# **Parallelizing the Browser: Synthesis and Optimization of Parallel Tree Traversals**

by

Leo A. Meyerovich

A dissertation submitted in partial satisfaction of the
requirements for the degree of
Doctor of Philosophy

in

Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Fall 2013

The dissertation of Leo A. Meyerovich, titled Parallelizing the Browser: Synthesis and Optimization of Parallel Tree Traversals, is approved:

|  |  |  |  |
|---|---|---|---|
| Chair | ________________________ | Date | ____________ |
|  | ________________________ | Date | ____________ |
|  | ________________________ | Date | ____________ |
|  | ________________________ | Date | ____________ |

University of California, Berkeley

**Parallelizing the Browser: Synthesis and Optimization of Parallel Tree Traversals**

## Abstract

Parallelizing the Browser: Synthesis and Optimization of Parallel Tree Traversals

by

Leo A. Meyerovich

Doctor of Philosophy in Electrical Engineering and Computer Science

University of California, Berkeley

Professor Rastislav Bodik, Chair

From low-power phones to speed-hungry data visualizations, web browsers need a performance boost. Parallelization is an attractive opportunity because commodity client devices already feature multicore, subword-SIMD, and GPU hardware. However, a typical webpage will not strongly benefit from modern hardware because browsers were only designed for sequential execution. We therefore need to redesign browsers to be parallel. This thesis focuses on a browser component that we found to be particularly challenging to implement: the layout engine.

We address layout engine implementation by identifying its surprising connection with attribute grammars and then solving key ensuing challenges:

1. We show how layout engines, both for documents and data visualization, can often be functionally specified in our extended form of attribute grammars.

2. We introduce a synthesizer that automatically schedules an attribute grammar as a composition of parallel tree traversals. Notably, our synthesizer is fast, simple to extend, and finds schedules that assist aggressive code generation.

3. We make editing parallel code safe by introducing a simple programming construct for partial behavioral specification: schedule sketching.

4. We optimize tree traversals for SIMD, MIMD, and GPU architectures at tree load time through novel optimizations for data representation and task scheduling.

Put together, we generated a parallel CSS document layout engine that can mostly render complex sites such as Wikipedia. Furthermore, we scripted data visualizations that support interacting with over 100,000 data points in real time.

To You

Hey you! out there in the cold Getting lonely, getting old, can you feel me Hey you! Standing in the aisles With itchy feet and fading smiles, can you feel me Hey you! don't help them to bury the live Don't give in without a fight.

# Contents

# List of Figures

# List of Tables

## Acknowledgments

I want to thank my advisor for advising me.

# Chapter 1

# Introduction

## 1.1 application area 1: low-power browsers

## 1.2 application area 2: big data visualization

## 1.3 application area 3: layout engineering

## 1.4 generality: parallel algorithms and programming

# Chapter 2

# Modernized Attribute Grammars

## 2.1 Motivation

## 2.2 The HBox Language

## 2.3 Classical Attribute Grammar

## 2.4 Interfaces

## 2.5 Traits

## 2.6 Loops

## 2.7 Embedding Foreign Functions

# Chapter 3

# Parallel Programming as Partial Behavioral Specification: Schedule Sketching

## 3.1 Motivation

## 3.2 Target Language of Visit Schedules

## 3.3 Input Language of Visit Schedules

## 3.4 Examples

### Guiding Parallelization

### Preventing Serialization

### Designing Algorithms

# Chapter 4

# Parallel Schedule Synthesis

## 4.1 Modular Algorithm Design

## 4.2 Optimized Algorithm: Finding One Schedule

## 4.3 Optimized Algorithm: Autotuning Over Many Schedules

## 4.4 Complexity Analysis

# Chapter 5

# MIMD and SIMDTree Traversals

## 5.1 MIMD: Semi-static work stealing

#### Scheduling

#### Data representation

#### Evaluation

## 5.2 GPU: Staying on-device

#### Level-synchronous breadth-first traversal

#### Rendering

#### Evaluation

## 5.3 Sub-word SIMD: Clustering

#### The problem

#### Instruction clustering

#### Data clustering

#### Evaluation

Limit study and benchmarks

# Chapter 6

# Case Studies

## Rendering

Dynamic (HTML5) and static (WebGL)

## Radial

Sunburst

## Charts

GE

## Animation and Interaction

Treemap

## Flow-based Layout

CSS

## Grid-based Layout

Tables

# Chapter 7

# Conclusion