Efficient, Parallel Mobile Web Browser

Justin Bonnar, Seth Fowler, James Ide, Chris Jones, Rose Liu, Leo Meyerovich, Doug Kimelman (IBM), Krste Asanovic, and Rastislav Bodik

ParLab
UC Berkeley
The Transition to Handhelds

Power wall: Previous generations reused software of their ancestors. Mobiles will need parallel software.

Soon on mobile: 4-cores x 2-threads x 8-SIMD = 64-way parallelism
Problem: Power Wall for Individual Threads
Handhelds may soon replace laptops

A guy walks into a bar, asks for a cup, and starts his browser.

Let’s see why this “tablet phone” may actually appear soon...
Can handhelds make laptops unnecessary?

**Hypothesis:** laptops become largely unnecessary if handhelds can provide *laptop-quality web browser*

For that, we need

- **network:** 50Mbps
- **display:** at least 1024x768
- **input:** keyboard-like rate

All three are forthcoming …
Input: idea for tablet input for a handheld

**Inspiration:** mimio, a whiteboard recorder (mimio.com)

**How mimio works:**
triangulates in the same way that one measures lightning distance

1. marker simulates a lightning strike: simultaneously emits light and sound signals;
2. capture bar measures sound travel time: yields marker distance to each mic;
3. the two distances determine marker location on the whiteboard; goto step 1
Dasher + picomimio = keyboard-rate input

Dasher: replacement for traditional keyboards

- Input rates up to ~30 words/minute
- Only needs 1 input axis (up/down) to work
  - can be controlled by picomimio, eyes, tilt sensor, ...

See [http://www.inference.phy.cam.ac.uk/dasher/](http://www.inference.phy.cam.ac.uk/dasher/) for more info, online demo
Why Parallelize a Browser?

Dominant application platform
- easy deployment: apps downloaded, JS portable
- productive programming: scripting, layout

... but not on handhelds
- slow: for Slashdot, Laptop: 3s => iPhone: 21s
- native frameworks for: iPhone, Google Android

Parallel browser may need new architecture
- ex: JavaScript relies on “gotos”, is too serial
Run Browser on Server?

On the cloud or on nearby devices

- bandwidth: must support *all* users
- connectivity
- latency
- ... *short-term* solution?
Anatomy of a Browser

Frontend

- page?
  - decompress
    - lex
      - parse + build DOM
        - layout
          - render

Scripting

web servers

Layout

- plugin
  - (decode image, ...)

- script
Anatomy of a Browser

Frontend

- page?
  - decompress
    - lex
      - parse + build DOM

Plugin (decode image, ...)

- layout
  - render

10%

1E8 2008 Stockwell

5%

web servers

Scripting

- script

3%

Layout

9% + 43% 

27%
Parallel algorithms for

- **CSS rule matching**: parallel-map with a tiling optimization
- **Page layout**: break up tree traversal into five parallel ones
- **Lexing**: speculation to break sequential dependencies
- **Parsing**: parallel version of the popular packrat

*work-efficient*: no more work than sequential algo.

**Designed a new scripting language**

- **programmer productivity**: from callbacks to actors to **constraints**
- **performance**: adding structure to detect dependences
Frontend: Lexing

- Plugin (decode image, ...)
- Decompress
- Lex
- Parse + build DOM
- Layout
- Render
- Web servers
- Frontend: Lexing
- Page?
Lexing, from 10,000 feet

**Goal:** given lexical spec and input, find lexemes

\[
\text{STag} ::= <[^>]*> \\
\text{Content} ::= [^<]+ \\
\text{ETag} ::= </[^>]*>
\]

(label each character with its state)
Inherently Sequential?

\[\text{STag} \ ::= \ <[^>]*>\]
\[\text{Content} \ ::= \ [^<]+\]
\[\text{ETag} \ ::= \ </[^>]*>\]
An observation

In lexing, almost regardless of where DFA starts, it converges to a stable, recurring state.

Lexing:  

```
< b > Berkeley! < / b >
```

Parallel scans thus need not scan from all possible states, just one, yielding a work-efficient algorithm.
Our solution (1/2): Partition

- split input into blocks with $k$-character overlap
- scan in parallel; start block from a tolerant state
Our solution (2/2): Speculate

• split input into blocks with $k$-character overlap
• scan in parallel; start block from a *tolerant* state
• check if blocks converge: expected in $k$-overlap
• speculation may fail; if so, block is rescanned
Speedup: Flex vs Cell

today’s page sizes: 5 cores are 4.5x faster than flex

baseline: (sequential) flex on the CELL main CPU
Layout Solving (1/2)

1. plugin (decode image, ...)
2. decompress
3. lex
4. parse + build DOM
5. page?
6. web servers
7. script
8. layout
9. render
**Goal:** Match rules with nodes:

- a rule: `p img { fontsize: 7px}`
- rule-to-path matching:
  - rule is a substring of path
  - ends with same node

<table>
<thead>
<tr>
<th>selectors</th>
<th>p</th>
<th>img</th>
<th>p img</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>height=83%</td>
<td>width=100px float=left</td>
<td>fontsize=7px</td>
</tr>
</tbody>
</table>
Parallelization

• 1000s nodes, 1000s rules
• assign nodes to cores

<table>
<thead>
<tr>
<th>selectors</th>
<th>p</th>
<th>img</th>
<th>p img</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>height=83%</td>
<td>width=100px float=left</td>
<td>fontsize=7px</td>
</tr>
</tbody>
</table>
Tiling for Caches

Problem: all the nodes + selectors might not fit in cache!

<body>
  <p>hello</p>
  <img>
  <b>ok</b> ok ok ok
  <p>world</p>
</body>
Performance: Slashdot

4 cores x 2 thread (2.66 GHz, 256KB L2, 8MB L3)

Speedup

<table>
<thead>
<tr>
<th>Hardware Contexts</th>
<th>opt cilk++</th>
<th>opt g++</th>
<th>naïve cilk++</th>
<th>naïve g++</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>8ms</td>
<td>84ms</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>31ms</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>2ms</td>
<td>31ms</td>
<td>8ms</td>
<td>84ms</td>
</tr>
</tbody>
</table>

Strong Scaling

<table>
<thead>
<tr>
<th>Hardware Contexts</th>
<th>strong (ideal)</th>
<th>opt cilk++</th>
<th>naïve cilk++</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Layout Solving (2/2)

- plugin (decode image, ...)
- decompress
- lex
- parse + build DOM
- script
- layout
- render

web servers
Problem: Layout a Page

<body>
  <p>
    <img src="hello" width="100", float="left" x="0", y="0">
    <b>ok ok ok</b>
  </p>
  <p>
    w=100, fs=12
    x=0, y=0
    w=100, fs=12
  </p>
  <p>
    w=50, float="left"
    w=100, fs=12
    x=0, y=10
    h=40
  </p>
  <p>
    h=10
    h=20
    w=30, fs=12
    x=50, y=10
    h=10
  </p>
</body>
It looks rather sequential..
But not entirely
5 Phases: Each Exhibits Tree Parallelism

- **Phase 1:** font size, temporary width
- **Phase 2:** preferred max & min width
- **Phase 3:** solved width
- **Phase 4:** height, relative x/y position
- **Phase 5:** absolute x/y position
Results: layout (modeled)

Baseline: Cilk++ model on 1 core.

- Eight socket x 4 core AMD Opteron 2356 Barcelona Sun X4600
- Dual socket x 4 core AMD Opteron 2356 Barcelona Sun X2200
- Preproduction 2 socket x 4 core x 2 thread Intel Xeon Nehalem

Modeled Speedup w/ Cilk++

Average Speedup vs. # Hardware Threads
Scripting

page?

decompress

lex

parse + build DOM

plugin (decode image, ...)

layout

render

script

web servers
Why parallelize scripting (example)

Example: animate between different views
- each transition: recolor, resize each state or county
- animation rate 30fps \(\Rightarrow\) 33ms for 1000s of nodes
The browser programming model

- Nonpreemptive event model
- Handlers respond to events
- Handlers execute atomically
  - document changes cause relayout
  - style changes cause relayout

- To parallelize, must understand how the document is shared
  - document-carried dependencies:
    handler A: california.x = 100;
    handler B: varz = california.x;
  - layout-carried dependencies:
    handler A: america.w = 200%;
    layout: california.w = 200%;
    handler B: varz = california.w;
Concurrent bugs

1. GUI animations and interactions
   - several animations modifying an object simultaneously
2. Server interactions
   - responses to requests may be delayed, reordered
3. Eager script loading
   - executing a script on a document before done loading
“Gotos” in JavaScript

```
<div id="box" style="position:absolute; background: yellow;"> My box </div>

<script>
document.addEventListener('mousemove', function (e) {
    var left = e.pageX;
    var top = e.pageY;
    setTimeout(function() {
        document.getElementById("box").style.top = top;
        document.getElementById("box").style.left = left;
    }, 500);
}, false);
</script>
```
Dataflow language (version 1)

Program structure is clearer when data and control is explicit
- in dataflow version: **changing mouse coordinates are streams**
- coordinate streams adjust box position after they are delayed
- **structured names** of document element allow analysis

```
mouse
```

```
delay 500
```

```
body.column.div
```

```
delay 500
```

```
delay 500
```

1. Developed work-efficient algorithms
   - Rule matching: parallel-map with a tiling optimization
   - Layout: break up tree traversal into five parallel ones
   - Lexing: speculation to break sequential dependencies

2. Reexamining the scripting programming model
   - programmer productivity: from callbacks to toactors
     • influenced by Flapjax, Ptolemy, Max/MSP, LabVIEW
   - performance: adding structure to detect dependences
     • current browsers: JIT compilation, font vectorization, task parallelism eg for image rendering – all these are useful, too.