Parallel Webpage Layout

Leo Meyerovich, Chan Siu Man, Chan Siu On, Heidi Pan
Krste Asanovic, Rastislav Bodik
and many others from the UPCRC Berkeley project

UC Berkeley
Par Lab Research Overview

- **Applications**
  - Personal Health
  - Image Retrieval
  - Hearing, Music
  - Speech
  - Parallel Browser

- **Productivity Layer**
  - Motifs

- **Efficiency Layer**
  - Composition & Coordination Language (C&CL)
  - C&CL Compiler/Interpreter
  - Parallel Libraries
  - Parallel Frameworks
  - Efficiency Languages
    - Sketching
    - Autotuners
  - Legacy Code
    - Schedulers
    - Communication & Synch. Primitives
  - Efficiency Language Compilers

- **Diagnosing Power/Performance**
  - Legacy OS
  - OS Libraries & Services
    - Hypervisor
  - Multicore/GPGPU
  - RAMP Manycore

- **Correctness**
  - Static Verification
    - Type Systems
  - Directed Testing
  - Dynamic Checking
  - Debugging with Replay
Parallel Web Browser

Why the browser?
- an important application platform
- browser wars again: competing on performance (latency)
- how important? handheld pageload is tens of CPU seconds

Why a parallel browser?
- soon in your phone? 4 cores x 2 threads x 8-wide SIMD = 64
- parallelism is more energy efficient

Technical challenge
- Parallelize the browser to run with 100-way parallelism
This Talk: Parallelize **Single** Page Layout

- Page layout (HTML+CSS) is the LaTeX of the Web
  - latex takes seconds to format a document
  - but pageload should be 20-100ms
  - pageload is a bottleneck: 51% of CPU time on IE8
- Page layout is a challenging “desktop” application
  - not parallelized before
  - specifications: often ambiguous and **sequential**
  - low-latency: problems are short-running
  - less understood motif: tree computation
- Knuth: “Multiprocessors are no help to **T\text{E}X**”
Our Contributions

1. Analyzed browser performance
   - layout is a bottleneck; we identified its critical motifs
2. Distilled essential CSS and wrote a declarative spec for it
   - crucial step for exposing parallelism hidden by today’s spec
3. Developed first parallel page layout algorithms
   (1) matching: task parallel, 20x speedup, strongly scales to 16
   (2) solving: task parallel, 4x speedup, strongly scales to 3 cores
4. Future steps – components and algorithms
Overall Page Layout Problem

What the browser does

```
<body>
  hello
  <img src="http:...">
  <p><b>world</b></p>
  ok ok ok ok ok
</body>
```

```

p  { width: 100% }
img { width: 100px;
    float: left }
p img { width: 10pt }
```

HTML

CSS styling rules

Our page layout subproblem

**Input:** document tree + CSS rules

**Output:** sizes and positions of tree nodes

**Steps:** determine styling rules; solve constraints

```
25%  25%
```

```
<table>
<thead>
<tr>
<th>p</th>
<th>width=100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>width=100px float=left</td>
</tr>
<tr>
<td>p img</td>
<td>width=10px</td>
</tr>
</tbody>
</table>
```

What the browser does

<table>
<thead>
<tr>
<th>p</th>
<th>width=100%</th>
</tr>
</thead>
<tbody>
<tr>
<td>img</td>
<td>width=100px float=left</td>
</tr>
<tr>
<td>p img</td>
<td>width=10px</td>
</tr>
</tbody>
</table>

XX=?, y=?
The layout spec is confusing

Example of spec:

– “In general, the left edge of a line box touches the left edge of its containing block... However, floating boxes may come between [them].”

Hard to implement correctly, even sequentially.
Flow: sequential layout in today’s browsers

simplest way to implement the spec seems to be to (mostly) flow the elements sequentially in order
Flow is guided by a cursor

Cursor $\Delta$ points to where next element goes
Flow’s dependences

Constraints not specified if equality (e.g., inherited) or intrinsic (e.g., default image size or aspect ratio)
Dependencies prevent parallelism

<body>

</body>

Hello

World

ok ok ok

fs, Δ, w

fs, Δ, w

fs, Δ, w

fs, Δ, w

fs, Δ, w

fs, Δ, w

fs, Δ, w
Enable parallelism by doing part of work.
Parallel Layout Solving: Five Phases

Extensive analysis led us to five phases
These enable parallelism

1. font size, tentative widths
2. preferred widths: max, min
3. final widths: break cycles by over-specifying CSS
4. heights, relative x/y positions
5. absolute x/y positions
Each Phase Exhibits Tree Parallelism

Phase 1: font size, temporary width
Phase 2: preferred max & min width
Phase 3: width
Phase 4: height, relative x/y position
Phase 5: absolute x/y position
Parallel Layout: Speculative Evaluation

• Did not break dependencies for floats
  – might stick out of paragraphs
Parallel Layout: Speculative Evaluation

• Did not break dependencies for floats
  – might stick out of paragraphs

• Speculate: assume no floats

• Check

• Patch up as needed
Parallel Layout: Speculative Evaluation

- Did not break dependencies for floats
  - might stick out of paragraph

- Speculate: assume no floats

- Check

- Patch up as needed
  - floats rare
  - We believe overflow is minimal
Berkeley Style Sheet Layout Language

- Can compile essential CSS into it
- Refactored CSS to separate features
- Simplifies: correctness, parallelization, use

\[
V \rightarrow \\
\{ y_{acc} \leftarrow 0 \} \\
(\{ \$1.x = 0; \\
\quad \$1.w = \phi(\$0.w, \$1.tempw, \$1.m, \$1.p) \} \\
\quad (V \mid H) \\
\quad \{ \$1.y = y_{acc}; \\
\quad \quad y_{acc} \leftarrow y_{acc} + \$1.h \}^* \\
\quad \{ \$0.h = \$0.temph \bowtie y_{acc} \})
\]
Analysis

- Model: sequential speed $\sim$ Firefox speed
- Cilk++: 4x speedup, scales to 3 cores
- Need to SIMDize leaves

![Diagram](image)

**Modeled Speedup w/Cilk++**

- 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
Rule Matching: Problem Statement

- **Matching**
  - Tag path (img: <body> <p> <img> )
  - Rule Selectors
  - For each tag path: which selectors are ~substrings?

- **Rule resolution**
  - Prioritize properties by rule order: lower overrides

<table>
<thead>
<tr>
<th>selectors</th>
<th>p</th>
<th>img</th>
<th>p img</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>width=100%</td>
<td>width=100px</td>
<td>width=100px</td>
</tr>
<tr>
<td></td>
<td></td>
<td>float=left</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>width=10px</td>
<td></td>
</tr>
</tbody>
</table>
Rule Matching: Parallelization

- ~600 nodes, 1000s rules
- Assign nodes to cores
  - load balancing: random assignment
- SIMDizable?

<table>
<thead>
<tr>
<th>selectors</th>
<th>p</th>
<th>img</th>
<th>p img</th>
</tr>
</thead>
<tbody>
<tr>
<td>properties</td>
<td>width=100%</td>
<td>width=100px</td>
<td>width=100px</td>
</tr>
<tr>
<td></td>
<td>float=left</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analysis

• Results
  – perfect scaling: up to 10 cores
  – 20x speedup on 32 cores
  – ... but with python
    • interp. overhead (seq.)
    • procs., not threads

• Future
  – C++ implementation
  – SIMD rule matching
Takeaways

• Artifacts
  – BSS/CSS specification & dependency decomposition
  – 4x solving speedup (untuned), 20x matching (in python)

• Lessons
  – 4x << 100x → SIMDize low-level libraries (e.g., fonts)
  – motifs: low latency tree ops, vectors, pixel blending
  – attribute grammars helped

• Next steps
  – tune tasks, SIMD kernels, bigger scope of model
  – implications for concurrent scripts using layout?
(questions?)