The Trace Cache, One Year Later

Sanjay Jeram Patel   Daniel Holmes Friendly   Marius Evers
Advisor: Yale N. Patt

High Performance Systems Group
Advanced Computer Architecture Laboratory

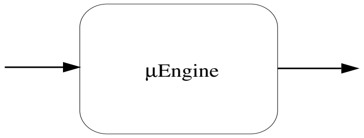
Ninth Annual IPoCSE Research Review
March 26, 1998
http://www.eecs.umich.edu/HPS/

Outline

- What's a trace cache good for anyway?
- Technique 1: Partial Matching
- Technique 2: Inactive Issue
- Technique 3: Branch Promotion
- Technique 4: Trace Packing
- Conclusions and Future Directions

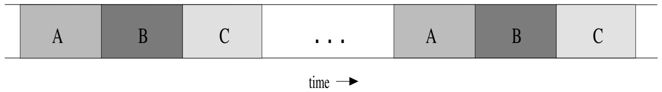
Fundamental Principle

Flynn's Bottleneck



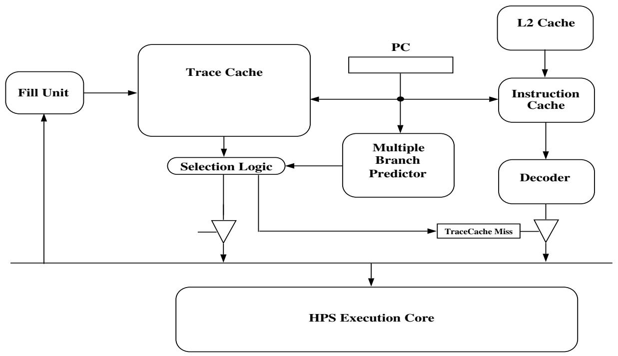
Can't get out more than you put in.

Caching the Dynamic Instruction Stream

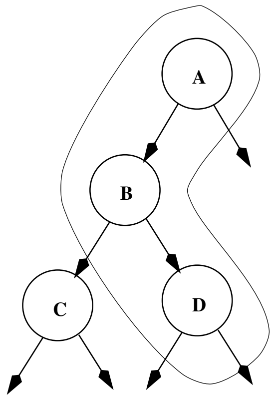


Dynamic Instruction Stream

Trace Cache Fetch Mechanism



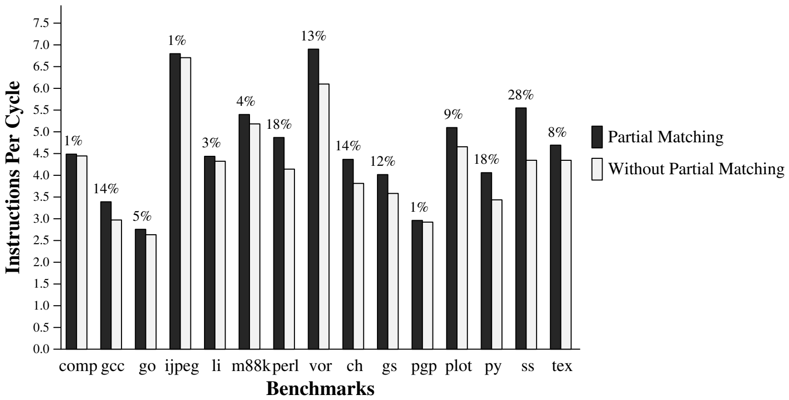
Technique 1: Partial Matching



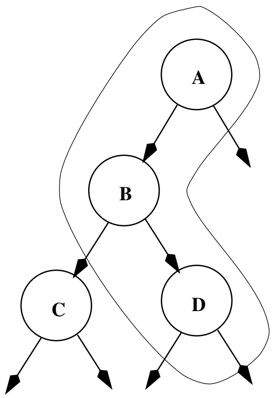
Predicted path: ABC
Fetched segment: ABD

No partial matching: miss
Partial Matching: AB

Partial Matching Results

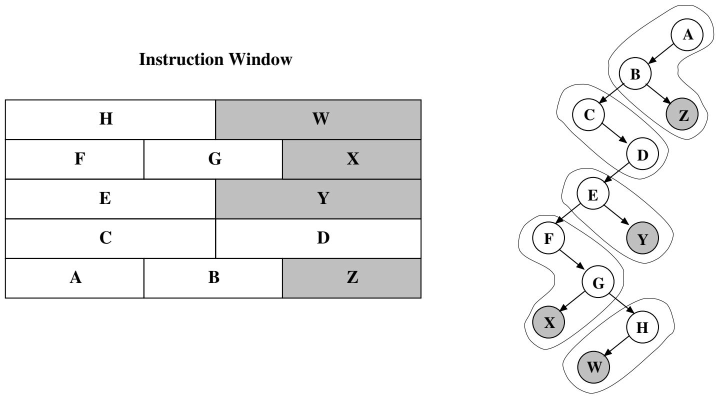


Technique 2: Inactive Issue

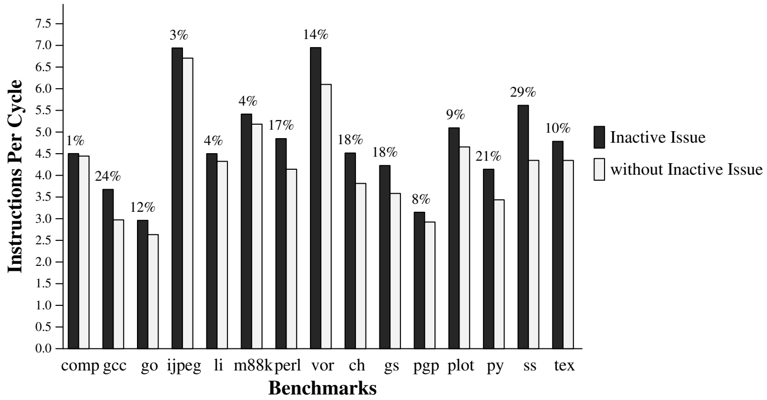


Predicted path: ABC
Fetched segment: ABD

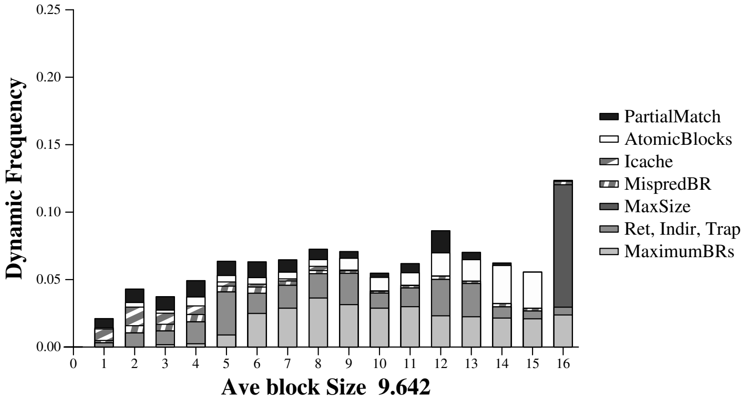
No partial matching: miss
Partial matching: AB
Inactive Issue: AB (active) D(inactive)



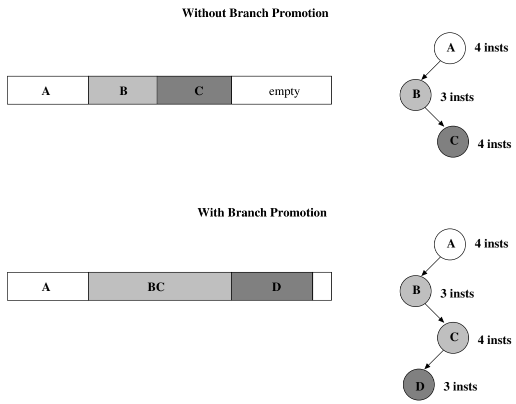
Inactive Issue Results



Limitations on trace size: gcc



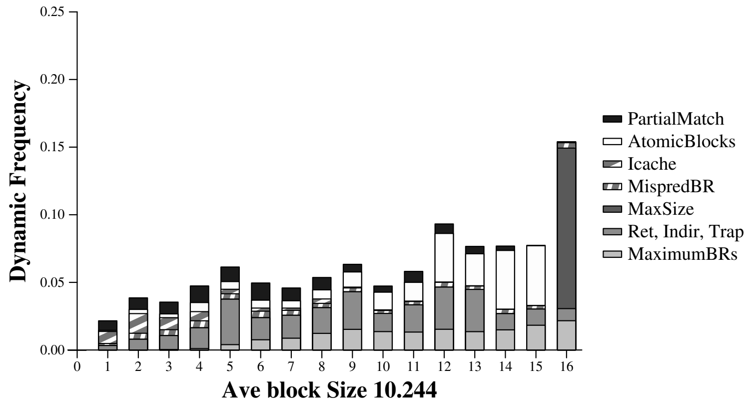
Technique 3: Branch Promotion



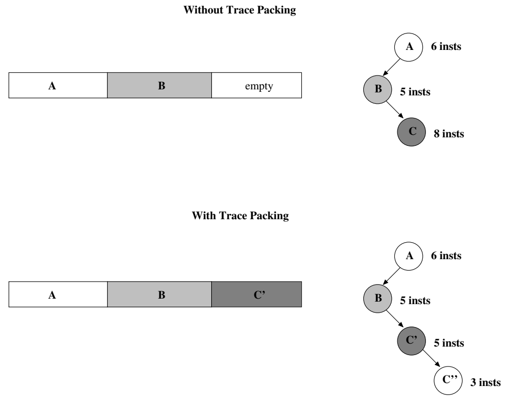
Number of dynamic predictions per fetch

Branches	without Promotion	with Promotion
0	25%	56%
1	29%	29%
2	18%	12%
3	28%	3%

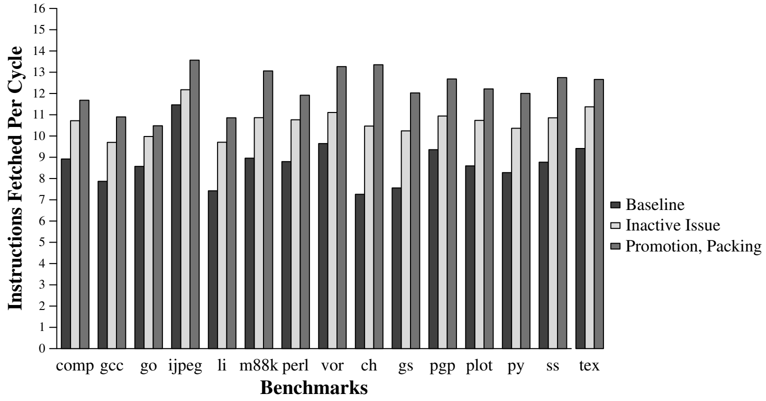
Limitations on trace size: gcc with promotion



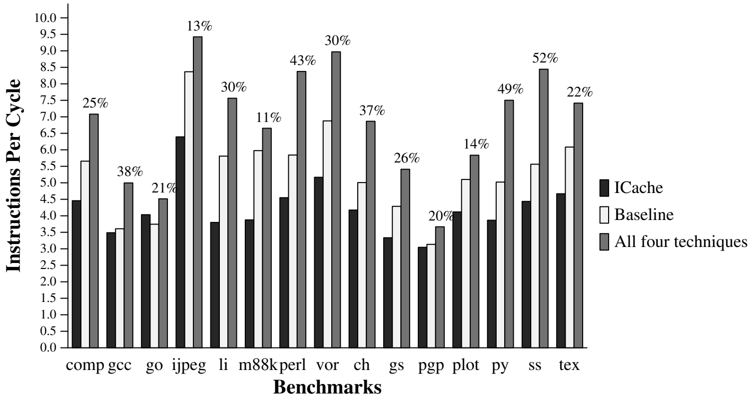
Technique 3: Trace packing



Effective fetch rate



Overall Performance (aggressive core)



Conclusions

- 10% performance increase with partial matching
- 13% performance increase with inactive issue
- 29% performance increase with all techniques
- 41% fetch rate increase with all techniques
- Promotion provides a good branch framework

Future Directions

- A branch predictor which utilizes promotion
- Longer traces
- Dynamic trace optimizations
- Implementation issues