NC STATE UNIVERSITY

A Study of Slipstream Processors

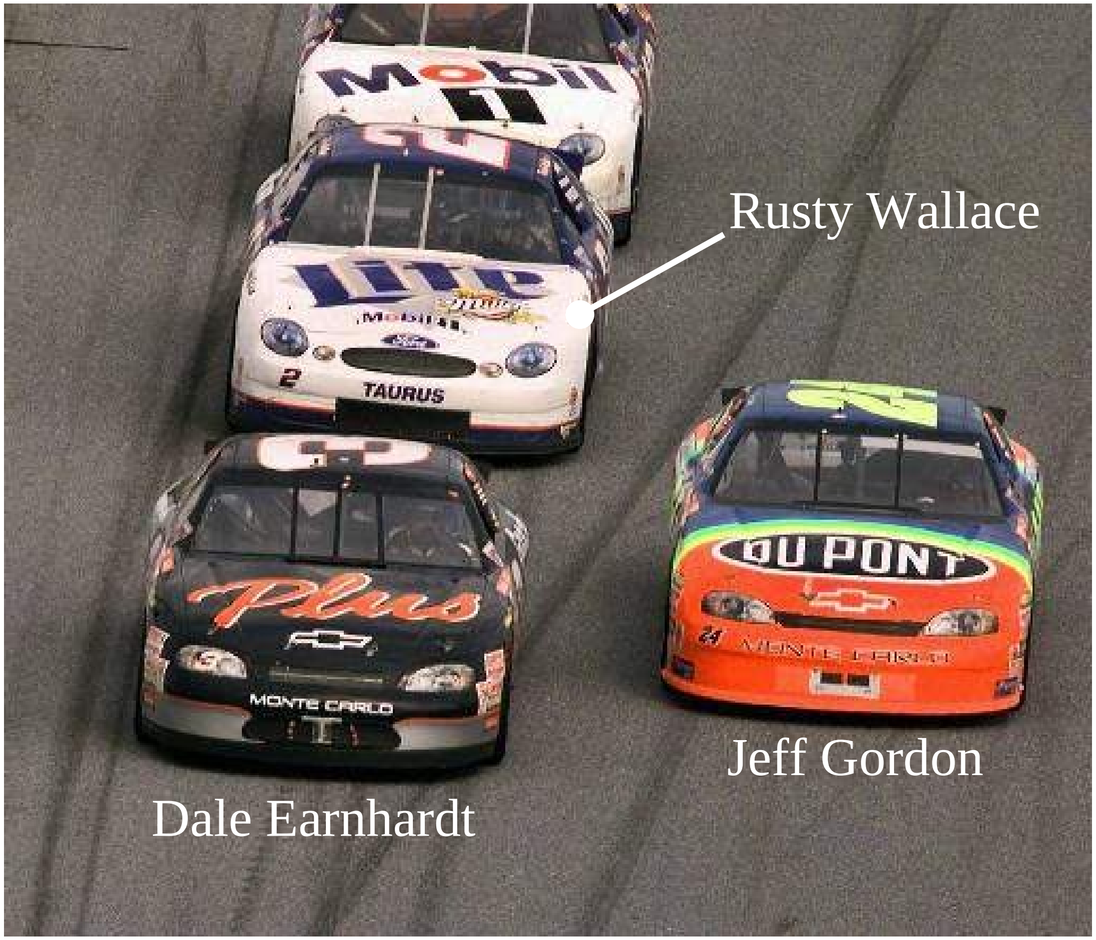
Zach Purser

Karthik Sundaramoorthy, Eric Rotenberg

Dept. of Electrical and Computer Engineering
North Carolina State University
www.tinker.ncsu.edu/ericro/slipstream
{zrpurser,ksundar,ericro}@ece.ncsu.edu

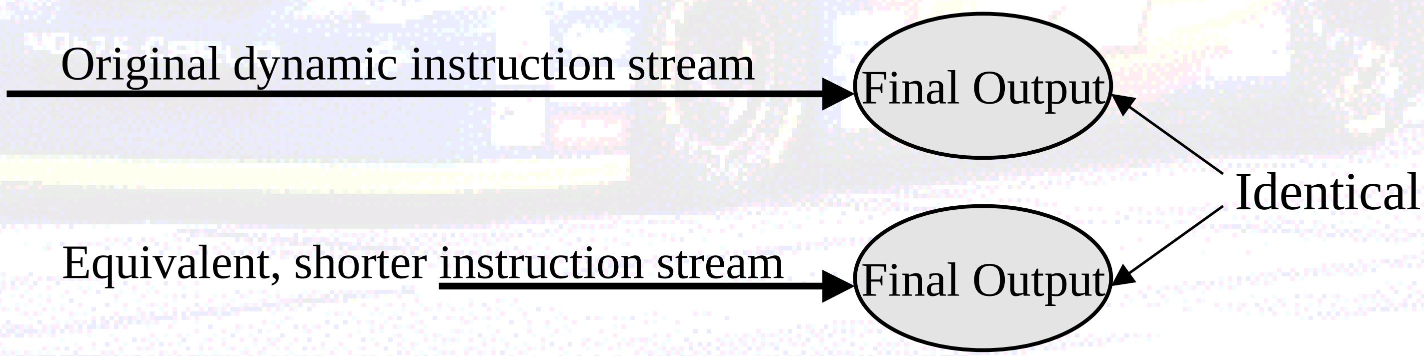
NASCAR and Computers

- "Slipstreaming"
  - Two cars race nose-to-tail to speed up both cars



Reducing the Program

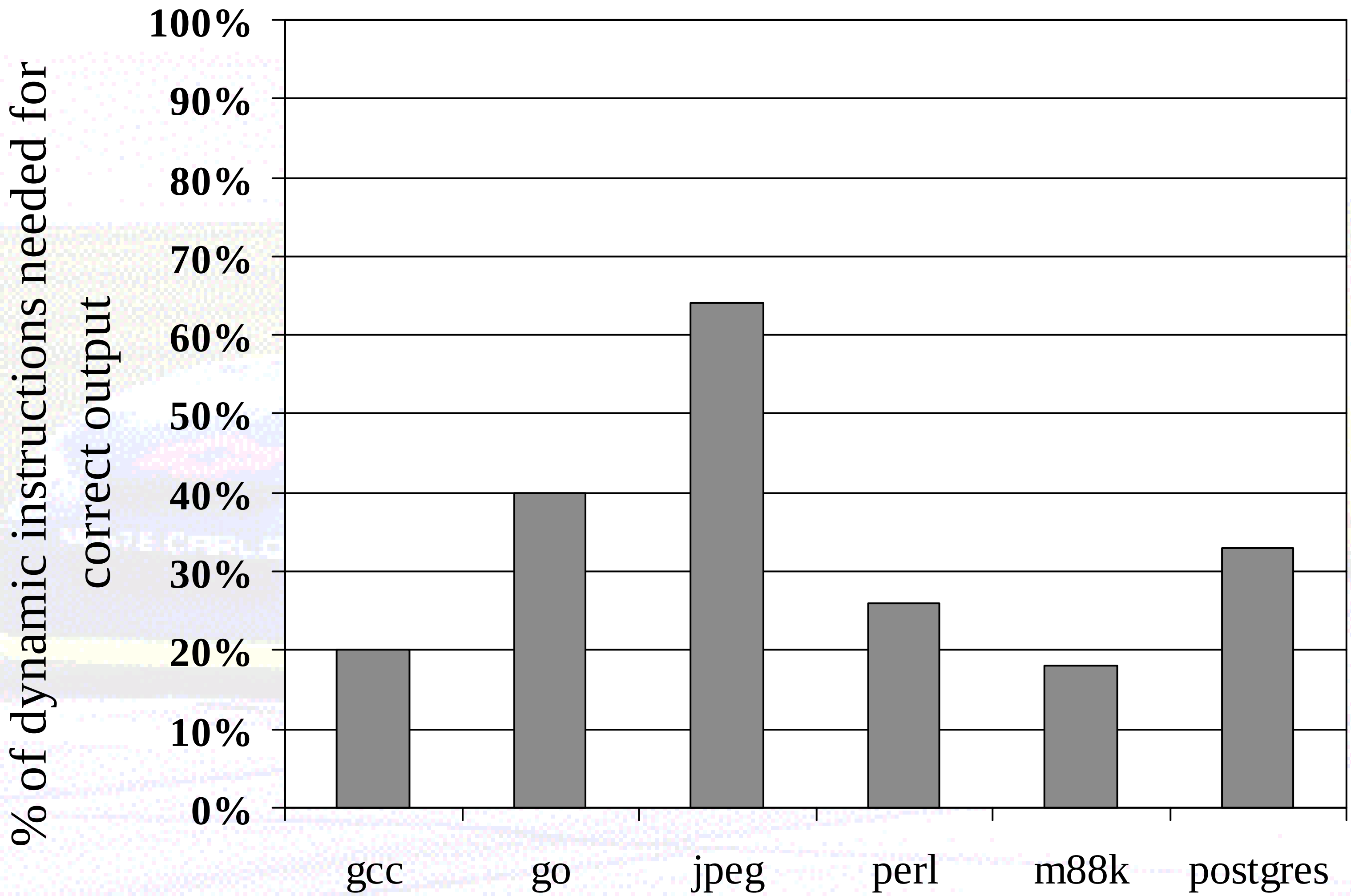
- Processors execute full instruction stream to produce final output
- Is it possible to construct a shorter instruction stream with the same effect?



Reducing the Program (cont.)

- Ideal experiment
  - Run full program
  - Then pick out instructions that (in retrospect) were unnecessary
- What were unnecessary for correct forward progress?
  - Non-modifying writes
  - Unreferenced writes
  - Correctly-predicted branches
  - …and their dependence chains

Reducing the Program (cont.)



Catch-22

- Only need a small part of program to make full, correct, forward progress
- The catch
  - Skipping instructions is speculative…
  - …AND lose ability to verify instructions were skippable
- Answer: run both programs! (redundant execution)
  - Check results of short program against full program

NC STATE UNIVERSITY

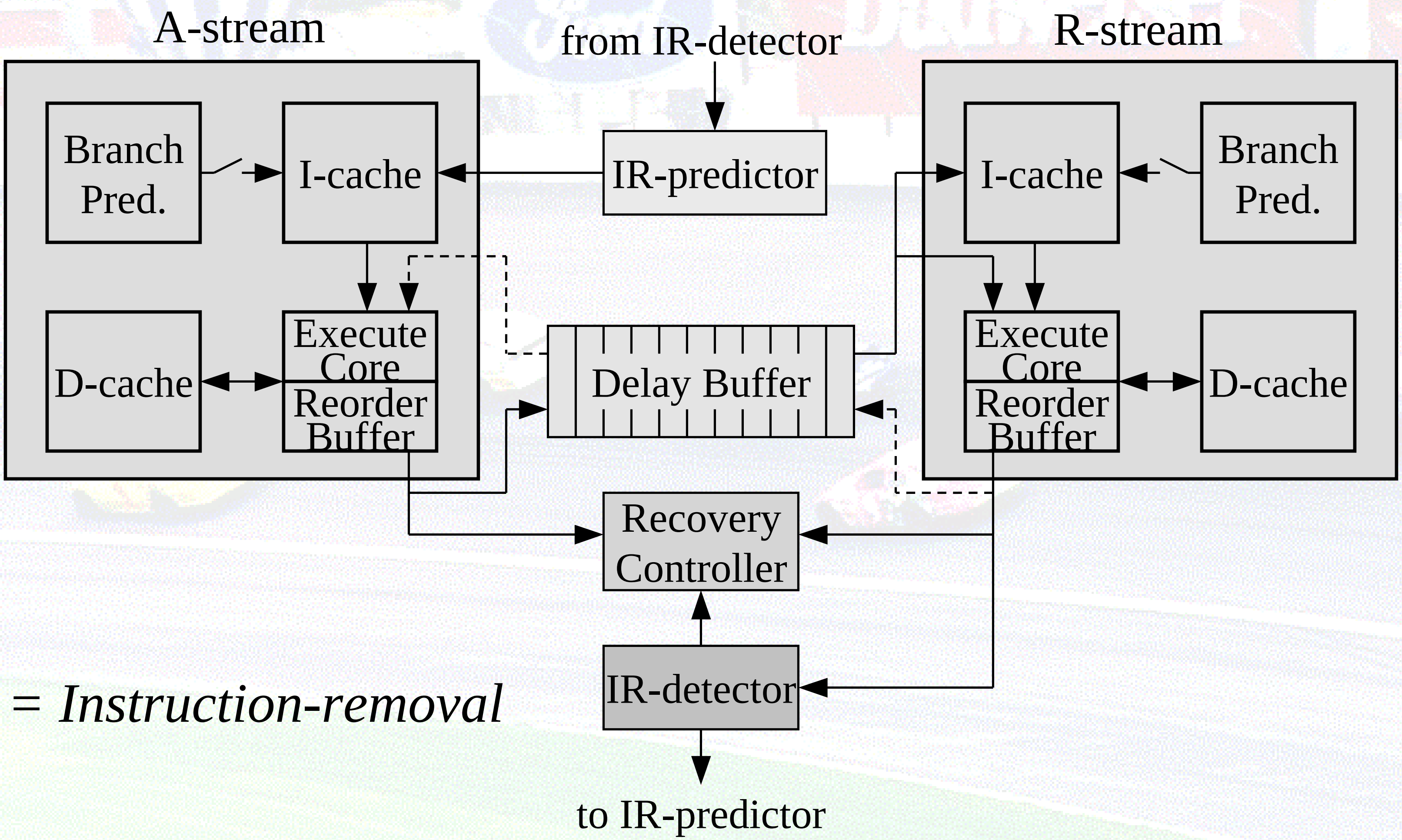
Slipstream Paradigm

- Operating system creates two redundant processes
  - Programs run concurrently on single-chip multiprocessor (CMP) or simultaneous multithreading processor (SMT)
  - One program always runs slightly ahead of other
    - *Advanced stream* (A-stream) leads
    - *Redundant stream* (R-stream) trails

Slipstream Paradigm

- Step 1: reduce the A-stream
  - Monitor R-stream to detect past-removable computation
  - Use knowledge to speculatively reduce A-stream in future
  - A-stream fetches/executes fewer instructions
- Step 2: check the A-stream
  - A-stream passes control/data outcomes to R-stream
  - R-stream checks outcomes: if A-stream deviates, it’s context is recovered from R-stream
- Step 3: speedup R-stream while it checks
  - R-stream uses A-stream outcomes as predictions
    - Leverage existing speculation mechanisms to do checks
    - R-stream fetches/executes more efficiently
- Both programs finish sooner (roughly same time)

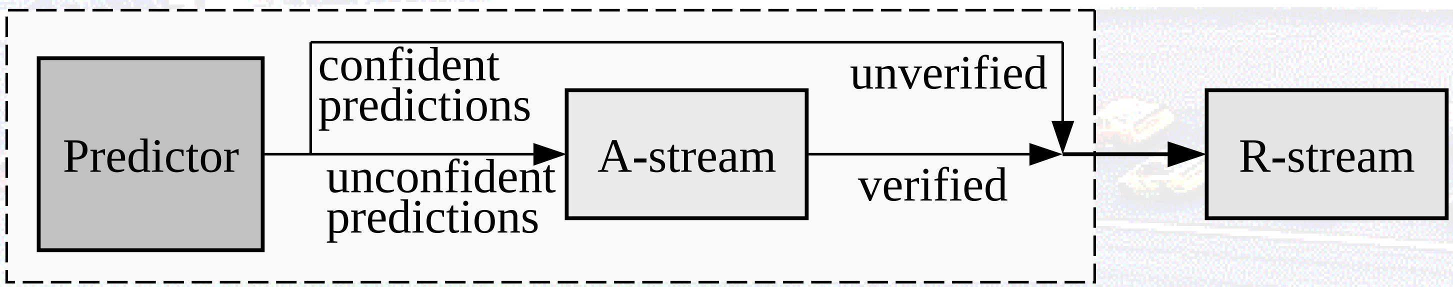
Slipstream Microarchitecture



IR = Instruction-removal

Where's the speedup?

- A-stream's perspective: It is a shorter program
  - A-stream runs faster
  - R-stream is a fast checker (doesn't slow A-stream down)
- R-stream's perspective: It has accurate program-based prediction

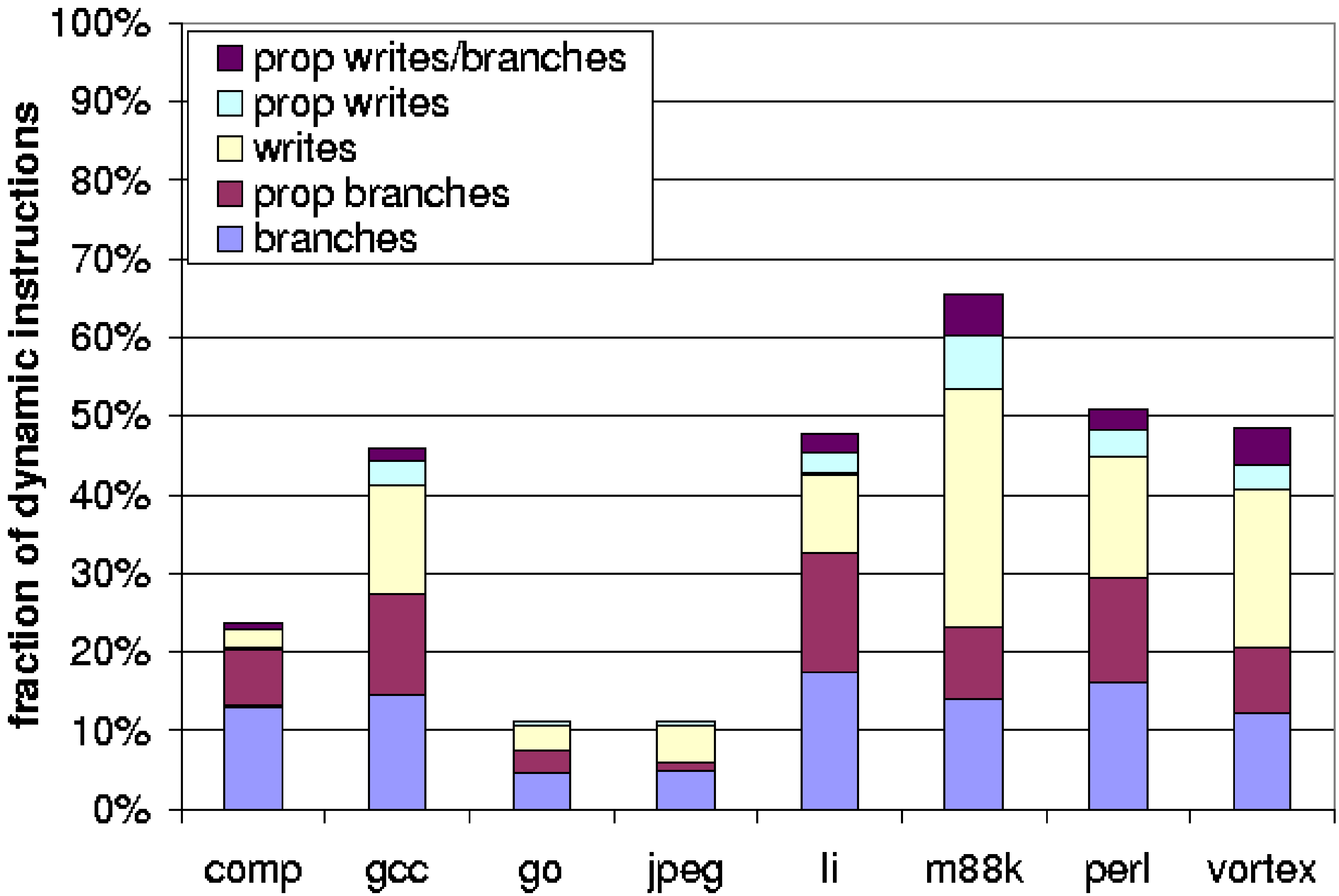


  - Related work
    - [Roth, Moshovos & Sohi] - Prefetching linked data structures
    - [Roth & Sohi] - Speculative data-driven multithreading
    - [Zilles & Sohi] - Backward slices of performance-degrading instr.
    - [Farcy, Temam, Espasa & Juan] - Early branch resolution
    - [Chappell, Stark, Kim, Reinhardt, Patt] - SSMT

Contributions

- More effective instruction removal
  - Previous trace-based approach was conservative
    - Insufficient removal
    - Overall confidence reflects least-confident instructions in trace
  - New instruction-based approach => majority of benchmarks reduced by half

Breakdown of Instruction Removal



NC STATE UNIVERSITY

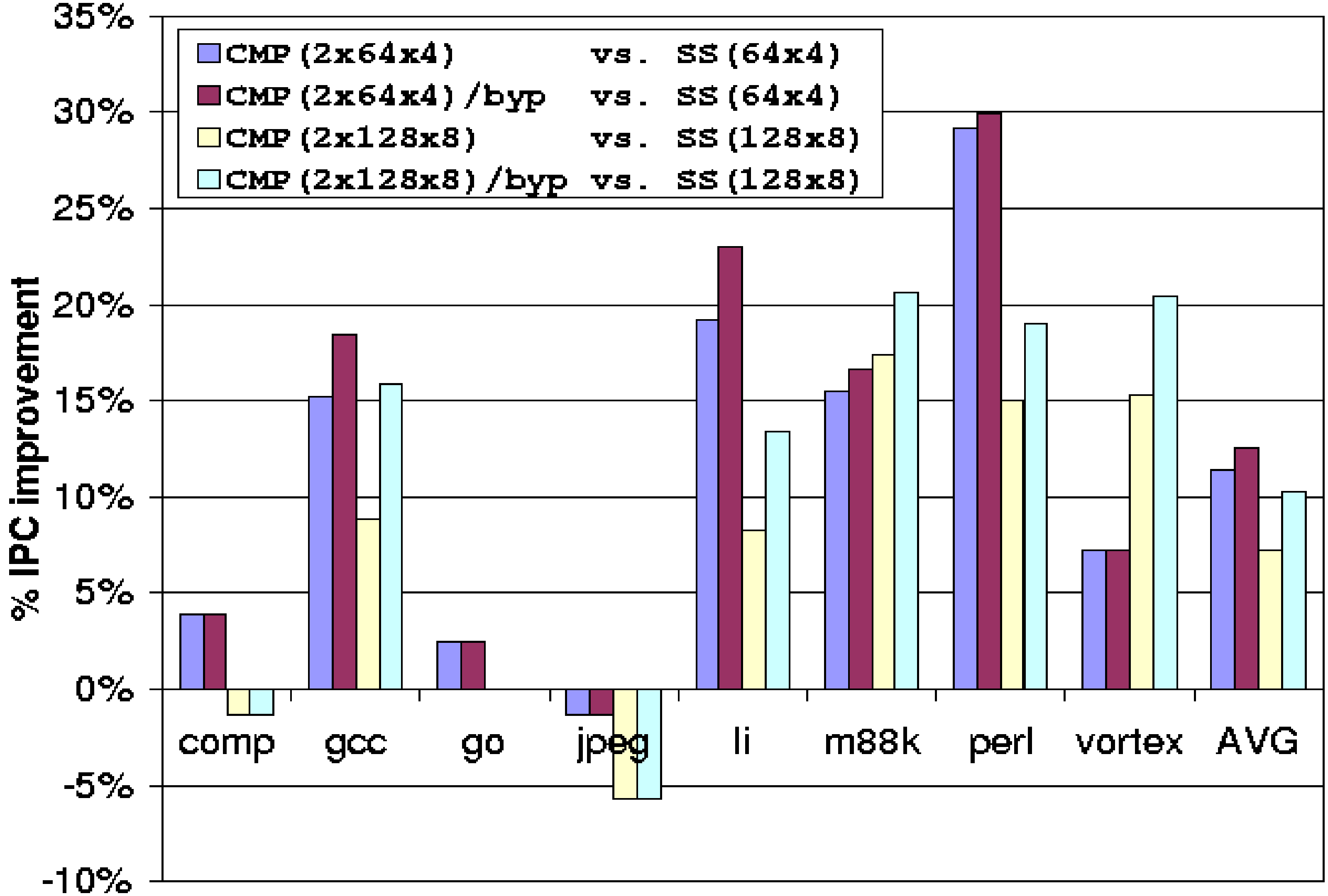
Contributions (cont.)

- Performance (CMP)
  - 12% average speedup using second free superscalar core
  - Comparable to larger, more complex, inflexible superscalar
- Bypassing instruction fetch
  - Important to skip instructions *before* they are fetched
  - Novel method for bypassing instruction fetch
    - Simple modifications to conventional branch predictor
    - Skip basic block if all instructions in block are predicted removed

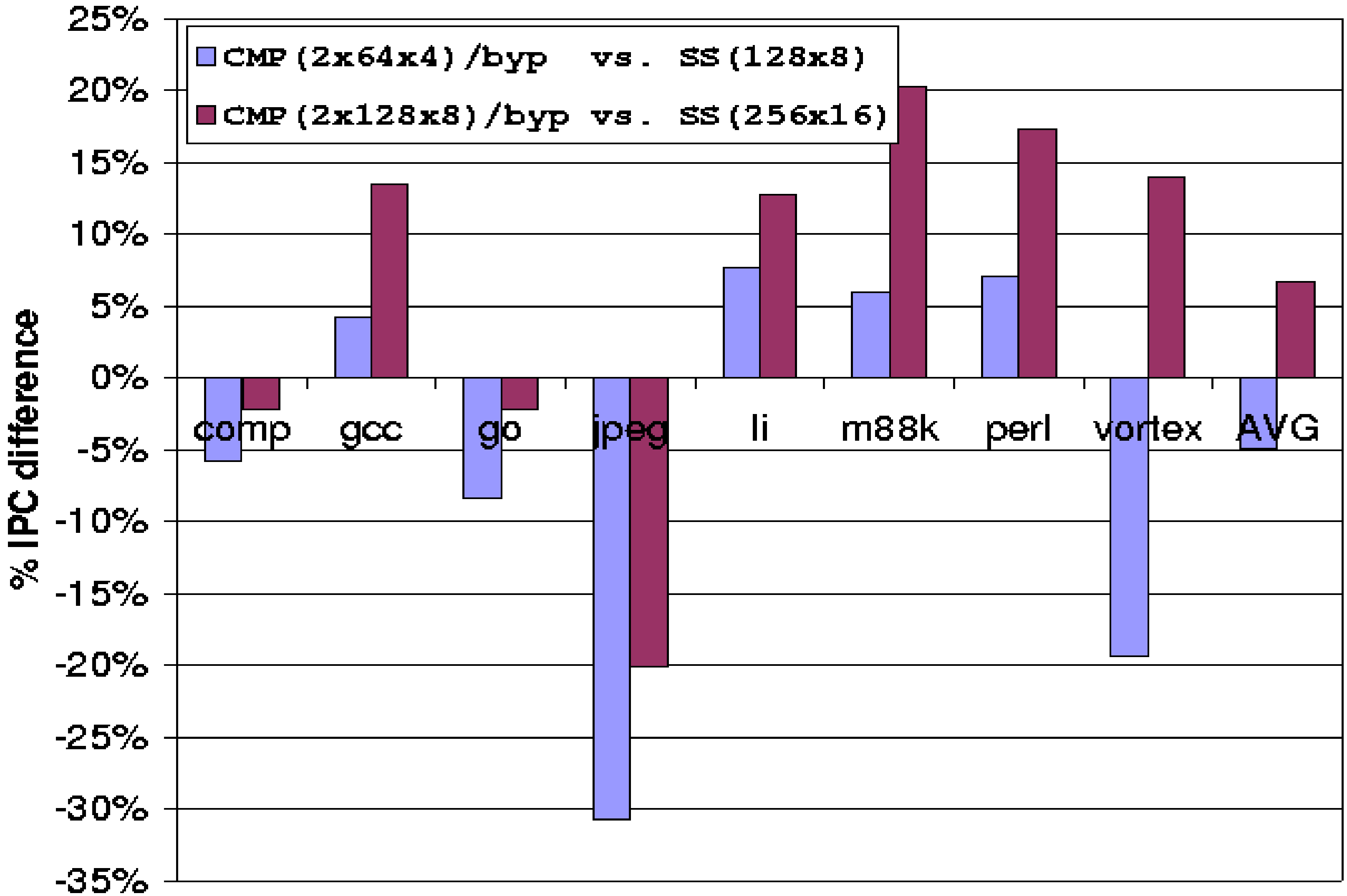
Models

- SS(64x4): Single 4-way superscalar proc with 64 ROB entries
- SS(128x8): Single 8-way superscalar proc with 128 ROB entries
- SS(256x16): Single 16-way superscalar proc with 256 ROB entries
- CMP(2x64x4): Slipstream on a CMP composed of two SS(64x4) cores
- CMP(2x64x4)/byp: Same as previous, but A-stream can bypass instruction fetch
- CMP(2x128x8): Slipstream on a CMP composed of two SS(128x8) cores
- CMP(2x128x8)/byp: Same as previous, but A-stream can bypass instruction fetch
- SMT(128x8)/byp: Slipstream on SMT, where SMT is built on top of SS(128x8)

Slipstream Performance (CMP)



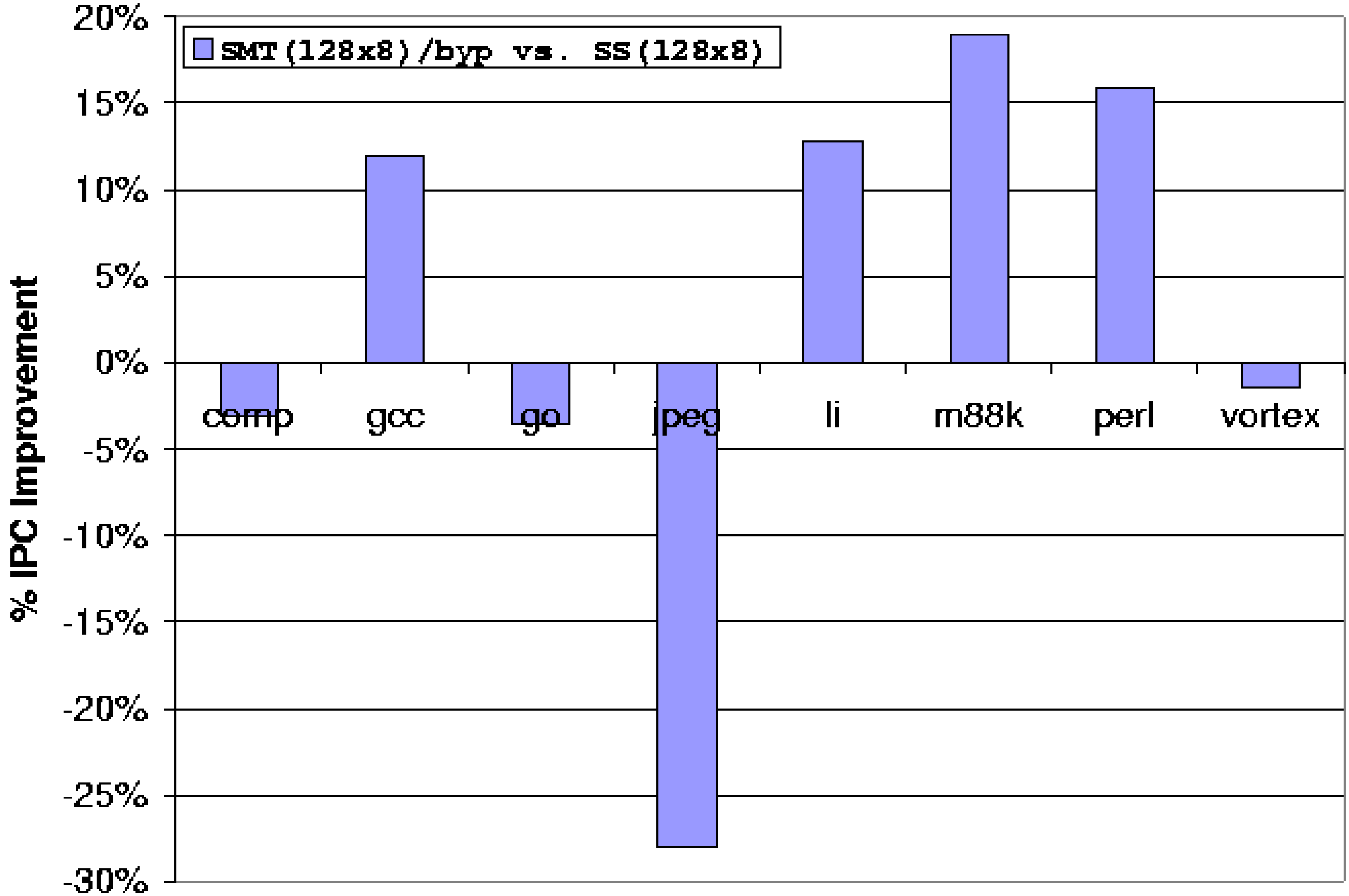
Slipstream on 2 cores VS. 1 large core



Contributions (cont.)

- SMT-based slipstream implementation
  - SMT infeasible before due to insufficient A-stream reduction
  - 10-20% improvement for benchmarks with significant removal

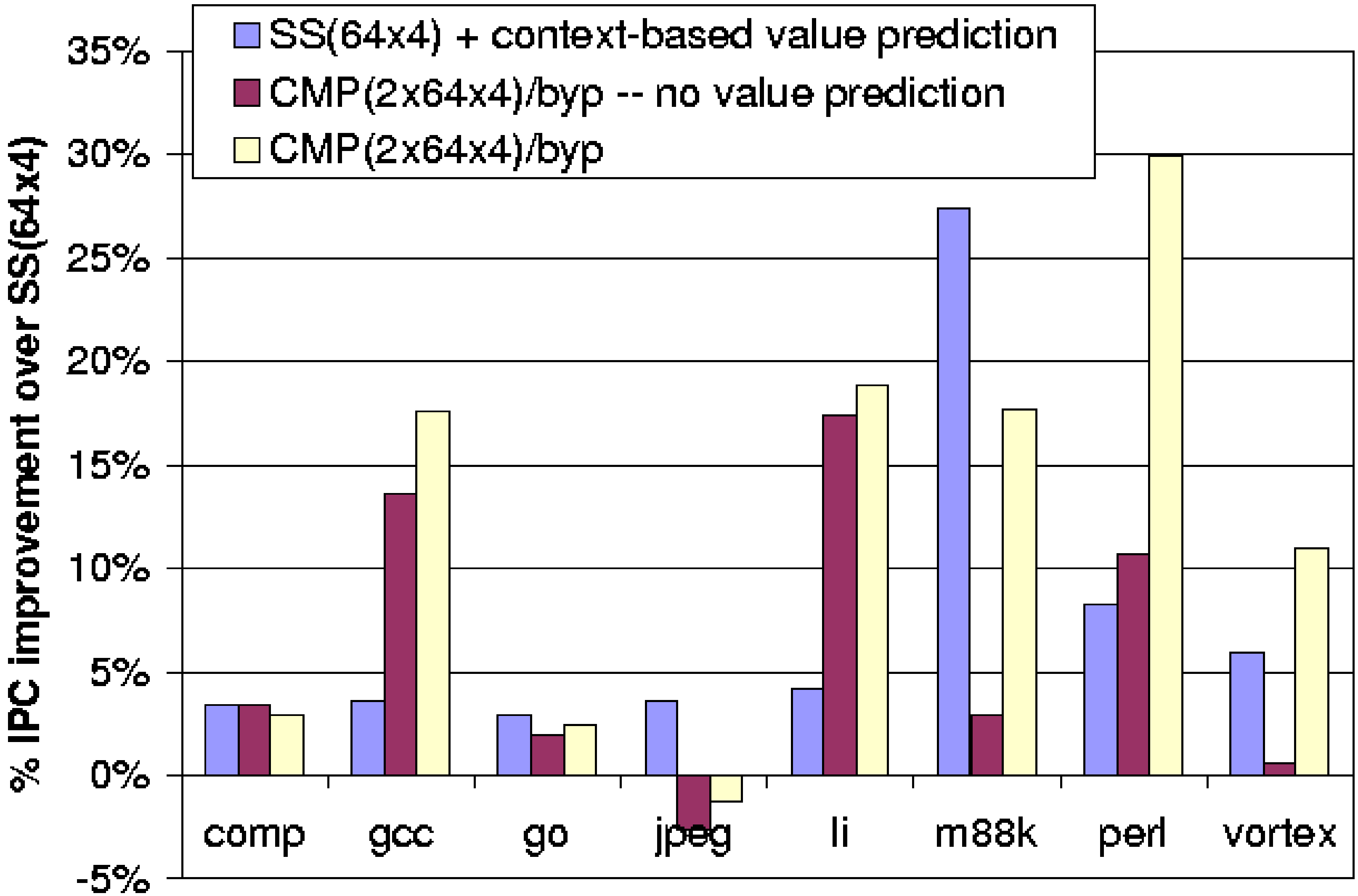
Slipstream Performance (SMT)



Contributions (cont.)

- Quantify program-based prediction
  - Some benchmarks benefit from resolving branch mispredictions ahead of time
  - Others benefit from value predictions, not always reproducible by conventional value predictor

Program-Based Prediction



Summary

- Results
  - 12% average improvement harnessing otherwise unused PE
  - Slipstreaming on 2 small cores has comparable IPC to 1 large core, but with faster clock and more flexible architecture
  - Bypassing instruction fetch is important
  - Majority of benchmarks show significant A-stream reduction (50%); Slipstreaming on 8-way SMT improves their performance 10%-20%
  - Quantified program-based prediction: resolving branch mispredictions in advance + quality value prediction
- Slipstream Processors: novel method for harnessing CMP/SMT to speed up single programs

Future Work

- Slipstream Processors
  - Further understanding performance
  - Microarchitectural design space
  - Pipeline organization
  - Fault tolerance
  - System-level issues
  - Adaptivity
- Fundamental variations of Slipstream Paradigm
  - Streamlining R-stream
  - Other A-stream shortening approaches
  - Scaling to N threads
  - Approximate A-streams
- Other novel CMP/SMT applications