Carnegie Mellon

"Replenishing the Microarchitecture Treasure Chest"

Prof. John Paul Shen
Electrical and Computer Engineering Department
Carnegie Mellon University

CMµART
CARNEGIE MELLON MICROARCHITECTURE RESEARCH TEAM

Carnegie Mellon

CMuART Members

Current Ph.D. Students:

1. **Bryan Black**
2. **Yuan Chou**
3. **Alex Dean**
4. **Ryan Rakvic**
5. **Bob Rychlik**

Current M.S. Students:

1. **Candice Bechem**
2. **Jonathan Combs**
3. **Jeffrey Heid**
4. **Kyle Oppenheim**

CMuART (PhD) Alumni:

1. **Ron Bianchini (FORE & CMU)**
2. **Mauricio Breternitz (Moto)**
3. **Trung Diep (Intel)**
4. **F. Joel Ferguson (UCSC)**
5. **Andrew Huang (Moto)**
6. **Mikko Lipasti (IBM & UW)**
7. **Chris Newburn (Intel)**
8. **Derek Noonburg (S3)**
9. **Scott Robinson (Intel)**
10. **Mike Schuette (Moto)**
11. **Kent Wilken (UC-Davis)**
12. **Andy Wolfe (S3)**

Carnegie Mellon

Microprocessor Performance

Unmatched by Any Other Industry!

[John Crawford, Intel, 1993]

Doubling every 18 months (1982-1996): total of 800X

- Cars travel at 44,000 MPH; get 16,000 miles/gal.
- Air travel: L.A. to N.Y. in 22 seconds (MACH 800)
- Wheat yield: 80,000 bushels per acre

Doubling every 24 months (1971-1996): total of 9,000X

- Cars travel at 600,000 MPH; get 150,000 miles/gal.
- Air travel: L.A. to N.Y. in 2 seconds (MACH 9,000)
- Wheat yield: 900,000 bushels per acre

Carnegie Mellon

Leveraging the Treasure Chest

All originally invented in the 1960's

- Pipelining
- Cache Memories
- Multiple Instruction Issue
- Out of Order Execution
- Dataflow Machines
- Vector Machines
- Virtual Memory
- Optimizing Compilers
- Operating Systems

Iron Law of Processor Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

Architecture --> Implementation --> Realization

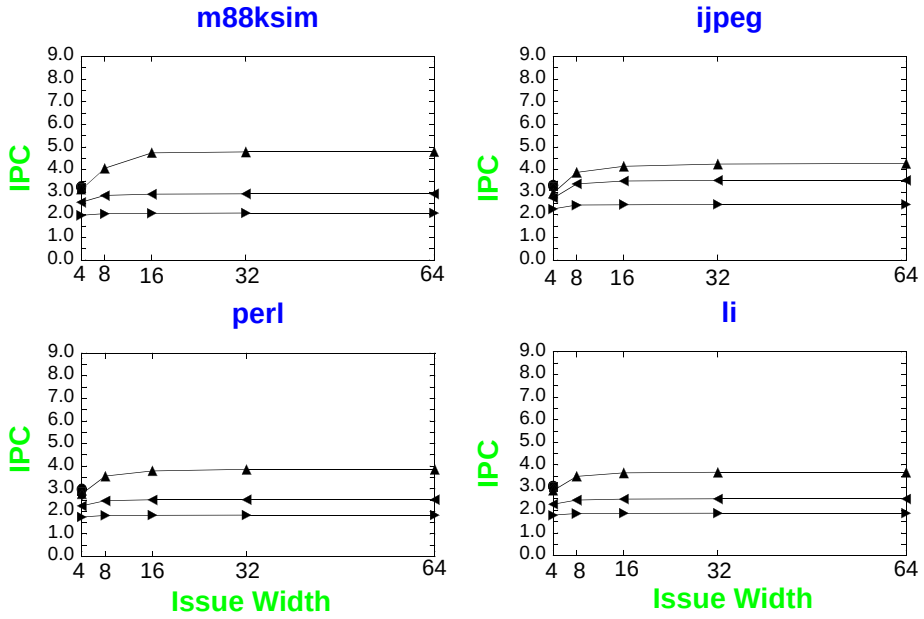
Compiler Designer Processor Designer Chip Designer

Evolution of Microprocessors

	1970-1979	1980-1989	1990-1999	by 2009
Transistor Count	10K-100K	100K-1M	1M-30M	1,000M
Clock Frequency	0.2-2MHz	2-20MHz	20-600MHz	10GHz
Instruction/cycle	<< 0.1	0.1-0.8	0.8- 2.4	10 (?)
MIPS/MFLOPS	<< 1	1-20	20-1,400	100,000

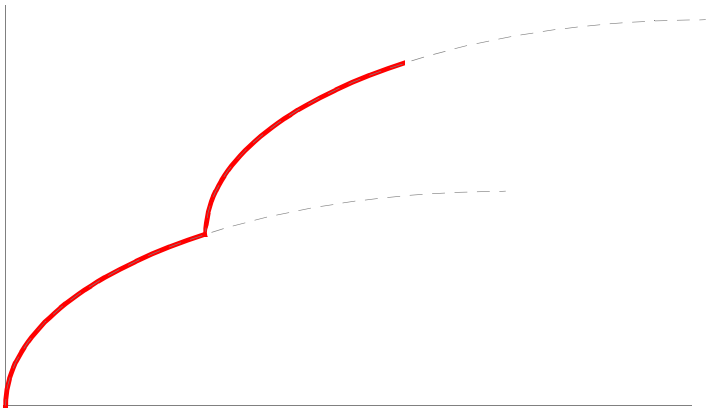
Carnegie Mellon

Strong Diminishing Returns on IPC



Carnegie Mellon

Looking for A Paradigm Shift



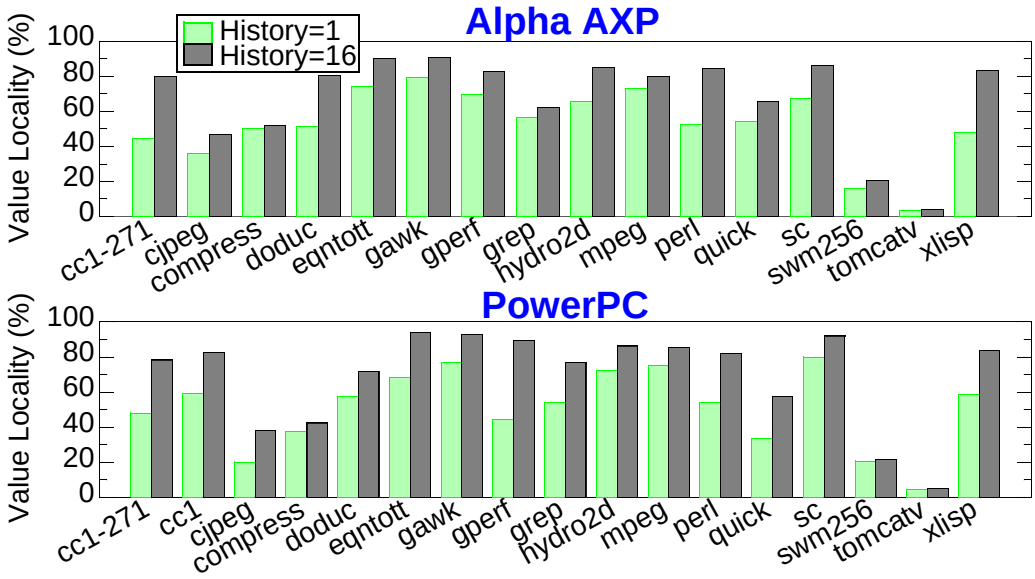
Revisit previously-assumed limits and try to go beyond these limits.

1970's - "Flynn's Bottleneck" ..... Branch Prediction

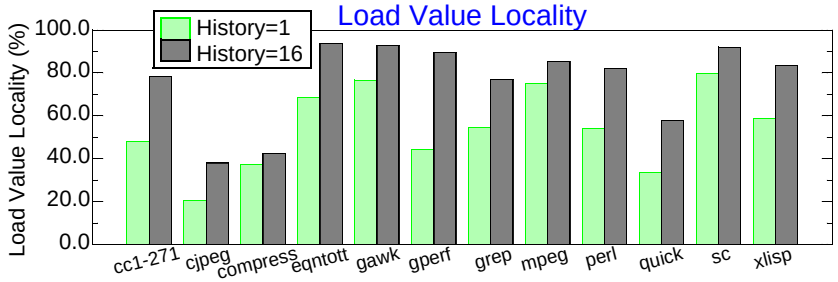
1990's - "Dataflow Limit" ..... Value Prediction

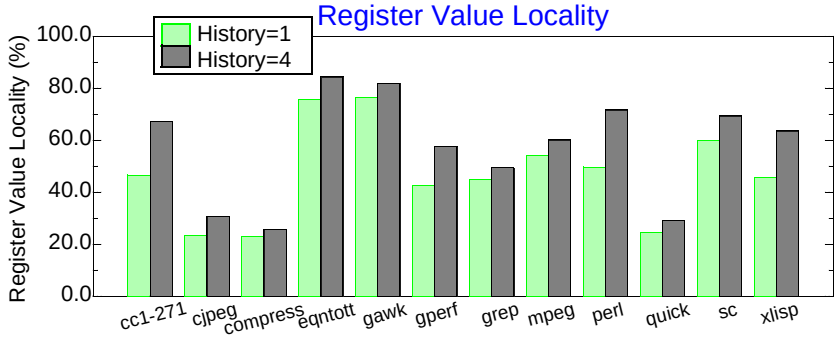
Once Upon a Time...Fall 1995

Load Value Locality:

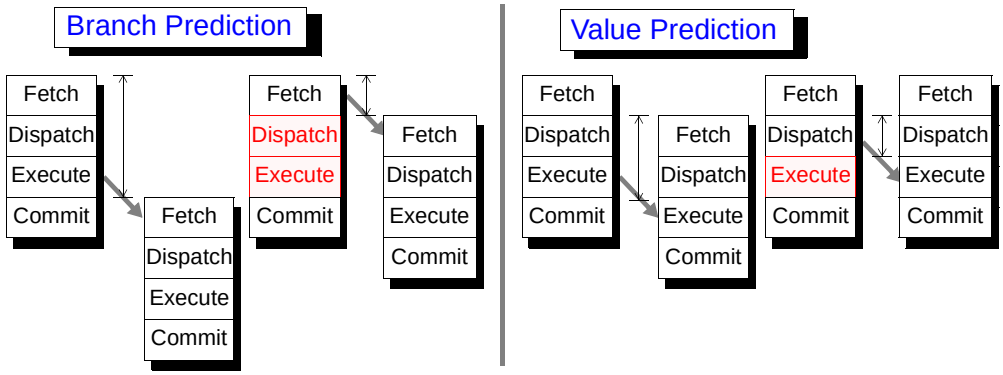


Concept of "Value Locality"

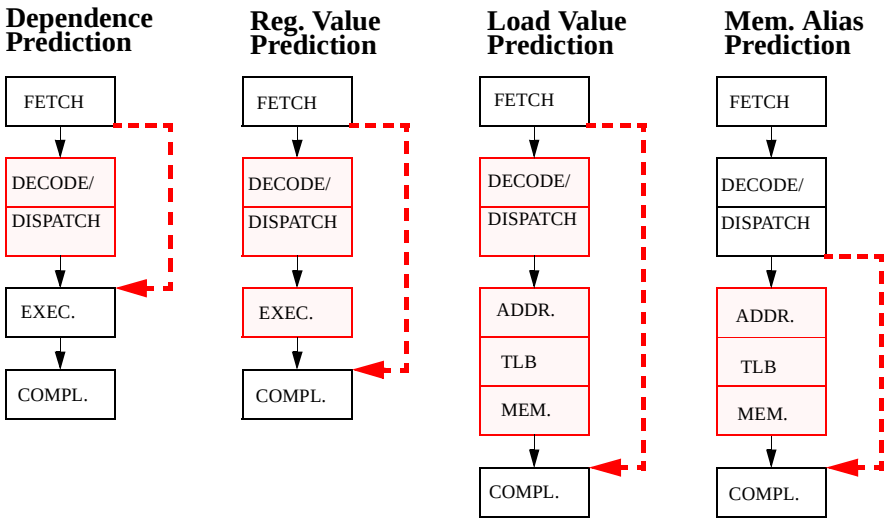




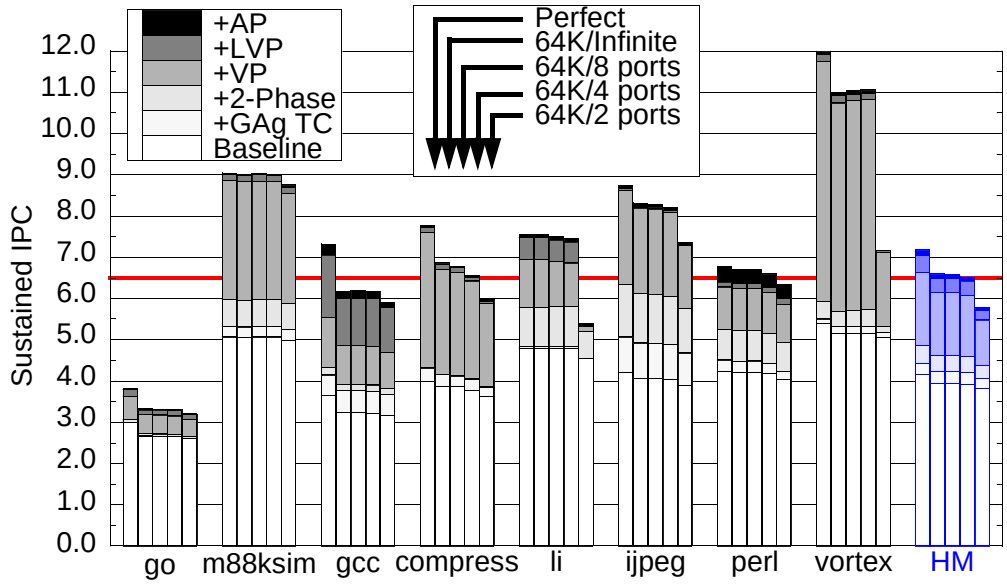
Dynamic “Pipeline Contraction”



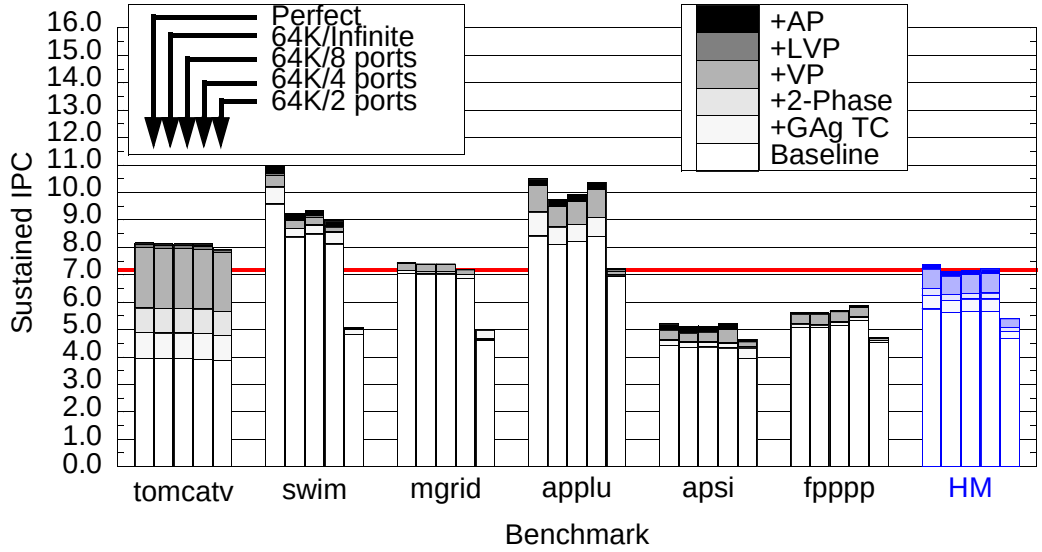
“Superspeculation” Techniques



SPECint95 Performance (16 wide)

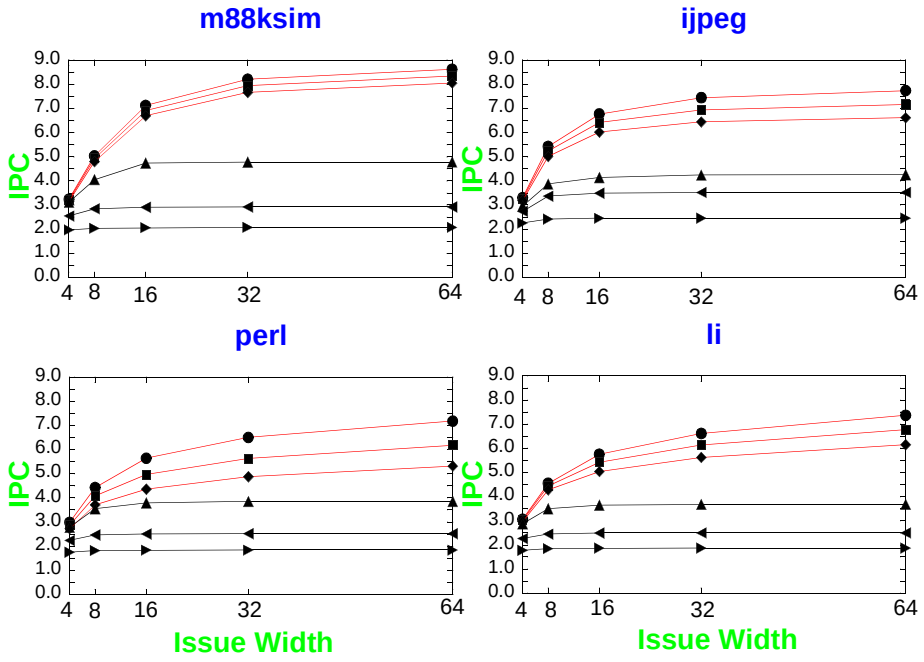


SPECfp95 Performance (16 wide)



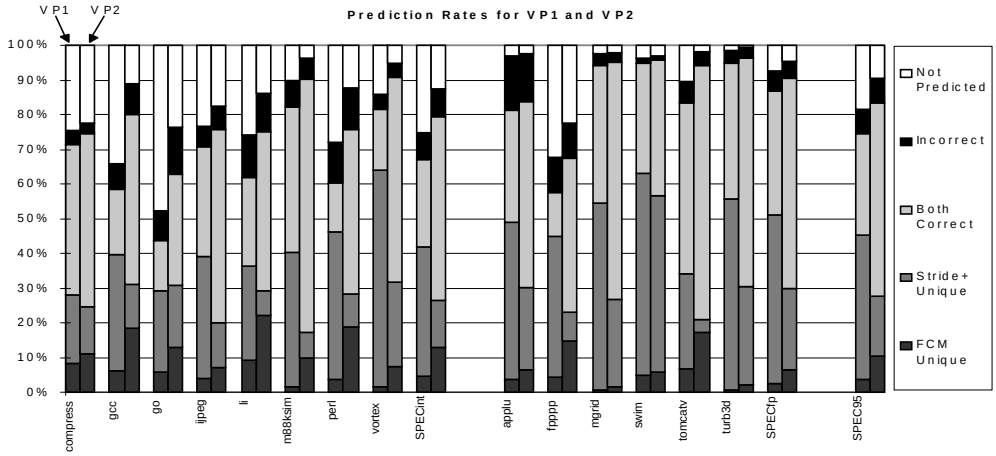
Carnegie Mellon

A Possible New Paradigm

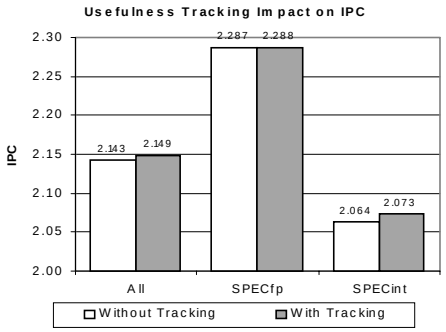


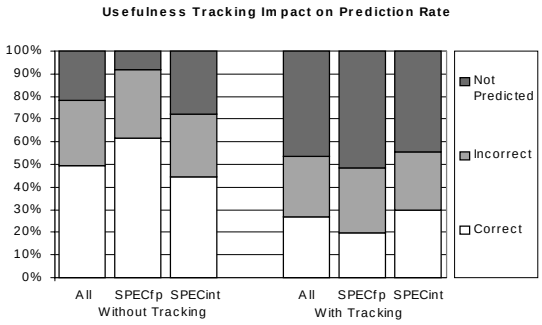
Carnegie Mellon

Hybrid Value Predictors



Prediction "Usefulness"





Current Value Prediction Landscape

Increasing Sophistication

- Hybrid Predictors
- Adaptive Predictors

Increasing Efficiency

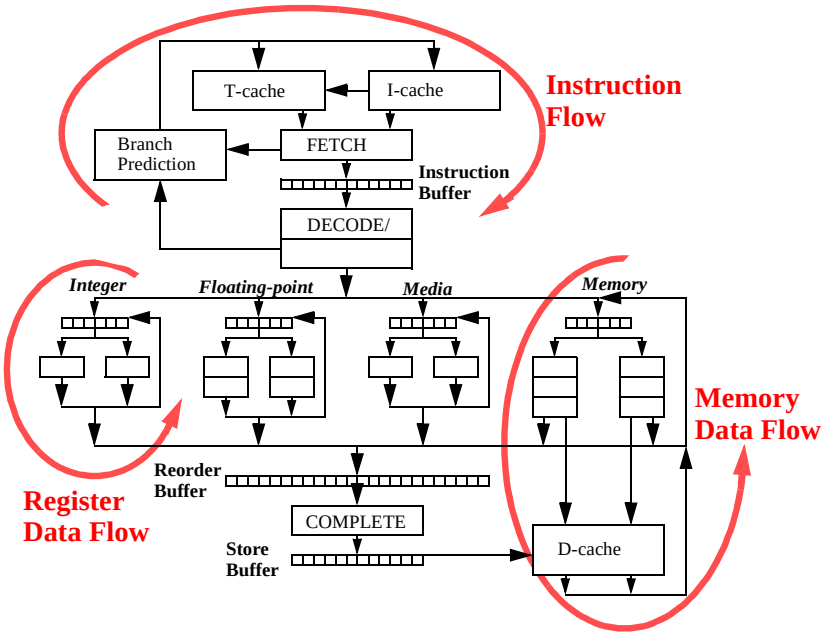
- Selective Prediction
- Register-based Prediction
- Compiler Assistance

Extensions Into Other Domains

- VLIW-based Value Prediction
- Dynamic Instruction Reuse
- Aggressive Partial Evaluation

Carnegie Mellon

Instruction Supply Problem

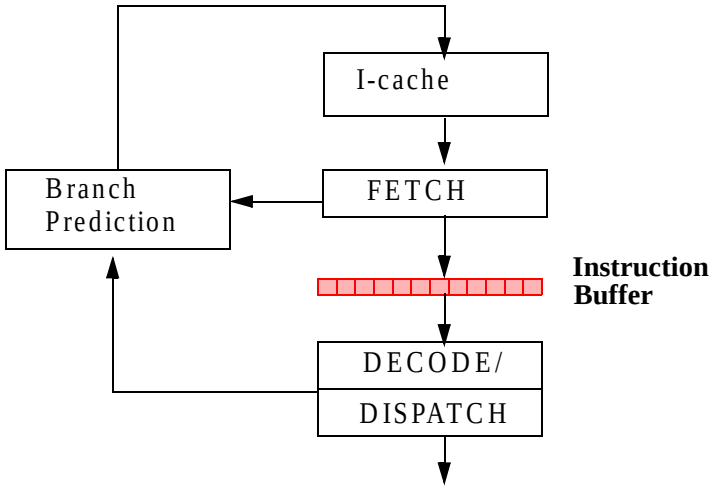


Carnegie Mellon

Wide-Machine Instruction Fetch

Three Major Challenges:

1. Multiple-Branch Prediction
2. Multiple Fetch Groups
3. Alignment and Collapsing



High-bandwidth Instruction Fetch

Fetch Time

Enhanced Instruction Cache

- Multi-branch prediction
- Multi-ported instruction cache fetch
- Instructions align & collapse

Trace Cache

- Next trace prediction
- Trace cache fetch

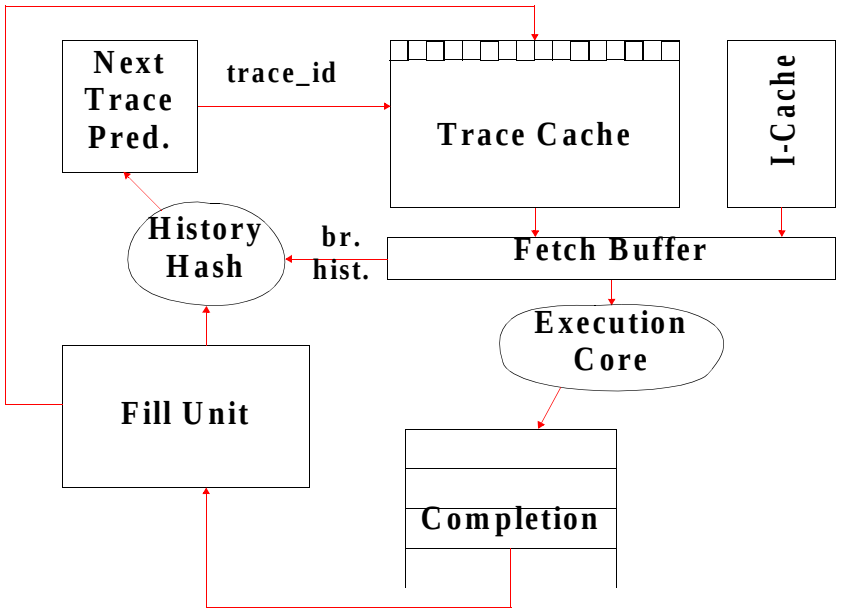
Execution Core | **Execution Core**

Completion Time

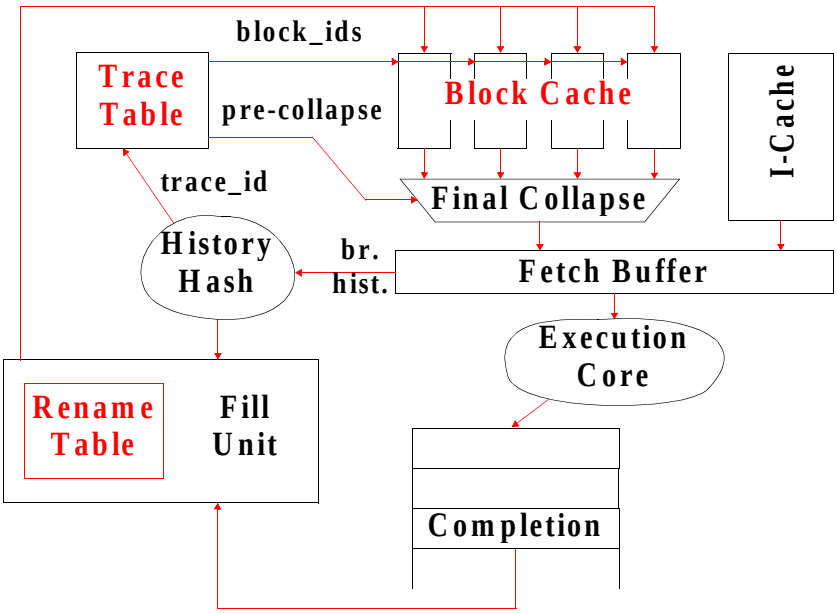
- Branch predictor update

- Instructions align & collapse
- Trace construction
- Trace cache fill
- Trace predictor update

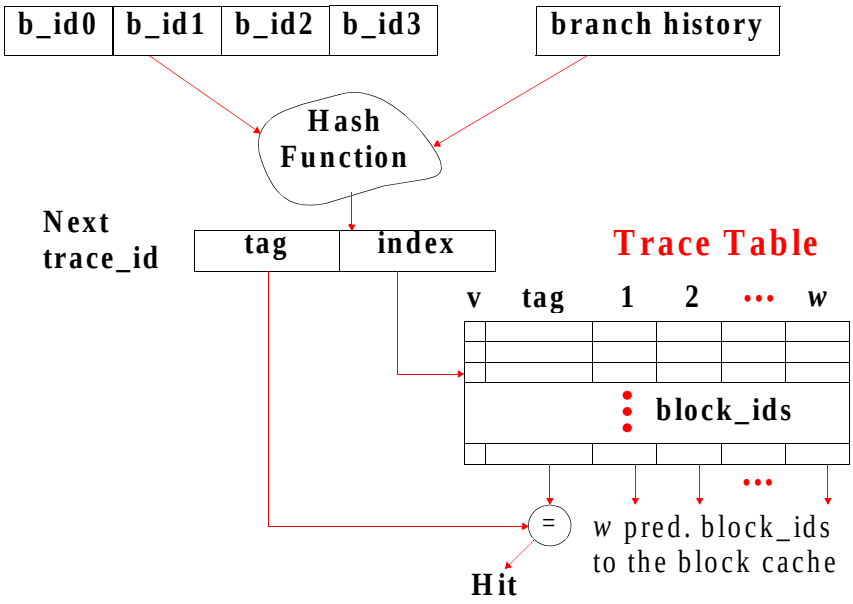
Conventional Trace Cache



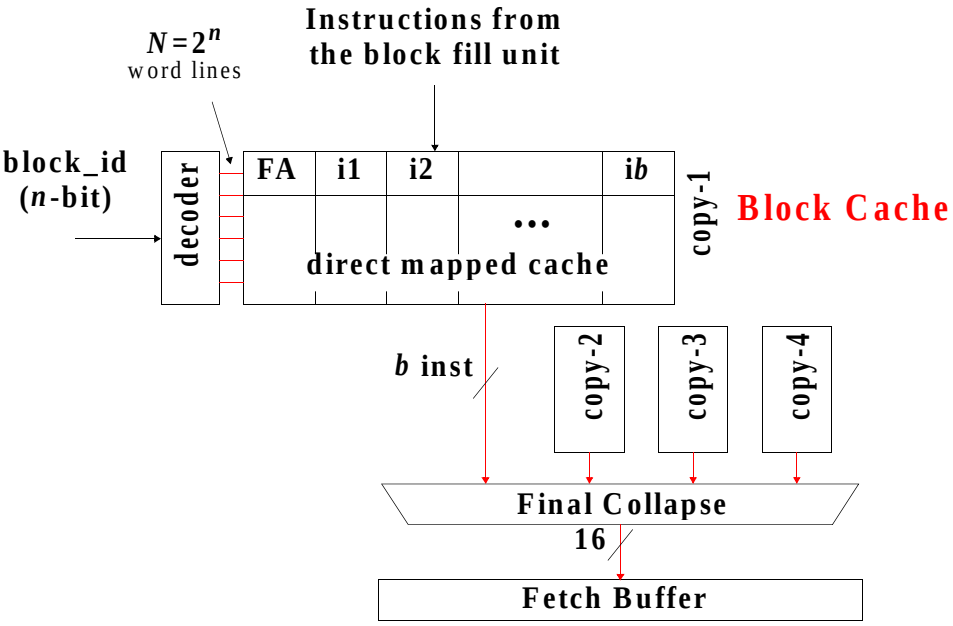
The Block-Based Trace Cache



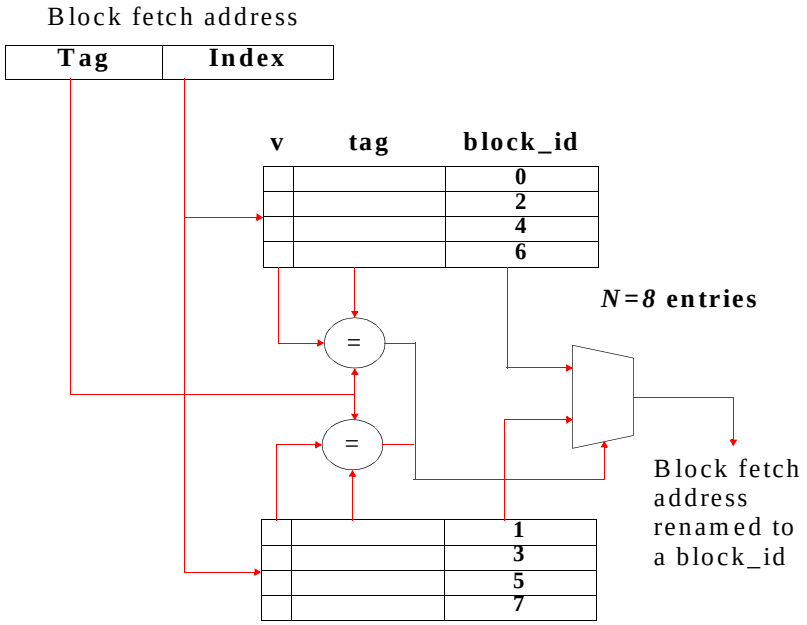
Next Trace Prediction



Replicated Block Cache

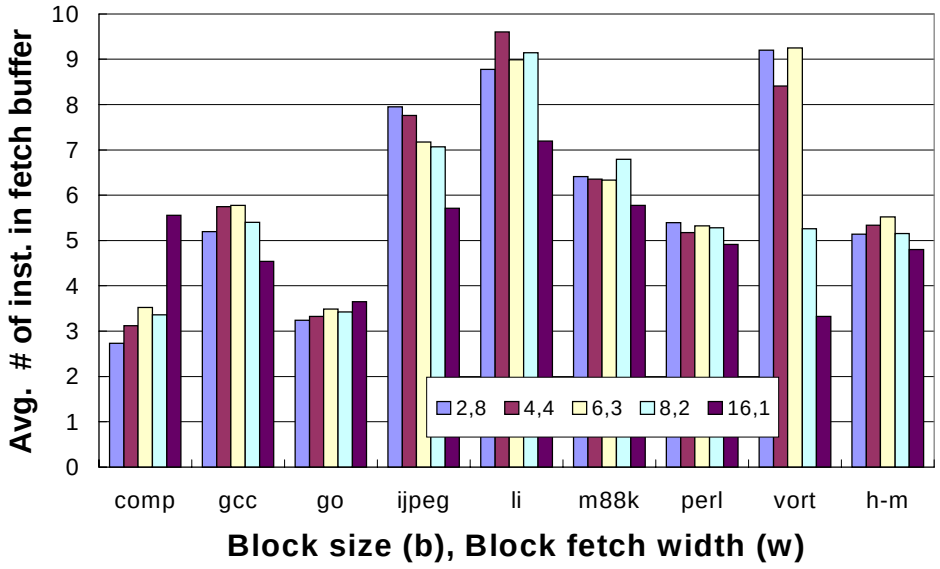


Block Rename Table



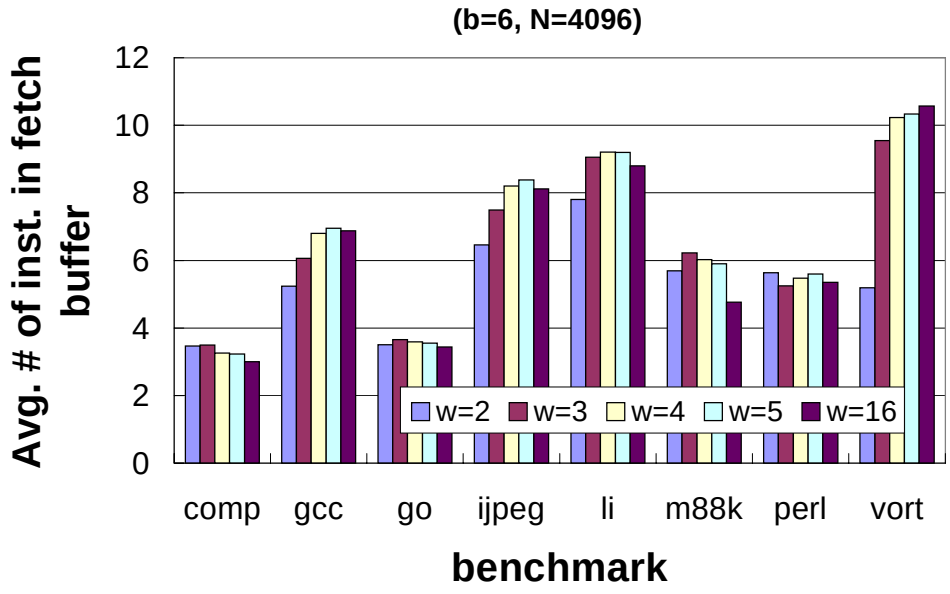
Carnegie Mellon

Block Cache Fragmentation

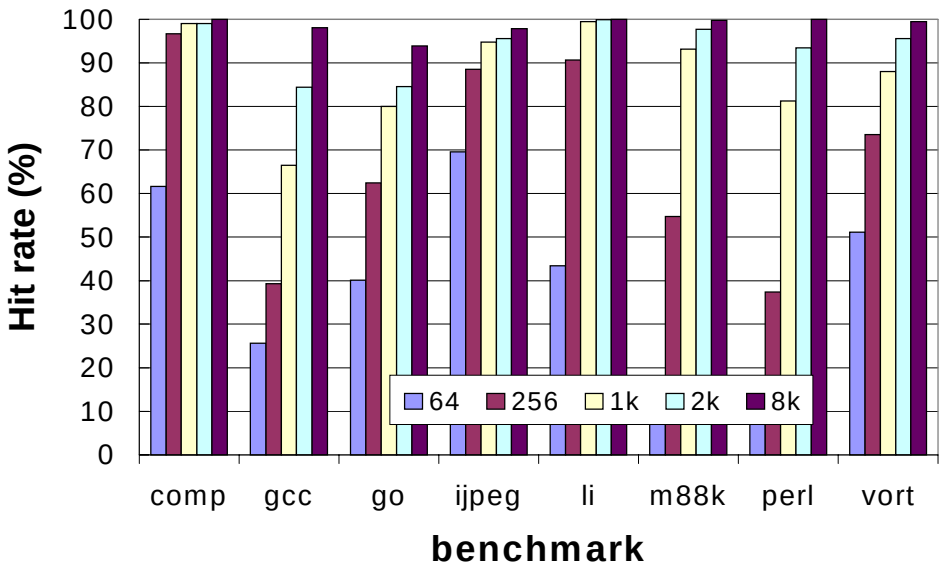


Carnegie Mellon

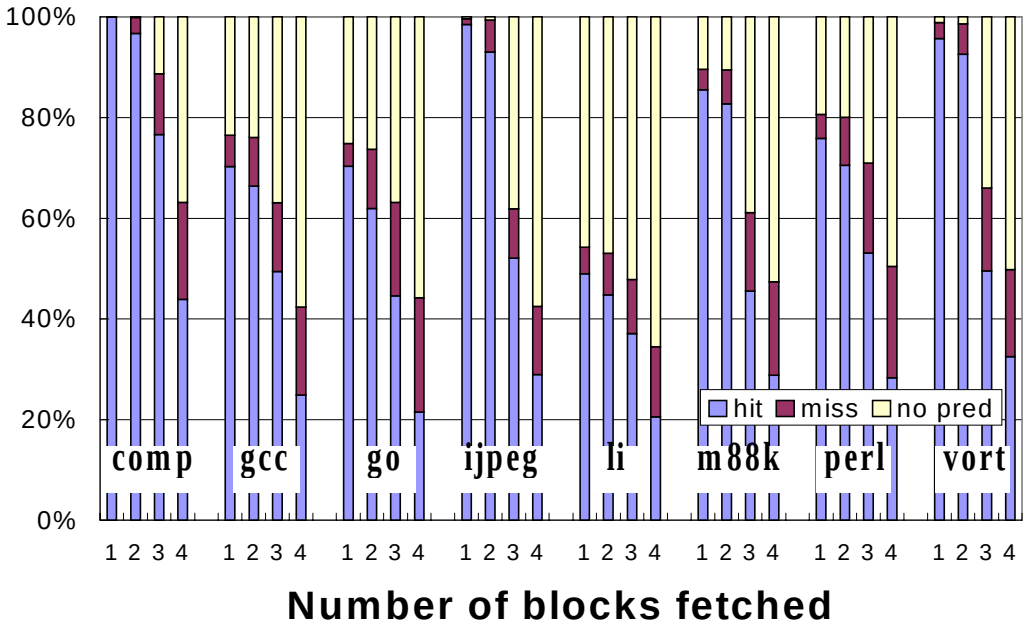
Block Cache Replication



Trace Table Size

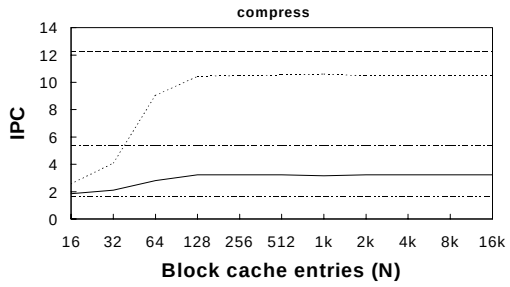


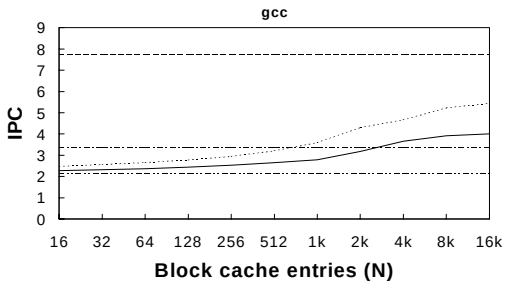
Block Cache Hit Rate

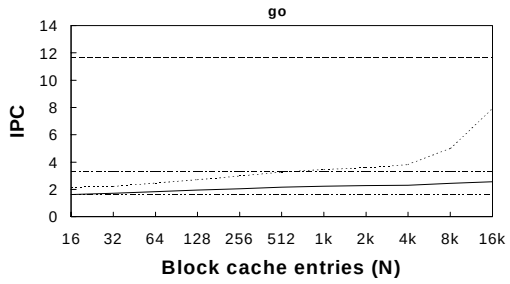


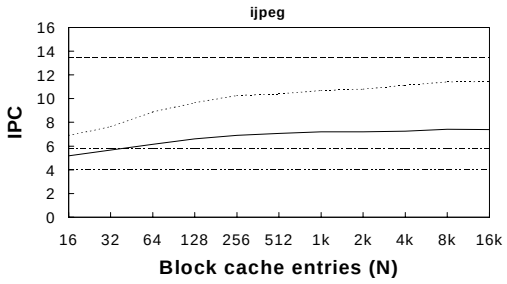
Carnegie Mellon

Performance vs. Block Cache Size



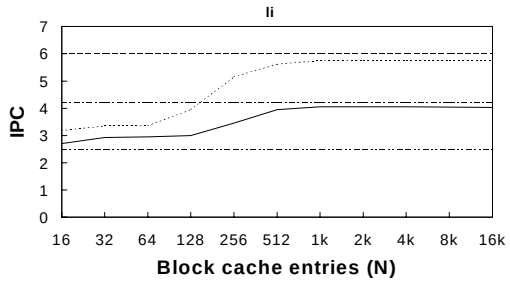


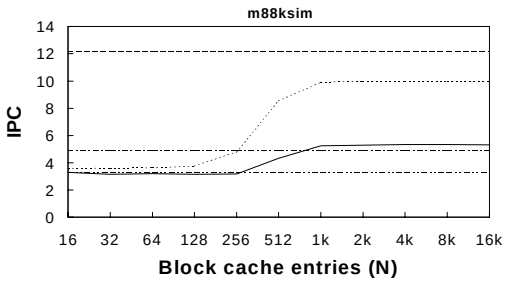


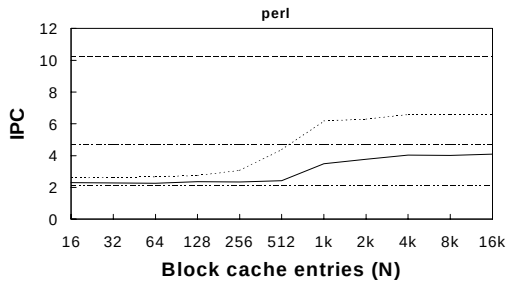


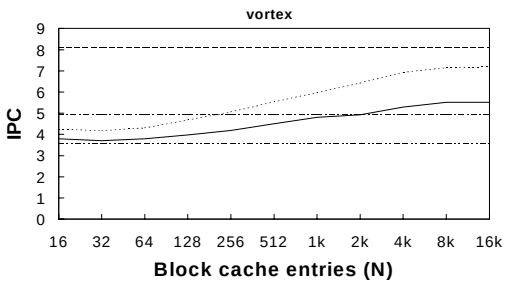
Carnegie Mellon

Performance vs. Block Cache Size

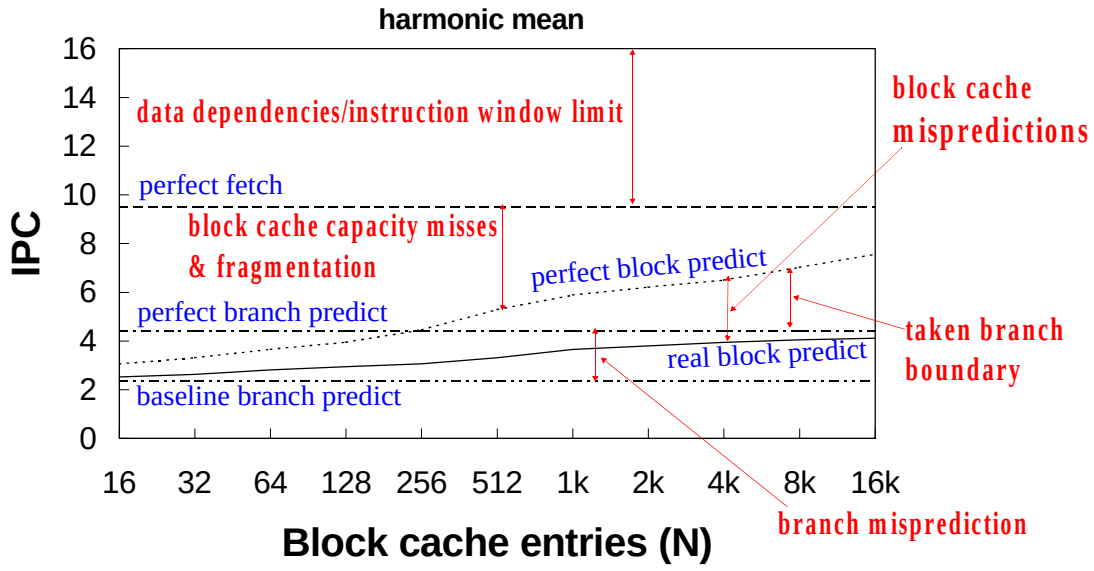




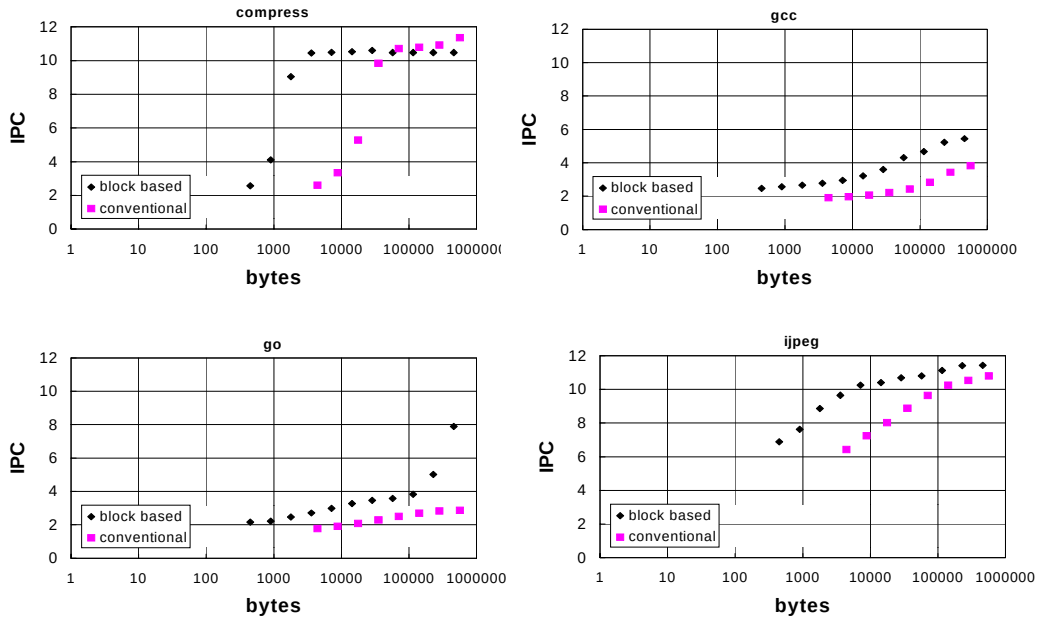




Aggregate IPC vs. Block Cache Size

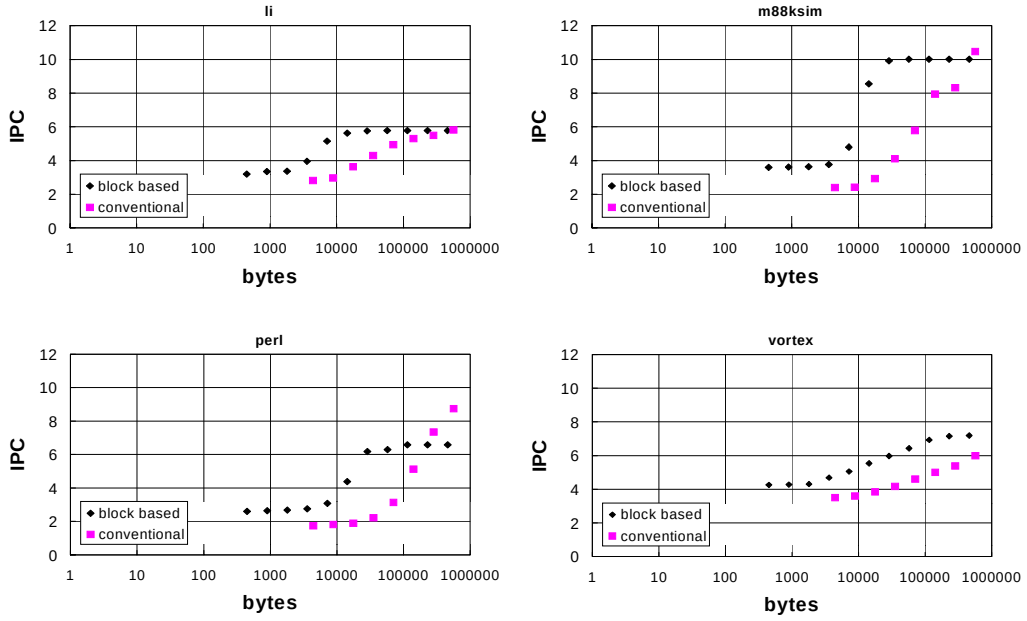


Conventional vs. Block-Based



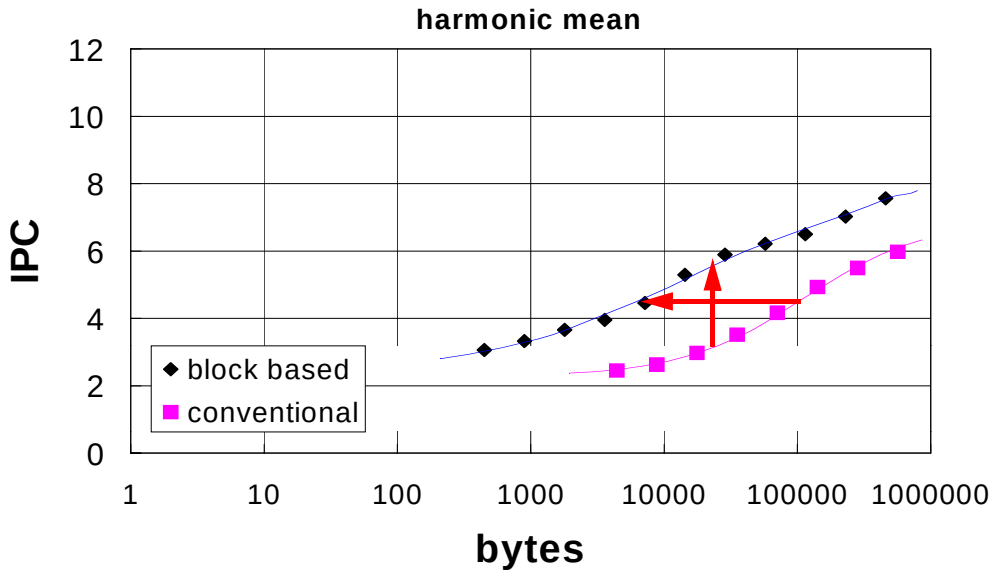
Carnegie Mellon

Conventional vs. Block-Based



Carnegie Mellon

Aggregate Comparison



Carnegie Mellon

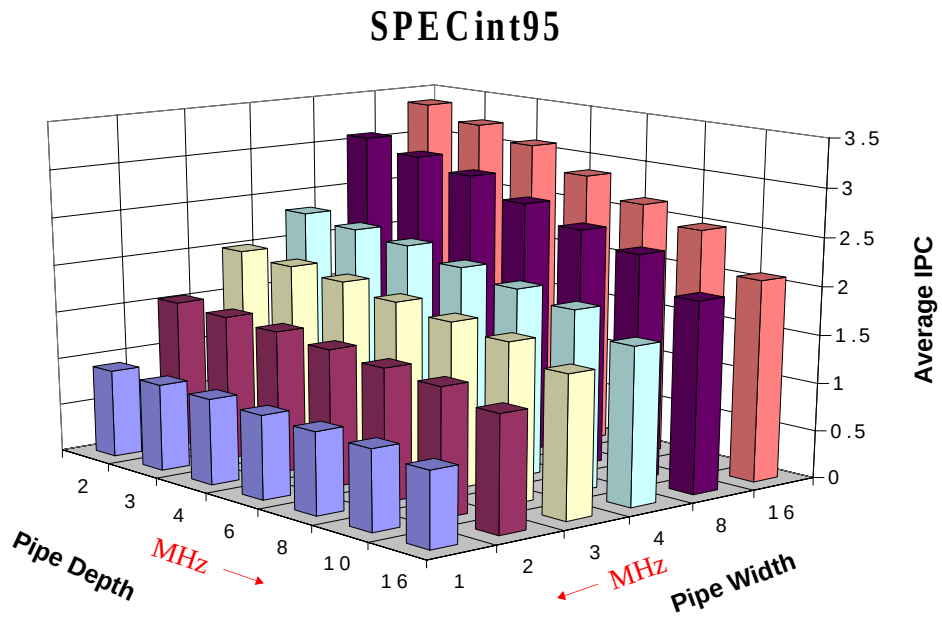
Where Do We Go From Here...

New Forcing Functions:

- **High Frequency Instruction Level Parallelism**
  - **Simultaneously optimize IPC and MHz**
- **Silicon and Time Efficient Performance (STEP)**
  - **Increase both IPC/die_area and δ(IPC)/δ(year)**
- **Dynamic/Static and Hardware/Software fusion**
  - **Reconcile generational latencies of H/W and S/W**
  - **Bridge compiled object code & machine executable**

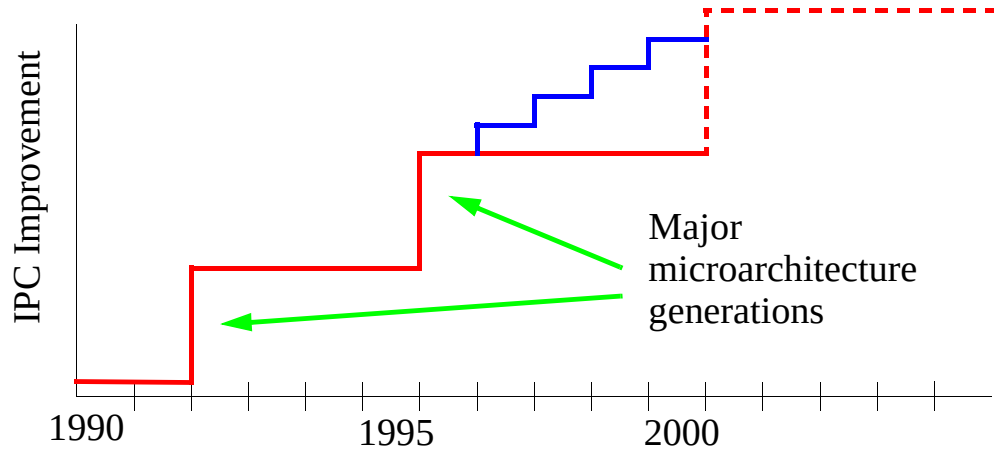
Carnegie Mellon

How to have your cake and eat it too



Carnegie Mellon

Journey of Many Small Steps



Need to Increase δ(IPC)/δ(year)

Carnegie Mellon

Get the Best of Both Worlds

Microarchitecture generational latency: 4-5 years
Compiler generational latency: 8-10 years (?)