: A Framework to Evaluate Mobile Platforms

Nachiappan Chidambaram Nachiappan[†] Praveen Yedlapalli[†] Niranjan Soundararajan[§]
Mahmut T. Kandemir[†] Anand Sivasubramaniam[†] Chita R. Das[†]
[†]The Pennsylvania State University [§]Intel Corp.
{nachi, praveen, kandemir, anand, das}@cse.psu.edu {niranjan.k.soundararajan@intel.com}

ABSTRACT

As the demand for feature-rich mobile systems such as smartphones and tablets has outpaced other computing systems and is expected to continue at a faster rate, it is projected that SoCs with tens of cores and hundreds of IPs (or accelerator) will be designed to provide unprecedented level of features and functionality in future. Design of such mobile systems with required QoS and power budgets along with other design constraints will be a daunting task for computer architects since any ad hoc, piece-meal solution is unlikely to result in an optimal design. This requires early exploration of the complete design space to understand the system-level design trade-offs. To the best of our knowledge, there is no such publicly available tool to conduct a holistic evaluation of mobile platforms consisting of cores, IPs and system software.

This paper presents GemDroid, a comprehensive simulation infrastructure to address these concerns. GemDroid has been designed by integrating the Android open-source emulator for facilitating execution of mobile applications, the GEM5 core simulator for analyzing the CPU and memory centric designs, and models for several IPs to collectively study their impact on system-level performance and power. Analyzing a spectrum of applications with GemDroid, we observed that the memory subsystem is a vital cog in the mobile platform because, it needs to handle both core and IP traffic, which have very different characteristics. Consequently, we present a heterogeneous memory controller (HMC) design, where we divide the memory physically into two address regions, where the first region with one memory controller (MC) handles core-specific application data and the second region with another MC handles all IP related data. The proposed modifications to the memory controller design results in an average 25% reduction in execution time for CPU bound applications, up to 11% reduction in frame drops, and on average 17% reduction in CPU busy time for on-screen (IP bound) applications.

SIGMETRICS'14, June 16–20, 2014, Austin, Texas, USA.

http://dx.doi.org/10.1145/2591971.2591973.

Categories and Subject Descriptors

C.1.4 [**Processor Architectures**]: Parallel Architectures—*Mobile Processors*; I.6.5 [**Computing Methodologies**]: Simulation and Modeling—*Model Development*

General Terms

Measurement, Performance

Keywords

Memory Optimization; Simulation; SoC Modeling; Metrics;

1. INTRODUCTION

There is an exploding demand for mobile systems, which include smartphones, tablets, and wearable devices. Gartner research projects that 2 billion of these units will be sold in 2013 [15] and there will be over 10 billion mobile devices by the end of 2017 [8]. Moreover, it is projected that global mobile data will increase 13-fold between 2012 and 2017 reaching 11 Exabytes per month and two-thirds of this data is projected to be video data [8]. These numbers clearly indicate the importance of designing feature-rich mobile devices to cope up with the market demand. The ITRS roadmap for designing System-on-Chip architectures (SoCs) over the next decade projects that a mobile device could have up to 50 processing cores with about 300 TFLOPS computing capability and more than 400 IP blocks for enabling such feature-rich mobile platforms [14]. Thus, major companies like AMD, ARM, Apple, Intel, NVidia, Qualcomm, and Samsung have already ventured into this growing market targeting devices ranging from wearable wrist watches, glasses, to hand-held smartphones, phablets and tablets. Design and analysis of these devices with required QoS provisioning, power budgets and evolving technology artifacts is a daunting task that computer architects have to deal with in the coming years.

Mobile systems are based on the SoC design philosophy, having the core(s) (CPUs) and multiple accelerators on die or as part of the complete platform. These accelerators, or IPs [1] as they are commonly called, are customized to implement specific functionalities very efficiently, and hence

The authors would like to confirm that this work is an academic exploration and does not reflect any effort within Intel.

[1]We use the term accelerator or IP interchangeably throughout this paper.

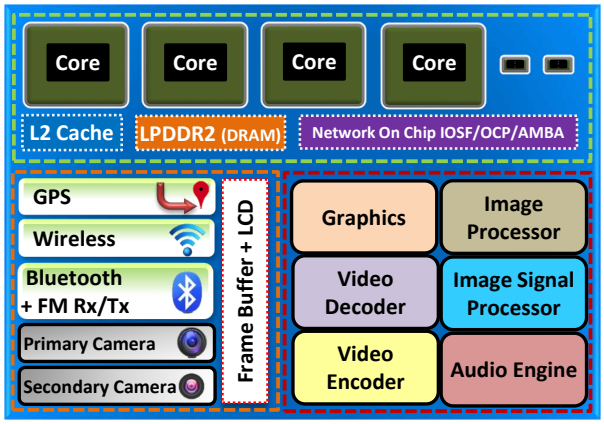


Figure 1: A sample SoC Platform with a high-level view of different functional blocks.

off-load the work from the core. As the IPs are task specific hardware components, they provide high performance delivering superior performance-per-watt compared to running the same task on cores. The set of IPs employed by a typical mobile platform includes the graphics, video encoder, video decoder, imaging, modem, communications (e.g., Wi-Fi, bluetooth) and others as shown in Figure 1.

Given the growing need for these mobile devices that have become an essential part of our daily life, it is essential that we understand the underlying platform issues better to develop more efficient designs from the performance and power standpoints. Also, the growing number of use-cases which get integrated into these devices and the associated software complexity can often lead to conflicting performance and power requirements all of which need to be analyzed carefully for optimal design decisions. The important research questions relevant to emerging mobile platforms include: (i) topology exploration that determines the ideal number and types of cores, and types of IPs to best match the demands of the target workloads; (ii) design of scalable on-chip networks to cater to the divergent needs of IPs and cores; (iii) design of memory schedulers and scheduling algorithms that can handle both IP and core requests; and (iv) workload mapping and scheduling algorithms to maximize performance and minimize energy consumption. Clearly, answering these types of questions requires access to a simulation framework, capable of *capturing such issues and enabling exploration of design alternatives/tradeoffs at the complete platform level.*

While there are existing individual IP [9,42] and core simulators [5], to the best of our knowledge, there is *no* open source integrated simulation platform to conduct holistic studies of a complete mobile platform that can (i) capture activities across multiple IPs and cores, (ii) run an operating system (like Android [16]), (iii) execute real-world applications to capture and analyze realistic events, (iv) simulate architectural features such as core, caches, network and memory, in detail to understand the application-architecture interactions, and (v) provide various application level and IP-specific metrics in addition to global (chip-wide) performance metrics.

In this work, we intend to fill this critical void by making the following contributions:

• We propose a comprehensive simulation infrastructure, called **GemDroid**, which incorporates the GEM5 architecture simulator [5], Attila graphics simulator [9] and internal models for the other IPs, and, is capable of running the Android mobile OS [16] for facilitating mobile platform design and optimization research. GemDroid is comprised of two primary layers. The first layer provides emulation of Android OS by the Google Android Emulator [17] and allows us to capture system-level interaction between multiple IPs and I/O devices, including OS activities. What the emulator cannot provide is the timing information of different IP activities and therefore, as our second layer, we integrate/build the timing piece using existing simulation platforms or model them analytically as needed for different IPs. The framework is flexible for integrating models of varying complexities for the cores and IPs.

• Using several smartphone/tablet applications such as games, video-playback, video-recording, as well as core-centric workloads that run on Android, we demonstrate that it is possible to simultaneously capture the activities of the cores and IPs for conducting a multitude of design and optimization studies, and focus on analyzing the memory system performance in this work.

• We demonstrate through extensive workload analysis, that the shared memory subsystem is a critical bottleneck because of the combined memory requests from the core and IPs with different characteristics. For example, memory access patterns of IPs exhibit high levels of regularity (e.g., sequential data accesses by frame-buffers), as opposed to the memory access patterns of, say, well-known SPEC benchmarks [21]. Similarly, the memory bandwidth demanded by cores and IP when running a video on YouTube are very different from each other as depicted in Figure 2. Specifically, while the core's bandwidth demand is more or less constant (requiring $<$ 0.2GBPS for the studied workloads), display IP's needs are very bursty in nature (needing around 0.8GBPS), and the total bandwidth demand for a video recorder can be much higher than that of the memory system. Moreover, these requests not only differ in terms on bandwidth demands, but also their latency demands are also very different from each other. While IPs have strict latency deadlines that *need* to be met, cores do not have deadlines. On the other hand, IPs have time window before which they can be served without affecting the user-experience, but, each cores memory request directly affects the performance of the system.

• Based on the insights obtained from the characterization of memory requests, we propose a novel heterogeneous memory controller (HMC) design for SoCs, where one MC is dedicated for latency critical core requests and the second MC is optimized to enhance the bank-level parallelism of the memory requests it serves. The two memory controllers are still responsible for two distinct (non-intersecting) address ranges. Our evaluation of this new MC design results in better performance and user experience; specifically, it results in average 25% reduction in execution time for CPU bound applications, up to 11% reduction in frame drops, and on average 17% reduction in CPU busy time for on-screen applications.

2. SYSTEM OVERVIEW

Mobile platforms are system-on-chip (SoC) devices with at least one processor core and specialized accelerators/IPs to which computations get offloaded by the operating system for performance and/or power efficiency. Based on the application characteristics, the work gets split between the

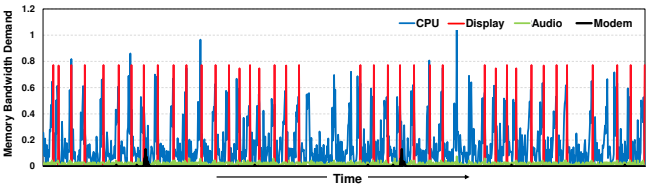


Figure 2: Memory bandwidth demand of cores and IPs when YouTube is played.

core and multiple IPs. Data transfers between the cores and the IPs take place through the main memory. Figure 3 provides an overview of video playback on YouTube highlighting the tasks that are run on the core and those that are run on individual IPs (audio and video HW). The interactions between core, IPs and OS are described next.

2.1 OS-IP Interaction

The phone and tablet operating systems such as Android and iOS include different software drivers to interact with the respective IPs. The device maker optimizes these drivers according to their requirements. Note that a driver acts as the link between the different applications running on the OS and the underlying hardware. Besides the functionality, the drivers also control the power states of the IPs. Android also includes an additional framework layer, which provides an interface for applications to interact with the drivers and the underlying libraries which control the IPs. Further discussion of Android OS can be found elsewhere [16], but it is important to note that the software complexity and features have greatly increased with the newer devices and contribute to an important portion of the overall power, performance and user experience with the device. Together, these play an important role in determining the battery life of the device, which is critical in the mobile ecosystem.

2.2 Core-IP Interactions

One of the main difficulties in analyzing SoCs is that the individual IPs are owned by third party vendors. These are licensed by the phone and tablet makers to build their products. The intricate details of the architecture and the working details of the IPs are not released to the public to maintain the competitive edge. In this part, we briefly explain how an IP works without getting into intricacies about each IP specifically.

Each IP gets a region of memory allocated for it, where its input and output data get stored. The software (OS and IP-driver) stores the data that is supposed to be sent to an IP at the region allocated, and the corresponding address is set in IP registers. The IP independently accesses the data through DMA (direct memory access).

It is important to emphasize that IPs do not directly communicate with the cores. They work based on scheduling triggered from their driver. Different modes of interaction exist – some of the IPs like the display panel operate at a constant rate, where they read their frame buffer at 60 FPS irrespective of when the updates to the frames happen by a core, while others, such as the graphics and imaging IPs, are asynchronously triggered when required.

To get better clarity about the interaction and the consequent impact on overall performance, let us consider Figure 3 that involves video streaming to display.

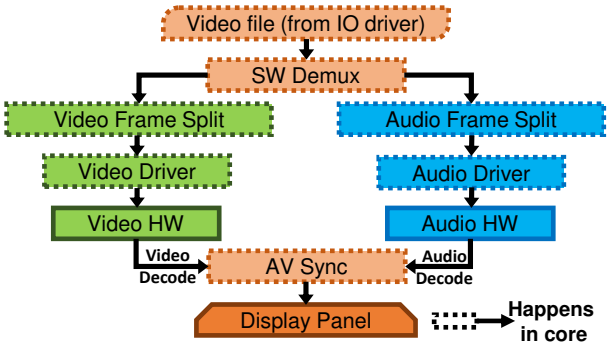


Figure 3: Simplified work-flow diagram of video streaming use case highlighting the core-IPs interaction.

2.2.1 Core-Video/Audio Interaction

Incoming video file gets demultiplexed by the core and the individual frames are marked. The video and audio drivers then direct their respective IPs to pick the frames and decode them at a specific rate. These IPs wait for the core to complete the first step of splitting the audio and video data before starting their specific activities. The performance of the core determines the overall processing rate since it handles these critical portions of the overall use-case.

2.2.2 Video Decoder-Display Interaction

The display panel operates independently of the other IPs since it refreshes the screen continuously. There are three possible scenarios of relative speeds. In the first scenario, where both video decoder (after completing the AV sync) and display work at the same rate, a two-entry frame buffer (FB0 and FB1) is sufficient. When display reads from FB0, video decoder can write to FB1. In the next time frame, display reads from FB1 and video decoder writes to FB0, and this process repeats in a cyclic fashion. In the second scenario, where the video decoder (producing 60 frames per second (FPS)) is working much faster than the display panel (30 FPS), the frame buffer needs to have a larger number of entries to be consumed by the slower display panel. Note that, in the steady state, n/2 frames from the video decoder are read by the display and the other n/2 frames are dropped (50% frame drops). Finally, in the third scenario, where the video decoder is working at a slower pace than the display IP, the system performance is bottlenecked by the video decoder performance. In this case, while computed frames need not be dropped, many frames are skipped from being computed to maintain the sync between audio/video with real-time. Further, the display panel does unnecessary refreshes as the frames are not updated, which is clearly non-optimal from an overall platform power perspective. While the first scenario is the most preferable one, the third scenario is often encountered in most SoC devices.

2.3 Application Requirements

Table 1 lists a set of our target applications and specifies the IPs they employ at some point during their executions. Note that an application can access multiple IPs at different portions of its execution, or send requests to different IPs at the same time. As user requirements increase, the application complexity correspondingly is scaled to cater to their needs for different platforms to provide competing solutions.

Table 1: Shows 9 IPs and two classes of applications (CPU-bound and On-screen applications) evaluated using GemDroid. The table shows IP usage across applications. High/Low indicates a particular IP's dominance in IP utilization compared to others.

	OnScreen-bound								CPU-bound								
Apps / IPs	Game 1	Ar-Game 2	Browser	Video Rec.	Soundplay	Youtube	Video Player	Gallery	Antutu-RAM	RG Bandwidth	Linpack	RLBench	CFBench	Caffein Mark	And-EBench	Antutu-Core	Antutu-GFX
Core/MEM	Low	Low	High	Low	Low	High	Low	Low	High	High	High	High	High	High	High	High	Low
Display-Out	High	High	-	High	-	High	High	Low	-	-	-	-	-	-	-	-	-
Touch-In	Low	Low	-	-	-	-	-	Low	-	-	-	-	-	-	-	-	-
NW-Out	-	-	Low	-	-	Low	-	-	Low	-	Low	-	-	-	-	Low	Low
NW-In	-	-	Low	-	-	High	-	-	Low	-	Low	-	-	-	-	Low	Low
Cam-In	-	High	-	High	-	-	-	-	-	-	-	-	-	-	-	-	-
Aud-Out	Low	Low	-	-	High	-	-	-	-	-	-	-	-	-	-	-	-
GPU	High	High	-	-	-	-	-	-	-	-	-	-	-	-	-	-	High
Vid/Img-Dec	-	-	-	-	-	High	High	High	-	-	-	-	-	-	-	-	-
Aud-Dec	Low	Low	-	-	High	-	-	-	-	-	-	-	-	-	-	-	-

An example is, emerging applications in the augmented reality space [31], where the devices and applications attempt to enhance the surroundings with additional details to help the user with their specific needs. These applications place heavy requirements on core, graphics, network, memory and other IPs on the platform. Further, slowdown in any portion of the platform or any specific IP will affect the overall experience. Hence, it is critical to analyze the entire SoC platform as a whole when optimizing features, which in turn, makes a case for a complete simulation infrastructure that can enable this.

3. A COMPREHENSIVE EVALUATION PLATFORM

Given the diversity in the types of phones and tablets that get built and used, the goal of the proposed framework is to provide flexibility for evaluating these designs with multiple cores and IPs. The framework is agnostic to the details in the IP model, which can be a simple analytical model or a complex cycle-accurate model of the IP's micro-architecture.

3.1 GemDroid - Simulation Infrastructure

Currently very limited infrastructure exists for enabling platform level studies across multiple IPs running realistic and/or relevant workloads. GEM5 framework is the closest, that the authors are aware of, which can simulate an ARM or x86 cycle-accurate core with Android/Linux Kernel running on top of it [5, 19]. Currently, GEM5 can simulate only a limited set of IPs (core and only display panel). This limited support and drastic simulation slow down severely restricts the number of apps that can be run. Further, it is not possible to do IP-centric evaluations. While incorporating GEM5 in its infrastructure, GemDroid expands on the number of IPs modeled to get close to a complete device (see Figure 4). Further, GemDroid makes it flexible in terms of the modeling technique adopted for the IPs for which cycle-accurate models are hard to build mainly due to unavailable public information (see Figure 4).

GemDroid relies heavily on the Google Android's open-source emulator and it has been enhanced for our needs. Android emulator meets two of our essential goals – booting an operating system and running commonly used applications on top. The emulator runs the latest version of Android compiled for ARMv7 ISA with Neon instructions. The core of the emulator, based on the Qemu tool [17], translates each ARM instruction to a set of native machine instructions and executes them on the host. During this translation, instruction level traces are captured. The framework also emulates other IPs such as the imaging (handles the images captured using the camera), display, network, audio and there are hooks available to emulate sensors such as accelerometer, gyrometer, etc.

However, the emulator misses out on the crucial part needed for performance studies: it does not incorporate the simulation time for any of the IPs. The emulator's goal is to enable application development for Android and hence has a different set of goals than ours. GemDroid integrates existing performance models - GEM5 for the core and memory subsystem, Attila for graphics [9] - and analytical models for the other IPs missing a model (like Video, Network and others). We *do not* claim to have developed performance/power models for all IPs, but our proposed framework is extensible, and will allow for other users in the community to incorporate their models as needed. We are looking to make our framework open for others in the community to contribute and do their studies on this platform.

3.2 Trace-based simulation

Unlike server workloads or widely available benchmarks like SPEC, PARSEC [4], etc., mobile applications are more user interactive. Providing user inputs and studying the system is not an easy task due to the associated non-determinism; for that, one possible method is to capture user inputs that are sent to the OS, replay it exactly while evaluating the system [33, 40]. In our infrastructure, we use Android emulator as the front end, where one can install almost all applications available on Google Play and provide inputs like the way it is done on SoCs. The emulator has been instrumented to capture the ARM instructions and IP calls along with their interactions with the memory in a trace file. Using such a trace, provides determinism in evaluating such applications with user inputs.

3.3 Model characteristics

While a cycle-accurate full system simulation meets the accuracy goals for micro-architectural and system level studies, they cannot simulate considerable durations of our target mobile workloads due to complexity associated with handling multiple IPs. On the other hand, while development boards can meet the speed requirements they fail to provide control for exploring the system by changing the underlying parameters. As described in [5], GEM5 can cycle-accurately simulate 200K instructions per second, potentially leading to 800X slowdown for a processor-core based system. If the simulator is augmented with accurate models of GPU, audio/video encoder/decoder, and imaging IPs, the simulation times would become unreasonable. Hence GemDroid looks to keep the infrastructure flexible for integrating models with differing levels of complexity and allowing them to interact. Depending on the IP of interest, users can inte-

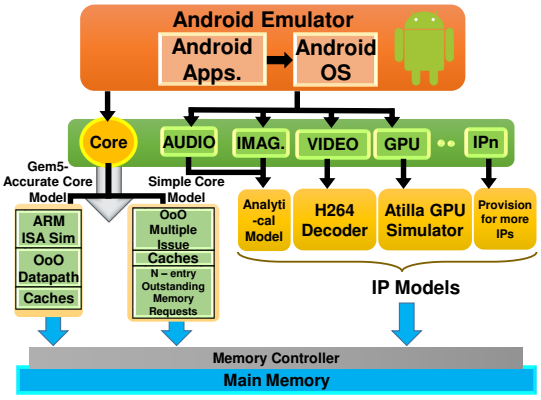


Figure 4: Detailed Infrastructure Diagram.

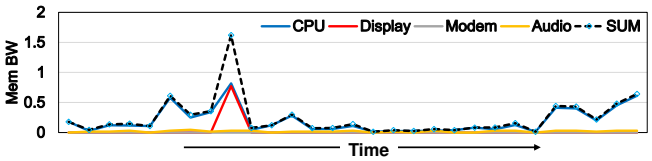


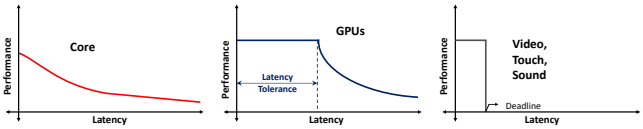
Figure 5: Execution where display IP reads a frame from memory at the same time core is writing a new frame.

grate accurate models for specific IPs and integrate *less accurate models* for the rest of the system. The less accuracy is with respective to not modeling the micro-architecture details of IPs, but having enough information to capture timing associated with different activities. In our work, for studying system-level memory characteristics across IPs, we developed an alternate simplified core model in our infrastructure which assumes a 1-IPC model. This does not affect the timing accuracy of the execution significantly as many frequently used ARM ISA instructions are single-cycle instructions [2]. Such a system had only a 180X slowdown compared to real hardware. Users though have the flexibility to switch between the highly accurate GEM5 core model or our simplified core model based on their requirements. Note that when the system is extended with cycle-accurate core model, significant slowdowns were observed resulting in only a short duration of execution time being simulated. Such a simulation is unsuitable for IP based system studies as not many IP calls are seen in such short duration.

For the graphics IP, we used the Attila graphics simulator [9], which handles the openGL calls issued by applications. These OpenGL calls that are used for rendering different images to the screen, are captured in the trace. For video IP, we used the open-source H264 RTL model [42] to capture the timing associated with decoding. For audio, and imaging (applications that use camera for capturing pictures or recording video), we use the emulator to capture the calls to audio and imaging IP. These calls provide us with the sizes of the frames, and the arrival rate of the frame requests (based on the number of instructions between frames). Reasonable amount of application time was simulated by capturing the system level metrics with at least 2 billion core instructions.

3.4 Capabilities of GemDroid

Our infrastructure can be used to conduct multiple types of studies starting from the core, memory and individual



(a) Cores *(b)* GPUs *(c)* IPs(display, sound)

Figure 6: Effects of varying memory latency on performance for (a)cores, (b) GPUs, and (c) other IPs.

IPs to system-level performance/power analyses. The first insight we can get from using GemDroid is the usage pattern of IPs for different applications. Currently, we have incorporated 9 IP models in GemDroid and have analyzed the behavior of a wide spectrum of applications such as games, video recording and video playback. Table 1 already shows the application-specific usage of IPs.

In addition to understand IP usage, the platform can help in studying contention for shared resources. Consider the example shown in Figure 5. The figure illustrates a scenario when YouTube video playback traces were simulated on the system and different IP's memory accesses were analyzed. We note that there are instants when two IPs perform memory access at the same time. Further, multi-core studies are also possible once we collect application-level traces for multiple applications. We leave this as a future work. Instead, in this paper, we analyze the memory system of current SoCs to quantify its impact on application performance, and explain how a heterogeneous memory controller design can help mitigate some of the problems the memory system brings.

4. EVALUATION METRICS

Performance evaluation of SoCs is more challenging than the conventional CPU-centric platforms because of the emergence of new IPs and use-cases with their different characteristics and requirements. In particular, having one global performance metric does not provide a right picture on how the platform is behaving or to identify the bottlenecks in the system. In this section, we classify target applications into following classes, and define appropriate metrics for each application class.

CPU/GPU-bound Applications – IPC/CPI: CPUs are the traditional processing cores, where each load or store instruction that goes to memory critically stalls the core, because out-of-order processing capability is very limited in these cores due to power constraints. Figure 6a depicts how the performance of such a core degrades as the load/store delays increase. Typically these cores do not operate under any deadlines. For our core-bound benchmarks like Linpack and Antutu-CPU, since job execution time is a standard performance metric, we can use IPC (Instructions Per Cycle) or CPI (Cycles per Instruction) for gauging their performance.

GPU workloads (that are not display refresh limited) also involve throughput-oriented computation; but, they slightly differ from CPU cores. Specifically, they have some inherent latency tolerance because of the high thread level parallelism (TLP) [24]. GPUs can hide memory access latency up to a point (marked with dotted lines in Figure 6b), beyond which their performance starts to drop, following the pattern of the CPU cores. Typically, GPUs do not have any deadline

when they are used for compute purposes. However, while they are used for graphics/rendering purposes, they act like the other IPs with well-defined QoS targets. Thus, both IPC/CPI can serve the purpose for evaluating GPUs.

Onscreen Applications – Frame Drops: For onscreen applications which have a visual aspect to them, such as video/audio playback or graphics oriented applications, job turnaround time may not be an effective metric. For example, when a game is being played, the smoothness of the transitions and game playback are commonly defined performance metrics. These applications are limited by the rate at which the display panel refreshes the screen (at say 60 FPS). Even if the components compute faster, it does not change the user perception but will have an impact on overall power. Similarly, when playing a video or audio file, execution time is dependent only on the clip's duration, and execution should be controlled such that execution time is equal to the clip's duration. For example, a 1-minute video has to be played for exactly 1-minute. Any deviation from this may result in distortion in quality or lags become visible. In these situations, one of the ways of quantifying performance is to check whether the system is capable of playing X frames per second (FPS) consistently, if the video file had been encoded in X-FPS (24 FPS is the most commonly encoded video frame rate, though 30 and 60 FPS videos and games are also becoming increasingly popular).

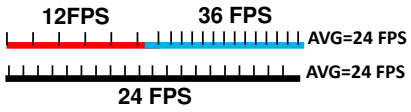


Figure 7: Two different scenarios with different FPS rates over time.

Consider the scenario shown in Figure 7, where in case (1) the system manages to play a 1-minute video file at two different FPS rates, 12 FPS for the first half (30 seconds), and 36 FPS for the second half. The average FPS for the video file being played would be 24 frames per second. In case (2) on the other hand, the system manages to play the file at a constant 24 FPS. Although both the videos are played at 24 FPS on the average, certainly the user would not be satisfied with a choppy frame rate experience delivered by the first scenario. From this, it is clear that FPS may not be the right metric. That is, the average FPS value does not capture the overall behavior of the system.

Instead of FPS, if *frame drops per second* (FDPS) is used as a metric, one can distinguish between the two cases depicted in Figure 7. More specifically, if the required frame rate is 24 FPS, case (i) would leave us 6 FDPS, whereas case (ii) would have 0 FDPS. This clearly shows that case (ii) is preferable over case (i).

Real-time Applications – Response Time: Several IPs in the SoC are responsible for meeting the immediate deadlines like touch, accelerometer (and other sensors) and interrupt-handling to user pressing different buttons. These are different from the CPU and GPU cores as they are response time oriented in order to meet certain deadlines. Their performance is captured by monitoring if they meet their strict deadlines or not. For an input request, their performance is determined through a yes or no question: *if the IP met the deadline set for the request.* Figure 6c shows this. The width of the bar shown corresponds to the available latency before which the deadline expires. However, GemDroid does not model these IPs at this point. This metric is provided only for the sake of completeness.

Core-Utilization: In addition to the above metrics, we also measure core-utilization (percentage of busy-cycles), which measures the total amount of time for which the core is working. By lowering core-utilization without affecting performance, one can make the system more energy efficient.

5. IMPACT OF MEMORY SUBSYSTEM IN MOBILE PLATFORMS

As depicted in Figure 4, memory is a shared resource between the cores and IPs and any performance issues in its path is more likely to manifest as a system bottleneck, as has been the case for traditional computing systems. Design of high performance memory systems including memory controllers (MCs) has been an active area of research for the uniprocessor and CMP (chip multi-processor) domain [26, 43]. However, memory access patterns of our target mobile applications are different from scientific workloads like SPEC, Parsec [4] or server workloads, as we will illustrate shortly. To our knowledge, no other work has investigated the memory system design for such multiple IP SoC systems. This section describes our analysis of these memory systems.

5.1 Memory Access Patterns

To show how SoC applications differ in their memory behavior, we plot in Figure 8 the memory access patterns of a SPEC applications (*h264.ref*) and a video recording application running on our SoC platform plotted over time. The Y-axis shows a sample address range, and only a portion of the full execution has been shown for clarity. We see similar trends for many other SPEC benchmarks. One can make two important observations from these plots:

First, while SPEC applications have quite irregular memory address patterns over time, the video recording and the browser applications use a specific set of data over and over again, indicating good data locality. We observed similar patterns for many other display-bound applications. We investigated the address regions and found that, this reuse pattern was caused by two reasons: (i) for the display bound applications, frames were written into the same physical address region repeatedly almost every 1/60th of a second; and (ii) the source devices (like cores/ GPUs/ video decoder) write into the same address region, from where the sink devices (display, audio output) read data.

Second, at some time instants, we see two regions of address being used concurrently. This happens when the core is accessing two different regions, or when a core and an IP

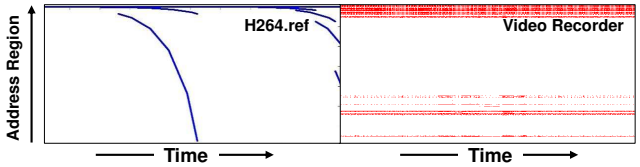
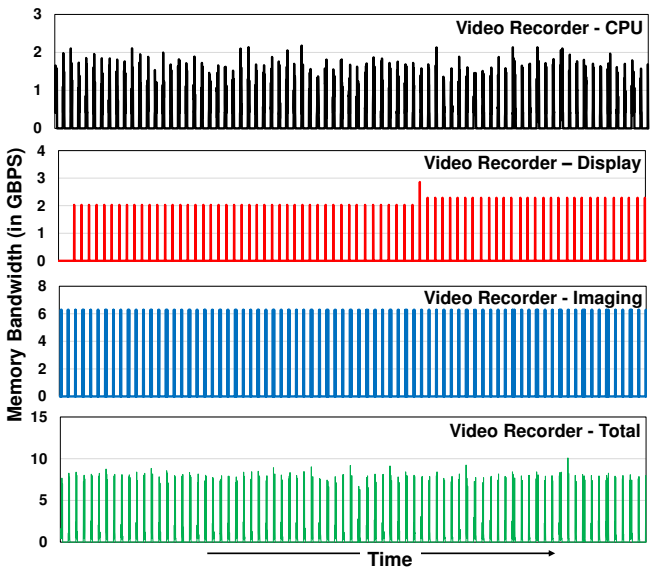
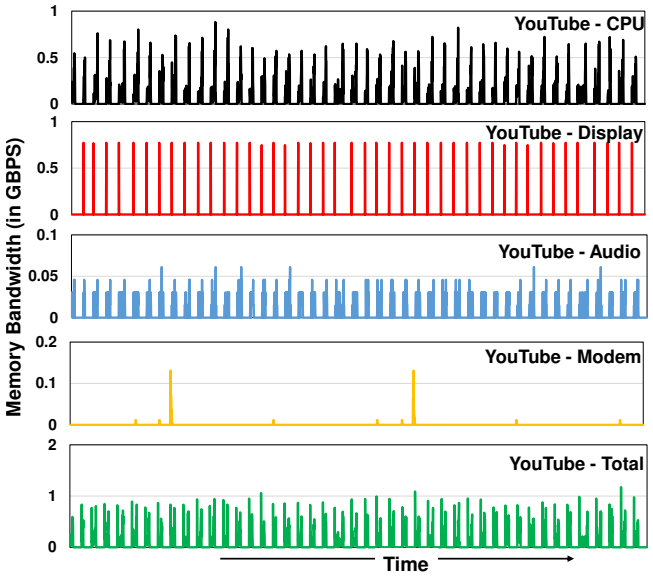


Figure 8: Comparing the memory access patterns of SPEC application (h264.ref) with video recorder mobile application from our experimental suite.



(a) Bandwidth demand of VideoRecorder.



(b) Bandwidth demand of video playback on YouTube.

Figure 9: Variations in the bandwidth demand of applications over time.

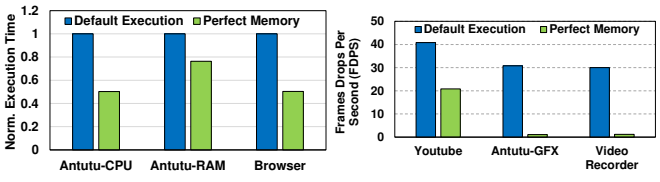
are accessing the memory concurrently. Such a scenario increases the bandwidth demand placed by the application on the memory system. We observed that the number of concurrent accesses to memory can increase depending on how many IPs are used by an application. Thus, the peak memory pressure could change significantly across applications, making the memory system design complex. This is because, unlike SPEC, PARSEC and many other applications, which typically do not have hard deadlines, most of the mobile applications and hence IPs have real-time constraints. Thus, provisioning only for the average memory bandwidth may not be adequate to meet the real-time constraints.

Analyzing Memory Access Characteristics of IPs: We analyzed the memory access characteristics of IPs requests belonging to our applications, and noted some interesting characteristics. While modem, audio, display and camera IPs primarily sent and received fixed size packets for an application, across applications their sizes varied. For example, an application involving HD video capture or photo capturing applications had different sizes of frames being sent compared to a 720p video capturing. One interesting feature that we observed was, when the Android Emulator writes data into the frame buffer region for the display to read, it does not write "framesize-amount" of data for every frame. Instead, it only overwrites parts of the frame that need to be changed. For applications like browsers, if only parts of the screen are changed due to animation, then data only for that part is overwritten in memory. For a YouTube video running on a browser, we plotted the distribution of frame sizes in Figure11a. Typically, the distribution shows that 3 sizes of frames were transmitted. Whenever full sized frames were transferred, we observed 0.73 MB of data per frame being written by the CPU. If most of the frame, except the borders/system panel changed, then 0.6 MB data per frame was transmitted. Otherwise, negligible amount was transferred (this includes writing and reading from memory-mapped registers of the display IPs, or some small part like clock display being changed on the screen). In Figure 11b, we plotted the inter-arrival time of the display frames software-rendered by the CPU (the frames are sorted in ascending order based on their inter-arrival times). Ideally, CPU is expected to produce the frame exactly every 1/30th of a second (in this experiment, the required FPS limit was set as 30 FPS). But, we can observe that the delays vary. This variation is attributed to many reasons like the complexity of the frame needed to be rendered, interference at shared resources at that instant, etc. One crucial point to be noted is that, while most of the frames arrive before the deadline time ($1/30 = 3.33 \times 10^7 ns$), some arrive very late (towards the right most side of the graph). Frames that arrive later than this point forces the following frame to be dropped. When we curve-fit this plot, we observe the distribution follows an exponential distribution, with an R^2 value of 0.943 for this application.

Similar characterizations are possible for other applications as well. These results are interesting because one can use these inter-arrival time distributions along with frame rates and packet size distributions to simulate a YouTube workload on the GemDroid simulator.

5.2 Memory Demand of Applications

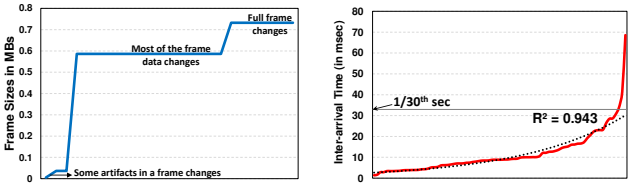
Figure 9 shows the memory bandwidth demand of two applications over time. Only a part of the full timeline is shown for clarity. Most of the other applications are similar to at least one of these two applications as far as their bandwidth demand is concerned. In this figure, we separately plot the memory demand of each IP during the execution of an application, and then show the total demand. For example, in Figure 9b, we see that the YouTube videoplayback application uses CPU, display, audio device and the GSM network for data, and their corresponding bandwidth demand is shown in the top four sub-graphs. The last graph of Figure 9b shows the total bandwidth demand placed on the memory. We observe that, at some instants, the bandwidth demand is much higher than the average. These peaks are primarily observed when the data request bursts from different IPs overlap with each other. In applications like video recording, we notice that the peak demand requested



(a) Execution time difference of CPU-bound applications in a system with a *perfect memory*.

(b) Difference in FPS of Display-bound applications in a system with a *perfect memory*.

Figure 10: Impact of a *perfect memory* on applications.



(a) Shows the distribution of the frame sizes when a sample YouTube video is played.

(b) Shows distribution of inter-arrival times between two frames when a sample YouTube video is played.

Figure 11: Characteristics of frames in YouTube application.

is much higher than the peak bandwidth (3.2 GBPS) provided by the memory system (LPDDR2-400). These are the instants when a frame can possibly get dropped. Note that, in the video recording application, all frames demand more than 3.2 GBPS, but not all get dropped. This is because, the required frame rate is 60 FPS, each frame gets 1/60th of a second to be processed. If the frame currently being processed is not served within that limit, the next frame that arrives is dropped.

Investigating core-bound benchmarks like Linpack and AntutuCPU, we observed that the impact on memory is substantially lesser than display bound applications[2]. We also noted that such applications have higher instruction throughput than the rest, due to fewer memory stall cycles.

5.3 Impact of Memory on Applications

To understand how memory contributes towards application's stalls, we analyzed how much improvement a *perfect* memory can provide. We define a *perfect* memory to have 0 cycles turn-around time for requests. We found that, for core-bound applications (Figure 10a), the execution time decreases substantially for many workloads, and moderately for some. For display bound applications on the other hand, we noted that the system improved its frames-per-second and lesser number of frames were dropped with the perfect memory (see Figure 10b). This is primarily attributed to two reasons – (1) the core that produces the data for the frame, produces it earlier, thus, avoiding frame drops; and (2) the IPs (producers/consumers) are able to write/read the data much faster, not exceeding the deadline imposed. Note that the display bound applications are limited to 60 FPS. Once the required 60 FPS is reached, applications are throttled to remain at that rate. If the FPS drops, throttling is stopped.

Consider the scenario, where the base case has 60 FPS. Then, no performance improvements can be observed. Here, we use the number of busy cycles in core and IP devices to quantify the impact of perfect memory. The lower the number of busy cycles, the better would be the power savings. Similarly, the improvements seen with applications reaching 60 FPS are not the true maximum. For these, benefits should be seen through the reduced number of busy cycles as well.

[2]We noted that for some CPU bound benchmarks, there was noticeable network activity during every run of the benchmark. They were found to be the addresses that were rendered during the execution, or when the application's communicated the results back to a server towards the end of the execution.

5.4 Summary of Observations:

Specifically, we observe the following primary differences between the traffic from cores and the IPs:

1 - IPs have more or less regular request inter-arrival times, with their requests coming in bursts. CPUs have irregular arrival rates, and are typically not bursty.

2 - IPs requests have substantial memory latency tolerance compared to CPU cores. Thus, they can be stalled for sometime without any effect on performance or user-experience.

3 - Arrival rates of memory requests from CPU and IP are very different. While CPUs requests are fewer in number, IP requests come in bursts of tens or hundreds or even thousands. Their inter-arrival time distributions can be used to simulate any specific IP.

4 - IPs demand bandwidth. The higher the bandwidth, the faster they read/write data; achieving two purposes: (1) they move to low power states sooner, allowing for some power savings, (2) it lets the next component (core or another IP) that needs to feed on what was provided by this IP to start its work sooner, therefore reducing frame drops, and improving response time.

Based on the core and IP request properties described above, we present in the next section a Heterogeneous Memory Controller (HMC) design that has been tailored for SoC systems.

6. A CASE FOR A HETEROGENEOUS MEMORY CONTROLLER

In this section, we provide a brief overview of the baseline memory design, explain the proposed heterogeneous memory controller (HMC) design, and finally evaluate our proposal.

6.1 Locality-Parallelism Tradeoff in Memory Design

Baseline Memory System: Figure 12 (A) shows the memory design of our baseline system. It consists of 2 memory controllers (MCs) controlling two distinct regions of memory. As shown, the cores and IPs *share* this memory subsystem. Traditionally, these MCs are the gateway to access data in memory, which is logically organized as DRAM banks. Each bank has cells (memory elements) laid out in arrays of rows and columns. The data can be striped across banks at various granularity, for example, at *page-level* or at *cache-line-level*. In *page-level*, the distribution of data across banks is at a granularity of a OS page, which is a chunk of multiple consecutive cache lines. For example, if page size is 4KB (used in our paper), the first 4KB of consecutive data is mapped to the first bank, and the next 4KB

to the next bank, and so on. In our baseline system, we use this page-level striping for both the memory controllers, as shown in Figure 12 (A). In *cache-line-level* striping, the distribution of data across banks is at a much finer granularity – at cache-line granularity. In this case, every other cache line is mapped to a different bank.

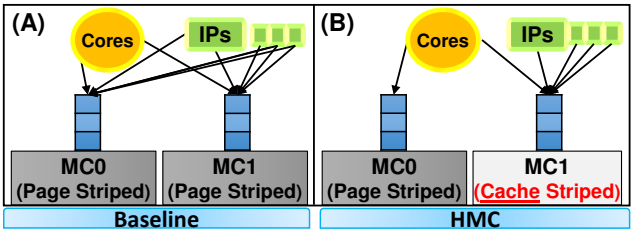


Figure 12: Schematic of (A) Baseline memory design and (B) Proposed HMC memory design.

Locality vs. Parallelism: When accessing a cache line from memory, the row that contains the cache line is brought to a buffer called *row buffer*, which is associated with every DRAM bank. Once the contents are placed in the *row buffer*, subsequent memory requests to the same row are served from the row buffer (row-buffer hits), instead of fetching them again from the memory array. This reduces access latency, improves performance, and saves the energy-expensive job of reading the row from memory array. Instead, if a different row from the one in the row buffer is requested, the current row is closed, and the new row gets placed in the buffer. This incurs high memory latency and is a high energy consuming task. Therefore, it is optimal to receive and serve requests from the row buffer. For this very reason, the most popular form of data distribution in CMPs is page-level striping, where up to 4KB of consecutive data can be mapped to a bank, and if requests are scheduled timely, all the data can be fetched from the row-buffer, thereby improving DRAM locality and energy efficiency. On the other hand, page-level striping restricts parallelism, as not many DRAM banks can be utilized in parallel. This is because if the requests possessing good locality are scheduled roughly at the same time, only a limited set of the banks will be accessed and the other DRAM banks will be idle. This limitation can be addressed by cache-line striping, where the same 4KB of data is striped across banks, and hence the same requests will access multiple banks. Such striping, although increases parallelism, it reduces locality. It is apparent that both techniques of data-distribution have pros and cons.

Auxiliary Metrics: In this context, we define two auxiliary metrics, which will be used to understand this trade-off. First, Bank Level Parallelism (BLP), which is defined as the average number of memory banks that are busy when there is at least one request being served at this memory controller [27]. Improving BLP enables better utilization of DRAM bandwidth. Second, Row-Buffer Locality (RBL), which is defined as the average row-buffer hit rate across all memory banks [27]. Improving RBL decreases average latency for memory requests, and increases the memory service rate.

6.2 Overview of the Proposed Design

As discussed in Section 5, the IPs have significantly higher memory bandwidth requirements compared to cores. This is shown in Figure 9a, where imaging IP demands more bandwidth compared to CPUs (note that the y-axis scales are different). This manifests into two primary problems: (1) the IP requests arrive in bursts thereby causing large queuing delays for CPU requests reducing the core performance, and (2) the IP memory requests interfere with core requests, thereby impacting the row-buffer locality of all the requests. Due to these two issues, the DRAM bandwidth utilization is severely affected leading to degradation of system performance. To address this, we propose having separate memory regions for mobile systems.

6.2.1 Memory Region Separation

In this design, we divide the address space into two regions: first region – associated with a dedicated memory controller (MC1) for CPU data which is accessed only by the CPUs, and the second region – associated with MC2 for IP data, which can be accessed by both cores and IPs. Note that, we cannot have completely dedicated memory controllers for IP and CPU requests, because the data produced by IPs need to be used by cores (or vice versa).

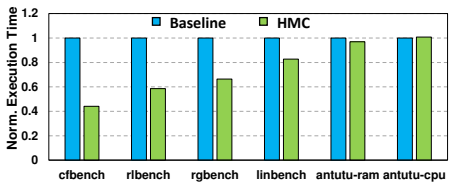
The goal of this design is to offer dedicated memory controller for core requests, as these requests are more latency critical. On the other hand, requests for the IP region are bandwidth intensive, as they arrive in bursts and access large chunks of data. Now, with separate memory regions, the bursts of requests coming to IP regions access consecutive cache lines. Due to this, these requests have very good row buffer locality. But, the downside of such an access pattern is that the bank-level-parallelism is very low.

6.2.2 Heterogeneity in Data Striping

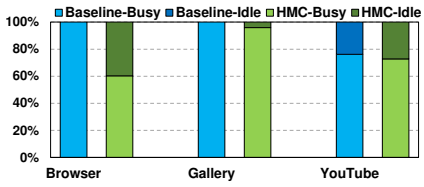
To address the above problem, we enhance the design of memory with appropriate data striping. We adopt two different data striping techniques for MCs: MC0 uses page-level striping, and MC1 uses cache-line striping as shown in Figure 12(B). The motive of having two different striping techniques is to increase the BLP for IP memory region, while retaining the row-locality at CPU memory region. Note that, in general, cache-line striping reduces row-buffer locality. However, in this scenario (especially for IPs), typically the row-buffer locality is not affected, because these regions receive requests to large chunks of contiguous data. Consider the example where the system has cache-line striped n memory banks, and the display IP is accessing a large frame region. In such a system, consecutive cache lines are mapped to different banks in a cyclic manner, such that every n^{th} cache line is mapped to the same bank. Because the IPs typically access consecutive cache lines, the requests that are mapped to the same bank are likely to hit in the row buffer, leading to high row-buffer locality. Also, as the IP requests are sent to different banks, they take advantage of BLP. Note that in the proposed design, there are no extra overheads in terms of data copies or data duplication. All IP-associated data are written to DRAM through a separate memory controller(MC1) by cores or other IPs. While MC0 can be accessed by the cores, MC1 can be accessed by the cores and the IPs.

6.3 Evaluation of HMC Design

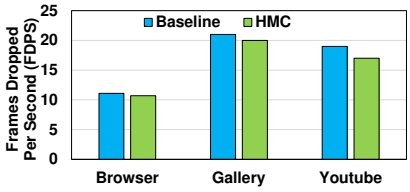
We compare our HMC design to an iso-resource baseline system with 2 memory controllers which are page-striped. In the baseline system, the memory controllers are not aware



(a) Reduction in execution time for CPU-bound applications.



(b) Reduction in CPU busy cycles with HMC.



(c) Improvement in average FDPS for Onscreen applications

Figure 13: Performance improvements of HMC with respect to baseline system. *Lower is better.*

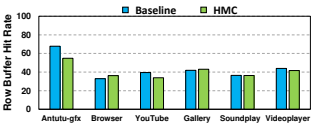
of the characteristics of IPs' and cores' requests. We do not consider cache-line striped memory controllers as they increase the memory latency for all core's memory operations, thus reducing system performance and energy efficiency. In the proposed design, *Heterogeneous Memory Controller* (HMC), we isolate the requests targeted to IP and CPU memory regions.

Figure 13a shows the performance comparison of HMC with the baseline system for representative CPU bound and Onscreen applications. We report their respective evaluation metrics (execution times for CPU bound applications, and CPU-busy cycles and FDPS for Onscreen applications). From Figure 13a, we observe that, on average, the execution time of CPU-bound applications is reduced by 25% (up to 56% in cfbench). This improvement is primarily attributed to two reasons. First is the reduced interference from IP accesses on the CPU requests at MC0, because of memory region separation (discussed in Section 6.2.1). Second is the reduction in latency at MC1 because of increased RBL and BLP as discussed in Section 6.2.2.

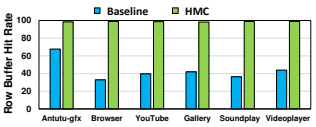
The variance in reduction in execution times are attributed to the impact of IP accesses on the CPU accesses. If an application has relatively more number of IP accesses, it is likely to perform better with our HMC design. Note that for core bound applications, which do not have *any IP calls* (antutu-ram and antutu-cpu), will mostly not take advantage of HMC's optimizations. In fact, in some cases, they might lose performance due to reduced memory channel parallelism for CPU requests. In our studies, we find that the execution time of Antutu-CPU application increases by less than 1%.

The graph in Figure 13b shows the CPU activity under different memory system designs. In on-screen applications, CPU has to process data before an IP can consume it or vice versa. By employing our HMC design, the CPU processes the data quicker leaving it idle for more cycles. This can be seen in the second set of bars in the graph. This reduction in busy cycles directly translates to power savings.[3] Figure 13c shows the metric Frames Dropped Per Second (FDPS) under different memory system designs. HMC design makes the memory subsystem faster for both CPU and IP memory requests leading to fewer frame drops per second.

To understand the impact of our HMC design, we analyze some auxiliary metrics below. First, we look at how the locality at the memory controllers is affected due to HMC in Figure 14. Sub-graph (a) shows that the locality (row-buffer hit rates) at MC0 which is receiving only CPU requests in the HMC case did not change much, while (b) shows the row-buffer hit rates increase to almost 100%. This is mainly because, when the address regions are partitioned, only requests to IP memory space arrive at MC1. These requests typically access consecutive cache lines, contributing to high number of row-buffer hits.



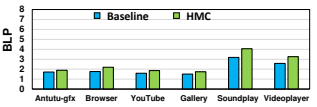
(a) Effect on locality in CPU-region addresses.



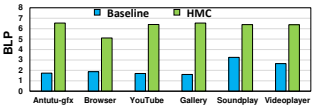
(b) Effect on locality in IP-region addresses.

Figure 14: Impact of HMC on locality. Baseline has both MCs serving both CPU and IP requests without distinguishing between them. *Higher is better.*

In HMC, though there is significant locality, because consecutive accesses go to the different banks due to cache-line striping, the bank level parallelism is also observed to be substantially higher than the base case. Particularly, this can be seen in Figure 15 (b), where the BLP for base-case averages around 1.25 banks only, whereas for HMC averages around 5.8 banks across all applications. In this IP memory region, as the requests that arrive typically go to consecutive banks in cyclic fashion, BLP tends to remain so high. Thus, Figures 14 and 15 together clearly show that our design did not lose locality when striping cache lines across banks. It is also clear from these graphs that with intelligent data mapping in memory, like in HMC, we can get the benefits of both, locality and parallelism.



(a) Effect on bank-level parallelism in CPU-region addresses.



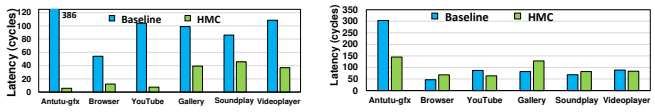
(b) Effect on bank-level parallelism in IP-region addresses.

Figure 15: Impact of HMC on Bandwidth. Baseline has both MCs with default page-striped addresses. *Higher is better.*

We observe that locality and parallelism in this system have significant impact on the latency of memory requests. Figure 16 shows the average latency of requests arriving at the memory controllers. We observe that, with HMC, in MC0, average latency improves because the core's requests are isolated from IP's requests. Thus, the latency critical core requests are served much faster, leading to performance improvements. In IP-region memory controller, the latencies were not affected significantly even though the requests are coming in bursts.

Finally, in Figure 17, we plot the cumulative distribution function of latencies of memory requests that arrived at the IP-region memory controller for YouTube and Browser application. The x-axis in this plot is the memory latency in

[3]In this work, we focus on performance and do not have a comprehensive power model for the system components.

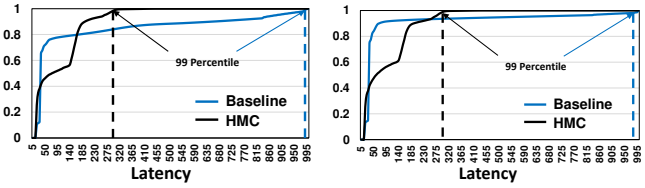


(a) Effect on latency in CPU-region addresses.

(b) Effect on latency in IP-region addresses.

Figure 16: Impact of HMC on Latency. Baseline has both MCs serving both CPU and IP requests without distinguishing between them. *Lower is better.*

cycles. We can observe that, with HMC, 99 % of the requests have latencies less than 300 cycles, while in baseline system, only 82% (youtube) and 95% (browser) of the requests have this latency. This clearly shows the benefits of HMC in reducing the memory latencies.



(a) CDF of latencies of memory accesses sent to IP address region in YouTube application.

(b) CDF of latencies of memory accesses sent to IP address region when using Browser application.

Figure 17: Impact of HMC showing how increase in bank-level parallelism reduces latency of requests in IP-address region.

7. RELATED WORK

Simulation Infrastructure and Application Characterization: A closely related work to ours by Gutierrez et al. [19] analyse the micro-architectural characteristics of smartphone applications without focusing on IP behavior. Another recent work by Sunwoo et al. [40] proposes an infrastructure to simulate smartphone cores, by integrating the architectural simulator GEM5 and OS, to study emerging smartphone workloads. Again this study is only core centric and lacks IP analyses. In this paper, we develop an infrastructure that can simulate multiple IPs as well as cores with OS, and can be easily extended to include more IP models. Also, we characterize the memory accesses generated by CPUs and IPs, and design a memory scheduling mechanism based on this characterization. Several works have investigated the power consumption of different applications [35], and different IPs [6] in smartphones, and proposed simulation infrastructure to simulate mobile networks [7,18]. These works neither characterize the memory sub-system, nor look at commonly used mobile applications.

IP Design and Optimizations: Ozer et al. [32] describe the steps involved in the design and verification of ARM IPs. Saleh et. al [37] discuss reusability, integrity, and scalability of IPs used in SoCs. Along with IP design and analysis, several works have proposed IP-specific optimizations [13, 20, 25, 34, 38, 39]. Our work does not consider specific IP design or optimizations, instead focuses on characterizing the interaction of different IPs and cores, and intelligently schedules their memory requests to improve overall system performance. A large body of work on power in smartphones include propose a system-call-based power model [36], power consumption of network devices and protocols in smartphones [3], a network-based power reduction technique for smartphones [12], and the power consumption of various IPs and applications in smartphones [22, 41, 44].

Memory Controller Design and QoS: Several works have investigated memory scheduling techniques in the context of smartphones. Lee and Change [28] describe the essential issues in memory system design for SoCs. Lee et al. [29] propose a memory scheduling mechanism that provides latency and bandwidth guarantees for memory accesses. Akesson et al. [1] propose a memory scheduling technique that provides a guaranteed minimum bandwidth and maximum latency bound to IPs. Lin et al. [30] employ a hierarchical memory scheduler that improves system throughput. Jeong et al. [23] provide QoS guarantees to frames by balancing memory requests at the memory controller. In the context of CMPs and uni-processor systems, several works have proposed low-power memory designs [10, 11] that can be applied in smartphones for better energy efficiency. In this paper, we propose a MC design specifically tailored for both CPUs and IPs.

8. CONCLUSIONS

In this paper, we present a comprehensive simulation framework for exploring the SoC design space, targeted specifically for mobile systems. The proposed GemDroid platform primarily consists of the Android emulator that enables collecting core traces and IP calls for mobile applications and the GEM5 core simulator that enables in-depth analysis of the core and memory subsystems. In addition, we have included several IP models for characterizing the execution profile of IPs, invoked by different applications.

To demonstrate the capabilities of the infrastructure, in this paper we focused on the memory system analysis of SoCs since it is a known performance bottleneck for both latency critical core executions and bandwidth critical IP executions. Thus, we present a heterogeneous MC design, where one MC is optimized for core requests and the other MC is dedicated to enhance bank-level parallelism of IP requests. The proposed modifications to the MC design results in improving both core and IP performance.

The proposed GemDroid simulator is expected to fill a void in the mobile system design space by facilitating a holistic performance and power analyses of evolving SoC designs. The novelty of the framework is that it is flexible where users can add more IPs, different simulation and analytical models for IPs either to make detailed or faster evaluation, simulate multiple cores, on-chip interconnect design and emerging memory technologies for system-wide performance and power optimization. We are currently looking into some of these issues for making GemDroid a more powerful tool. We propose to make the entire GemDroid framework available in the public domain, together with modular capabilities to allow the broader academic community to undertake numerous studies.

Acknowledgments

We thank the anonymous reviewers, Ashutosh Pattnaik, Adwait Jog, Onur Kayiran, Prashanth Thinakaran and other HPCL members for their feedback on this paper. This research is supported in part by the following NSF grants – #1302557, #0963839, #1205618, #1213052, #1320478, #1317560, #1302225, #1017882, and grants from Intel.

9. REFERENCES

[1] B. Akesson, K. Goossens, and M. Ringhofer. Predator: A predictable sdram memory controller. In *CODES+ISSS*, 2007.

[2] ARM. ARMv7-A Technical Reference Manual. 2011.

[3] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *IMC*, 2009.

[4] C. Bienia. *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[5] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood. The gem5 simulator. *SIGARCH Comput. Archit. News*, 2011.

[6] A. Carroll and G. Heiser. An analysis of power consumption in a smartphone. In *USENIXATC*, 2010.

[7] L. Chen, W. Chen, B. Wang, X. Zhang, H. Chen, and D. Yang. System-level simulation methodology and platform for mobile cellular systems. *Communications Magazine, IEEE*, 2011.

[8] Cisco. Cisco Visual Networking Index: Forecast and Methodology, 2012:2017. 2013.

[9] V. del Barrio, C. Gonzalez, J. Roca, A. Fernandez, and R. Espasa. Attila: a cycle-level execution-driven simulator for modern gpu architectures. In *Performance Analysis of Systems and Software, 2006 IEEE International Symposium on*, 2006.

[10] Q. Deng, D. Meisner, L. Ramos, T. F. Wenisch, and R. Bianchini. Memscale: Active low-power modes for main memory. In *ASPLOS*, 2011.

[11] B. Diniz, D. O. G. Neto, W. M. Jr., and R. Bianchini. Limiting the power consumption of main memory. In *ISCA*, 2007.

[12] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *IMC*, 2010.

[13] S. Fenney. Texture compression using low-frequency signal modulation. In *HWWS*, 2003.

[14] T. I. T. R. for Semiconductors. 2008 update, 2008.

[15] Gartner. Worldwide PC, Tablet and Mobile Phone Combined Shipments to Reach 2.4 Billion Units in 2013.

[16] Google. Android Developers, 2013.

[17] Google. Android SDK - Emulator, 2013.

[18] P. Guo. Simulation and testing of mobile computing platforms using fujaba.

[19] A. Gutierrez, R. Dreslinski, T. Wenisch, T. Mudge, A. Saidi, C. Emmons, and N. Paver. Full-system analysis and characterization of interactive smartphone applications. In *IISWC*, 2011.

[20] K. Han, A. Min, N. Jeganathan, and P. Diefenbaugh. A hybrid display frame buffer architecture for energy efficient display subsystems. In *ISPLED*, 2013.

[21] J. L. Henning. Spec cpu2006 benchmark descriptions. *SIGARCH Comput. Archit. News*, 34(4), Sept. 2006.

[22] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid. Web search using mobile cores: Quantifying and mitigating the price of efficiency. In *ISCA*, 2010.

[23] M. K. Jeong, M. Erez, C. Sudanthi, and N. Paver. A qos-aware memory controller for dynamically balancing gpu and cpu bandwidth use in an mpsoc. In *DAC*, 2012.

[24] A. Jog, O. Kayiran, N. C. Nachiappan, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das. OWL: Cooperative Thread Array Aware Scheduling Techniques for Improving GPGPU Performance. In *ASPLOS*, 2013.

[25] H. b. T. Khan and M. K. Anwar. Quality-aware Frame Skipping for MPEG-2 Video Based on Inter-frame Similarity. Technical report, Malardalen University.

[26] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter. ATLAS: A Scalable and High-performance Scheduling Algorithm for Multiple Memory Controllers. In *HPCA*, 2010.

[27] Y. Kim, M. Papamichael, O. Mutlu, and M. Harchol-Balter. Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior. In *MICRO*, 2010.

[28] K.-B. Lee and T.-S. Chang. *Essential Issues in SoC Design Designing - Complex Systems-on-Chip*, chapter SoC Memory System Design. Springer, 2006.

[29] K.-B. Lee, T.-C. Lin, and C.-W. Jen. An efficient quality-aware memory controller for multimedia platform soc. *Circuits and Systems for Video Technology, IEEE Transactions on*, 2005.

[30] Y.-J. Lin, C.-L. Yang, T.-J. Lin, J.-W. Huang, and N. Chang. Hierarchical memory scheduling for multimedia mpsocs. In *ICCAD*, 2010.

[31] T. Olsson and M. Salo. Online user survey on current mobile augmented reality applications. In *ISMAR*, 2011.

[32] E. Ozer, N. Chong, and K. Flautner. *Processor and System-on-Chip Simulation*, chapter IP Modeling and Verification. Springer, 2010.

[33] D. Pandiyan, S.-Y. Lee, and C.-J. Wu. Performance, energy characterizations and architectural implications of an emerging mobile platform benchmark suite : Mobilebench. In *Workload Characterization (IISWC), 2013 IEEE International Symposium on*, 2013.

[34] K. Patel, E. Macii, and M. Poncino. Frame buffer energy optimization by pixel prediction. In *ICCD*, 2005.

[35] A. Pathak, Y. C. Hu, and M. Zhang. Where is the energy spent inside my app?: Fine grained energy accounting on smartphones with eprof. In *EuroSys*, 2012.

[36] A. Pathak, Y. C. Hu, M. Zhang, P. Bahl, and Y.-M. Wang. Fine-grained power modeling for smartphones using system call tracing. In *EuroSys*, 2011.

[37] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. Pande, C. Grecu, and A. Ivanov. System-on-chip: Reuse and integration. *Proceedings of the IEEE*, 2006.

[38] H. Shim, N. Chang, and M. Pedram. A compressed frame buffer to reduce display power consumption in mobile systems. In *ASP-DAC*, 2004.

[39] H. M. Siqueira, I. S. Silva, M. E. Kreutz, and E. F. Correa. Ddr sdram memory controller for digital tv decoders. In *SBESC*, 2011.

[40] D. Sunwoo, W. Wang, M. Ghosh, C. Sudanthi, G. Blake, C. D. Emmons, and N. Paver. A structured approach to the simulation, analysis and characterization of smartphone applications. In *IISWC*, 2013.

[41] Y. Xiao, R. S. Kalyanaraman, and A. Yla-Jaaski. Energy consumption of mobile youtube: Quantitative measurement and analysis. In *NGMAST*, 2008.

[42] K. Xu. Nova : H.264/avc baseline decoder. OpenCores, Apr 2008. RTL verified.

[43] P. Yedlapalli, J. Kotra, E. Kultursay, M. T. Kandemir, C. R. Das, and A. Sivasubramaniam. Meeting midway: Improving cmp performance with memory-side prefetching. In *PACT*, 2013.

[44] Y. Zhu and V. J. Reddi. High-performance and energy-efficient mobile web browsing on big/little systems. In *HPCA*, 2013.