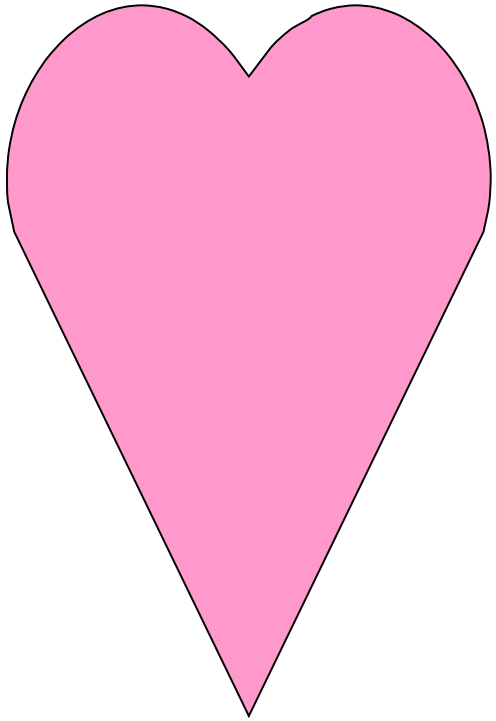
A metrics-based evaluation of SSA

Jeremy Singer
University of Manchester, UK

we  SSA

Happy birthday, SSA

21st birthday?

An Efficient Method of Computing Static Single Assignment Form

Ron Cytron*
Jeanne Ferrante*
Barry K. Rosen*
Mark N. Wegman*
F. Kenneth Zadeck†

Introduction

optimizing compilers, data structure choices directly fluence the power and efficiency of practical program timization. A poor choice of data structure can in- bit optimization or slow compilation to the point where lvanced optimization features become undesirable. Re-

functions instead, SSA form leads to simpler formulations of works like [CLZ86, WZ85] that are based on the precursor.

Property 1 has been exploited by a constant propagation algorithm that deletes branches to code proven unexecutable at compile-time [WZ88]. Without SSA form, data flow information might have to be recomputed each time branches are deleted, rendering the algorithm excessively

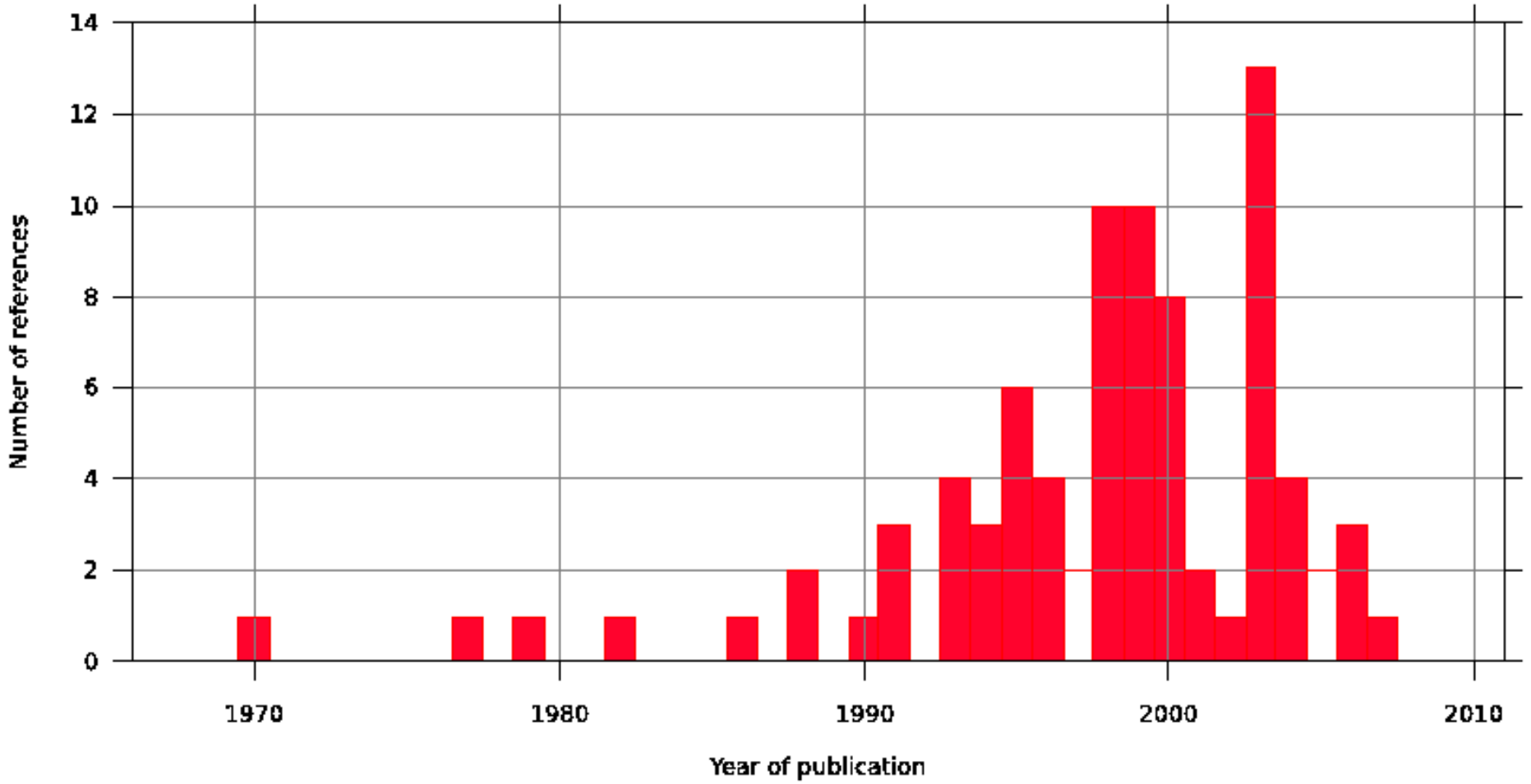
18th birthday?

Efficiently Computing Static Single Assignment Form and the Control Dependence Graph

RON CYTRON, JEANNE FERRANTE, BARRY K. ROSEN, and MARK N. WEGMAN
IBM Research Division
and
F. KENNETH ZADECK
Brown University

In optimizing compilers, data structure choices directly influence the power and efficiency of practical program optimization. A poor choice of data structure can inhibit optimization or slow compilation to the point that advanced optimization features become undesirable. Recently, static single assignment form and the control dependence graph have been proposed to represent data flow and control flow properties of programs. Each of these previously unrelated techniques lends efficiency and power to a useful class of program optimizations. Although both of these structures are attractive, the difficulty of their construction and their potential size have discouraged their use. We present new algorithms that efficiently compute these data structures for arbitrary control flow graphs. The algorithms use *dominance frontiers*, a new concept that may have other applications. We also give analytical and experimental evidence that all of these

SSA is popular



Usual answers

- Simplification of live ranges / du-chains
- Explicitly encode control and data flow in variable naming scheme
- More efficient / more effective analysis
- Easier to understand and write code for handling it
- Overhead of increased vocab, pseudo-assignment fns

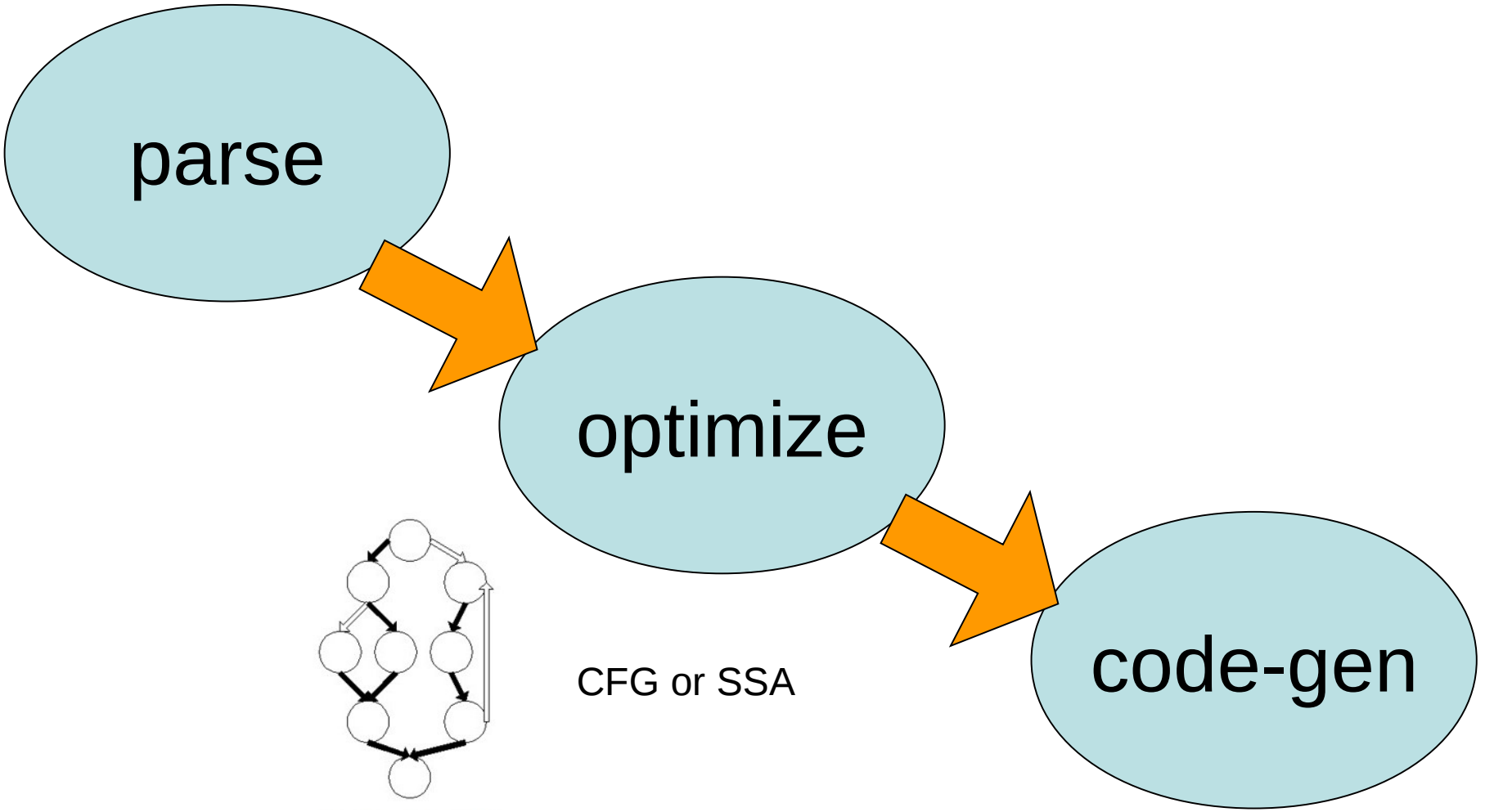
All qualitative answers

- Any quantitative answers always tied to particular
  - System
  - Compiler
  - Analysis
  - Optimization

My aim

- A platform-independent quantitative evaluation of SSA
- How?

Software metrics


parse
optimize
code-gen
CFG or SSA

Which metrics?

- McCabe's cyclomatic complexity
  - CFG and SSA have same control flow
- Object-oriented metrics (coupling etc)
  - Generally too high-level for procedural IR
- Simple data-flow based metrics
  - Identify difference in variable naming
  - Identify difference in live ranges

Halstead's metrics

$n1$—the number of distinct operators

$n2$—the number of distinct operands

$N1$—the total number of operators

$N2$—the total number of operands

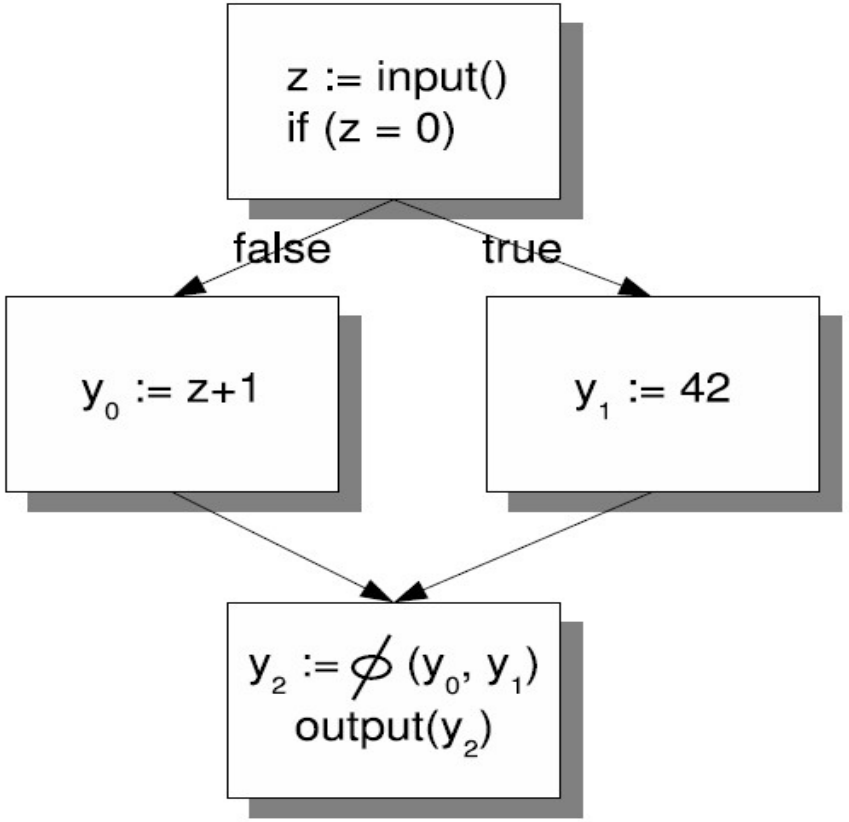
Halstead's metrics

Measure	Formula
Program length	$N = N1 + N2$
Program vocabulary	$n = n1 + n2$
Volume	$V = N * (\log_2 n)$
Difficulty	$D = (n1/2) * (N2/n2)$
Effort	$E = D * V$

Information Flow Complexity

$$\mathrm{IFC} = \mathrm{length} * (\mathrm{fanIn} * \mathrm{fanOut})^2$$

Information Flow Complexity



$$f1() = \textbf{let } z \leftarrow \text{input}() \textbf{ in}$$
$$\textbf{if } (z = 0)\ f2(z)$$
$$\textbf{else } f3()$$

$$f2(z) = \textbf{let } y_0 \leftarrow z + 1 \textbf{ in}$$
$$f4(y_0)$$

$$f3() = \textbf{let } y_1 \leftarrow 42 \textbf{ in}$$
$$f4(y_1)$$

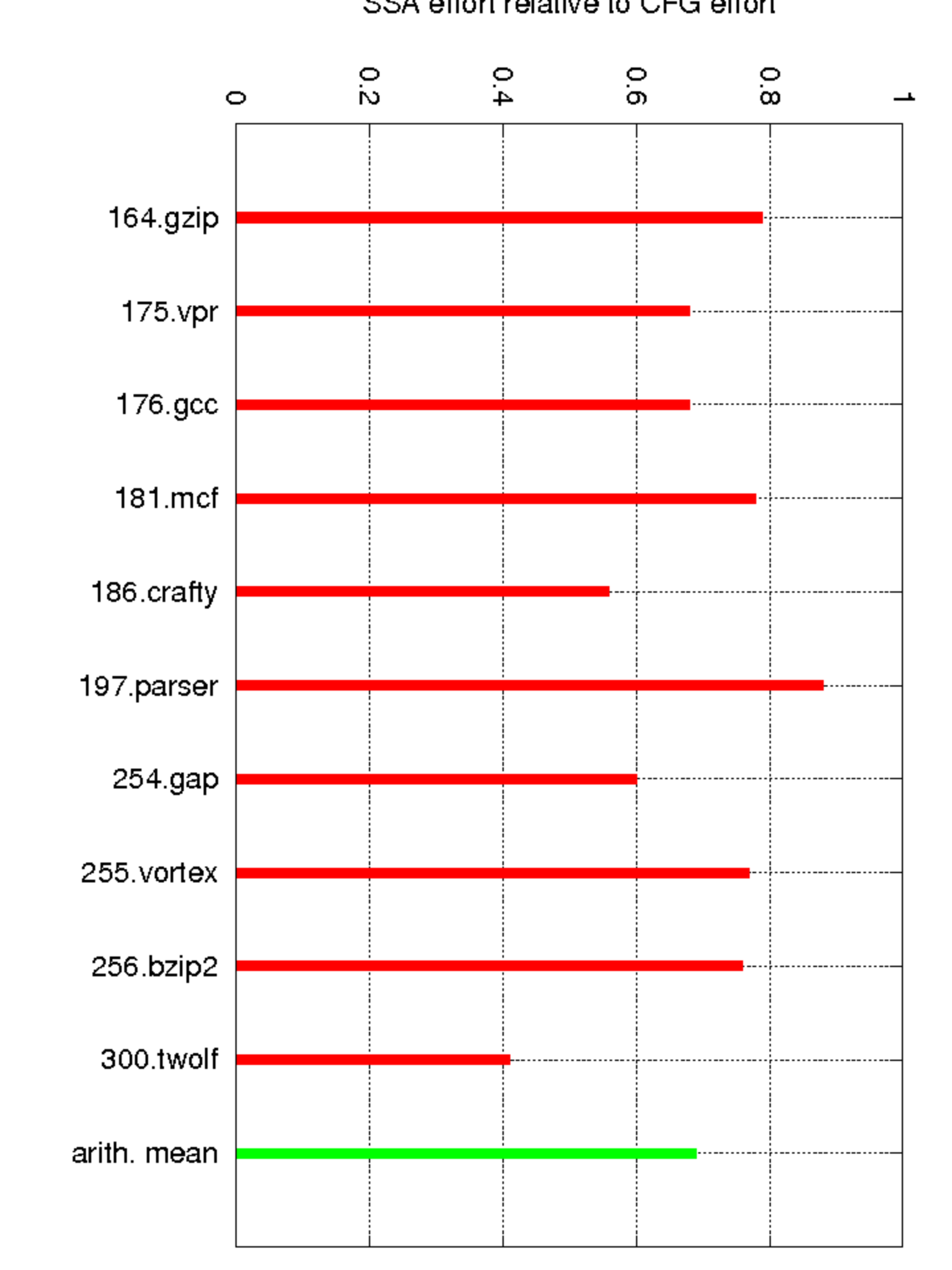
$$f4(y_2) = \text{output}(y_2)$$

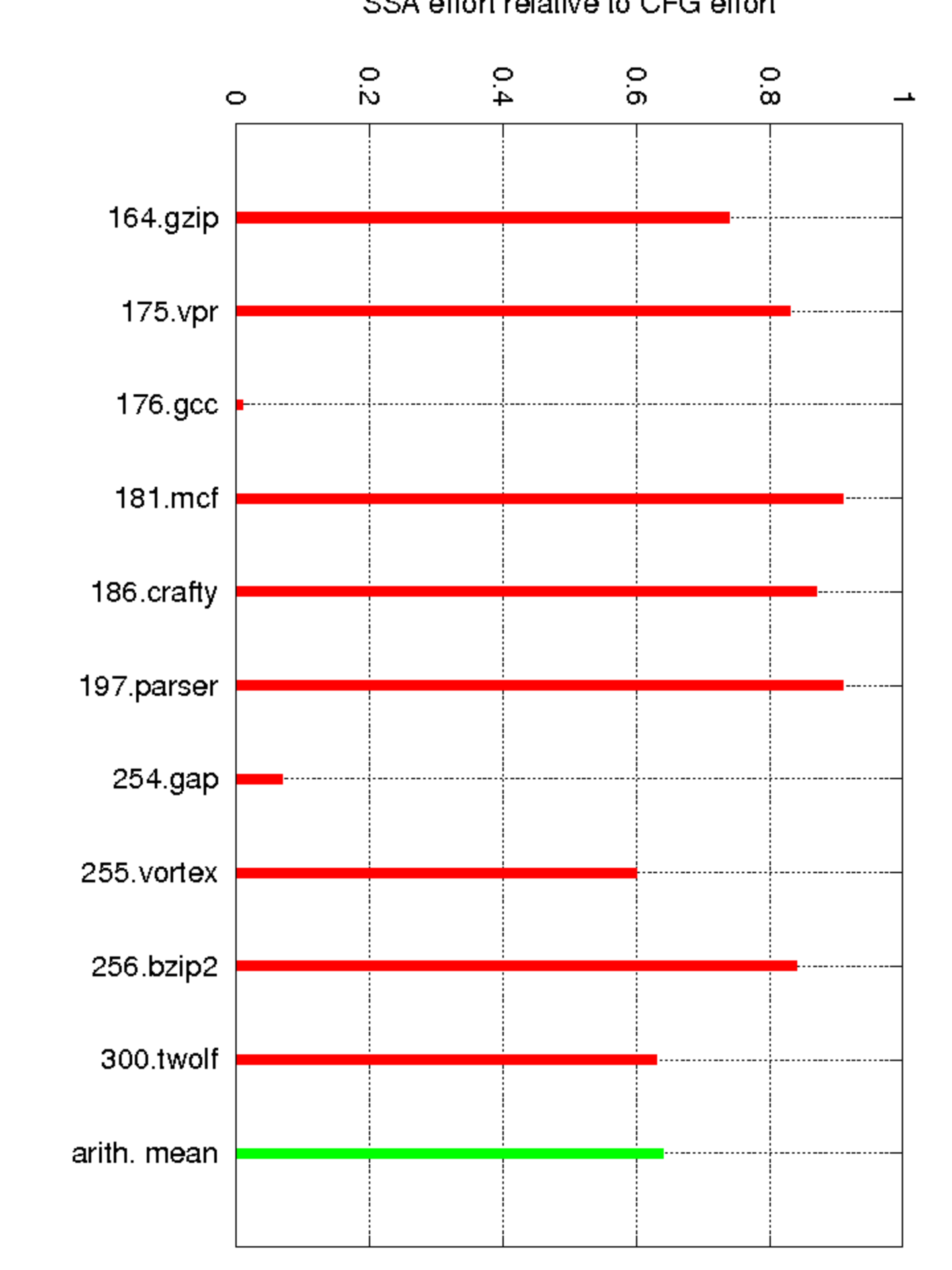
Evaluation

- Compute metrics score for each procedure in SPEC CINT 2000 benchmarks
- Compiled to CFG and SSA with the gcc compiler
- Examine change in scores over CFG to SSA transformation

SSA reduces complexity

- According to both Halstead and IFC metrics


SSA effort relative to CFG effort
0
0.2
0.4
0.6
0.8
1
164.gzip
175.vpr
176.gcc
181.mcf
186.crafty
197.parser
254.gap
255.vortex
256.bzip2
300.twolf
arith. mean


SSA effort relative to CFG effort
0
0.2
0.4
0.6
0.8
1
164.gzip
175.vpr
176.gcc
181.mcf
186.crafty
197.parser
254.gap
255.vortex
256.bzip2
300.twolf
arith. mean

Revisit earlier qualitative reasons

- Halstead: although vocab increases, difficulty decrease is greater.

- IFC: aggressive live range splitting reduces scope of variables.

Paper draft

- Joint work with Martin Ward, Christos Tjortjis
- http://www.cs.man.ac.uk/~jsinger
- jsinger@cs.man.ac.uk

Personal SSA interests

- Quantitative evaluations of SSA
- SSA extensions, other renaming schemes
  - Taxonomy of SSA extensions
  - Taxonomy of construction algorithms