Workload Characterization of Commercial Mobile Benchmark Suites

Victor Kariofillis, Natalie Enright Jerger
University of Toronto
viktor.karyofyllis@mail.utoronto.ca, enright@ece.utoronto.ca

Abstract—Mobile devices are essential in our daily lives. The hardware and software of these devices differs from their desktop and server counterparts. Evaluating these devices fairly and accurately requires special benchmarks. However, the computer architecture community lacks studies analyzing the performance characteristics of mobile benchmarks. This paper presents an extensive workload characterization of commercial mobile benchmark suites using several hardware performance counters. We analyze the temporal behaviours of benchmarks. Our findings show the diverse load patterns across CPU core clusters. Our clustering analysis reveals a benchmark subset that reduces evaluation time by close to 75%, while preserving the richness of benchmark coverage. This work contributes insights for refining mobile benchmarking methodologies, providing a valuable compass for researchers navigating the landscape of mobile system architecture assessments.

Index Terms—Mobile benchmark suites, workload characterization, performance evaluation

I. INTRODUCTION

Mobile devices, ranging from smartphones and tablets to smartwatches, are projected to exceed 18 billion units by 2025 [1]. In comparison, the total number of desktop, laptop and server computers worldwide is estimated at over 2 billion [2]. Despite this, a mere 1% of top computer architecture conference papers in 2018 delved into mobile computing, highlighting a research gap [3].

Mobile System-on-Chips (SoCs) are distinct from their desktop and server counterparts. They feature tight integration, significant heterogeneity [4] and rapid evolution. Accelerators in mobile SoCs have more than quadrupled in the last decade, while multiple new CPU designs are released each year [5]. Mobile operating systems (OS) evolve at a comparable pace to take advantage of new hardware capabilities. One challenge faced by architects studying mobile SoCs is the absence of benchmarks specifically designed for evaluating mobile hardware.

Benchmarks commonly used in the architectural community, like SPEC CPU [6] and PARSEC [7], often fall short in representing real-world mobile applications [8]. They lack the interactivity, heterogeneity, and reliance on shared libraries typical of mobile workloads [9], [10]. Academic benchmark suites targeting the mobile space are often narrowly focused on specific aspects or domains (e.g., web browsing [10], [11], augmented reality [12], race events [13], deep learning [14], etc.), thus limiting their utility. Furthermore, code changes introduced by new OS and API versions can render them incompatible [15].

Commercial benchmarks are commonly employed in industry [16]. However, challenges arise when incorporating them into academic settings. In contrast to desktop benchmark suites, these benchmarks lack prior characterization, impeding computer architects’ understanding of their impact on system performance. Additionally, these benchmarks are not tailored for optimal use in system simulations, a common practice in hardware studies. Their lengthy execution times pose challenges for the efficient evaluation of new hardware designs. Closing this gap in mobile benchmark characterization is essential for fair evaluations of mobile platforms.

In this study, we conduct a comprehensive analysis of the execution and performance characteristics of commercial mobile benchmarks. Our objective is to provide researchers with in-depth insights into the behavioural patterns of these benchmarks and their effect on mobile hardware. Additionally, we offer perspectives on the overall similarities and differences among benchmark suites. This nuanced understanding empowers researchers to judiciously select benchmarks aligned with their specific requirements, thereby streamlining the evaluation process.

In summary, we make the following contributions:
- Analysis of the performance characteristics of widely used commercial mobile benchmark suites.
- Analysis of the temporal behaviours and heterogeneity exhibited by the workloads.
- Assessment of the similarity among individual benchmarks.
- Proposal of a reduced benchmark set, allowing researchers to reduce evaluation time by up to 75%.

II. MOTIVATION

Commercial mobile benchmark suites provide insights into how users interact with their mobile phones, offering a realistic representation of real-world usage scenarios. These benchmarks, widely adopted by industry to showcase the capabilities of new devices and remain up-to-date with the latest features integrated into mobile System-on-Chips (SoCs). As industry standards, they mirror evolving user behaviours and mobile hardware.

Existing benchmarks such as SPEC and PARSEC are designed for traditional computing environments. They fall short when evaluating the rapidly evolving and heterogeneous
TABLE I: Commercial mobile benchmark suites analyzed.

<table>
<thead>
<tr>
<th>Benchmark Suite</th>
<th>Benchmark Names</th>
<th>Targeted HW / Workload</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Mark v2</td>
<td>Slingshot</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>Slingshot Extreme</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>Wild Life</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>Wild Life Extreme</td>
<td>GPU</td>
</tr>
<tr>
<td>Antutu v9</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>GPU</td>
<td>GPU</td>
</tr>
<tr>
<td></td>
<td>Mem</td>
<td>Memory subsystem</td>
</tr>
<tr>
<td></td>
<td>UX</td>
<td>Everyday tasks (e.g., data/image processing, video decoding)</td>
</tr>
<tr>
<td>Antutu v2</td>
<td>-</td>
<td>AI-related tasks</td>
</tr>
<tr>
<td>Geekbench 5</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>Compute</td>
<td>GPU</td>
</tr>
<tr>
<td>Geekbench 6</td>
<td>CPU</td>
<td>CPU</td>
</tr>
<tr>
<td></td>
<td>Compute</td>
<td>GPU</td>
</tr>
<tr>
<td>GFXBench v5</td>
<td>High Level</td>
<td>GPU (overall graphics performance)</td>
</tr>
<tr>
<td></td>
<td>Low Level</td>
<td>GPU (specific graphics performance, e.g., tessellation)</td>
</tr>
<tr>
<td></td>
<td>Stress Test</td>
<td>GPU (render quality performance)</td>
</tr>
<tr>
<td>PCMark</td>
<td>Storage 2.0</td>
<td>Storage subsystem</td>
</tr>
<tr>
<td></td>
<td>Work 3.0</td>
<td>Everyday activities (e.g., browsing, videophoto editing)</td>
</tr>
</tbody>
</table>

components of modern mobile SoCs. Beyond conventional CPUs, these SoCs include digital signal processors (DSPs) for accelerated vector instructions, GPUs dedicated to tasks like media processing, and AI accelerators tailored for machine learning applications. The iPhone XS’s A12 chip, for instance, integrates 42 accelerators [17] with the number steadily rising in later models [18]. The limitations of traditional benchmarks highlight the necessity of turning to commercial mobile benchmark suites. For researchers, understanding these benchmarks is crucial. There is presently no comprehensive workload characterization study of these benchmarks; thus, emphasizing the need for one.

III. BENCHMARK SUITES: BRIEF OVERVIEW

In our analysis, we include some of the most popular mobile benchmark suites [19]–[30] (Table I). In this section, we provide an overview of them.

3DMark Android [31], published by UL, is a benchmark suite that measures the CPU and GPU performance of mobile devices. It is composed of two benchmarks, Wild Life and Slingshot. Both have Extreme versions with higher resolutions, resulting in a total of four sub-benchmarks. Wild Life runs for approximately one minute and measures a device’s ability to provide high levels of performance for short periods of time. It mirrors mobile games that have short bursts of intense activity. Slingshot tests a range of graphics API features, such as volumetric lighting (i.e., adding lighting effects to rendered scenes) and instanced rendering (i.e., rendering multiple instances of a model in a single draw call).

Antutu [32], published by Cheetah Mobile, is an all-around benchmark suite stressing various hardware components. It is composed of 4 parts; however, the user cannot execute those parts individually. Antutu GPU contains five GPU benchmarks. Three of them, Swordsman, Refinery and Terracotta, have mobile game-like high-end graphics. The other two, Fisheye and Blur, are simpler image processing tests. Antutu CPU contains mathematical operations (e.g., GEMM), common algorithms (e.g., PNG decoding) and multi-core tests. Antutu Mem stresses the RAM and the storage subsystem. Antutu UX includes data processing and data security workloads, image and video processing, as well as a scroll delay test. Aitutu is a standalone benchmark, from the creators of Antutu. It focuses on AI workloads like image classification and object detection.

Geekbench [33], published by Primate Labs, is one of the most popular mobile benchmark suites. It is designed to evaluate and compare the performance of devices across different platforms and operating systems. It is split into two components, one for testing the CPU and the other for the GPU. Geekbench 5 CPU contains three parts: integer, floating point (FP), and cryptography workloads. Geekbench 6 CPU is split into five subsections: productivity, developer, machine learning, image editing, and image synthesis workloads. Geekbench GPU Compute benchmarks evaluate GPU performance. Geekbench 5 Compute contains 11 workloads, while Geekbench 6 Compute contains 8 workloads divided in four categories: Machine Learning, Image Editing, Image Synthesis, and Simulation. Despite the different categorizations, there are multiple workloads that are shared between the two versions.

GFXBench [34], published by Kishonti, is a graphics-based benchmark suite aimed at testing the GPU. It is split into three categories: High-Level, Low-Level, and Special tests. High-Level tests stress the GPU in a mobile game-like manner. There are four graphics scenes: Aztec Ruins, Car Chase, Manhattan, and T-Rex. Each scene is executed with tweaked settings (e.g., resolution, API used) resulting in 19 separate benchmarks. The Low-Level category consists of 8 benchmarks. They measure specific performance aspects, like tessellation and texturing. Special tests measure visual fidelity.

PCMark Android [35], published by UL, is a benchmark suite used for measuring the performance and battery life of Android phones and tablets. It comprises of two benchmarks, Work and Storage. Work is composed of web browsing, data manipulation, and video, document, and photo editing workloads. Storage measures IO performance in internal and external storage, as well as database performance.

IV. WORKLOAD CHARACTERIZATION METHODOLOGY

In this section, we present our experimental setup and explain our approach in characterizing existing commercial mobile benchmark suites.
A. Experimental Setup

Table [II] presents our experimental system configuration. We use a Qualcomm Snapdragon 888 Mobile Hardware Development Kit [36]. The Snapdragon 888 features a tri-cluster octa-core Kryo 680 CPU. The top core is a single Kryo 680 Prime processor. The second cluster comprises of three Kryo 680 Gold processors. The third cluster includes four Kryo 680 Silver processors. The first two core clusters feature out-of-order superscalar designs, while the latter is an in-order superscalar pipeline. We refer to them as CPU Big, CPU Mid and CPU Little, respectively. There are 4 MBs of L3 cache memory that is shared across all clustered cores and 3 MBs of system-level cache memory.\(^1\) It also features an Adreno 660 GPU and an AI engine (AIE) with a Hexagon 780 Processor. The AIE is intended for compute-intensive multimedia applications (e.g., video, audio, image processing), neural-network-related calculations and communications. There are 12 GB of LPDDR5 RAM and 256 GB of flash storage. The hardware board has Android 11 installed and is connected to an external display with a 1920×1080 pixels (Full HD) resolution.

We use Qualcomm’s Snapdragon Profiler [37] to capture various metrics. The tool’s real-time view option enables us to capture over 190 hardware performance metrics that cover the following categories: 1) CPU-related including cores, cache, and branch predictor information, 2) GPU-related including cores, shaders, GPU memory, and GPU stalls, and 3) metrics about the AIE, system memory and temperature.

Benchmark Suites. We have split each suite into individual benchmarks that users can execute independently, except for Antutu and GFXBench. The Antutu benchmark consists of four components, GPU, Mem, CPU, and UX. However, users are unable to execute them individually; instead, they must run the entire suite. During Antutu’s execution, micro-benchmarks from each component are bundled together and executed consecutively. Given the benchmark’s extended runtime, we’ve organized the collected statistics into four segments, aligning with its constituent parts. Regarding GFXBench, we group its 29 micro-benchmarks into three categories, aligning with the classification by the benchmark designers, as outlined in Section III. We ran all benchmarks three times and averaged their metrics across runs.

Limitations. We briefly describe the limitations of our current evaluation setup:

1) Our evaluation platform and the tools available preclude the inclusion of power or thermal information in our analysis. The absence of a battery and casing in the development board limits the representativeness of thermal readings for a mobile platform. Furthermore, conducting power readings necessitates external hardware, which is not within the scope of our current capabilities.

2) We are limited in our analysis by the available metrics. As such, we are unable to broaden our analysis to other IP components.

\(^1\)This is an SoC-wide accessible cache (i.e., accessible by all components).

<table>
<thead>
<tr>
<th>Development Board</th>
<th>Qualcomm Snapdragon 888 Mobile Hardware Development Kit</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1X Kryo 680 Prime processor (ARM Cortex-X1 based) @ up to 3.0GHz (CPU Big)</td>
</tr>
<tr>
<td></td>
<td>5X Kryo 680 Gold processors (ARM Cortex-A78 based) @ up to 2.42 GHz (CPU Mid)</td>
</tr>
<tr>
<td></td>
<td>4X Kryo 680 Silver processors (ARM Cortex-A55 based) @ up to 1.8 GHz (CPU Little)</td>
</tr>
<tr>
<td>Cache</td>
<td>L1 Big Core: 64 KB L1 Inst., 64 KB L1 Data &amp; 1 MB L2</td>
</tr>
<tr>
<td></td>
<td>Per CPU Mid Core: 512 KB L2</td>
</tr>
<tr>
<td></td>
<td>Per CPU Little Core: 128 KB L2</td>
</tr>
<tr>
<td></td>
<td>4 MB L3 (for CPU cores)</td>
</tr>
<tr>
<td></td>
<td>3 MB System-level</td>
</tr>
<tr>
<td>GPU</td>
<td>Adreno 660</td>
</tr>
<tr>
<td>AIE Engine (AIE)</td>
<td>Hexagon 780 Processor</td>
</tr>
<tr>
<td>RAM</td>
<td>12GB LPDDR5</td>
</tr>
<tr>
<td>Storage</td>
<td>56GB</td>
</tr>
<tr>
<td>Manufacturing Process</td>
<td>Samsung 5nm low power early (LPE)</td>
</tr>
<tr>
<td>OS</td>
<td>Android 11</td>
</tr>
<tr>
<td>External Display</td>
<td>1920×1080 pixels</td>
</tr>
</tbody>
</table>

Table III: Correlation values between metrics.

<table>
<thead>
<tr>
<th>IC</th>
<th>IPC</th>
<th>Cache MPKI</th>
<th>Branch MPKI</th>
<th>Runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>IC</td>
<td>1</td>
<td>-0.867</td>
<td>0.350</td>
<td>-0.174</td>
</tr>
<tr>
<td>IPC</td>
<td>0.400</td>
<td>-0.228</td>
<td>0.460</td>
<td>0.350</td>
</tr>
<tr>
<td>Cache MPKI</td>
<td>-0.228</td>
<td>-0.845</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Branch MPKI</td>
<td>-0.174</td>
<td>-0.672</td>
<td>0.967</td>
<td>1</td>
</tr>
<tr>
<td>Runtime</td>
<td>0.588</td>
<td>-0.242</td>
<td>0.460</td>
<td>0.350</td>
</tr>
</tbody>
</table>

3) Snapdragon Profiler provides information about total system memory usage, including the Android OS and the services running. We gathered statistics with the system being idle and computed the average memory usage. We then deducted this amount from all process specific information.

V. Evaluation

A. Metrics

Figure [I] shows the average values of a few important performance metrics for all benchmarks. These metrics are the Dynamic Instruction Count (IC), Instructions per Cycle (IPC), Cache Misses per Kilo Instructions (MPKI), Branch MPKI and Runtimes. We split and colour the benchmarks into 5 groups, based on the clustering in Section VII. Table III shows the correlation between metrics, calculated using the Pearson correlation coefficient [38]. The correlation coefficient is an important measure that quantifies the extent of interconnection among these metrics. Correlation values of above 0.8 or below −0.8 are strong positive and negative associations, respectively. Values between [0.4 − 0.8] are moderate associations, while lesser values indicate that there is no association present.

The dynamic instruction counts of the included benchmarks are in the order of billions. There is an order of magni-
tude difference between the smallest benchmark (GFXBench Special Tests at 1 billion) and the largest one (Geekbench 6 CPU at 57 billion). The average IC is 14 billion. For comparison, SPEC CPU 2017 benchmarks are in the order of trillions. We observe that newer benchmarks tend to have higher instructions counts (e.g., Geekbench 6 vs Geekbench 5, 3DMark Wild Life vs 3DMark Slingshot).

The IPC metric is a critical measure for assessing the efficiency and performance of processors. A high IPC is indicative of high instruction-level parallelism. In general, a CPU-centric benchmark’s IPC value tends to be higher than one. In our evaluation platform, the CPU Big core can theoretically achieve a maximum IPC value of 8. All benchmarks explicitly targeting the CPU (i.e., Antutu CPU, Geekbench 5 CPU and Geekbench 6 CPU) have an average IPC of 1.16. We observe that benchmarks with a focus on graphics (e.g., GFXBench High Tests) exhibit lower IPC values, averaging at 0.55. Typically, 3D-graphics applications employ a strategy of pre-allocating textures anticipated for imminent use, leading to heightened memory bandwidth usage and significant occupancy of cache memory space. Our analysis reveals a notable correlation between IPC values and the utilization of GPU shader cores. We posit that the diminished IPC values can be attributed to cache contention arising from the substantial loading of graphics-related data into the cache. The only outlier is Antutu Mem with an IPC of 0.45, affected by its high number of cache misses.

**Cache misses per thousand instructions** (MPKI) is a strong indicator of data retrieval inefficiency. Similarly, **branch MPKI** reflects the efficiency of the processor’s branch prediction mechanism. Both metrics impact performance by introducing delays, necessitating data retrieval from slower memory levels and causing missteps in program execution. Moreover, both metrics are dependent on the micro-architectural design choices and intricacies of the SoC. We capture the misses across all levels of the cache hierarchy. We see that these metrics have similar trends and they exhibit negative correlations with IPC.

It is important for researchers to be cognizant of benchmark **runtimes**, as they impact the time spent on evaluating new designs. The average runtime is slightly over 200 seconds. Dynamic instruction count is frequently used as a predictor for the runtime of a program. However, we see that there is only a moderate correlation of 0.588 with the benchmark runtimes, as shown in Table III.

### B. Temporal Behaviour

Averaging values across a time series condenses the information, enabling a more granular understanding of workload fluctuations and system behaviour.

Figure 2 shows the normalized values of six metrics across the normalized execution time for all benchmarks. We normalize the values of the metrics to the [0 – 1] range. The highest values recorded for each metric across all benchmarks serve as the normalization’s upper bound, while the inverse is true for the lower bound. Coloured regions indicate a value exceeding 0.5. Table IV shows these metrics with an explanation. We opt for CPU Load instead of CPU utilization as it incorporates the frequency the CPU cores are running at. High CPU utilization levels at low frequencies can be misleading in terms of the stress the CPU cores are under. Following the same reasoning, we pick GPU and AIE Load. Percentage Shaders Busy and Percentage GPU Bus Busy show the amount of execution time that the GPU Shaders and the GPU bus are used. We chose these metrics because they collectively serve as important indicators of heterogeneous components’ behaviour in a mobile SoC. We can make several observations.

**Observation #1**: Benchmarks that include multi-core or multi-threaded components show high CPU load levels.

Geekbench 5 CPU and Geekbench 6 CPU exhibit a spike in CPU load when the multi-core segment of the benchmark is running. The single-core part has a significantly lower CPU load of close to 30% for both benchmarks. Similarly, Antutu...
CPU contains a multi-core micro-benchmark near the end that focuses on multi-threading and multi-tasking performance. The uptick in the beginning of Antutu CPU is due to a general matrix multiplication (GEMM) routine, commonly used in benchmarks due to its intensity [44], [45]. Most efficient matrix multiplication routines are multi-threaded. The steep increase in CPU load in 3DMark Slingshot and 3DMark Slingshot Extreme is due to physics tests. This test measures CPU performance, while minimizing the GPU workload. It has three levels, successively more intensive, and is highly multi-thread.

**Observation #2:** Benchmarks that use the Vulkan API have lower GPU load than the ones that use OpenGL.

GFXBench benchmarks that use OpenGL have 9.26% higher GPU load compared to Vulkan ones. This is due to Vulkan being a more efficient API [46], [47].

\(^{3}\)GFXBench Special Test appears to have AIE Load over 50% near the end of its execution time. The region does not appear coloured due to the timestamps with high loads not being contiguous.
### TABLE IV: Performance Metrics.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Load</td>
<td>Load on CPU Core (CPU Frequency × CPU % Utilization)</td>
</tr>
<tr>
<td>GPU Load</td>
<td>Load on GPU (GPU Frequency × GPU % Utilization)</td>
</tr>
<tr>
<td>% Shaders Busy</td>
<td>Percentage of time that all Shader cores are busy</td>
</tr>
<tr>
<td>% GPU Bus Busy</td>
<td>Percentage of time the GPU’s bus to system memory is busy</td>
</tr>
<tr>
<td>AIE Load</td>
<td>Load on AIE (AIE Frequency × AIE % Utilization)</td>
</tr>
<tr>
<td>Used Memory</td>
<td>Percentage of total system memory used</td>
</tr>
</tbody>
</table>

**Observation #3: Usage of GPU resources is not limited to GPU-related benchmarks.**

We observe that GPU shaders are not used exclusively by benchmarks that include 3D graphics workloads (e.g., 3DMark and GFXBench). PCMark Work exhibits sustained periods where the majority of shaders are used. This is attributed to the video and photo editing workloads that it contains. Similarly, we see that the percentage of time the GPU memory bus is busy is not proportional to a workload’s graphical intensity.

**Observation #4: Newer benchmarks are not always more computationally intensive.**

Antutu version 9 introduced a new GPU-focused micro-benchmark, Swordsman, that is executed at the beginning of the benchmark. It is followed by the Refinery and Terracotta Warriors micro-benchmarks and two short image-processing micro-benchmarks. The normalized running times of the benchmarks are 15%, 30%, and 49% of the total duration of Antutu GPU, respectively. Figure 2 shows that the CPU load of Antutu GPU spikes at 16% and 49% of the execution. Both of these moments are not during the newest benchmark’s execution. Swordsman, Refinery, and Terracotta Warriors have 28%, 31%, and 35% CPU load, respectively.

**Observation #5: Benchmarks make little use of AIE.**

While there are parts of benchmarks that stress the AIE, the average load is just 5%. Antutu UX exhibits short peaks close to 50% in the scroll delay, webview rendering and video decode tests. In general, Antutu UX includes photo- and video-related tests, leading to increased AIE usage. Antutu CPU’s mathematical functions (e.g., fast Fourier transform (FFT), map) and the PNG decoding test result in increased load. All the previously mentioned workloads are encompassed within the domain of DSP tasks, supported by the AIE. Similarly, 3DMark Wild Life and 3DMark Wild Life Extreme use FFT operations as part of the post-processing techniques. Antutu focuses on AI workloads. It has three image-oriented tasks, image classification, object detection and super resolution. We also observe higher AIE load levels on GFXBench Special tests. They compare a single rendered frame against a reference frame using a Peak-Signal-to-Noise-Ratio (PSNR) metric, based on the Mean Square Error (MSE). It is split into two sections, with the latter doing computations in higher precision. We theorize that the increase in AIE usage is due to the computation of the PSNR metric at the end of each part. Lastly, PCMark Work has an uptick in AIE load due to the video editing part of the test.

**Observation #6: The memory footprint of benchmarks is moderate.**

The system memory usage across all benchmarks is 21.6%. This is 2.55GB out of the 11.83GB of system memory. GPU-oriented benchmarks have higher memory usage compared to others. Our analysis reveals a strong correlation between utilized memory, the processing load on GPU shaders, and L1 Texture misses. As highlighted in Section V-A, benchmarks with intensive graphics demands exhibit elevated memory occupancy, attributed to the loading of textures. The highest recorded usage is 4.3GB during the execution of Antutu GPU. The highest average memory consumption is 3.8GB (34.5%) during the execution of 3DMark Wild Life Extreme.

In the rest of the section, we focus on some interesting benchmark behaviour characteristics.

- **Antutu UX** has high CPU load near the end due to the video encoding and decoding tests. These micro-benchmarks use common video formats like H264, H265, VP9, and AV1 [49]. All formats except the last are supported by the SoC’s AIE component. We speculate that the lack of support for AV1 leads to a considerable increase in CPU load, as it cannot be processed by the AIE.
- **GFXBench High and Low** benchmarks have two varieties of tests, on-screen and off-screen. In the former, the GPU drawing operations are performed on a region of memory that goes directly to the display output. This means that the tests run at a Full HD resolution. In the latter, the computations in memory are not directed to the display output. Off-screen rendering is used to generate intermediary images in computer graphics, like post-processing filters (e.g., blur). The off-screen tests have various resolutions. All tests can be executed at Full HD. The Manhattan test can also be executed at 2K QHD, while Aztec Ruins contain all previous options and a 4K one. In our tests, we observe that High level off-screen benchmarks result in a 14.5% increase in GPU load. Low level off-screen benchmarks exhibit a 62.85% increase.

### C. CPU Heterogeneity Analysis

Designing and evaluating mobile SoCs comprehensively is challenging, due to the heterogeneity of both their hardware components and workloads [50]. It is imperative for researchers to be aware of the level of heterogeneity exhibited by mobile benchmarks. Thus, they can ensure the fair and comprehensive testing of proposed techniques by picking...
appropriate target workloads.

So far, we have looked into CPU load as one metric that is the mean of the load of all cores. However, there are 8 cores in the SoC we are using, divided into 3 clusters. Work is not spread equally among all these cores and clusters.

In this part of the analysis, we examine the degree of CPU load exhibited by the benchmarks. Figure 3 shows the results. We categorize normalized CPU core load metrics into four levels (each covering 25% of the [0 – 1] range) and count their occurrences. This is averaged across CPU core clusters, as the load values for cores belonging to the same cluster are almost identical. The x-axis indicates the load of the three CPU core clusters as defined in Table II, while the y-axis represents the normalized benchmark runtimes. We use colours to represent the four load levels. Table V shows the average percentage of execution time each CPU core cluster spends in each load level across all benchmarks.

**Observation #7:** Bigger, more powerful cores have higher load levels than medium cores.

CPU Big has high load levels (i.e., 50% – 100%) sustained for longer than CPU Mid in all but one of the benchmarks that they are actively used in. In 3DMark Wild Life, GFXBench Low and GFXBench High both CPU Mid and CPU Big clusters see minimal use. Aitutu is the only benchmark where the CPU Mid cluster exhibits high load levels sustained for longer compared to CPU Big.

**Observation #8:** CPU tests tend to use only the energy-efficient cores.

CPU Big and CPU Mid have fewer instances of high load than CPU Little in all GPU tests (lower row of Figure 3). The computational demands of GPU-oriented benchmarks can mostly be satisfied with just the energy-efficient CPU cores.

**Observation #9:** Workloads tend not to exploit more than one type of core concurrently [51].

We observe a consistent load on all CPU core clusters only in Aitutu, Antutu CPU, Geekbench 6 CPU and Geekbench 5 CPU. As highlighted in Section V-B under Observation #1, these benchmarks are distinct in having workloads explicitly designed for multi-core architectures. Among them, the latter is the only benchmark that exhibits sustained high load levels in CPU Mid for more than half of its execution time.

Overall, we observe that few benchmarks utilize all CPU core clusters, even though heterogeneous mobile architectures have existed commercially since at least 2011 [52], [53]. Using the cores in the CPU Little cluster proves adequate in most cases. There is potential for future benchmark designs to fully utilize all CPU core clusters.

### TABLE V: Percentage of execution time spent by the CPU core clusters in the load levels.

<table>
<thead>
<tr>
<th>CPU Cluster</th>
<th>0% – 25%</th>
<th>25% – 50%</th>
<th>50% – 75%</th>
<th>75% – 100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Little</td>
<td>21%</td>
<td>32%</td>
<td>25%</td>
<td>22%</td>
</tr>
<tr>
<td>CPU Mid</td>
<td>76%</td>
<td>8%</td>
<td>8%</td>
<td>8%</td>
</tr>
<tr>
<td>CPU Big</td>
<td>69%</td>
<td>7%</td>
<td>6%</td>
<td>18%</td>
</tr>
</tbody>
</table>

VI. SIMILARITY AND REDUNDANCY

The benchmark suites we have included in our analysis contain 41 sub-benchmarks that can be individually executed. Their combined runtime on a real device is over 110 minutes. Architectural research often employs simulators (e.g., gem5 [54]) that are thousand times slower than native execution [55]. While limiting simulation to smaller regions of interest is commonly employed [56], [57], it is not always an option. The closed-source nature of these commercial benchmark suites renders modifications to their source code nearly impossible. Moreover, choosing a Region of Interest (ROI) poses challenges, given our observations that these benchmarks can encompass various types of workloads (e.g., 3DMark Slingshot). Thus, simulating all benchmarks or even natively running them can result in prohibitively expensive execution times.

In this section, we perform a statistical analysis on the sim-
Clustering is a method for finding subgroups of observations within a larger dataset. In this section, we present some issues that arise with the use of clustering algorithms.

Optimal number of clusters. Selecting the optimal number of clusters is challenging in unsupervised learning. In scenarios like workload characterization with diverse benchmarks, the ideal number of clusters may not be evident. Most algorithms lack prior knowledge, often relying on user input for cluster count. However, real-world workload differences are not always clear-cut, making the number of clusters ambiguous. Consequently, evaluating the produced clusters using validation measures becomes crucial in ensuring meaningful insights.

We use two methods, Internal and Stability validation [58]. Internal validation measures the compactness, connectedness, and separation of the clusters. That is, we want the average distance within a cluster to be as small as possible and the average distance between clusters to be as large as possible. We use two popular measures that demonstrate the above characteristics, the Dunn Index [59] and the Silhouette Width [60]. Higher values are better for these measures. Stability validation evaluates the consistency of a clustering result by comparing it with the clusters obtained after each column is removed, one at a time. We use two cluster stability measures, the average proportion of non-overlap (APN) and the average distance (AD) [61]. Lower values are better for these measures.

Clustering Algorithms. There are several clustering algorithms, some of which see widespread use. However, there is no single method that is universally considered the best. Thus, we use three common clustering techniques, **K-means** [62], **Partitioning Around Medoids (PAM)** [63] and Agglomerative Hierarchical clustering [64]. Instead of just using a single algorithm, we try to find common patterns that emerge across different techniques.
Figure 4 shows the results of the clustering validation analysis. The two sub-figures to the left show the internal validation measures, while the two right ones show the stability validation measures. We observe that the optimal number of clusters is 5 for both the internal measures, regardless of the clustering technique used. As for the stability measures, APN shows a tie among various number of clusters with a general preference towards the lower range. The AD measure indicates a strong bias for a higher number of clusters. Considering that three out of four measures have 5 clusters as their top pick, we select this number.

Figures 6 and 5 depict the clusters created after applying a Hierarchical and a K-Means clustering algorithm, respectively. We omit the results of a PAM algorithm as they are similar to K-Means. We average the metrics across the benchmarks’ runtime. The resulting data are provided as input to all these techniques. We see that all three algorithms group the sub-benchmarks identically. This validates the correctness of our clusters.

B. Reduced Benchmark Set

Computer architects frequently subset benchmark suites to streamline evaluations and focus on specific aspects relevant to their research. The goal of a subset should be to provide the same information as the full benchmark suite. Researchers have to navigate a fine balance between efficiency and the need for comprehensive coverage.

A commonly employed subsetting technique is to select a single benchmark from every benchmark cluster. However, this method introduces potential challenges, as it does not guarantee coverage of all application domains. Furthermore, it does not take into account the unique circumstances of the benchmarks in our original set. Thus, we choose to showcase and evaluate multiple subsetting techniques.

Naive Subset. We select one benchmark from each cluster. We make the decision based on the benchmarks’ execution times, since we want to minimize time spent on evaluation. The Naive subset is comprised of PCMark Storage, Geekbench 5 CPU, GFXBench Special, 3DMark Wild Life and Geekbench 5 Compute.

Select Subset. As we explain in Section IV-A, the different parts of the Antutu benchmark cannot be executed individually. The user can only run it in its entirety. All of Antutu’s segments (i.e., Antutu CPU, Antutu Mem, Antutu UX) are grouped in the purple cluster except Antutu GPU. We begin our selection with Antutu instead of PCMark as it has the additional benefit of covering the GPU. Our goal is to ensure that the reduced set stresses all of the SoC components. Thus, we include GFXBench Special Tests as it provides the highest AIE load. Finally, we select Geekbench 5 CPU as it covers the need for stressing all CPU core clusters, while having a shorter execution time than Geekbench 6 CPU.

Select + GPU Subset. The previous reduced set included Antutu GPU for stressing the GPU subsystem. However, it does not provide the highest GPU load. We present a third reduced set option that includes Geekbench 6 Compute, as it has the highest average GPU load exhibited among all benchmarks.

Table VI shows the total running times for all benchmarks and the proposed reduced sets. It also shows the running time reduction percentages compared to executing all benchmarks. We observe that even the subset with the slowest running time (Select + GPU) results in a close to 75% reduction in execution time.

Careful consideration is crucial when creating a benchmark subset. The objective is to choose a subset of benchmarks that, when executed, yields nearly equivalent information as the full suite but in considerably less time. We measure the representativeness and coverage of our presented subsets by using a technique presented by Yi et al. [67]. We follow these steps:

1) Create a vector containing the values of all performance metrics of each benchmark.
2) Normalize the performance metrics to the maximum recorded value of each.
3) Compute the Euclidean distance between each benchmark vector not in the subset to each benchmark vector in the subset.
4) For each benchmark vector not in the subset, we pick the minimum Euclidean distance with a benchmark vector in the subset.
5) Sum the Euclidean distances for all benchmark vectors and assign that value as the total minimum Euclidean distance for that subset.

A smaller total minimum Euclidean distance means that the benchmarks that are not in the subset are very close to a benchmark that is in the subset. This means that every benchmark not in the subset is accurately represented by one in the subset. Also, it means that the coverage extends to all benchmarks in the original set.

Figure 7 shows the total minimum Euclidean distances for reduced benchmark subsets.
the three subsets. We start by having a single benchmark in each subset. For example, the Naive subset begins with PMark Storage, while the Select subset has Antutu CPU. We continue by adding one of the benchmarks that belongs in each subset at each step. After all benchmarks that belong in a subset have been added, we add the rest of the benchmarks. We measure the total minimum Euclidean distance at each step. The dashed lines indicate the total minimum Euclidean distance of each subset as they were presented in Section VI-B. Notably, the Select + GPU subset, comprising 7 benchmarks, exhibits a total minimum Euclidean distance of 11. This positions it towards the lower end of the range of Euclidean distances, specifically at the 32.5% percentile. This signifies a reduction of 22.96% and 9.78% when compared to the Naive subset with 5 and 7 benchmarks, respectively. Therefore, our technique for selecting a reduced benchmark set is proven to create a representative subset that accurately captures the characteristics of the entire starting set of benchmarks.

VII. CONCLUSION

In summary, our research thoroughly explores and explains commonly used commercial mobile benchmark suites, offering important insights for the computer architecture community. We analyze hardware performance metrics, uncovering the details of workload similarities and their impact on different parts of the system-on-chip (SoC). Our study not only looks at how workloads change over time but also presents the complexities in how CPU core clusters are used.

Moreover, we find a smaller set of benchmarks, the Select + GPU combination, which significantly reduces the time needed for evaluations by almost 75%. This reduction in time does not compromise the thorough assessment needed for robust designs. Our discoveries highlight the importance of choosing the right benchmarks for efficient evaluations, especially considering the ever-changing landscape of mobile system architectures.

We believe that these insights contribute to the ongoing discourse on benchmarking methodologies, providing valuable considerations for researchers engaged in mobile system architecture assessments.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their thoughtful and detailed reviews of this work. Thanks to members of the NEJ group for their valuable feedback on this paper. We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant RGPIN-2020-04179. This research was undertaken, in part, thanks to funding from the Canada Research Chairs program and through the support of the University of Toronto McLean Award.

REFERENCES


