Evaluating video game performance on entry-level devices

Muhammad Haris
Department of Computer Science
Lahore University of Management Sciences
Lahore, Pakistan
22100013@lums.edu.pk

Muhammad Hamza Sajjad
Department of Computer Science
Lahore University of Management Sciences
Lahore, Pakistan
22100025@lums.edu.pk

Zain Ali Babar
Department of Computer Science
Lahore University of Management Sciences
Lahore, Pakistan
22100204@lums.edu.pk

ABSTRACT
With the increasing popularity of entry level devices in developing countries, the demand for mobile games has also increased. The aim of this paper is to analyze the performance of these games on entry-level devices. Since no major prior work has been done in this area, we have started our research from scratch. The major component which we have used for measuring the performance of a game are its Frames Per Second (FPS). We analyzed the dips and rises in the FPS along with the effects of limiting CPU cores and inducing memory pressure on the device. In order to make sure that we had a stable connection during these experiments, we ran several continuous speed tests while performing these experiments to rule out the possibility of network being a bottleneck.

General Terms
Measurement, Documentation, Performance, Experimentation.

Keywords
Memory pressure, rendered Frames per second, App, Random Access Memory (RAM), processor.

1. INTRODUCTION
The video game market has grown considerably over the years. The increase in sales for gaming consoles, the hardware that is dedicated only to playing games, and high personal computer parts is an evidence for this. However, these things are a considerable economic investment especially for someone living in developing countries. But entry-level mobile phone market could be a solution for this problem. Since mobile phones have become integral to part of life, they have a huge demand everywhere in the world. Hardware manufacturers produce a wide variety of phones for consumers globally. Entry-level devices are common in developing countries as they can provide most of the functionalities of higher spec devices but at a lower cost.

These low-level devices have access to the same apps through the application stores as the high-end devices. As the entry-level devices increase in popularity so does the demand for gaming applications on it. Since the hardware can vary greatly in different price ranges the performance of games usually suffer on the low-level devices, especially as the games available increase in complexity. This negatively affects the user experience. Therefore, our project aims to analyze the performance of these games on entry-level devices.

We used Frame-per-second (FPS) rendered as the metric for measuring performance as higher FPS would provide a smoother and more responsive experience to the player. For our experiments we used two games for evaluation, MiniMilitia and Asphalt 8. Initially we used MiniMilitia for testing as it was the lighter game of the two. While the game was being run, we observed the CPU and memory usage. We found that CPU and memory usage show bursty behavior and varies according to the object being rendered. For example, in this game when a player dies the body breaks into pieces and current score is shown the CPU and memory usage would increase. To quantify the difference in performance because of this we then started measure the FPS of the game. This showed a better picture for the jitters experienced. To make sure network was not the bottleneck for a multiplayer game we ran a continuous speed test in the background, on the device, to make sure internet connectivity was stable. Internet connectivity remained stable which led us to conclude the variation in performance was due to the hardware only.

With this knowledge we moved on to designing experiments. Different variables were changed for the android devices running two video games previously mentioned. We varied the active number of CPU cores and memory pressure on the device to observe their effect on the gaming performance. Performance of the game was measured using a tool for recording FPS each time a CPU core was disabled. For each number of active cores we also varied the memory pressure on the device using a tool previously developed [1]. We found disabling cores had a much larger impact over all on the user experience than memory pressure being applied.

2. METHODOLOGY

In this section, we will outline our experimental setup and the tools and methods we utilized to carry out the tests on different games. We will also describe the metrics measured.

To determine the performance of the game and to find the causes for dips in fps we had to observe different factors. We used the adb shell to observe the CPU usage and free memory available to the mobile device. The “adb shell top -m” command allowed us to see the resource usage of top 10 applications, by CPU usage, running on the android device. To measure the FPS rendered by a game we used the tools KF Mark and Gamebench. KF Mark automatically provided the FPS vs time graphs, however because of its incompatibility with our main testing device we were unable to utilize it. Instead Gamebench was used. Gamebench’s free version displays FPS on screen in real-time however only provides the overall average of FPS measured during a session. We had to resort to recording the screen of the device while the game was running and then manually create a graph from those recordings. Adb command for gfxinfo did not prove useful as it did not provide the count of the frames being rendered by the games we tested. The stability of the network was measured by using StarTrinity- Continuous Speed Test tool. It ran an internet speed test in the background so it can be determined if the network was a bottleneck during our testing or not.

Initially, we measured the basic metrics which were CPU and Memory usage. We found out that at several occasions, while playing the games, there was increase in both these metrics. We then moved on to analyze the FPS in order to ensure that these changes actually affected the performance of the game. As we found out that the performance of the game was affected at these times, we designed several experiments to find out the cause for such behaviour.

The effect of hardware on video game performance was evaluated by varying the active number of CPU cores and creating artificial memory pressure on the device. The number of cores were varied by using adb, however this can only be done on a rooted android device. Artificial memory pressure was created by using a previously developed tool [1]. By decreasing the number of cores we were able to see the effect of reduced compute power on FPS and by increasing memory pressure we tried to simulate the environment of running a game on a device with low available memory. These controlled tests were done on an LG Nexus 5 smartphone. FPS for the game were then measured for each change in conditions to determine how large of an impact on performance it had.

3. EVALUATION

In this section, we will outline the tests we performed on the device LG Nexus 5.

We used FPS as a performance metric to measure the games’ performance across different resource constraints. We ran network tests as the game was being played to rule out network throughput as a potential bottleneck in the game performance. The average speed was 4-5 Mbps with an average packet loss of 0.0%, which reasonably ruled out network bandwidth as a bottleneck in performance.

We varied the number of enabled cores as well as applied varying amounts of memory pressure on the device to measure the changes in rendered FPS while playing the game. We first tested on the game MiniMilitia. Four settings were used initially: 4 cores enabled with no memory pressure, 4 cores enabled with 512 MB memory pressure, 1 core enabled with no memory pressure, and 1 core enabled with 512 MB memory pressure. The results of these tests are summarized by the following graphs:

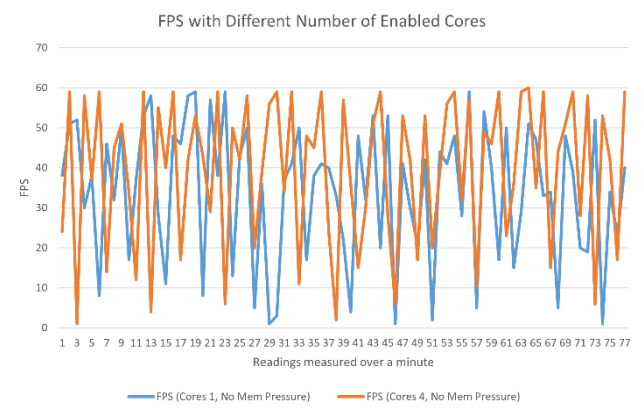

Fig 3.1

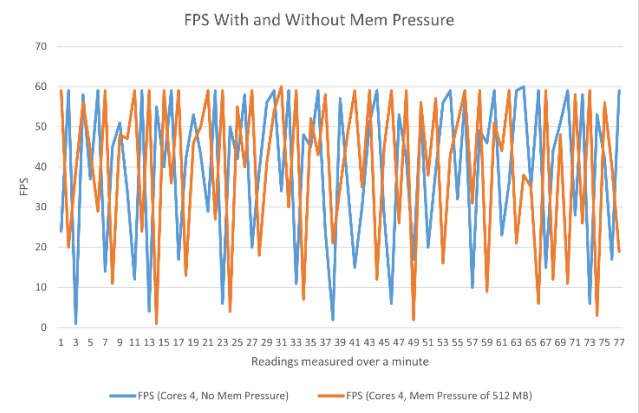

Fig 3.2

The average FPS for these settings were:
1 Core, No Mem Pressure: 33.96 FPS
4 Cores, No Mem Pressure: 39.53 FPS
4 Cores, 512 MB Mem Pressure: 39.83 FPS

These results indicate that in the case of Minimilitia, reducing the number of enabled cores reduced the average FPS slightly, whereas the applied memory pressure of 512 MB had a negligible effect on the average rendered FPS.

The same tests were run on the game Asphalt 8 too. The results are as follows:

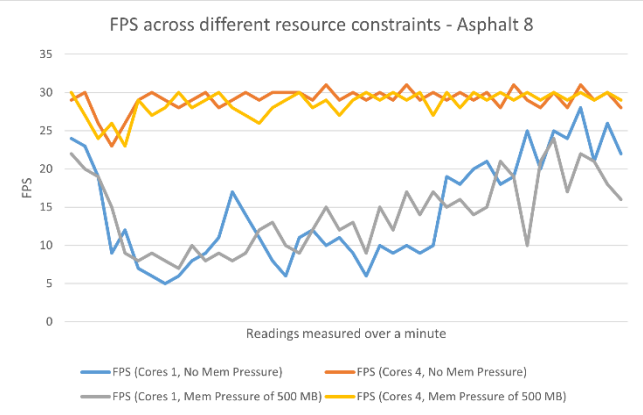

Fig 3.3

The average FPS for these settings were:
1 Core, No Mem Pressure: 14.48 FPS
1 Cores, 500 MB Mem Pressure: 14.05 FPS
4 Cores, No Mem Pressure: 29.10 FPS
4 Cores, 500MB Mem Pressure: 28.55 FPS

It is to be noted that the maximum FPS for Asphalt 8 on our device Nexus 5 was around 30 FPS.

The results in Fig 3.3 show an increasingly marked difference in the average FPS when the number of enabled cores was reduced, compared to applying memory pressure of 500 MB. The decrease in FPS when disabling cores (around 29 FPS to around 14 FPS) was greater here as compared to Minimilitia, and memory pressure of 500 MB again had a negligible effect on the average FPS (around 29 FPS in both cases).

There seemed to be an indication that the memory-daemons kswapd and lmkd had a role to play in reclaiming memory, especially when memory pressure was applied. Hence, tests were run with varying memory pressures of 0 Mb, 500 Mb, and 1000 Mb to see when these daemons get activated and whether that coincides with decreases in free available RAM in the device overall, as well as with app crashes.

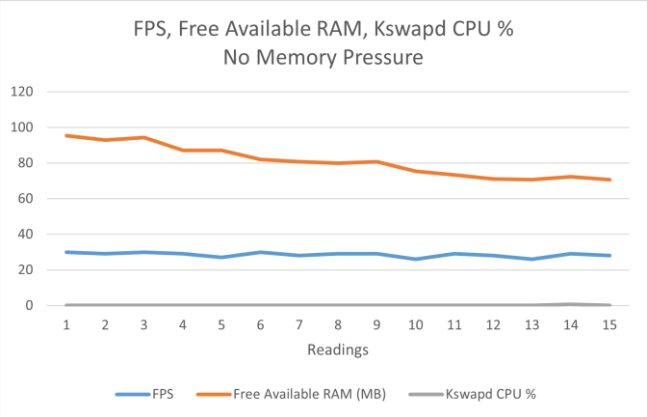

Fig 3.4

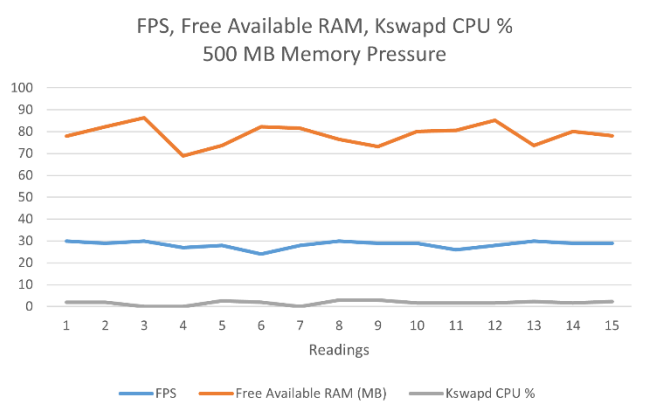

Fig 3.5

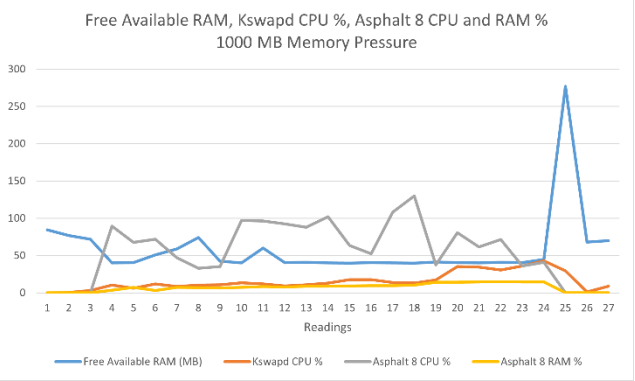

Fig 3.6

Fig 3.4 shows slight correlation with decrease in available RAM with average FPS. KSwapd was only activated once (0.6 % CPU) in this test.

Fig 3.5 showed a correlation between free available RAM and kswapd's CPU utilization percentage. Drops in available RAM corresponded with rises in kswapd's CPU utilization.

Fig 3.6 indicates the same correlation. The sharp rise in available RAM corresponds with when the app crashed (under high memory pressure of 1000 MB). The Kswapd CPU utilization peaked at about 43 % right before that, before decreasing to almost 0% once the used RAM was released. There was no visible usage of lmkd, at least in top 10 processes in adb shell. Moreover, average CPU utilization of Asphalt 8 was significantly high at around 80-90%, whereas RAM usage hovered around 10-15%.

4. DISCUSSION

The results obtained from the evaluations performed provide the major insight that the main bottleneck in video game performance is the processor or CPU (and GPU) of the device, rather than its RAM. This is substantiated by the graphs which show a significantly larger drop in the rendered FPS when three CPU cores were disabled, as compared to an induced memory pressure of 500 MB. Also, Asphalt 8's CPU usage was much higher than its RAM usage (as mentioned in the Evaluation section), which further corroborates this claim that CPU demands of these games are much higher than memory demands. An explanation for this is that these video games, especially more graphically intensive games like Asphalt 8, are more computationally expensive due to the increased computation requirements arising from higher graphics and complex game physics. Therefore, the limiting factor in these mobile multiplayer games is the processor used in the device.

If the CPU and GPU in the processor operate at higher frequencies, and perform computational operations at a quicker rate, then the computational operations in the mobile games would be carried out quicker. RAM is also important in enhancing the game's performance, but only to the extent that it is not fully used up. If there is sufficient free available RAM, the game can still run smoothly. This is indicated in the results when adding memory pressures of up to 500 MB, which is one-fourth of the total device memory (2 GB). These memory pressures caused a negligible change in the rendered average FPS when testing on two different games. Higher memory pressures (and thus lesser free available RAM) did cause stutters, and crashes at very high memory pressures. However, in normal circumstances, if the RAM is at least 2 GB, video games can still run relatively smoothly without crashing, if the processor is capable enough. Hence, the processor is the main bottleneck in the performance of these video games in smartphones, including entry-level devices.

These results also have several implications for both device manufacturers (OEMs) and Android game developers. OEMs should aim to provide minimum specs of 2 GB RAM, at least quad core CPU, and then prioritize these resources.

Game developers can run these or similar tests on their games across a heterogenous set of smartphones to gauge what are the general minimum requirements for CPU, RAM etc for the game to run relatively smoothly (e.g. at a certain minimum FPS) and not crash frequently. The developers can add these minimum requirements in the play store so only those devices can download these games who at least meet the minimum requirements (or still allow to download with the disclaimer that minimum requirements not met, so game will likely stutter and even crash). Game developers can also develop their games with the intent to utilize multi-core parallelism to use all cores and maximize the performance of the games by improving efficiency in computational operations.

5. RELATED WORK

According to our research there are no prior works that are directly related to video game performance on entry-level mobile devices. There were papers covering benchmarking of various process but not video games. However, some works were used indirectly. Mainly the papers "Mobile Web Browsing Under Memory Pressure"[1], "Performance analysis of game world partitioning methods for multiplayer mobile gaming"[2] and "Building high-performance smartphones via non-volatile memory: the swap approach"[3].

The first paper[1] focused on measuring performance of web browsers on entry-level android devices. Web browsers may run into the same problem of low resource availability a game running on an android device might. The tool for creating artificial memory pressure on a device also proved useful for us as it helped us simulate an environment with low ram available on a single device., allowing us to keep other variables constant.

The other two papers did not cover benchmarking performance of mobile devices. "Performance analysis of game world partitioning methods for multiplayer mobile gaming"[2] proposed a method of optimizing massively multiplayer games to reduce resource required to run them. This consisted of dividing a level into smaller parts. While "Building high-performance smartphones via non-volatile memory: the swap approach"[3] covered the limitations of swapping on mobile devices. Although this paper is old, we think the limitations still apply to the entry-level mobile devices.

6. CONCLUSION

We ran controlled experiments on the mobile device. We varied the number of cores and the free memory available to the device. FPS were measured to quantify the performance of the game being run. From our testing we concluded that there is a larger impact on performance of the game due to lesser number of compute resources than

less available RAM. This conclusion was made as the effect on FPS was larger when cores were disabled compared to when reasonable memory pressure was induced. Increasing memory pressure after a point just caused the game to not run. This shows that if less RAM is available, it is likely the game just will not run than provide a poor performance.

Future work can include running these tests on a larger number of devices to get a better idea of the causes of performance variance, since there is a large variation in hardware in the low-level devices that are available in the market.

7. ACKNOWLEDGMENTS

We would like to thank our instructors Ihsan Ayyub Qazi, Zafar Ayyub Qazi and Zartash Afzal Uzmi for guiding us in our research project.

8. REFERENCES

[1] Qazi et. al, “Mobile web browsing under memory pressure.” *ACM SIGCOMM,* 2020

[2] K. Prasetya and Z. D. Wu. “Performance analysis of game world partitioning methods for multiplayer mobile gaming.” *ACM SIGCOMM,* 2008

[3] K. Zhong et. al, "Building high-performance smartphones via non-volatile memory: The swap approach," *EMSOFT*,2014