Accurate Static Branch Prediction by Value Range Propagation

Jason Patterson
(Queensland University of Technology, Australia)

Static Branch Prediction

Static branch prediction is very important...

- global instruction scheduling
- code layout (branching & I-cache optimizations)
- very global register allocation
- to guide the application of optimizations (coagulation)
- other high-level optimizations

Branches are surprisingly predictable in nature.

Other Approaches

Execution profiling...

- extremely accurate
- ***too inconvenient*** for all but the most performance-aware programmers

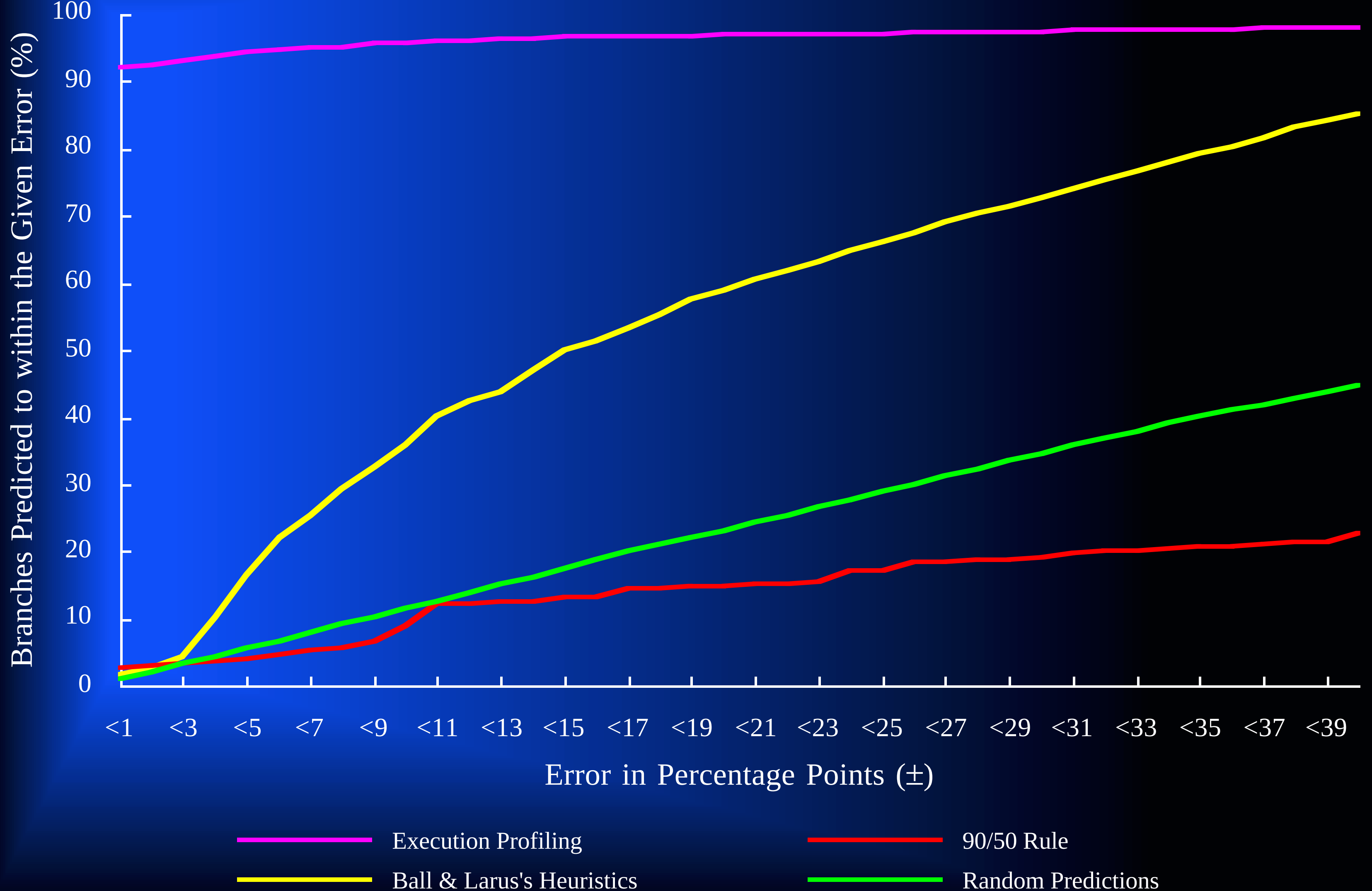
Heuristics based on nesting and coding styles...

- hit-and-miss
- simple heuristics are very inaccurate
- ***sophisticated heuristics are only mediocre***
- heuristics are really a bit of a hack

Programmer supplied hints...

- inconvenient and potentially inaccurate
- indicates a lack of suitable compiler technology (like “register” in C and “inline” in C++)

Other Approaches (SPECfp92 Unweighted)



Value Range Propagation

The basic idea...

- determine the ***weighted range of values*** each expression can have

- use these weighted value ranges to predict the ***probabilities*** of taking the conditional branches

- ***fallback to heuristics*** when the value range being branched on is unknown

Value Range Propagation

The algorithm...

- uses the same two-worklist algorithm as constant propagation with SSA form
- propagates ***value ranges*** rather than constants
- expression and ø-function evaluation are harder
- ***loop-carried expressions*** are handled specially
- associates a ***probability*** with each branch

This analysis is fast enough to be viable.

Range Representation

The representation must...

- handle the ***common cases***
  (constants, dense ranges, arithmetic sequences)
- handle both numeric and ***symbolic*** ranges
- be very ***efficient*** (fast and compact)

A good representation is a set of up to 4 ranges, where each range has...

- a probability
- a lower bound
- an upper bound
- a stride (arithmetic step size)

and each “value” is *SSAvariable op Constant*

Range Operations

{ 0.7[32:256:1], 0.3[3:21:3] }

\+ { 0.6[16:100:4], 0.4[8:8:0] }

= { 0.42[48:356:1], 0.28[40:264:1]
0.18[19:121:1], 0.12[11:29:3] }

Key: { P[L:U:S], ... }

- straightforward but tedious
- efficiency very important

Loop-Carried Expressions

- must be detected and handled specially to avoid “executing” the loops (which would be **very slow**)

- most are easy to detect during propagation

- most can be handled by simply matching the expression’s **derivation** to common looping scenarios...

  eg: new value = old value + set of possible incs
  assert (new value between certain bounds)

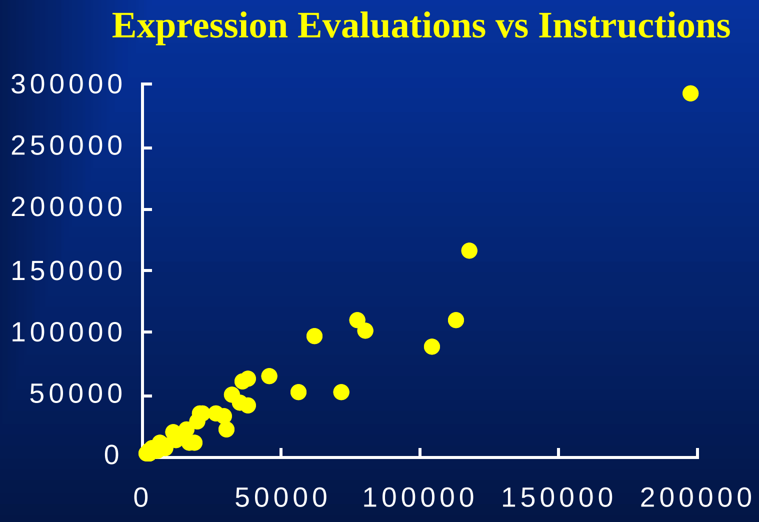
- rare situations can be handled by simply letting the propagation algorithm “execute” the loop

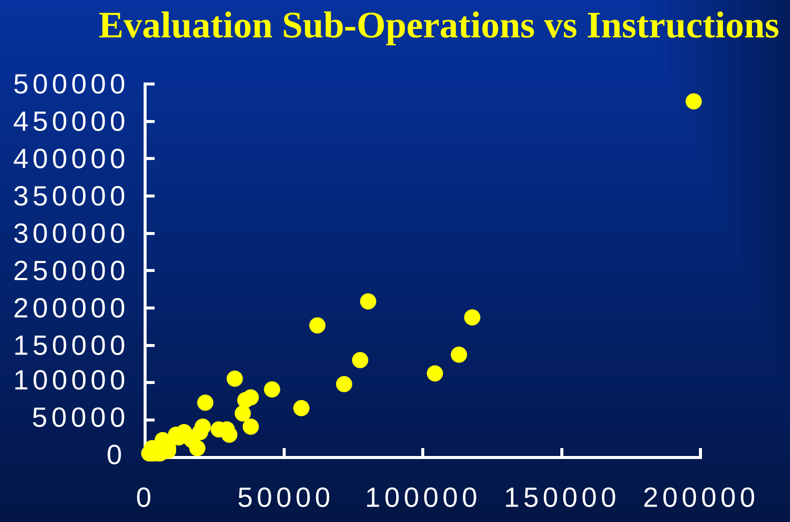
Algorithm Efficiency

Slower than constant propagation...

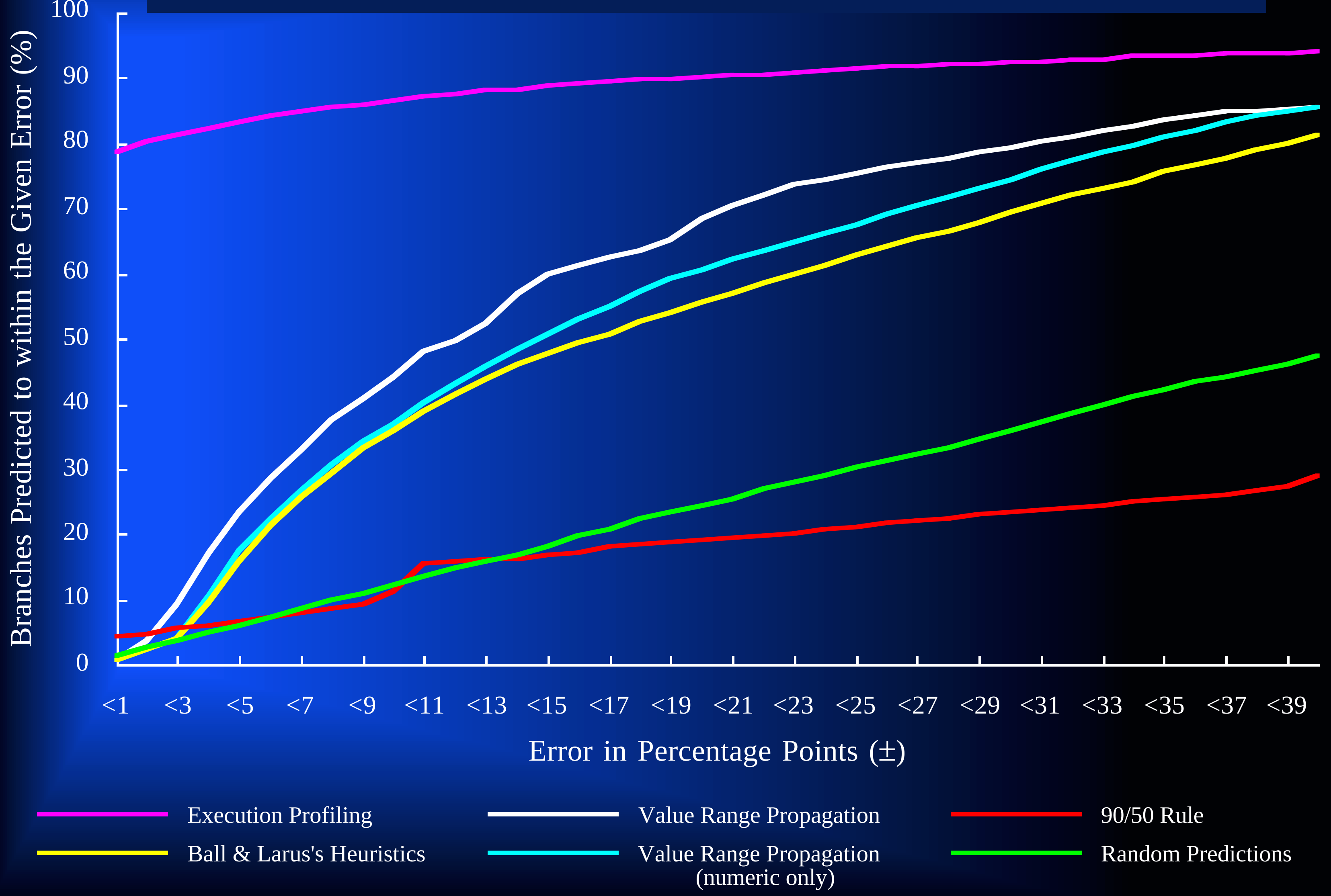
- expressions may need to be evaluated many times
- expression evaluation is slower than for constants

but still linear in the size of the program...

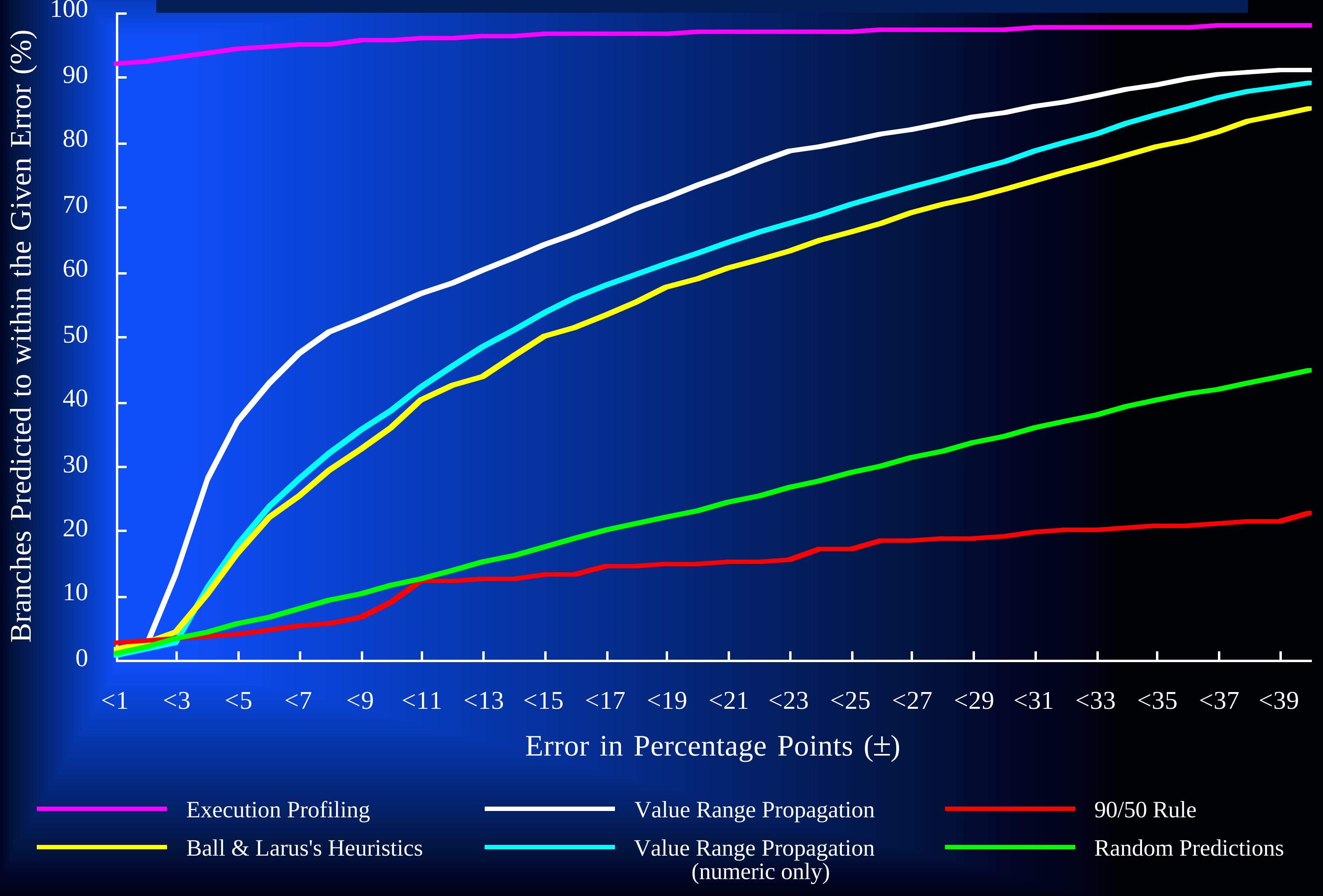




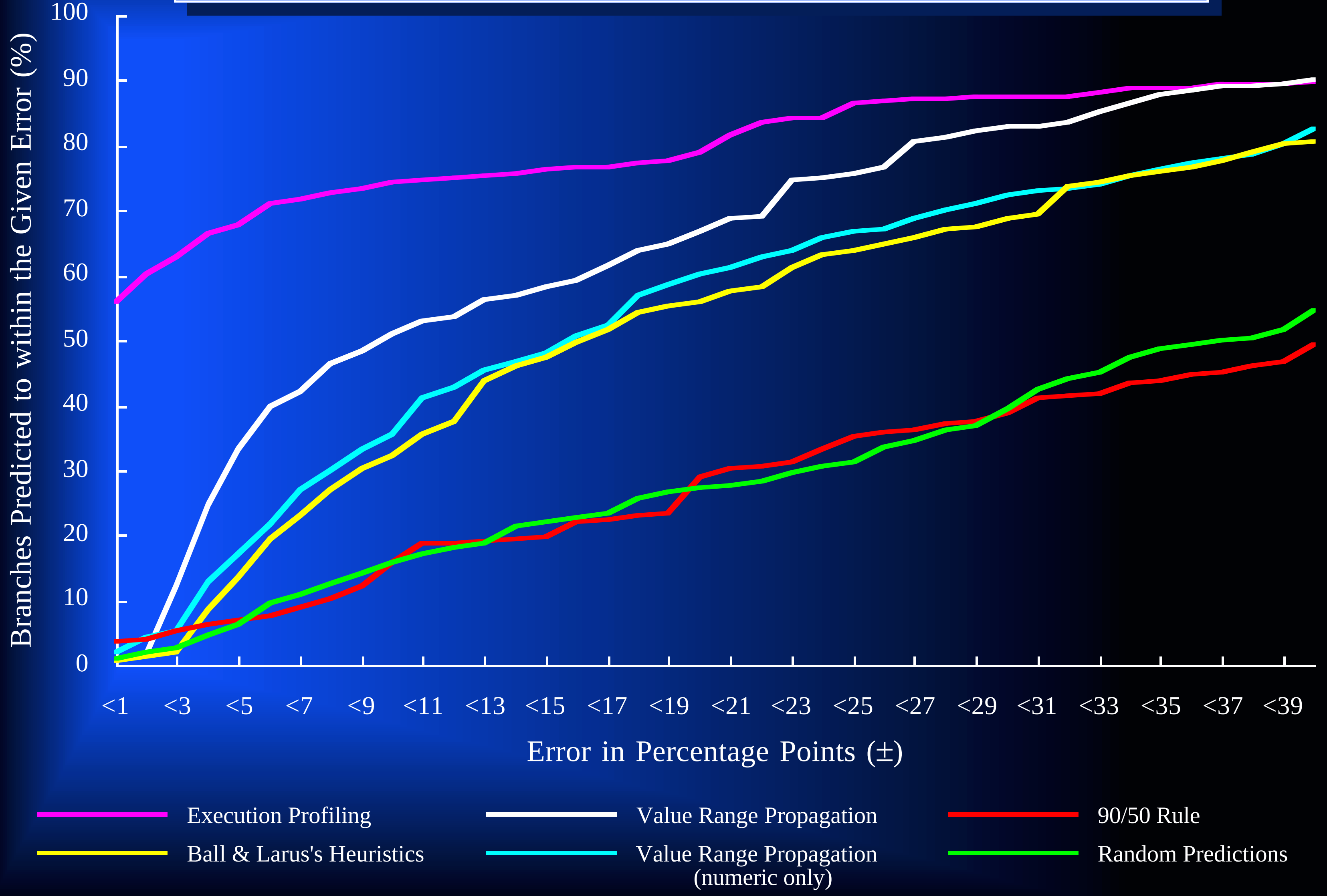
Results: SPECint92 (Unweighted)



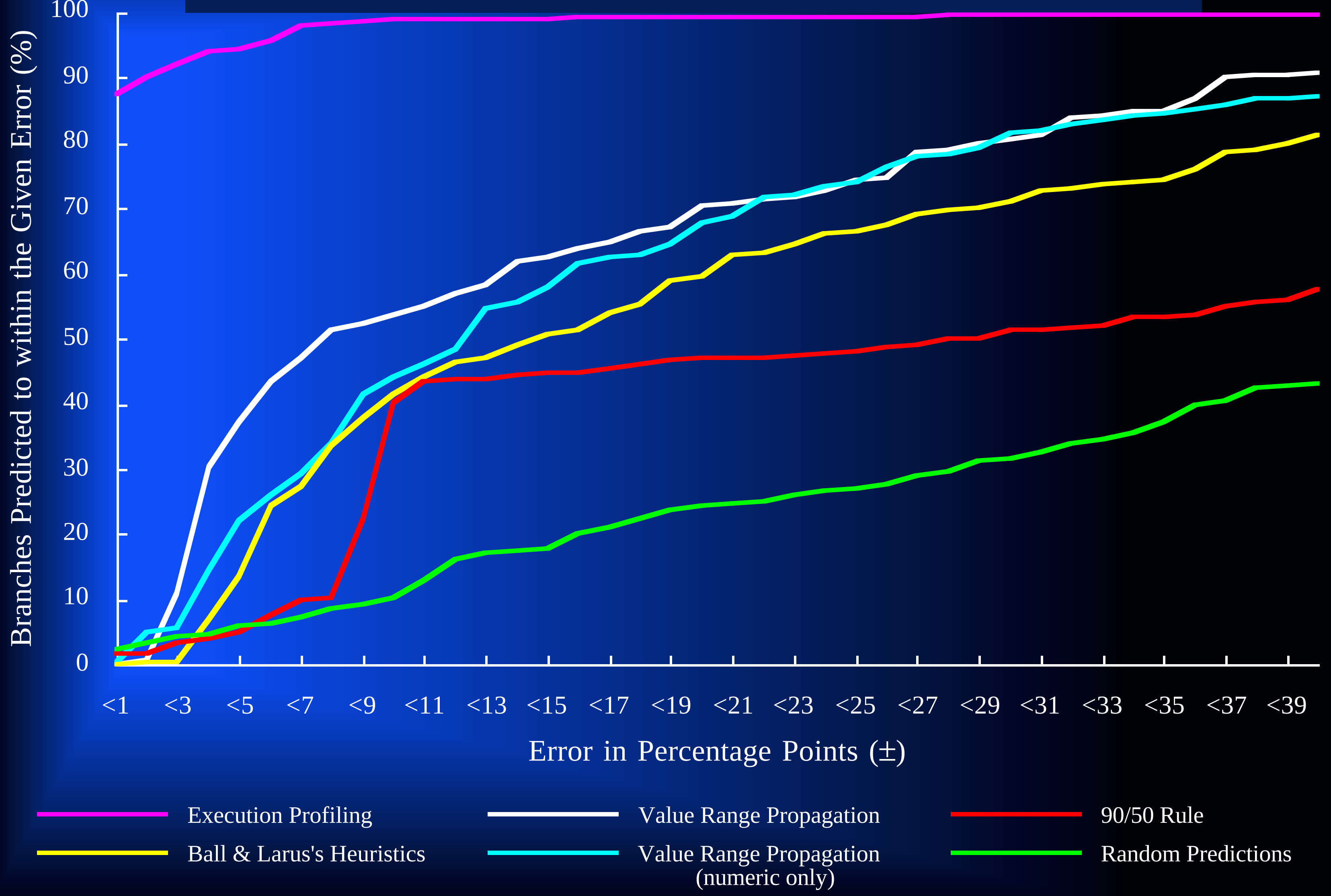
Results: SPECfp92 (Unweighted)



Results: SPECint92 (Weighted)



Results: SPECfp92 (Weighted)



Conclusions...

Value Range Propagation offers a significant improvement over the best existing heuristics (heuristics are still used as a fallback)

Most of this improvement comes from analysis involving ***symbolic*** ranges

Various engineering techniques can be used to make this analysis ***fast enough to be viable***...

- simple range representation
- symbolic analysis relative to a single variable
- handle **most** loop-carried expressions by matching derivations against common looping scenarios