WebCore: An Architectural Substrate for Enabling High-Performance and Energy-Efficient Mobile Web*

Yuhao Zhu Vijay Janapa Reddi

1. Introduction

The landscape of mobile computing has experienced a tremendous transformation over the past decade. The key enabler behind this transformation is the advancement in Web technologies, which foster rapid development productivity and strong connectivity. Enabling Web technologies, such as HTML, CSS, and JavaScript, are platform independent, and as such the Web browser acts as the intermediate layer that dynamically translates Web applications to different platforms. Such a "write-once, run-anywhere" feature is almost an absolute necessity in today's heavily fragmented mobile ecosystem.

The mobile platform portability that Web technologies bring for development and deployment productivity is often encumbered by large performance overhead and poses significant demand for processor capability. Prior work has shown that over the past decade, the client-side computational requirements for loading webpages have increased by up to 10X [16]. On average, mobile Web browsing demands over 80% CPU usage [11]. However, providing high CPU performance in the mobile context is challenging because of the marginal improvement on Lithium-ion battery density and battery capacity [14], combined with the demise of Dennard scaling.

We propose WebCore, a general-purpose processor customized and specialized for Web technologies. It provides an architecture substrate for supporting high-performance and low-energy mobile Web applications. WebCore starts from existing general-purpose mobile CPUs. Exhaustive design space exploration reveals that existing general-purpose designs have two inherent sources of energy inefficiency for mobile Web applications: instruction delivery and data feeding. We show that customizing current general-purpose designs by tuning key microarchitecture parameters improves energy efficiency.

To mitigate the two sources of energy inefficiency, WebCore applies two specialization techniques. The style resolution unit (SRU) accelerates the rendering engine of a Web browser, specifically the computationally intensive *Style* kernel. It aggregates computations into a few operations, and therefore greatly reduces the instruction footprint and mitigates the instruction delivery issue. The browser engine cache (BEC) achieves significant energy savings in data feeding by exploiting the Web browser's unique data access pattern using an extremely small scratchpad memory.

In summary, WebCore embodies our philosophy of designing a domain-specific processor for the mobile Web. First, WebCore retains the programmability of general-purpose processors. General-purpose programmability is essential for the modern Web browser, which is an extremely complex software system developed using more than 30 languages. Second, WebCore effectively incorporates specialized hardware by exploiting unique computation and communication patterns of the mobile Web browser to improve energy efficiency.

2. WebCore Design

In this section, we describe how WebCore customizes the general-purpose processors to uniquely suit mobile Web applications and how it incorporates specialized hardware units to support core computation kernels and communication patterns.

2.1. Customization

In the spirit of maintaining general-purpose programmability, WebCore starts from existing general-purpose designs and customizes the design parameters to identify an ideal general-purpose baseline architecture for mobile Web applications. We explore a vast design space by varying key microarchitecture parameters, which in total cover over 3 billion design points including both in-order and out-of-order cores.

Core Choice Design space exploration (DSE) reveals that the in-order design space has a narrow performance range of 1 second for webpage loading, whereas the out-of-order space covers a 4 second performance range. This indicates that out-of-order designs can better balance performance with energy, and therefore are better designs for mobile Web applications. Due to space limitations, we do not show the full data. We refer readers to Fig. 4 in the original paper for more details.

To understand the difference behind the in-order versus out-of-order designs, we study the kernel behaviors in Web browsers. There are four important kernels in a Web browser: i.e., *Dom*, *Style*, *Layout*, and *Render*. They contribute to about 80% of the webpage load time and energy consumption. Fig. 1 and Fig. 2 show the Pareto-optimal frontiers of the in-order and out-of-order design space for each kernel. We find that the kernel variance in the in-order designs is more pronounced than in the out-of-order designs. This indicates that as we push toward more performance in the in-order design space, some kernels stop scaling gracefully on the energy-versus-delay curve, and eventually become critical performance bottlenecks. In contrast, the out-of-order cores can cover the variances

*The original article is "WebCore: Architectural Support for Mobile Web Browsing" by Yuhao Zhu and Vijay Janapa Reddi, published in 41st International Symposium on Computer Architecture, June 2014, 541-552. It is also available for download at http://yuhaozhu.com/pubs/isca14.pdf

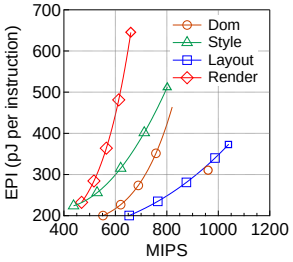

Figure 1: In-order space.

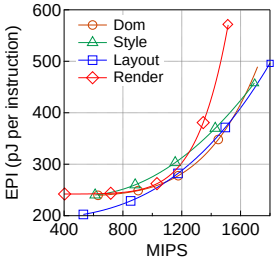

Figure 2: Out-of-order space.

across the different kernels through complex execution logic, and therefore achieve better overall energy efficiency.

Sources of Energy Inefficiency By examining the detailed microarchitecture configurations of the Pareto-optimal designs, we find two major sources of energy inefficiencies in general-purpose processors: instruction delivery and data feeding. Current mobile processors have a small L1 instruction cache that is typically 32 KB in size. However, our DSE results show that Pareto-optimal designs require a 64 KB to 256 KB instruction cache to alleviate the pressure on instruction delivery in mobile Web applications. The pathological front-end behavior mainly stems from the large instruction footprint and the prevalence of the irregular control flow path [9].

High-performance Pareto-optimal designs for mobile Web applications also necessitate a 64 KB data cache, doubling the typical L1 data cache size in current mobile CPUs. The large data cache leads to excessive energy consumption. The need for a large data cache mainly stems from the large working set size on principal data structures (e.g., the DOM tree) during webpage processing. For example, profiling results show that the average data reuse distance for DOM tree accesses is 4 KB.

2.2. Specialization

WebCore incorporates two specialized hardware units to mitigate the instruction delivery and data feeding inefficiencies in the customized out-of-order core designs. The specialized hardware units are closely coupled with the pipeline.

SRU The Style Resolution Unit (SRU) accelerates the Web browser's rendering engine. Specifically, it targets the *Style* kernel, which is one of the most computationally intensive kernels. The SRU is a 32-way data-parallel engine that leverages the *Style* kernel's unique fine-grained parallelism that is unexploited in prior art. This way, SRU aggregates sufficient computations into a few instructions. It therefore increases the arithmetic intensity and reduces the instruction footprint.

BEC The Browser Engine Cache (BEC) is a scratchpad memory designed for hosting principal data structures in a Web browser, particularly the DOM tree and render tree. They have strong locality that can benefit from a small and energy-efficient memory, rather than the large power-hungry traditional caches. For example, our profiling shows that about 90% of the DOM tree nodes are consecutively reused at least three times. The browser engine cache in the WebCore sits before the L1 cache, effectively behaving as an L0 cache.

Programmability In order to abstract the low-level details away from application developers, we provide a set of library APIs in high-level languages. Application developers use the APIs without having to be aware of the existence of the specialized hardware. The software library accesses the SRU and browser engine cache via a small set of ISA extensions.

3. WebCore Results

We focus on the popular WebKit browser engine used in Google Chromium (v30.0) for our studies. We implement the WebCore specializations in Verilog and synthesize our design in 28 nm technology using the Synposys toolchain. We use CACTI v5.3 [1] to estimate the memory structures' overhead. In total, the area overhead is about 0.59 mm^2, insignificant compared to a typical mobile SoC size [2]. The total power overhead is about 80 mW, insignificant compared to the total power consumption for Web browsing (about 1 W). The SRU logic latency is about 16 cycles under 1.6 GHz.

We use Fig. 3 to illustrate the effectiveness of WebCore. We start from an ARM A15-like processor, and progressively apply customization and specialization. Customization alone leads to 22.2% performance improvement and 18.6% energy saving. Specializations gain an additional 9.2% performance improvement and 22.2% energy saving. The accelerated *Style* kernel achieves up to 10X speedup. Overall, WebCore improves performance by 29.2% and saves 47.0% energy.

To further demonstrate the necessity and to justify the effort needed for specializing the WebCore, we also compare WebCore with conventional strategies that simply scale up the microarchitecture structures under the same area overhead of WebCore. Because instruction delivery and data feeding are the two major bottlenecks, as discussed earlier, we compare with designs that improve the I-cache and D-cache sizes.

Fig. 4 compares WebCore with designs that increase the I-cache size by 24 KB (I$), the D-cache size by 24 KB (I$), and both caches by 12 KB (I+D$). The figure normalizes the webpage loading time and energy consumption to WebCore without any specializations. We observe that simply improving the cache sizes in general-purpose cores achieves only negligible performance improvement (<1%) with a slightly higher energy consumption. However, WebCore specializations provide significantly better energy efficiency.

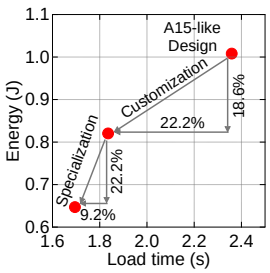

Figure 3: Energy-efficiency improvement over three designs.

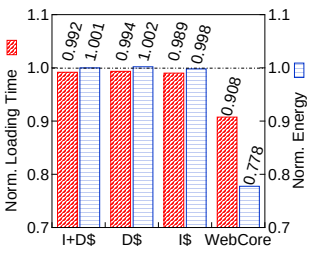

Figure 4: Allocating area for caches versus specializations.

4. Long-Term Impact

Dark Silicon Challenge The demise of graceful Dennard scaling, and consequently the dark silicon phenomenon, have been widely recognized in our community as an urgent challenge threatening the next generation of computing advancements. Improving energy efficiency is now becoming the first-class design consideration. Among all techniques, hardware customization and specialization is deemed an extremely effective approach to improve energy efficiency.

One of the most important questions to address as we enter the dark silicon era is to identify what application domains benefit the most from dedicated hardware resources and design efforts. In retrospect, software computational kernels that made their way into today's hardware have had a strong usage base and long-standing impact. Examples from recent proposals cover important domains such as scientific computing [13], video processing [10], and signal processing [15].

WebCore is the first attempt to comprehensively customize and specialize for the mobile application domain. Specifically, it identifies the Web software infrastructure (HTML, CSS, and JavaScript) as a promising target for hardware customization and specialization in the complex mobile software ecosystem.

The long-term impact of WebCore in supporting Web technologies stems from the Web becoming a universal application development platform. Virtually all mobile applications need Web connectivity. Web technologies provide a platform-independent way to interact with the Internet and manage local computations. They enable application portability to tackle the notorious device fragmentation issue [3]. Additionally, Web technologies are continuously being developed and standardized (e.g., HTML5, CSS3). The burst of development libraries and tools [4, 5] further flourish the entire Web ecosystem.

Principled Approach WebCore is not specific to a particular browser implementation; rather, it targets important computation and communication patterns that are fundamental to any Web browser. Those patterns are generally found across different Web browser engines, such as WebKit and Gecko. In addition, the kernel algorithms and data structures remain largely unchanged across browser versions. For example, the algorithm we study in the *Style* kernel has remained almost identical over the past two years, which includes over 10 versions of Chromium. Therefore, we do not expect software changes to dramatically impact our hardware design.

Balancing Programmability and Specialization Unlike prior research that takes either a fully software approach on general-purpose processors [8, 12] or a fully hardware approach [7], WebCore strikes a balance between the two. On one hand, WebCore retains the flexibility and programmability of a general-purpose core. The general-purpose programmability is essential to support the complex browser software system, and allows fast prototyping of new ideas and implementations. On the other hand, it incorporates modest hardware specializations that create closely coupled datapath and data storage to achieve energy-efficiency improvements.

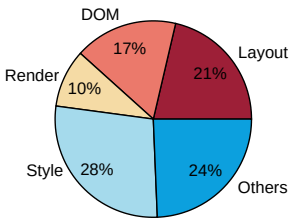

Figure 5: Execution time breakdown of the browser's kernels.

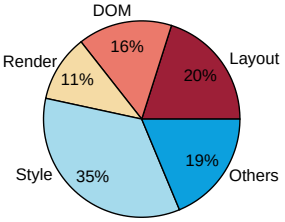

Figure 6: Energy consumption breakdown of the kernels.

WebCore Future Our vision of the WebCore is that it is one of the cores in the multicore SoC, which is already common in today's systems. WebCore executes mobile Web applications, and becomes "dark" when other tasks are active.

In the future, we expect that the WebCore will evolve and incorporate more specialization units for justified computation kernels and communication patterns. For example, Fig. 5 and Fig. 6 show that the *Style* kernel that is currently accelerated by the WebCore takes about one-third of the computation time and energy consumption. Other kernels are left as lucrative targets for acceleration. Recent industry efforts porting important Web functionalities into hardware, such as hardware support for WebRTC in Tegra 4 [6], reinforce this trend.

Citation in 2024 The WebCore evolved over the past decade as a crucial hardware structure to provide today's responsive user engagement and communication. It was based on an astute observation that mobile applications were increasingly built atop of underlying general-purpose Web technologies. By customizing and specializing the processor architecture for Web technologies, the WebCore continues to enable an engaging user experience without sacrificing energy efficiency.

References

[1] CACTI 5.3. http://www.hpl.hp.com/research/cacti
[2] Samsung details Exynos Octa at 60th ISSCC. http://goo.gl/C1NrS
[3] Android Fragmentation Visualized. http://goo.gl/ClV8Wr
[4] JQuery. http://jquery.com/
[5] Emscripten. https://github.com/kripken/emscripten
[6] Hardware Support for WebRTC in Tegra4. http://goo.gl/AiTDxd
[7] V. Bhatt *et al.*, "Sichrome: Mobile web browsing in hardware to save energy," *DaSi: First Dark Silicon Workshop*, 2012.
[8] C. Cascaval *et al.*, "Zoomm: A parallel web browser engine for multicore mobile devices," in *Proc. of PPoPP*, 2013.
[9] A. Gutierrez *et al.*, "Full-system analysis and characterization of interactive smartphone applications," in *Proc. of IISWC*, 2011.
[10] R. Hameed *et al.*, "Understanding sources of inefficiency in general-purpose chips," in *Proc. of ISCA*, 2010.
[11] J. Huang *et al.*, "A Close Examination of Performance and Power Characteristics of 4G LTE Networks," in *Proc. of MobiSys*, 2012.
[12] L. A. Meyerovich and R. Bodik, "Fast and parallel webpage layout," in *Proc. of WWW*, 2010.
[13] W. Qadeer *et al.*, "Convolution engine: Balancing efficiency & flexibility in specialized computing," in *Proc. of ISCA*, 2013.
[14] F. Schlachter, "No moore's law for batteries," in *Proc. of National Academy of Science of the United States of America*, 2013.
[15] M. Woh *et al.*, "Anysp: Anytime anywhere anyway signal processing," in *Proc. of ISCA*, 2009.
[16] Y. Zhu and V. J. Reddi, "High-performance and energy-efficient mobile web browsing on big/little systems," in *Proc. of HPCA*, 2013.