A Brief History of Speculation

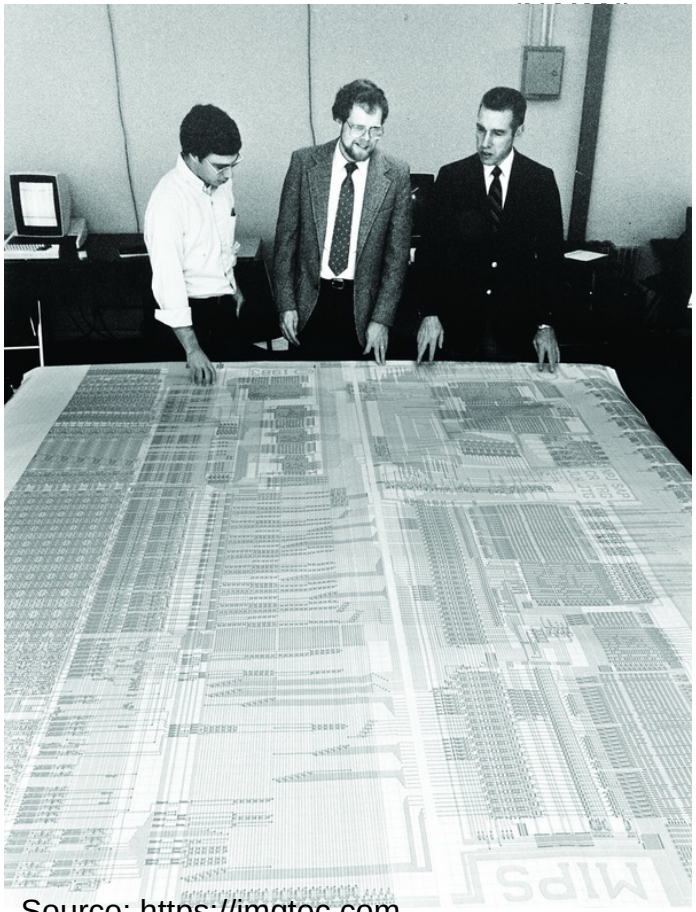
Based on 2017 Test of Time Award Retrospective for
"Exceeding the Dataflow Limit via Value Prediction"

Mikko Lipasti

University of Wisconsin-Madison

Pre-History, circa 1986

Stage	Phase	Function performed
IF	φ_1	Translate virtual instr. addr. using TLB
	φ_2	Access I-cache
RD	φ_1	Return instruction from I-cache, check tags & parity
	φ_2	Read RF; if branch, generate target
ALU	φ_1	Start ALU op; if branch, check condition
	φ_2	Finish ALU op; if ld/st, translate addr
MEM	φ_1	Access D-cache
	φ_2	Return data from D-cache, check tags & parity
WB	φ_1	Write RF
	φ_2	



Source: https://imgtec.com

- MIPS R2000, ~“most elegant pipeline ever devised” J. Larus
- No speculation of any kind

Iron Law

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

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

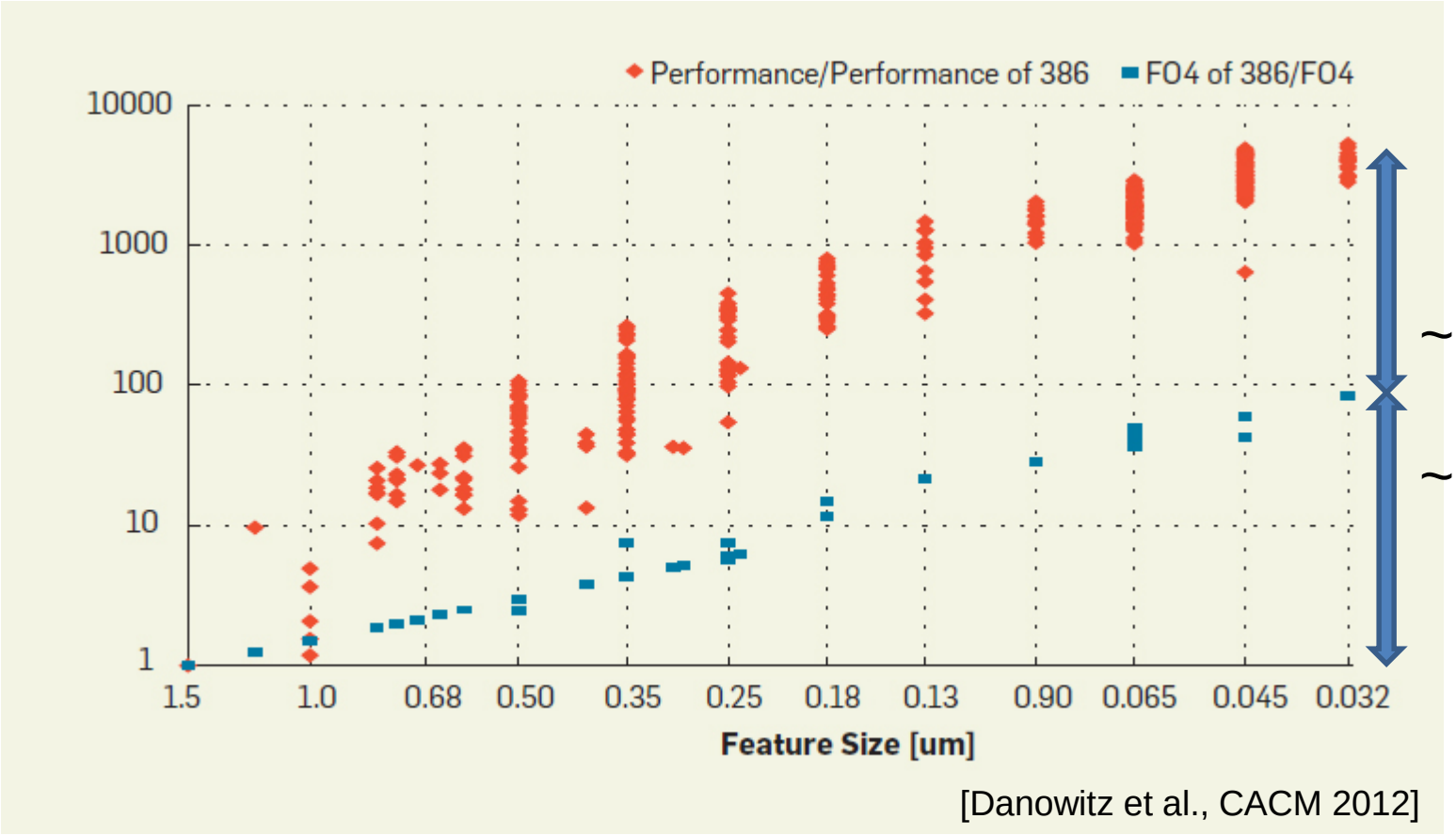
(code size) (CPI) (cycle time)

Microarchitecture

Architecture --> Implementation --> Realization

Compiler Designer **Processor Designer** **Chip Designer**

Performance Benefit of Microarchitecture?



[Danowitz et al., CACM 2012]

High-IPC Processor Evolution

Desktop/Workstation Market

Scalar RISC Pipeline	2-4 Issue In-order	Limited Out-of-Order	Large ROB Out-of-Order
1980s: MIPS SPARC Intel 486	Early 1990s: IBM RIOS-I Intel Pentium	Mid 1990s: PowerPC 604 Intel P6	2000s: DEC Alpha 21264 IBM Power4/5 AMD K8

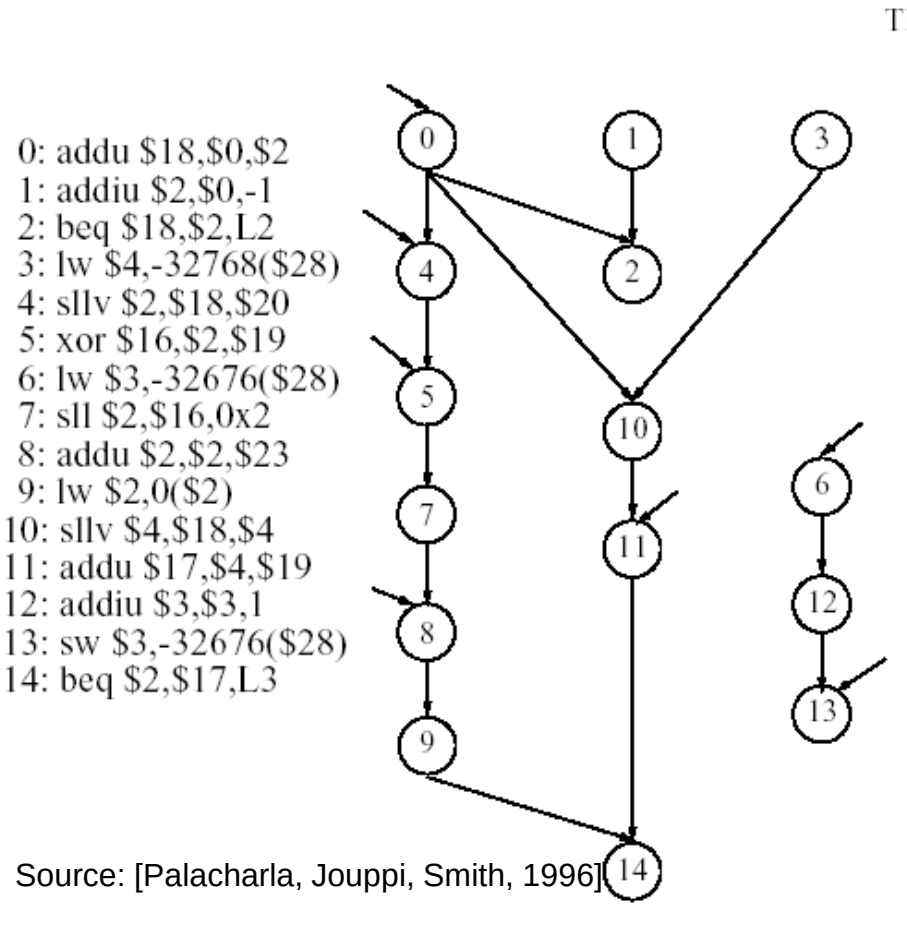
1985 – 2005: 20 years, 100x frequency

Mobile Market

Scalar RISC Pipeline	2-4 Issue In-order	Limited Out-of-Order	Large ROB Out-of-Order
2002: ARM11	2005: Cortex A8	2009: Cortex A9	2011: Cortex A15

2002 – 2011: 10 years, 10x frequency

What Does a High-IPC CPU Do?



Source: [Palacharla, Jouppi, Smith, 1996]

1. Fetch and decode
2. Construct data dependence graph (DDG)
3. Evaluate DDG
4. Commit changes to program state

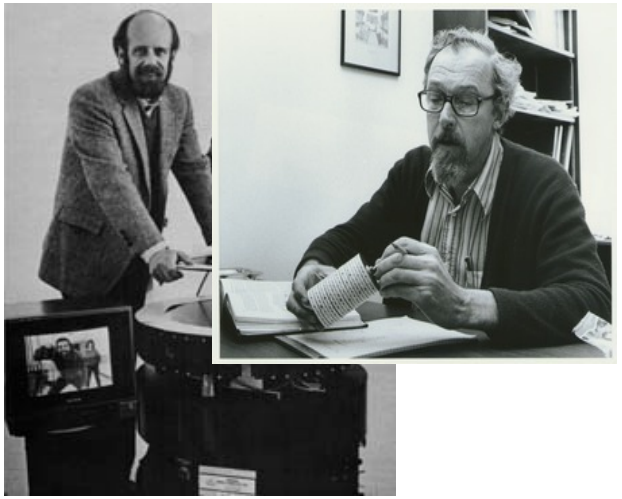
Limits on Instruction Level Parallelism (ILP)

1970: Flynn

Weiss and Smith [1984]	1.58
Sohi and Vajapeyam [1987]	1.81
Tjaden and Flynn [1970]	1.86 (Flynn's bottleneck)
Tjaden and Flynn [1973]	1.96
Uht [1986]	2.00
Smith et al. [1989]	2.00
Jouppi and Wall [1988]	2.40
Johnson [1991]	2.50
Acosta et al. [1986]	2.79
Wedig [1982]	3.00
Butler et al. [1991]	5.8
Melvin and Patt [1991]	6
Wall [1991]	7 (Jouppi disagreed)
Kuck et al. [1972]	8
Riseman and Foster [1972]	51 (no control dependences)
Nicolau and Fisher [1984]	90 (Fisher's optimism)

Riseman and Foster's Study

1970: Flynn
1972: Riseman/Foster



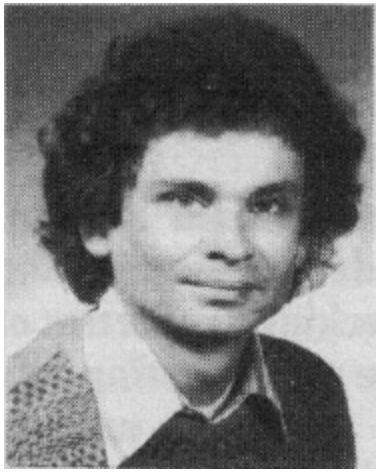
- 7 benchmark programs on CDC-3600
- Assume infinite machines
  - Infinite memory and instruction stack
  - Infinite register file
  - Infinite functional units
  - True dependencies only at dataflow limit
- If bounded to single basic block, speedup is 1.72 (Flynn's bottleneck)
- If one can bypass n branches (hypothetically), then:

Branches Bypassed	0	1	2	8	32	128	∞
Speedup	1.72						

Branch Prediction

1970: Flynn
1972: Riseman/Foster

1979: Smith Predictor



- Riseman & Foster showed potential
  - But no idea how to reap benefit
- 1979: Jim Smith patents branch prediction at Control Data
  - Predict current branch based on past history
- Today: virtually all processors use branch prediction

State of the art: Neural vs. TAGE

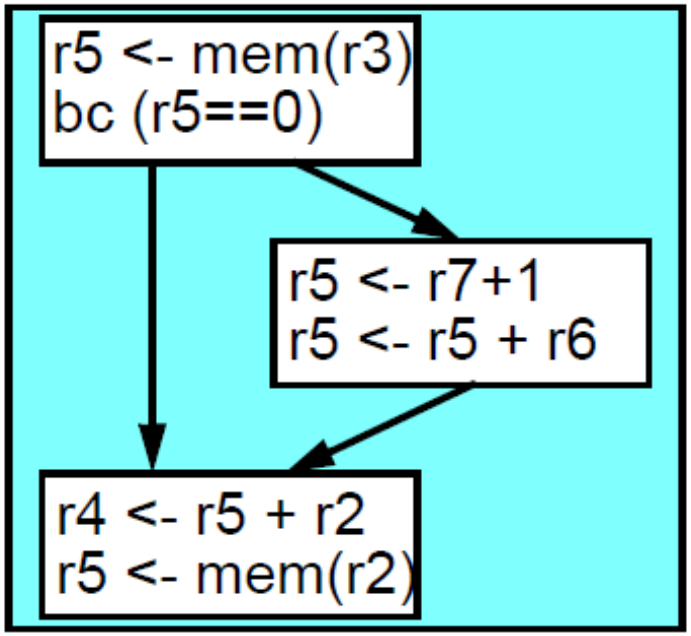
1970: Flynn
1972: Riseman/Foster
1979: Smith Predictor
1991: Two-level prediction
1993: gshare, tournament
1996: Confidence estimation
1996: Vary history length
1998: Cache exceptions
2001: Neural predictor
2004: PPM
2006: TAGE
2016: Still TAGE vs Neural

- Neural: AMD, Samsung
- TAGE: Intel?, ARM?
- Similarity
  - Many sources or “features”
- Key difference: how to combine them
  - TAGE: Override via partial match
  - Neural: integrate + threshold
- Every CBP is a cage match
  - Andre Seznec vs. Daniel Jimenez

Carnegie Mellon

ILP Limits

- ILP has two fundamental restrictions:
  - Control Flow
  - Data Flow
- Dealing with Control Flow:
  - Branch prediction
  - Speculative fetch
  - Speculative dispatch
- Dealing with Data Flow:
  - Removing false dependences
  - Reducing result latency
  - Collapsing ALUs
  - Removing true dependences (value prediction)
- Dependences must be detected and enforced.



Dependence Speculation, Collapsing

- Speculative disambiguation
  - Compile-time, e.g. [Huang et al., ISCA'94]
  - Later, Transmeta VLIW
- Famously, hardware prediction
  - Moshovos, Breach, Sohi patent

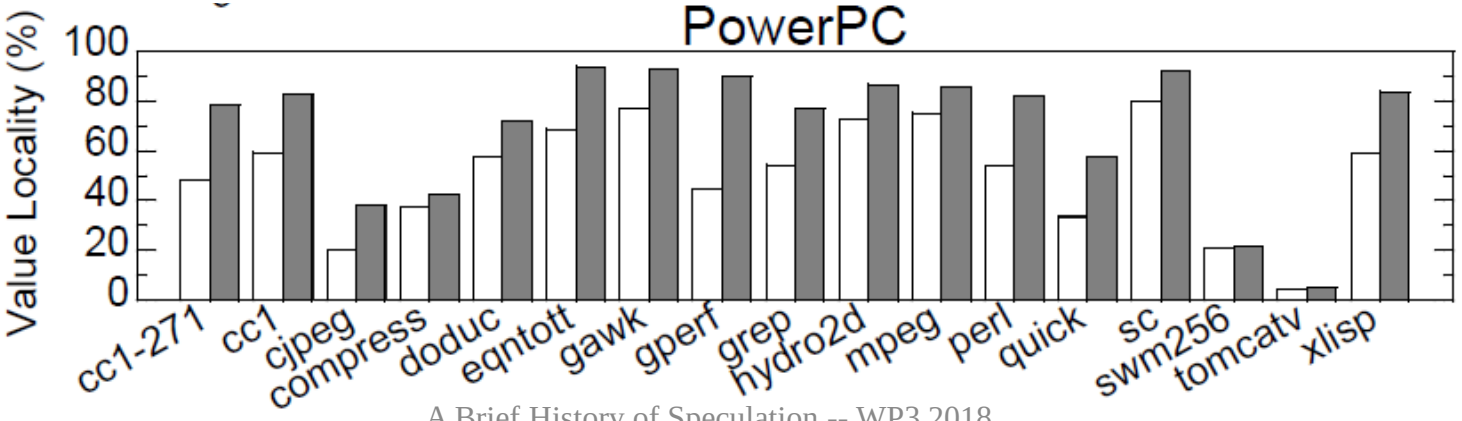
- Dependence collapsing
  - Collapsing ALUs, e.g. [Vassiliadis et al. '93]

Address Speculation

- Prior and concurrent work, e.g.
  - Stride prediction [Eickemeyer, Vassiliadis'93]
  - Zero cycle loads [Austin, Sohi '95]
  - Address prediction [Sazeides et al., '96]

Value Locality

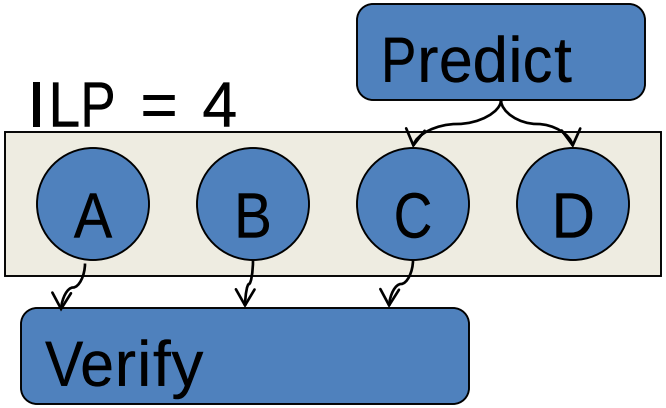
- Third dimension of locality
  - "There's a lot of zeroes out there." (C. Wilkerson)
  - Program tracing tools made values visible
    - It wasn't just zeroes
- Results of computation quite predictable
  - 50% of loads fetch same value as last instance
  - 40% of all instructions write same register value

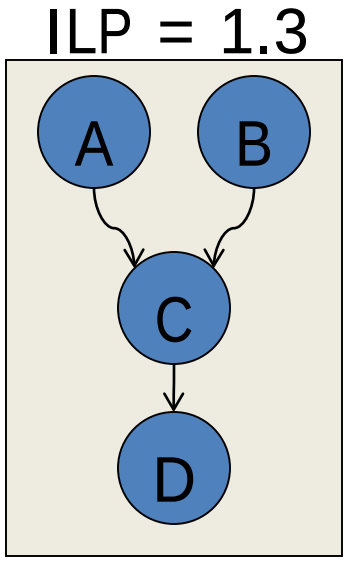


Causes of Value Locality

- Likelihood of same or similar values occurring repeatedly
- Why might this happen?
  - Data redundancy
  - Error checking
  - Program constants
  - Computed branches
  - Virtual function calls
  - Addressability
  - Call-subgraph identities
  - Memory alias resolution
  - Register spill code
  - Convergent algorithms
  - Polling algorithms
  - Etc.
- Programs are written to be general-purpose, error tolerant
- Compilers have to “play it safe”

Value Prediction





What is value prediction?

1. Generate a speculative value (predict)
2. Consume speculative value (execute)
3. Verify speculative value (compare/recover)

Goal: performance, i.e. expose more ILP

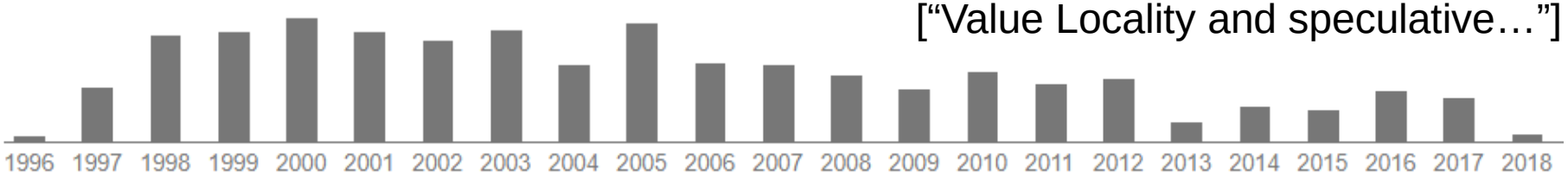
Some History

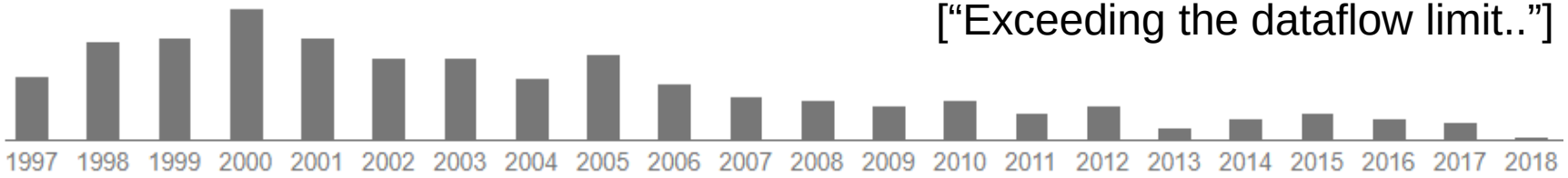
- "Classical" value prediction
  - Independently invented by 4 groups in 1995-1996
  1. AMD (Nexgen): L. Widigen and E. Sowadsky, patent filed March 1996, inv. March 1995
  2. Technion: F. Gabbay and A. Mendelson, inv. sometime 1995, TR 11/96, US patent Sep 1997
  3. CMU: M. Lipasti, C. Wilkerson, J. Shen, inv. Oct. 1995, ASPLOS paper submitted March 1996, MICRO June 1996
  4. Wisconsin: Y. Sazeides, J. Smith, Summer 1996

Why?

- Possible explanations:
  1. Natural evolution from branch prediction
  2. Natural evolution from memoization
  3. Natural evolution from rampant speculation
     - Cache hit speculation
     - Memory independence speculation
     - Speculative address generation
  4. **Improvements in tracing/simulation technology**
     - Values, not just instructions & addresses
       - TRIP6000 [A. Martin-de-Nicolas, IBM]

Citations by Year [scholar.google.com]





- ASPLOS paper has 786 citations, MICRO has 604
- Waxing and waning

Flurry of Advances (1)

- Predictor design, some examples
  - Stride [Gabbay/Mendelson'97]
  - 2-level [Wang/Franklin'97]
  - Last-n value [Burtscher/Zorn'99]
  - Finite Context Method [Sazeides/Smith'97]
  - Hybrid [Rychlik et al.'98][Burtscher/Zorn'02]
  - Block level [Huang/Lilja'99]
  - Storageless [Tullsen/Seng'99]
  - Global history [Zhou et al.'03]

Flurry of Advances (2)

- Software methods
  - Value Profiling [Calder et al.'97]
  - Compiler implementation [Fu et al.,'98][Larson/Austin'00]
  - Trace compression [Burtscher/Jeradit'03]
- Microarchitectural utilization
  - Selective [Calder et al.'99]
  - Critical path [Fields et al.,'01]
  - Recovery-free [Zhou/Conte'05]
  - L2 misses only [Ceze et al.'06]

What Happened?

- Considerable academic interest
  - Dozens of research groups, papers, proposals
- No industry uptake so far
  - Intel (x86), IBM (PowerPC), HAL (SPARC) all failed
- Why?
  - Modest performance benefit (< 10%)
  - Power consumption
    - Dynamic power for extra activity
    - Static power (area) for prediction tables
  - Complexity and correctness (risk)
    - Subtle memory ordering issues [MICRO ’01]
    - Misprediction recovery [HPCA ’04]

Performance?

- Relationship between timely fetch and value prediction benefit [Gabbay/Mendelson, ISCA'98]

Value prediction doesn't help when the result can be computed before the consumer instruction is fetched

- Accurate, high-bandwidth fetch helped
  - Wide trace caches studied in late 1990s
  - Late Ph.D. work looked at this
  - Much better branch prediction today (neural, TAGE)
- Industry was pursuing frequency, not ILP (GHz race)
  - Value Prediction got lost in the mix

Future Adoption?

- Promising trends
  - Deep pipelining, high frequency mania is over
  - Standard techniques have hit asymptotes
    - Bigger IQ/ROB/LSQ, more ALUs, more LD/ST ports
    - Better branch prediction, better prefetching
    - Not much opportunity left
- Bag of microarchitectural tricks is nearly empty
  - Value prediction may have another opportunity
  - Rumors of 4 design teams considering it as a “kicker”
- Much more benefit in spatial dataflow designs
  - Not currently popular

Some Recent Interest (1)

- VTAGE [Perais/Seznec, HPCA 14]
  - Solves many practical problems in the predictor
  - Inspired by IT-TAGE (indirect branch predictor)
  - Good coverage, very high confidence
    - Uses probabilistic up/down counters [Riley/Zilles'06]
  - No need for selective recovery
- EOLE [Perais/Seznec, ISCA 14]
  - Value predicted operands reduce need for OoO
  - Execute some ops early, some late, outside OoO
  - Smaller, faster OoO window

Some Recent Interest (2)

- Load Value Approximation

  [San Miguel/Badr/Enright Jerger, MICRO-47][Thwaites et al., PACT 2014]

- DLVP [Sheikh/Cain/Damodaran, MICRO-50]
  - Predicts addresses, accesses cache to predict values
- Compiler optimization effects [Endo et al.'17]
- GPUs [Sun/Kaeli'14]

If not value prediction, then…

- Value prediction presented some unique challenges:
  - Relatively low correct prediction rate (initially 40-50%)
  - Nontrivial misprediction rate with misprediction cost
- Confidence estimation
  - First practical application of confidence estimation
  - Focus area of early work, led to advances
- Selective recovery
  - Initial paper compared squash vs. selective recovery
  - Brute-force recovery (squash) not sufficient
    - EOLE work argues that better confidence estimation fixes this

If not value prediction, then…

- Focus on value-aware datapaths
  - Compression, encodings, operand significance
  - Newly resurgent in NN accelerators
- Value-aware memory system design
  - Silent stores, temporally silent stores, SLE, TM
  - Value-based replay, SVW, NoSQ
  - Advanced prefetchers

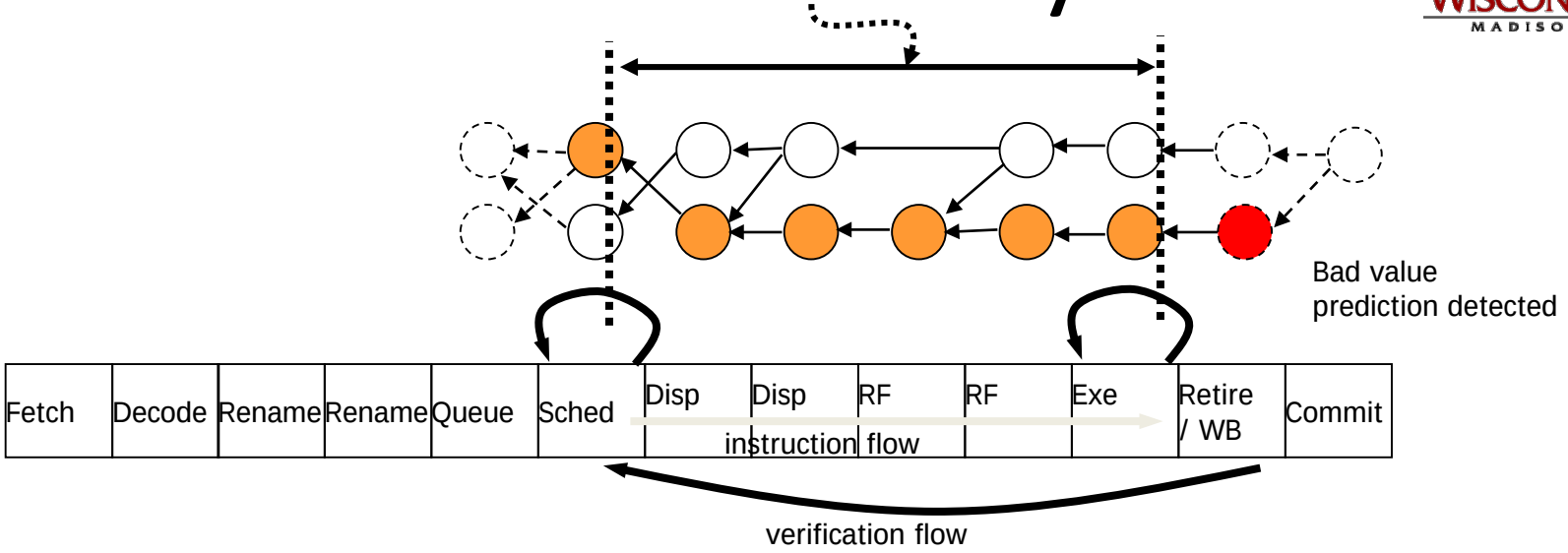
Remainder of Talk

- Selective recovery

- Value-aware memory system design
  - Silent stores, temporally silent stores,
  - Speculative Lock Elision, TM
  - Value-based replay, SVW, NoSQ

- Spectre

Selective Recovery



- Bad load (cache miss, incorrect value prediction) pollutes DFG
- Must identify *transitive closure of DFG*, e.g. *forward load slice*
  - Slice instructions could be anywhere in the back end
- In Ph.D. work, used bit vectors (1 bit/predicted value)
  - Propagated bit vectors to dependent ops in rename stage
  - Mispredicted op broadcasts ID, all ops with matching bit set replay

Runahead Execution

- Proposed by [Dundas/Mudge'97]
  - Used poison bit to identify miss-dependent *forward load slice*
- Checkpoint state, keep running beyond miss
- When miss completes, return to checkpoint
  - May need runahead cache for store/load communication [Mutlu et al, HPCA'03]
- Goal: expose *memory-level parallelism* by triggering subsequent cache misses
- Aside: later combined with LVP [Zhou/Conte'05]

Waiting Instruction Buffer

[Lebeck et al. ISCA 2002]

- Capture forward load slice in separate buffer
  - Propagate poison bits to identify slice
- Relieve pressure on issue queue
- Reinsert instructions when load completes
- Very similar to Intel Pentium 4 replay mechanism
  - But not publicly known at the time

WIB-like Recovery

- Enabled speculation mindset
  - Particularly among Intel Pentium 4 design team
  - Convenient, catch-all recovery mechanism
- Many forms of speculation
  - Cache hit/miss (7 cycles?), alignment, memory dependence, TLB miss, *access permissions*
- Tornado: same dep. chains issued many times! [Liu et al. ICS'05]
- Missed key requirement!
  - Parallel recovery (faster than issue) [HPCA'04]

Remainder of Talk

- *Selective recovery*

- Value-aware memory system design
  - Silent stores, temporally silent stores,
  - Speculative Lock Elision, TM
  - Value-based replay, SVW, NoSQ

- Spectre

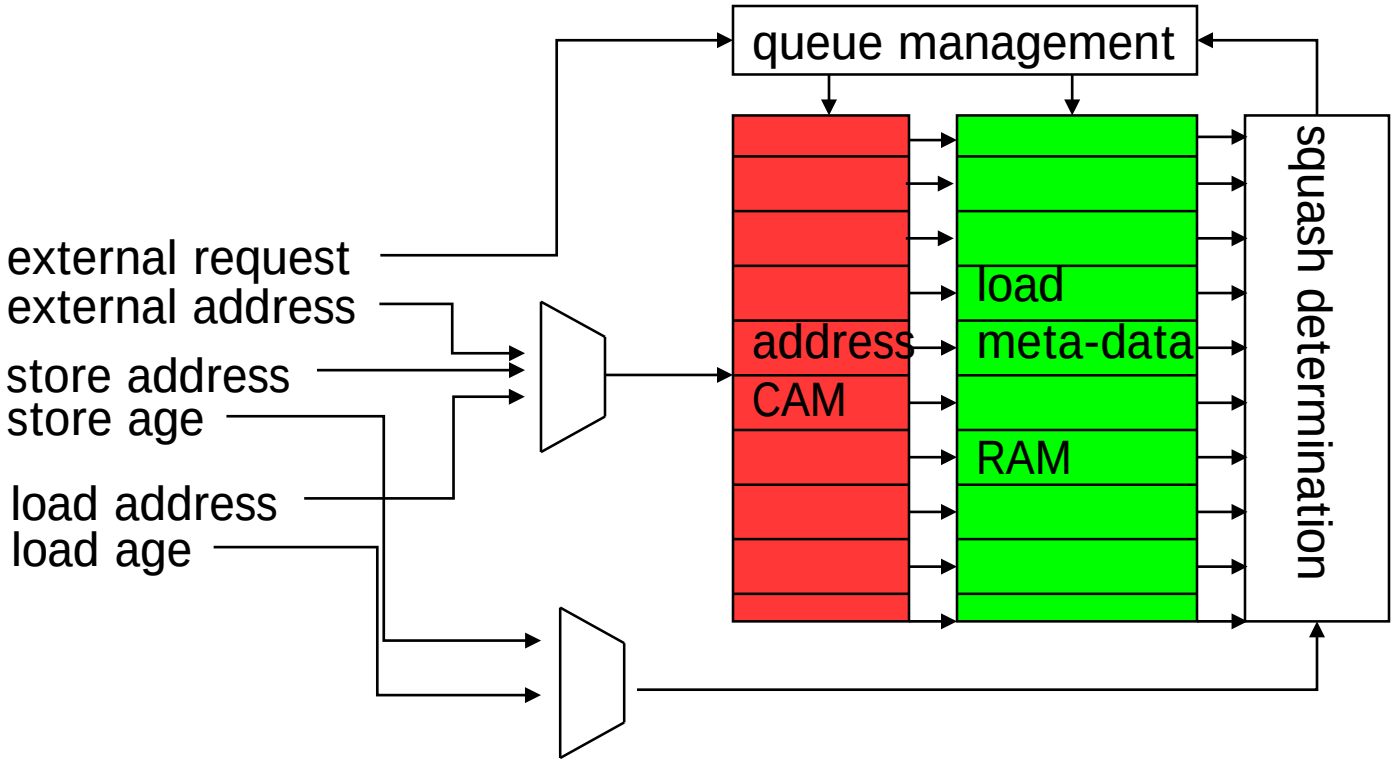
Silent Stores

- Loads and ALU ops redundant => stores also
- Many silent stores [ISCA'00, MICRO'00, PACT'00]
  - At least one IBM design squashes silent stores [Slegel et al. IBMJRD'04]
- Temporally silent stores [ASPLOS'02]
  - Values that change often revert
    - flags, counters, locks, etc.
  - Exploit in coherence domain to minimize traffic

Memory system: Speculative Lock Elision

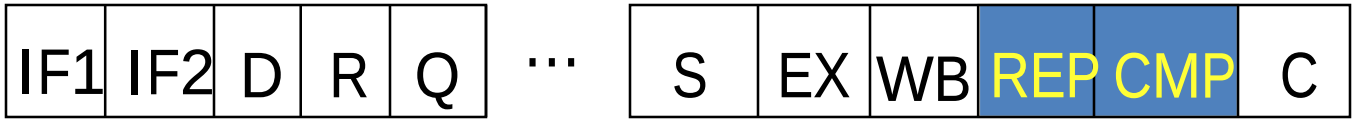
- Suggested as research topic in Fall 1999 at "get to know the faculty" UW seminar talk
  - Followup conversations with Ravi Rajwar
    - Ad hoc advisor while Jim Goodman on sabbatical
  - Led to SLE, Transactional Memory work

Load queue



- # of write ports = load address calc width
- # of read ports = load+store address calc width (+ 1)
- Current generation designs (32-48 entries, 2 write ports, 2 (3) read ports)

Value-based Memory Ordering

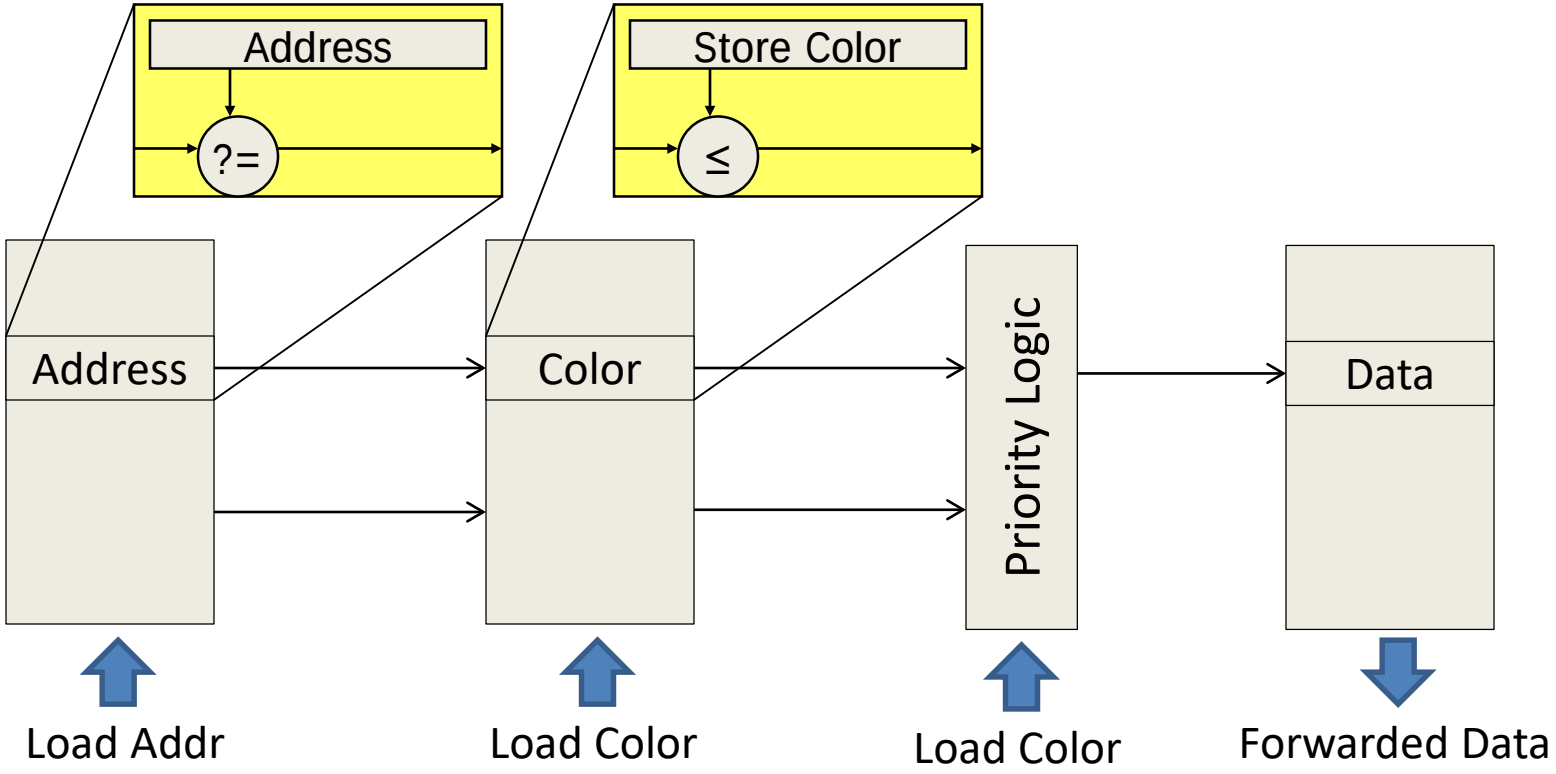


- Replay: access the cache a second time -rarely
  - Almost always cache hit
  - Reuse address calculation and translation
  - Share cache port used by stores in commit stage
- Compare: compares new value to original value
  - Squash if the values differ
- This is value prediction!
  - Predict: access cache prematurely
  - Execute: as usual
  - Verify: replay load, compare value, recover if necessary

Value-based Memory Ordering

- Proposed at ISCA 2004 [Cain/Lipasti]
- Key: clever replay filters
  - Sufficient conditions for avoiding replay
  - Less than 2% of instructions replay
- Goal: !Performance
- Triggered interesting follow-on work

Store Queue Implementation



- Store color assigned at dispatch, increases monotonically
- Load inherits color from preceding store, only forwards if store is older
- Priority logic must find nearest matching store

Store Vulnerability Window (SVW)

[Roth, ISCA 05]

Elegant extension/formalization of replay filters

1. Assign sequence numbers to stores
2. Track writes to cache with sequence numbers
3. Efficiently filter out safe loads/stores by only checking against writes in *vulnerability window*

NoSQ [Sha et al., MICRO 06]

- Rely on load/store alias prediction to directly connect dependent pairs
  - Memory cloaking [Moshovos/Sohi, ISCA'97]
- Use SVW technique to check
  - Replay load only if necessary
  - Train load/store alias predictor
- Similar concurrent proposals
  - DMDC [Castro et al., MICRO 06],
  - Fire-and-forget [Subramanian/Loh, MICRO 06]

Remainder of Talk

- *Selective recovery*

- *Value-aware memory system design*
  - *Silent stores, temporally silent stores,*
  - *Speculative Lock Elision, TM*
  - *Value-based replay, SVW, NoSQ*

- Spectre

Spectre

- Crisis in microarchitecture
  - Speculation leaves behind *cache footprint*
  - Timing side channel leaks privileged state
- Fundamentally hard problem
  - Cannot anticipate all possible side channels
- Places heavy burden on microarchitect
  - Now first-order design constraint
- Solution?
  - Can we redeploy VP recovery techniques?
  - Track microarchitectural state
  - Recover on mispredicts?

Conclusion

- Speculation critical for reaching 100x performance
- Value prediction seems like a promising idea
  - Best Paper Award 1996, Test of Time Award 2017
- Adoption thwarted by design trends, complexity
  - Inspired new research directions with real impact
  - May yet make it into a real design!
- You can help; please participate in CVP-1!
  - Toolkit, traces are available, submissions due 4/1:

https://www.microarch.org/cvp1/

Acknowledgments

First Trinity, Pittsburgh
Spiritual Home & Respite

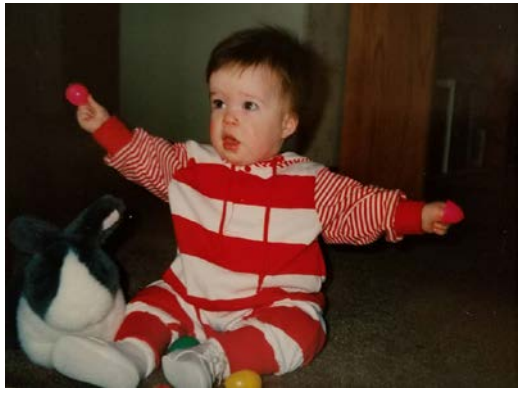
Erica Lipasti
Fount of love and support

No, let's not patent this. Let's publish it!

Prof. John Shen
Advisor, role model

There's a lot of zeroes out there!

Chris Wilkerson
Co-inventor, co-author



Emma Lipasti
Work-life balancer

Arturo Martin-de-Nicolas
Genius Toolmaker