Parallel Layout
Automatic Generation and Optimization
Why Generate a Layout Engine

Many and Growing Layout Languages

HTML, CSS, SMIL, XUL, Ext, jQuery, YUI, OpenOffice, JavaFX, Swing, Flex, Adam & Eve, Thermo, XAML/WPF, Word, WinForms, Qt, LaTeX, music, iPhone, Android, WAP, MathML, 3 competing CSS grid proposals

A lot of code

Firefox layout engine: 346111 lines
Approach

HTML

CSS

parser

selector matcher

cascade

CSS grammar specification

generator

sequential layout engine

scene graph

renderer

tree decorated with style constraints
HBox: Example of Specifying Layout

**HBox** with two child nodes

wInput = ShrinkToFit

\[
w := \text{case } w\text{Input}: \\
\quad n \text{ px: } n \\
\quad \text{shrinkToFit: } \text{sum (children.w)}
\]
HBox Traversal Functions

def pass0():
    child1.y = y
    child2.y = y

def pass1():
    cursor = child1.w
    w = if (wInput is shrink):
        child1.w + child2.w
    else:
        wInput
    h = max (child1.h, child2.h)

def pass2():
    child1.x = x;
    child2.x = x + cursor;
HBox Traversal Functions

def pass0():
    child1.y = y
    child2.y = y

def pass1():
    cursor = child1.w
    w = if (wInput is shrink):
        child1.w + child2.w
    else:
        wInput
    h = max (child1.h, child2.h)

def pass2():
    child1.x = x;
    child2.x = x + cursor;
Scheduling Traversals
Leveraging Generation: % Width

\[ w := \text{case } \text{wInput:} \]
\[ \text{n px: } n \]
\[ \text{n %: parentWidth * n\%} \]
\[ \text{shrinkToFit: sum(children.w)} \]
def pass0(): ...  
def pass1():  
    cursor = child1.w  
w = calculateWidth(wInput,  
    child1.w, child2.w)  
h = max (child1.h, child2.h)

def pass2():  
    child1.x = x  
    child2.x = x + cursor

def pass3():  
    cursor = child1.w  
def pass4():  
    child1.x = x  
    child2.x = x + cursor
Other Advantages

- **Correctness Wins**
  - Finds spec inconsistencies
  - Can visually debug spec
- **Performance Wins**
  - Optimal scheduling
  - Extract parallelism
Fast Tree Library

HTML
CSS
parser
selector matcher
cascade
tree decorated with style constraints

layout engine
fast tree library
tree traversals
scene graph
renderer

Compile Time
CSS grammar specification
ALE synthesizer

Fast Tree Library
Overview of Tree Eval Strategies

Sequential

Multicore

SIMD ("SIMTask")

core 1  core 2
2. Optimizing memory
2. Optimizing memory

Order within block: bfs, dfs

Traversal order

nodes per block

speedup

△ dfs

□ bfs
2. Optimizing memory

**Graph:**
- **x-axis:** Nodes per block
- **y-axis:** Speedup
- **Legend:**
  - ▲ dfs, rel pointers
  - △ dfs
  - □ bfs

**Observations:**
- The order within block is bfs, dfs.
- Pointer representation:
  - `leftChild = 0x00ffaa00`
  - `leftchid = 1200`
- How much compression
  - Hardware

**Note:**
- The graph shows the performance speedup with varying nodes per block for different algorithms.
2. Optimizing memory

- Packing
- Coallocation / Lazy defaults
- Structure splitting / Phasing

Order within block: bfs, dfs

Pointer representation:
leftChild = 0x00ffaa00,
leftChild = 1200
Challenge Problem for Task Parallelism?

- Dynamic task allocation?
- Runtime queues?
- Locality across traversals?

**TBB tree traversal on dual-core Atom 330**

<table>
<thead>
<tr>
<th>Threads</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.9</td>
</tr>
<tr>
<td>2</td>
<td>0.8</td>
</tr>
<tr>
<td>3</td>
<td>0.7</td>
</tr>
<tr>
<td>4</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Different TBB algorithms:
- base.h
- tbbcont.h
- tbbgraph.h
- tbb.h
- tbbopt.h
Semi-Static Work Stealing

1. **Before parallel traversal:** approximate work stealing schedule
2. **Traversal:** reuse schedule tuned locking scheme

Locality across passes!
Opteron Speedup
(2 sockets x 4 cores); 1,000 nodes

Strong scaling: small workload (1ms each)

10,000 nodes
1,000 nodes; repeat each 10x
SIMD Task Evaluation (MSR)

Microbenchmarks: 2-7x speedup

- 15-20
- 10-15
- 5-10
- 0-5

see poster for challenges and opportunities
Demo

- Parallel layout
Status and Future Work

- HTML
  - CSS
  - **multicore** parser
    - tree
    - style template
    - **multicore** selector matcher
    - **multicore** cascade
      - tree decorated with style constraints

- **Fast Tree Library**
  - layout engine
    - tree traversals
  - Fast Tree Library
    - scene graph
    - **OpenGL** Qt Renderer

- Widget definition
- MUD language
- Grammar specification
- ALE synthesizer incrementalizer