Rethinking Energy-Performance Trade-Off in Mobile Web Page Loading

Duc Hoang Bui, Yunxin Liu, Hyosu Kim, Insik Shin, Feng Zhao
Motivation

- Mobile web browsers: core apps but energy-hungry
- “Direct port” of desktop versions
Motivation

• Mobile web browsers: core apps but energy-hungry
  • “Direct port” of desktop versions
• Battery-powered smartphones: limited energy
  • Necessary to reduce energy consumption
Motivation

• Mobile web browsers: core apps but energy-hungry
  • “Direct port” of desktop versions

• Battery-powered smartphones: limited energy
  • Necessary to reduce energy consumption

• User experience: uncompromisable factor
  • 1-second delay in Bing search engine results in a 2.8% drop in revenue per user [1]
  • “As users migrate to mobile, page load time is perhaps the most important metric we have” [2]

Goal

Reduce energy consumption of web page loading without increasing page load time
Approach

• Analyze architectures and behaviors of popular mobile web browsers
  • Chromium, Firefox, UC Browser on Android
  • Note: Chrome = Chromium + proprietary technologies
Approach

• Analyze architectures and behaviors of popular mobile web browsers
  • Chromium, Firefox, UC Browser on Android
  • Note: Chrome = Chromium + proprietary technologies

• Identify energy inefficiency issues
  • Overhead and redundant computation
  • Underutilization of heterogeneous architectures
Approach

• Analyze architectures and behaviors of popular mobile web browsers
  • *Chromium, Firefox*, UC Browser on Android
  • Note: Chrome = Chromium + proprietary technologies

• Identify energy inefficiency issues
  • Overhead and redundant computation
  • Underutilization of heterogeneous architectures

• Develop energy saving techniques
Approach

• Analyze architectures and behaviors of popular mobile web browsers
  • *Chromium, Firefox, UC Browser on Android*
  • Note: Chrome = Chromium + proprietary technologies

• Identify energy inefficiency issues
  • Overhead and redundant computation
  • Underutilization of heterogeneous architectures

• Develop energy saving techniques

• Evaluate on top 100 U.S. websites
  • Save significant system energy (e.g., 24% on average) while not increasing page load time
Energy inefficiency issues

1. High energy cost of progressive web resource processing
2. Unnecessary high painting rate
3. Underutilization of energy-efficient little cores on big.LITTLE architecture
Energy inefficiency issues (1/3)

• High energy cost of progressive web resource processing
  • For each small data, the whole data rendering pipeline executed
  • E.g., *read* system calls return only 1.3 KB data on average from network
Energy inefficiency issues (1/3)

• High energy cost of progressive web resource processing
  • For each small data, the whole data rendering pipeline executed
  • E.g., *read* system calls return only 1.3 KB data on average from network
• High inter-process communication (IPC) overhead
  • Multi-process architecture browsers

![Chromium web browser architecture diagram](image)
Energy inefficiency issues (2/3)

- Unnecessary high painting rate
  - Visible screen changes can be very small during web page loading
  - Painting: *from models* in memory to *pixels* on screen
Energy inefficiency issues (2/3)

• Unnecessary high painting rate
  • Visible screen changes can be very small during web page loading
  • Painting: *from models* in memory *to pixels* on screen
  • E.g. Loading instagram.com, containing no animation
    • Average 23-32 frames/s (Chromium, Firefox), fixed 60 fps on UC Browser
    • 90% of paints generate zero visible changes on screen (in Chromium)
      • Off-screen paints

![Histogram of screen changes per paint (%)](image)
Energy inefficiency issues (3/3)

• Underutilization of energy-efficient little cores on big.LITTLE architecture
  • Current OS scheduler schedules threads based on load instead of quality of service (QoS)

(a) Energy consumption on Samsung S5 Exynos
Energy inefficiency issues (3/3)

• Underutilization of energy-efficient little cores on big.LITTLE architecture
  • Current OS scheduler schedules threads based on load instead of quality of service (QoS)
  • E.g. Loading instagram.com
    • Chromium: 89% of threads’ time on big cores
    • Firefox: 84% of Gecko rendering engine on big cores

![](image_url)

(a) Energy consumption on Samsung S5 Exynos
(b) Execution time of Chromium threads
Rethink energy-performance trade-off

- Energy consumption: first-class citizen on smartphones

- Reduce redundant computation
  - Adjust processing to the user-perceived content changes

- Utilize energy efficiency on heterogeneous architectures

- Develop 3 energy saving techniques
Network-aware Resource Processing

• Perform batch processing of web resources
  • Reduce overhead on small data sizes

• Trade-off: energy saving vs. delay
  • Large batch size: lower energy but high delay
  • Progressive processing: lower delay but high overhead
Network-aware Resource Processing

• Batch size: adaptive to downloading speed
  • Download speed: light-weight approximation of content changes
  • Increase on fast networks to save more energy
  • Decrease on slow networks to reduce delay

• Batch size determined by a buffer threshold

\[
\text{buffer\_threshold} = \alpha \times \text{network\_goodput}
\]

• \(\alpha\): determines the maximum delay for a chunk of data (e.g., 0.5 sec)

\(\alpha\): maximum delay time in the buffer
Adaptive Content Painting

• Aggregate multiple content paints
  • Reduce unnecessary computation of small-visible-change paints
Adaptive Content Painting

• Aggregate multiple content paints
  • Reduce unnecessary computation of small-visible-change paints

• Trade-off between user experience (UX) and energy
  • Low frame rate: less energy but worse UX
  • High frame rate: smoother UX but higher energy consumption
Adaptive Content Painting

• Aggregate multiple content paints
  • Reduce unnecessary computation of small-visible-change paints

• Trade-off between user experience (UX) and energy
  • Low frame rate: less energy but worse UX
  • High frame rate: smoother UX but higher energy consumption

• \textit{paint\_rate} parameter: maximum content painting rate
  • Dynamically adapt to content changing speed
  • Light-weight approach
    • Increase linearly when content changes fast
    • Decrease to a minimum value when content changes slowly
Application-Assisted Scheduling

• Better utilize little cores on big.LITTLE architecture
  • Leverage internals of browsers for scheduling threads
  • Schedule threads according to QoS
• QoS requirement: frame painting time of browser
Application-Assisted Scheduling

• Better utilize little cores on big.LITTLE architecture
  • Leverage internals of browsers for scheduling threads
  • Schedule threads according to QoS
• QoS requirement: frame painting time of browser
• Dynamic thread-to-core assignment
  • Move threads to big cores: when QoS about to be violated
  • Bring threads back to little cores: when QoS satisfied
Implementation

- Prototype based on Chromium version 38 (16 million lines of code)
- **Buffered resource handler**: Network-aware Resource Processing
- **VSync monitor**: Adaptive Content Painting
- **Thread management module**: Application-Assisted Scheduling
Evaluation

• Experiment setup
  • Emulated testbed: repeatable experimentation
  • Common 3G network condition
    • 2 Mbps download, 1 Mbps upload bandwidth, 120 ms RTT
    • Web Page Replay tool: record and replay pages
• Data set
  • Top 100 websites in the U.S. by Alexa.com in May 2014
• Devices
  • S5-E: Galaxy S5 Exynos (big.LITTLE processor)
  • S5-S: Galaxy S5 Snapdragon (symmetric processor)
• Metric: Page load time (W3C Navigation Timing specification)
• Automation tool
  • Two modules: on smartphone and on PC controlling Monsoon power monitor, time synchronized
  • Each configuration and website tested at least 5 times
Video demo: facebook.com

cps.kaist.ac.kr/eBrowser
Effectiveness of all techniques

- Galaxy S5 Exynos (big.LITTLE processor)
  - 24.4% system energy saving, including LCD screen
  - Page load time decreased by 0.38% (29 ms)
Effectiveness of all techniques

- Galaxy S5 Exynos (big.LITTLE processor)
  - 24.4% system energy saving, including LCD screen
  - Page load time *decreased* by 0.38% (29 ms)

- Galaxy S5 Snapdragon (symmetric processor)
  - 11.7% system energy saving (without Application-Assisted Scheduling technique)
  - Page load time *increased* by only 0.01% (6.7 ms)
Effectiveness of each technique

• Energy saving
  • Application-Assisted Scheduling (AAS): most effective
  • Network-aware Resource Processing (NRP) and Adaptive Content Painting (ACP): similar effectiveness
Effectiveness of each technique

• Energy saving
  • Application-Assisted Scheduling (AAS): most effective
  • Network-aware Resource Processing (NRP) and Adaptive Content Painting (ACP): similar effectiveness

• Page load time increase of individual technique is small
  • Maximum 0.76% average increase (NRP) on Galaxy S5 Snapdragon
User perceived experience

• User study: 18 users, compare our vs. default browsers
  • Test 1: observe loading speed and smoothness of 10 random websites
  • Test 2: do real web browsing for 5 minutes
User perceived experience

• User study: 18 users, compare our vs. default browsers
  • Test 1: observe loading speed and smoothness of 10 random websites
  • Test 2: do real web browsing for 5 minutes

• Results: Minimal difference between our and default browsers
  • All users want to use our revised browser
    • 72% users would always use, 28% users would use when low battery

<table>
<thead>
<tr>
<th>User experience</th>
<th>Page loading speed</th>
<th>smoothness</th>
<th>Page loading speed</th>
<th>Overall speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Better</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Same</td>
<td>-0.02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Worse</td>
<td>-0.12</td>
<td>-0.18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case study: All techniques

• Significant *reduction of power consumption*
• E.g., System power reduction: 5.3 W (Default) vs. 1.4 W (ours)
Case study: AAS technique

- Significant increase of utilization of little cores
  - E.g., 25% (Default) vs. 60% (Application-Assisted Scheduling)
Case study: NRP and ACP techniques

• Significant *reduction of threads’ execution time*
  • E.g., *Chrome_ChildIO* thread execution time reduced by 65% (NRP only)
Evaluations on other environments and browser

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Average system energy saving (%)</th>
<th>Average page load time increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast network (20 Mbps download, 10 Mbps upload, 50 ms RTT)</td>
<td>21.8</td>
<td>0.1</td>
</tr>
<tr>
<td>3G network (3G network interface used)</td>
<td>22.5</td>
<td>0.4</td>
</tr>
<tr>
<td>Web page loading with cached content</td>
<td>19.6</td>
<td>-1.7</td>
</tr>
<tr>
<td>Firefox web browser</td>
<td>10.5</td>
<td>1.7</td>
</tr>
</tbody>
</table>

• Significant system energy saving without page load time increase
• Applicable for other web browsers
Evaluations on other environments and browser

<table>
<thead>
<tr>
<th>Evaluation</th>
<th>Average system energy saving (%)</th>
<th>Average page load time increase (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast network (20 Mbps download, 10 Mbps upload, 50 ms RTT)</td>
<td>21.8</td>
<td>0.1</td>
</tr>
<tr>
<td>3G network (3G network interface used)</td>
<td>22.5</td>
<td>0.4</td>
</tr>
<tr>
<td>Web page loading with cached content</td>
<td>19.6</td>
<td>-1.7</td>
</tr>
<tr>
<td>Firefox web browser</td>
<td>10.5</td>
<td>1.7</td>
</tr>
</tbody>
</table>

- Significant system energy saving without page load time increase
- Applicable for other web browsers
- Speed Index metric increased only slightly (1.8%, on average)

(Above numbers are on Galaxy S5 Exynos big.LITTLE)
Related work

• Energy saving for mobile web browsers
  • Chameleon [MobiSys11]: changes color to save energy on OLED screens
  • Thagarajan et al. [WWW12]: measures energy and provides guidelines (e.g., avoid complex JavaScripts)
  • Zhu et al. [HPCA13]: uses statistical inference models
    • Limitations: Ignored JavaScript and dynamic contents
Related work

• Energy saving for mobile web browsers
  • Chameleon [MobiSys11]: changes color to save energy on OLED screens
  • Thagarajan et al. [WWW12]: measures energy and provides guidelines (e.g., avoid complex JavaScripts)
  • Zhu et al. [HPCA13]: uses statistical inference models
    • Limitations: Ignored JavaScript and dynamic contents

• Our work
  • Deal with trade-offs inside web browsers
    • Others focus on the characteristics of web pages (primitives, colors, network accesses)
  • Orthogonal with other approaches (e.g., changing color)
    • Ours can be integrated with others to further improve energy efficiency
  • Tested on real-world websites and smartphones
Conclusion

• Identify energy inefficiency issues in mobile web browsers
• Propose energy saving techniques
  1. Network-aware Resource Processing
  2. Adaptive Content Painting
  3. Application-Assisted Scheduling
• Implement on popular mobile web browsers (Chromium and Firefox for Android) on commercial smartphones (Samsung Galaxy S5 phones)
• Evaluate on top 100 U.S. websites: save significant system energy while not increasing page load time
  • 24.4% system energy saving while decreasing 0.38% page load time on a big.LITTLE phone
Thank you
Case study: AAS technique

• Significant increase of little cores utilization on big.LITTLE architecture
  • E.g., 25% (Default) vs. 60% (Application-Assisted Scheduling)

• Decrease of load on big cores