Demystifying Hardware Bottlenecks in Mobile Web Quality of Experience

Mallesham Dasari, Conor Kelton, Javad Nejati, Aruna Balasubramanian, Samir R. Das
Computer Science Department, Stony Brook University, Stony Brook, NY

ABSTRACT

Mobile web page load time depends on three key factors: (1) complexity of website, (2) underlying network conditions, and (3) processing capability of devices. While there is substantial work focusing on Web complexity and network, there is little work in understanding the hardware bottlenecks in page load process. In this poster, we analyze the effect of hardware bottlenecks of Web pages. We also analyze the effect of GPU offloading, a commonly used solution to speed up Web page loads.

1. INTRODUCTION

Typically, mobile web users experience a low quality page performance compared to desktop users. Our goal is to determine the root cause for this poor performance. There are three possible reasons for this root cause. Fig. 1 shows factors influencing at different layers of page load : Application layer (website complexity, encryption overheads), OS/HW layer (memory and processing bottlenecks) and Network sub layer (network wide parameters).

With the increasing speeds of Wi-Fi and Cellular connection, mobile page loads are no longer bottlenecked just by network. Often, due to limited resources, mobile application processor may be overwhelmed with processing of other applications and prone to slow down page rendering process. This leads to degraded web quality of experience even when there is enough network bandwidth available. In fact, on mobile devices, it has been shown that the computational tasks are the bottlenecks on the critical path during page load process [4]. Upon receiving Web objects from the cloud, the browser has to go through different stages of rendering such as parsing (e.g HTML, Javscript), scripting (CSS, JS), layout, and painting that introduces huge computation bottleneck. To understand this we anatomize the impact of hardware on web experience in mobile devices.

This problem is especially important for low-end devices. Over 62%-68% [1] of mobile users from developing regions (specifically India and African countries) use low quality smartphones and experience bad web experience. This motivates us to explore hardware bottlenecks in browsing path and improve Page Load Time (PLT) for low quality mobile users. In this poster, we show evidence to suggest that computation bottlenecks are still present in high-end mobile devices and it becomes far worse for low-end hardware.

Prior work explored that given the same network resources for mobile and desktop browsers, mobile devices perform poor because of the limited processing capabilities and computation interference from other applications [7]. In [4], the performance of mobile and desktop browsers are investigated and found that while the desktop browsers are limited mostly by network, mobile browsers have computation bottlenecks. Webcore [7] optimizes hardware architecture for mobile web to improve PLT as well as to minimize energy consumption. Also, many mobile browsers (Chrome, Firefox and Edge) have come up with GPU accelerated compositing page layers. Each browser uses software rendering or exploit hardware accelerated rendering depending on the availability of hardware or after user preference. Our contributions include:

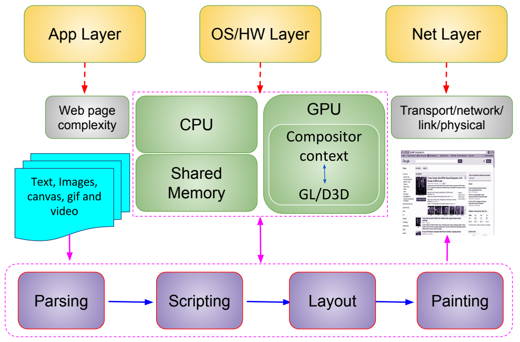


Figure 1: Architecture of Page Load Process

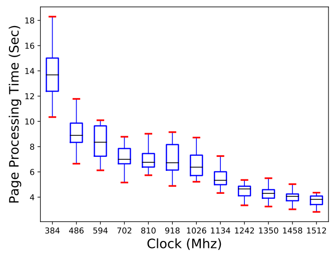
- Investigating mobile web page load performance with respect to underlying hardware resources.
- Dissecting the page load time into critical stages (loading, parsing, scripting, layout and painting) and finding critical blockage points in terms of compute.
- Exploring the use of GPU accelerated compositing (hardware rendering) over software rendering.

2. MOTIVATION

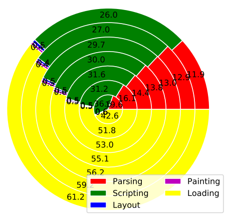
We first study the effect of the underlying hardware on page load time (PLT). To this end, we exclude loading time and measure only Page Processing Time (PPT) at the client, with respect to different frequency governors and processor clock rate to emulate different low-end mobile devices.

SetUp.

Our experiments are performed on a Google Nexus5 Android smartphone with the chromium browser. We use Google chrome developers tools [3] to trace browser events and calculate start and end times for all corresponding events. From this, we get total CPU time spent on individual activity (parsing, scripting, layout and painting). We collect hardware resource consumption statistics using Snapdragon Profiler from Qualcomm [5]. We modify CPU clock frequency to emulate different low-end mobile devices clock using android debug bridge tool on Linux. Typically, android governors control frequency to dynamically adapt to balance performance and power consumption. We experiment with

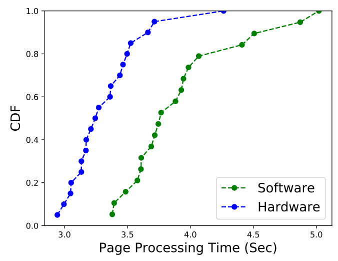


(a)

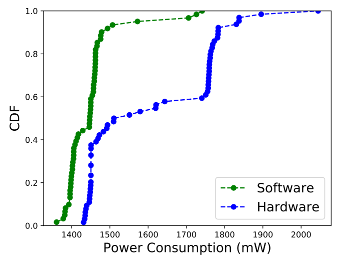


(b)

Figure 2: (a): PPT versus Clock Frequency, (b): Dissection of PLT into Loading, Parsing, Scripting, Layout, Painting



(a)



(b)

Figure 3: (a): PPT versus Hw-Sw Rendering, (b): Power Consumption versus Hw-Sw Rendering

different frequency governors available on Nexus5. To minimize variances due to Internet delays, we emulate network conditions using Linux Traffic Control tool [2]. To understand impact of memory, we change memory availability by creating different RAM disk levels (in steps of 256MB) from available memory and assign RAM disk to memory intensive workloads to occupy completely.

Results.

We observe a greater extent of (10sec) median PPT difference from a low-end (384Mhz which is the least available frequency on Nexus5) to a high-end frequency (1512Mhz) (Figure 2(a)). This is averaged for four available governors on Nexus5 (powersave, performance, interactive and on-demand) over 50 runs. It also shows that even at a high-end frequency, there is a median 3sec processing delay incurred and its way worse (14sec) at low-end frequency. This shows that computation is serious issue in mobile devices and there is a lot of potential to improve architectural support for good web experience.

Next, we analyze various components that make up the PLT including parsing, scripting, rendering, painting along with loading (networking) of objects, as a function of the clock rate. These results are obtained using the WProf tool [6], that provides the fine-grained component level breakdown of the PLT on the critical path. Figure 2(b) shows that, as the clock frequency increases, the fraction of time spent on network decreases. When moving from the highest to the lowest clock frequency, the fraction of network loading decreases by a median of 18.6%.

For lower frequencies, scripting (36.1%) and parsing (19.6%) make up a large amount of the critical path, while layout and painting contribute negligibly at all frequencies(0.5% each). Furthermore, with increase of hardware availability, the fraction of time spent on scripting and parsing computation decreased by a median of 7.8% each.

We further find that memory availability is also a severe issue for page performance. Memory is constrained on smartphones, with 56.7% of the devices having less than 1GB of RAM [1]. When comparing the PLT on smartphone devices from 512MB of RAM to 2GB of RAM, the PPT reduced by 8 seconds in the median case, which is a reduction of 12.5%.

3. EFFECT OF GPU OFFLOADING

Next we study the effect of offloading browser computation to the GPU. Due to highly parallel nature of web content, modern browsers are embracing GPUs to minimize page juddering. Compositing page layers on the GPU helps Webpages with more images, video and heavy page drawing. However, using this power hungry device is definitely a problem for battery life particularly when run with high performance CPU and GPU governors. We study this trade-off.

We measured page processing time with respect to software and hardware accelerated compositing in chrome browser by loading the Alexa top 20 websites (averaged over 50 runs) on Nexus5. We set the clock frequencies to 1512Mhz for CPU clock and 400Mhz for GPU clock with interactive frequency governor. Figure 3(a) shows that offloading compositing to the GPU reduces a median PPT of 0.5sec i.e, 12.5% reduction) for 90% of the time.

We measure power consumption (using Snapdragon Profiler) during both software and hardware rendering. Figure 3(b) shows that by using the GPU for rendering, the power consumption increases by 22% for 90% of the time. This gap between performance and power consumption is the key trade-off for GPU offloading.

4. ONGOING AND FUTURE WORK

We are currently working on bridging the gap between resource consumption and mobile web performance. Our end goal is to identify critical bottlenecks and extraneous components in the page load process. Our work is especially focused on low-end mobile devices that are popular in developing regions.

5. REFERENCES

[1] http://hwstats.unity3d.com/mobile/.

[2] Werner Almesberger. Linux traffic control. Technical report, 1998.

[3] https://developer.chrome.com/devtools.

[4] Javad Nejati and Aruna Balasubramanian. An in-depth study of mobile browser performance. In *Proc. WWW 2016*, pages 1305–1315, 2016.

[5] Qualcomm Development Network. developer.qualcomm.com/software/snapdragon-profiler.

[6] Xiao Sophia Wang, Aruna Balasubramanian, Arvind Krishnamurthy, and David Wetherall. Demystifying page load performance with wprof. In *NSDI*, pages 473–485, 2013.

[7] Yuhao Zhu and Vijay Janapa Reddi. Optimizing general-purpose cpus for energy-efficient mobile web computing. *ACM Transactions on Computer Systems (TOCS)*, 35(1):1, 2017.