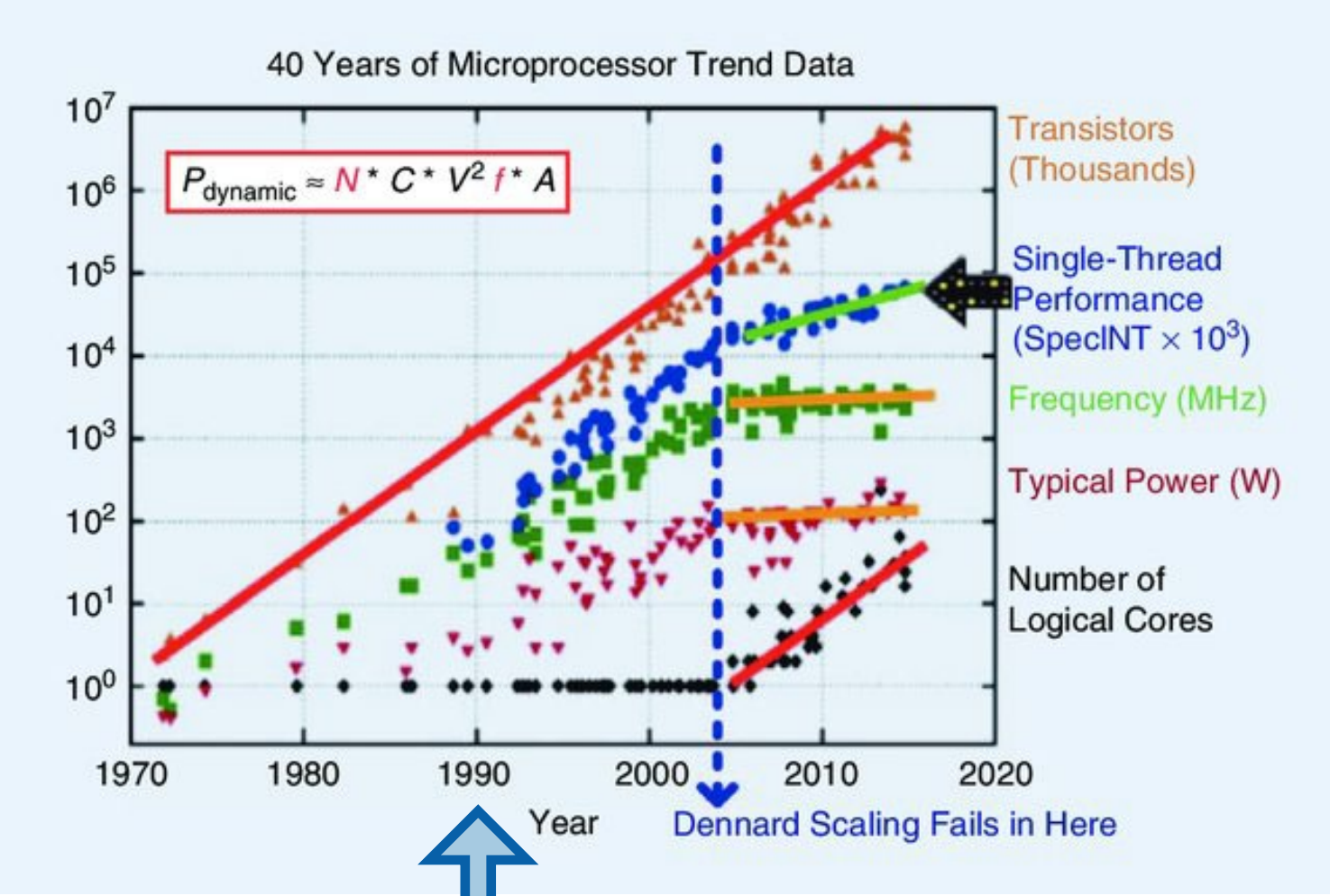
18-742:
Computer Architecture & Systems

WaveScalar

Prof. Phillip Gibbons

Spring 2025, Lecture 12

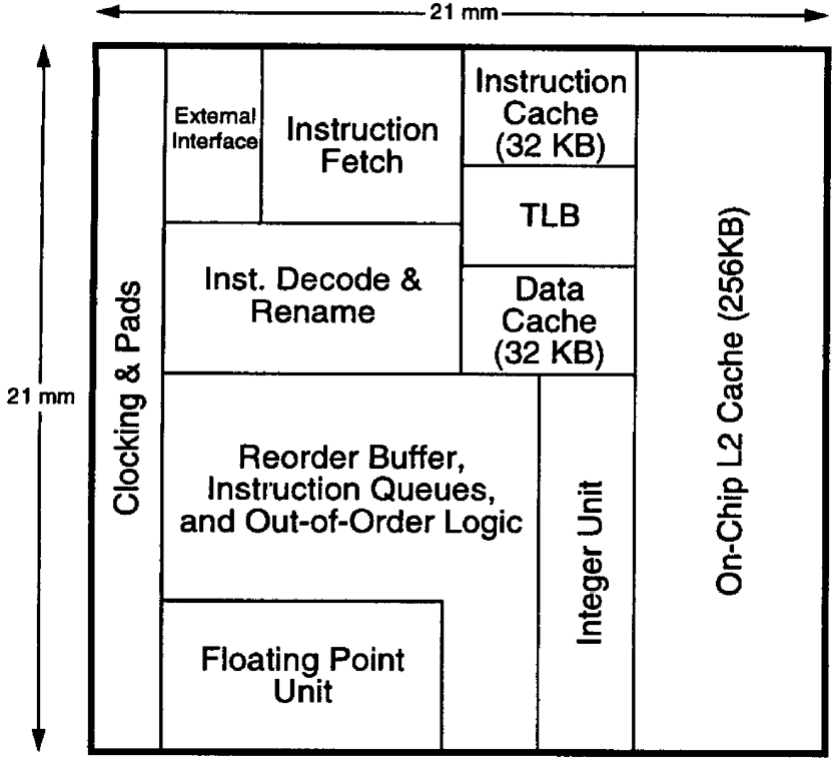
Moore's Law w/o Dennard Scaling



Out-of-Order Superscalars

- **Exploit Instruction-level parallelism**

- **Limitations:**
  - Significant complexity
  - High port requirements in register file & caches
  - Branch prediction must be extremely accurate
  - Inter-instruction dependencies limit application ILP

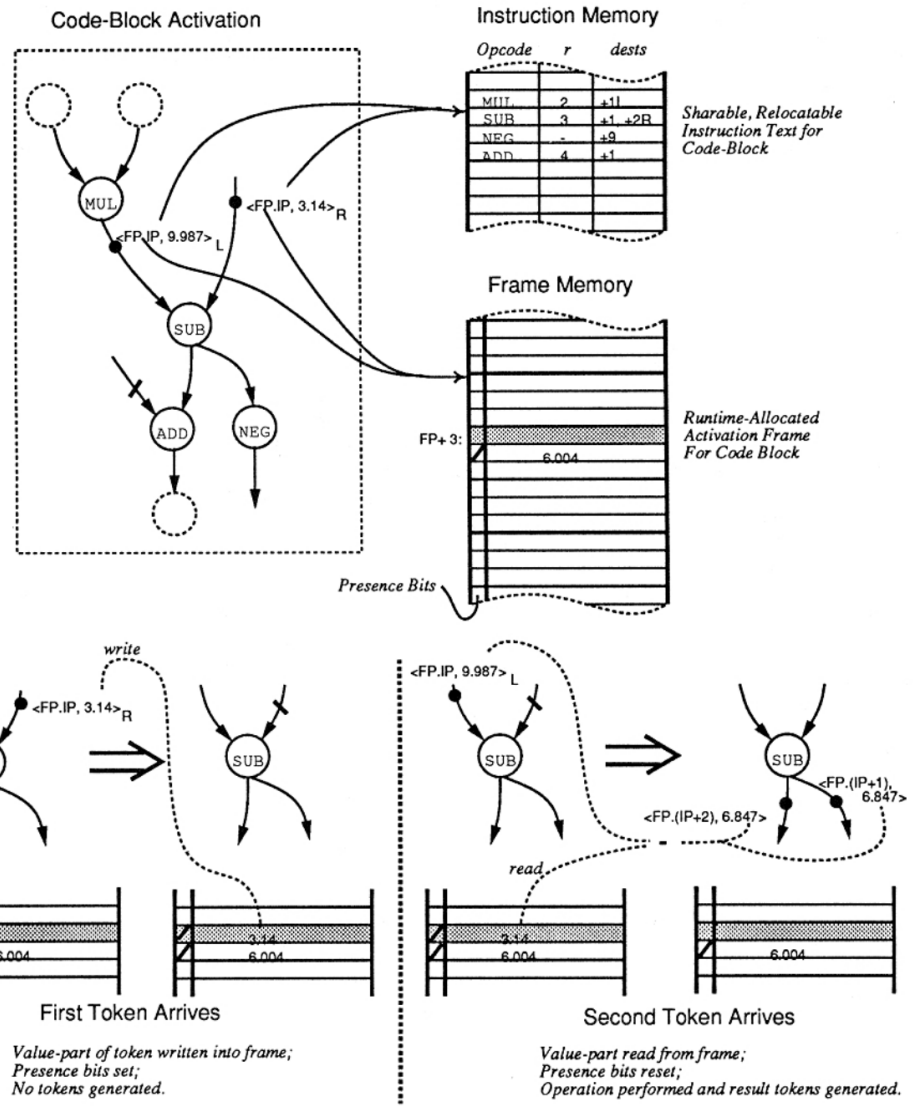


From CMP paper

“Monsoon: An Explicit Token-Store Architecture”

Gregory M. Papadopoulos, David E. Culler 1990

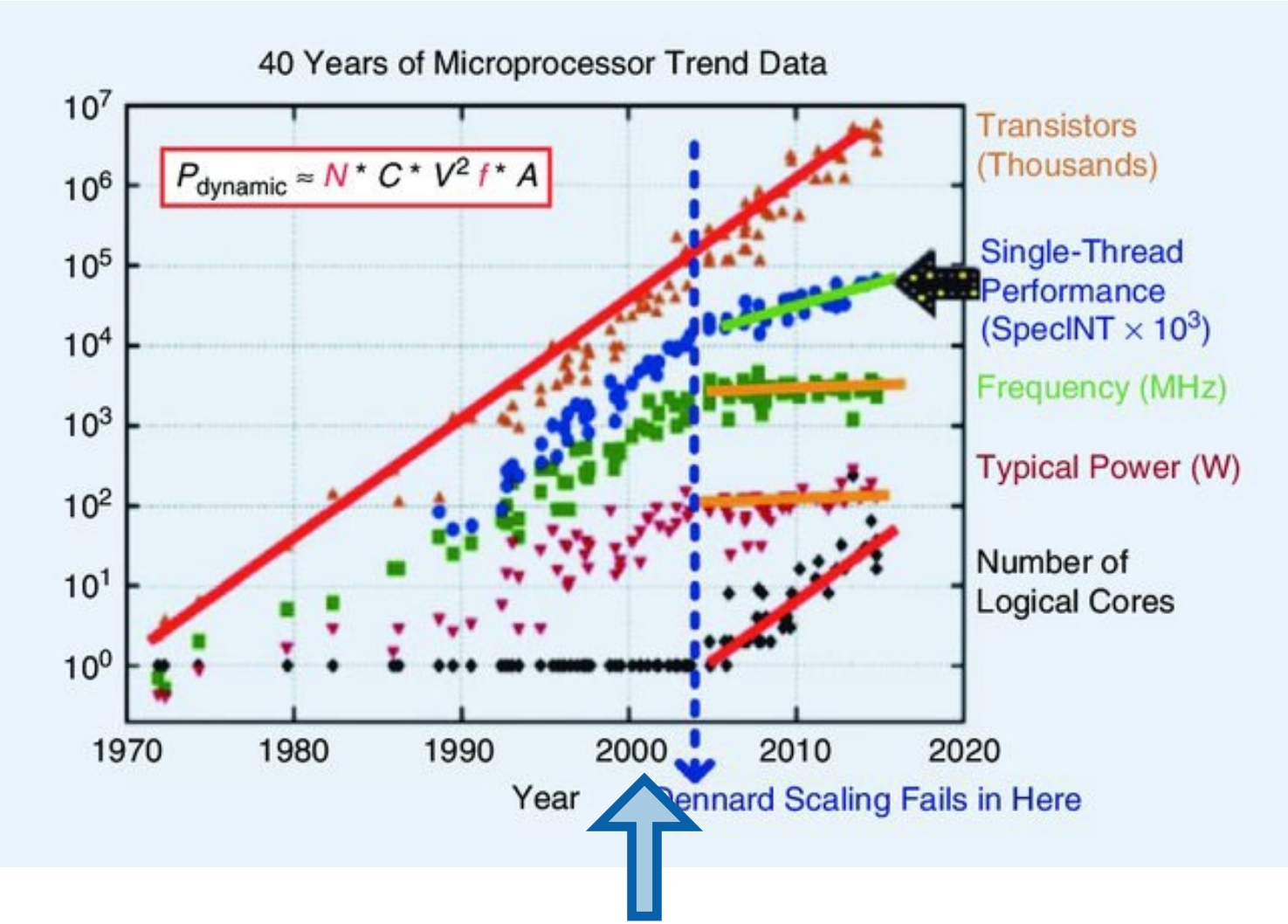
- **Dataflow architectures directly execute dynamic dataflow graphs**
  - Compiled from **Id** language
- **Goal: Tolerate long unpredictable communication delays, in HW**
  - (Tagged) Dataflow firing rule
- **Monsoon introduced Explicit Token Store**
  - Compiler allocates Tokens to Activation Frames



Monsoon: Authors' Retrospective (1998)

- "Clearly the paper did not establish dataflow as the dominant principle in instruction set design."
- One of Monsoon's clear shortcomings was the lack of power in its basic instructions.
- Modern microprocessors construct a small window of dynamic dataflow execution on-the-fly.
- Split-phase memory references foreshadow a future where: Memory reference are carried out asynchronously with some probability of failure, and complete with a well-defined event.
- Future work: Branching on the result of a load being 'not yet present' or branching on a cache miss. Nondeterministic.

Moore's Law w/o Dennard Scaling



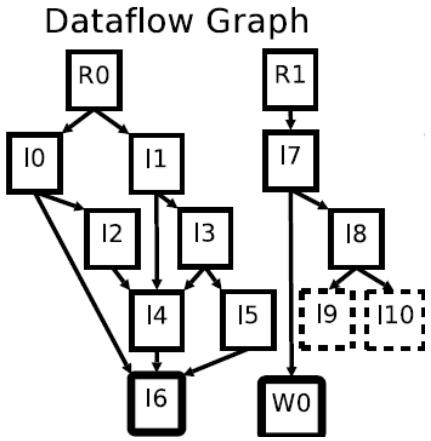
We move to here

"An Evaluation of the TRIPS Computer System"

Mark Gebhart, Bertrand Maher, Katherine Coons, Jeff Diamond, Paul Gratz, Mario Marino, Nitya Ranganathan, Behnam Robatmili, Aaron Smith, James Burrill, Stephen Keckler, Doug Burger, Kathryn McKinley 2009

[First TRIPS paper is in 2001]

- **Explicit Data Graph Execution (EDGE)**
  - Block-atomic execution model
  - Blocks are composed of dataflow instructions

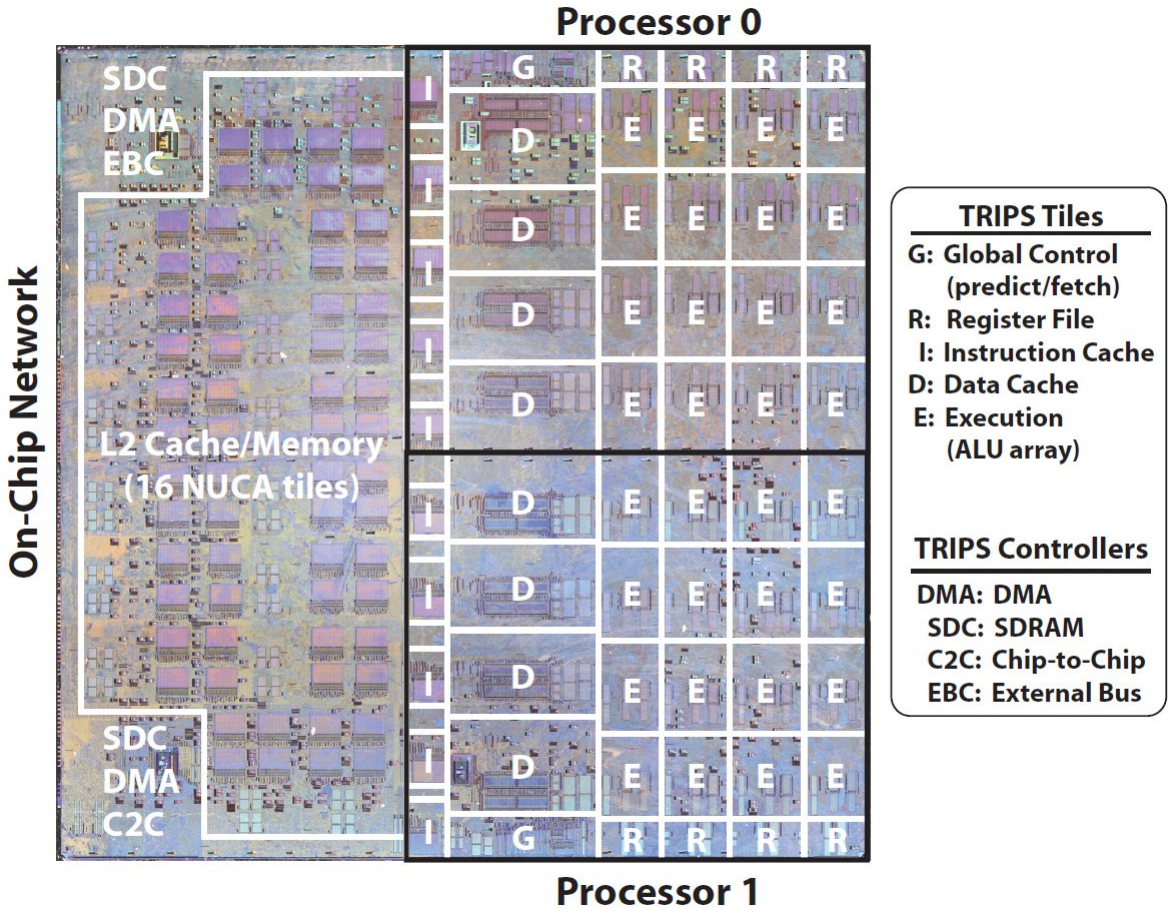


- **Advantages over traditional architectures:**
  - Larger instruction window (up to 8 x 128)
  - Increases concurrency (at cost of more instructions)
  - Register & memory access => inst-to-inst communication

Single-thread performance matters even in multicore era. Why?

TRIPS Chip

- 130nm ASIC w/170M transistors
- Dataflow in a block. Blocks communicate via registers & memory



TRIPS vs. Alpha RISC

(Simulator Results)

Suite	Count	Benchmarks
Kernels	4	transpose (ct), convolution (conv), vector-add (vadd), matrix multiply (matrix)
VersaBench	3 of 10	bit and stream (fmradio, 802.11a, 8b10b)
EEMBC	28 of 30	Embedded benchmarks
Simple	15	Hand-optimized versions of Kernels, VersaBench, and 8 EEMBC benchmarks
SPEC 2K Int	9 of 12	All but gap, vortex and C++ benchmarks[1]
SPEC 2K FP	9 of 14	All but sixtrack and 4 Fortran 90 benchmarks[1]

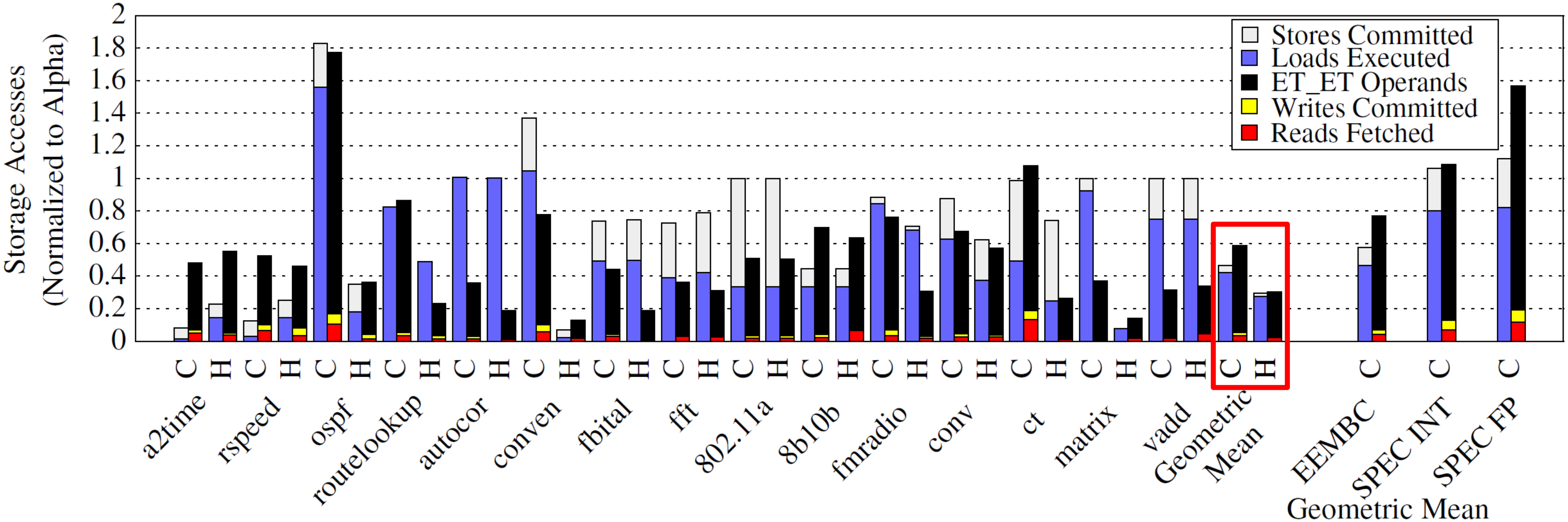


Figure 5. Storage accesses normalized to Alpha for compiled (C) and hand-optimized (H) benchmarks.

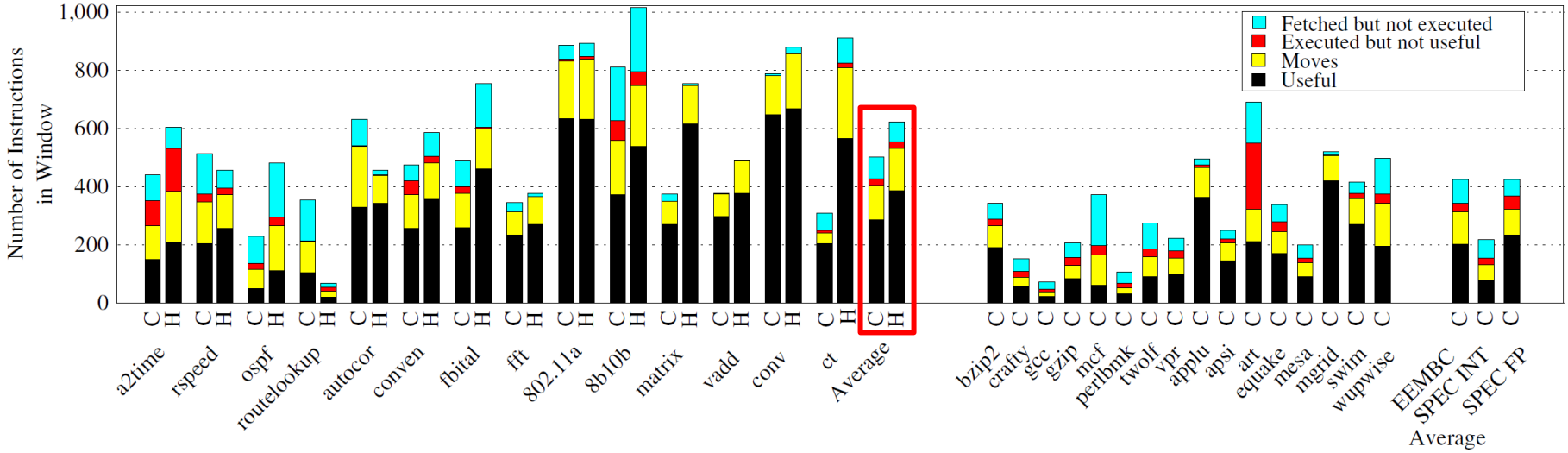
TRIPS cuts memory accesses in half.

TRIPS register accesses + direct ET-ET ~ Alpha register accesses.

TRIPS: In-flight Instructions

(Simulator Results)

Suite	Count	Benchmarks
Kernels	4	transpose (ct), convolution (conv), vector-add (vadd), matrix multiply (matrix)
VersaBench	3 of 10	bit and stream (fmradio, 802.11a, 8b10b)
EEMBC	28 of 30	Embedded benchmarks
Simple	15	Hand-optimized versions of Kernels, VersaBench, and 8 EEMBC benchmarks
SPEC 2K Int	9 of 12	All but gap, vortex and C++ benchmarks[1]
SPEC 2K FP	9 of 14	All but sixtrack and 4 Fortran 90 benchmarks[1]



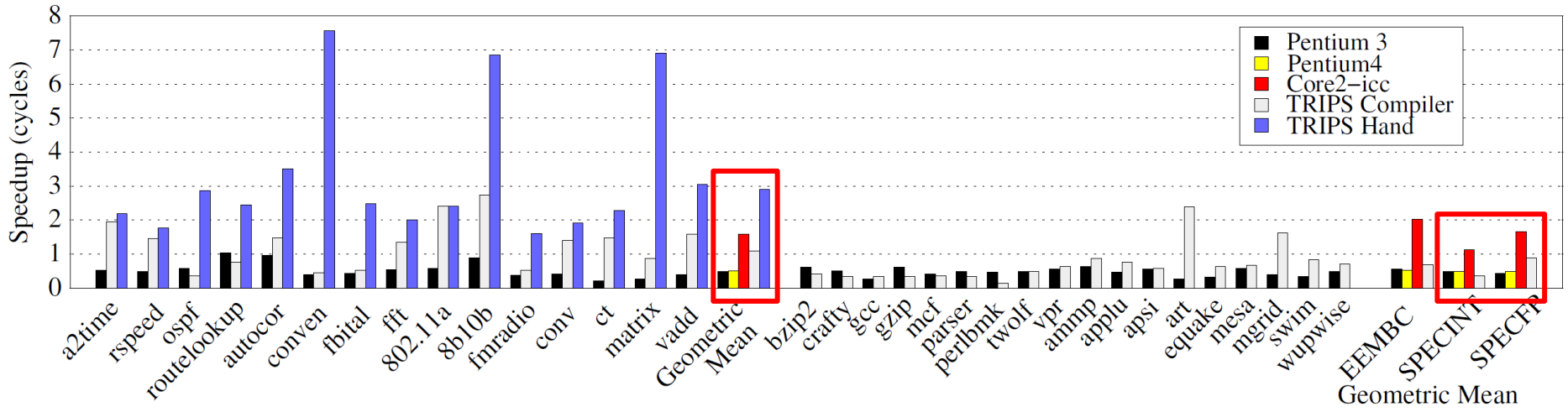
C = compiled code. H = hand-optimized code.

On average: 400 (C) and 600 (H) in-flight valid instructions.

Speedup Relative to Core 2 (gcc)

(Actual Hardware)

System	Issue Width	Proc Speed (MHz)	Mem Speed (MHz)	Proc/Mem Ratio	L1 Cap. (D/I) (KB)	L2 Cap. (MB)	Mem Cap. (GB)
TRIPS	16	366	200	1.83	32 / 80	1	2
Core 2	4	1600	800	2.00	32 / 32	2	2
Pentium 4	4	3600	533	6.75	16 / 150	2	2
Pentium III	3	450	100	4.50	16 / 16	0.5	0.256



TRIPS is 2.9x fewer cycles on Simple, but slower on SPEC.

Core2-icc much faster than TRIPS on SPEC.

TRIPS: Lessons Learned

+ Large window, OOO execution w/less complexity than Superscalar

- Needs operand broadcast support (many targets)

- Code overhead is too large: needs smaller block headers, variable sized blocks

- Challenge: How to form large blocks in control-intensive code?

+ Distributed, tiled architecture can be effective & easy to validate

- Needs predicate prediction

- Needs better mapping to minimize inter-tile communication

- Increase memory BW by interspersing it with all the execution tiles

Authors: EDGE's improvements will not justify deployment in full-power desktop systems. May justify for low-power systems.

“WaveScalar”

Steven Swanson, Ken Michelson, Andrew Schwerin, Mark Oskin
2003

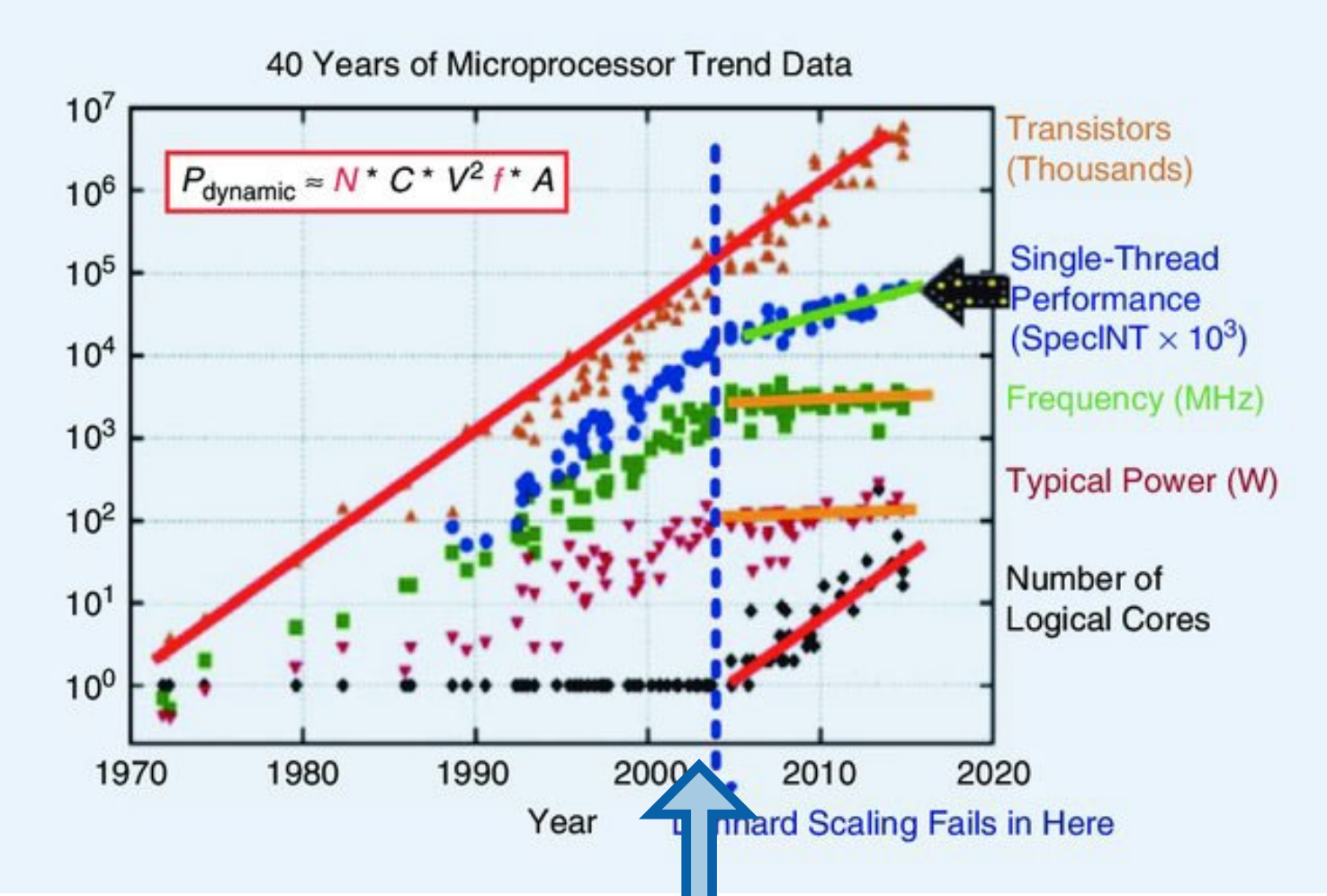
- **Steven:** U.Washington PhD (WaveScalar was his dissertation), UCSD prof since 2006
  - Co-founder NVMW

- **Ken:** UW undergrad, now prof of Pediatrics @ Northwestern

- **Andrew:** UW PhD, now @MongoDB (VP-Engineering, now Distinguished Engineer)

- **Mark:** UW prof, Co-founder Jakobia, Inc for SW defined HW
  - Advisor for Brandon Lucia

Moore's Law w/o Dennard Scaling



Processor Scaling Wall

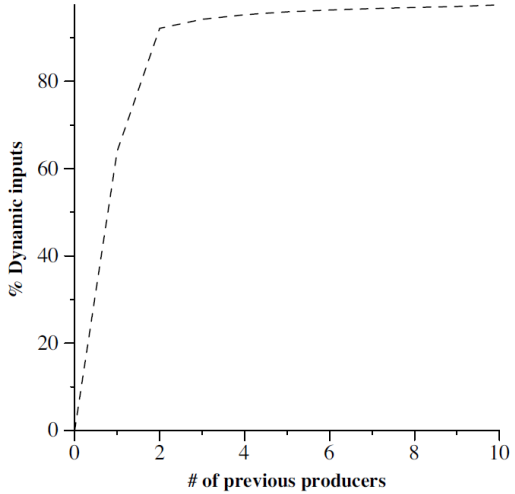
- **Faster transistors but slow wires**
- **Circuit complexity: longer design times, schedule slips, more bugs**
- **Decreasing circuit technology reliability**

Modern Superscalar designs will not scale

- Built atop slow broadcast networks, associative searches, complex control logic, inherently centralized structures

Dataflow Locality

- Predictability in the dynamic data dependencies of a program



- Static dataflow locality: producers & consumers of register values are precisely known

- Dynamic dataflow locality: arises from branch predictability

- What about data values in Memory?

Superscalar Destroys Dataflow Locality

- By changing the physical registers an instruction uses, renaming requires each instruction has fast access to the entire register file

- Consequences: Broadcast communication, bypass network, complex instruction scheduling

von Neumann Serialization Points

- **Instruction fetch**
  - Linear sequence of operations for execution
- **Memory reads/writes**
  - Must guarantee load-store ordering

Waves

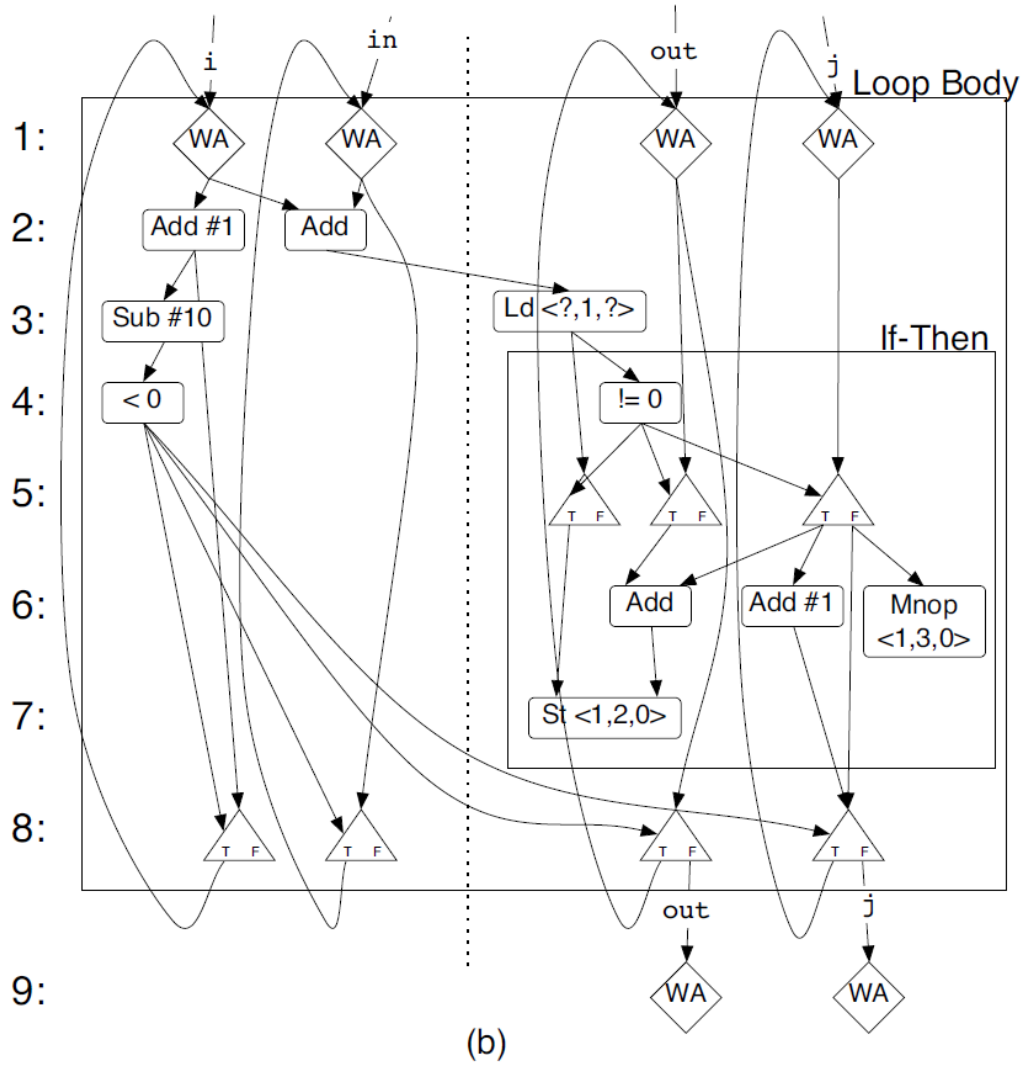
- Each instruction in the wave executes at most once
- Instructions are partially ordered
- Single point of entry

```
function s(char in[10], char out[10]) {
      i = 0;
      j = 0;
      do {
            int t = in[i];
            if (t) {
                  out[j] = t;
                  j++;
            }
            i++;
      } while (i < 10);
      // no more uses of i
      // no more uses of in
}
```

```
;; r0 = i
;; r1 = j
;; r2 = in
;; r3 = out
;; r4 = t

loop:
add  r6, r2, r0
ld   r4, r6(0)
bne  r4, L1
add  r6, r3, r1
st   r4, r6(0)
addi r1, r1, #1

L1:
addi r0, r0, #1
subi r7, r0, #10
blt  r7, loop
```

WaveScalar

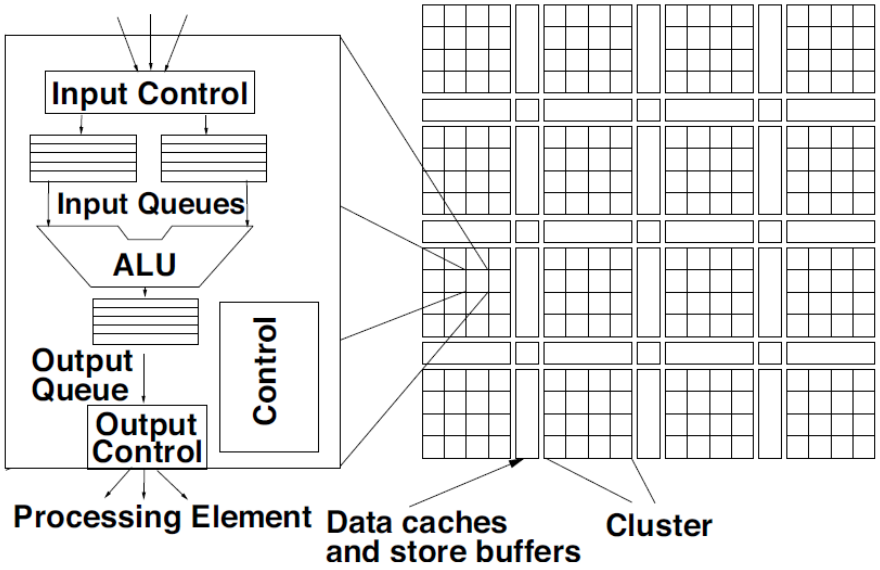
```
;; r0 = i
;; r1 = j
;; r2 = in
;; r3 = out
;; r4 = t

loop:
add  r6, r2, r0
ld   r4, r6(0)
bne  r4, L1
add  r6, r3, r1
st   r4, r6(0)
addi r1, r1, #1

L1:
addi r0, r0, #1
subi r7, r0, #10
blt  r7, loop
```

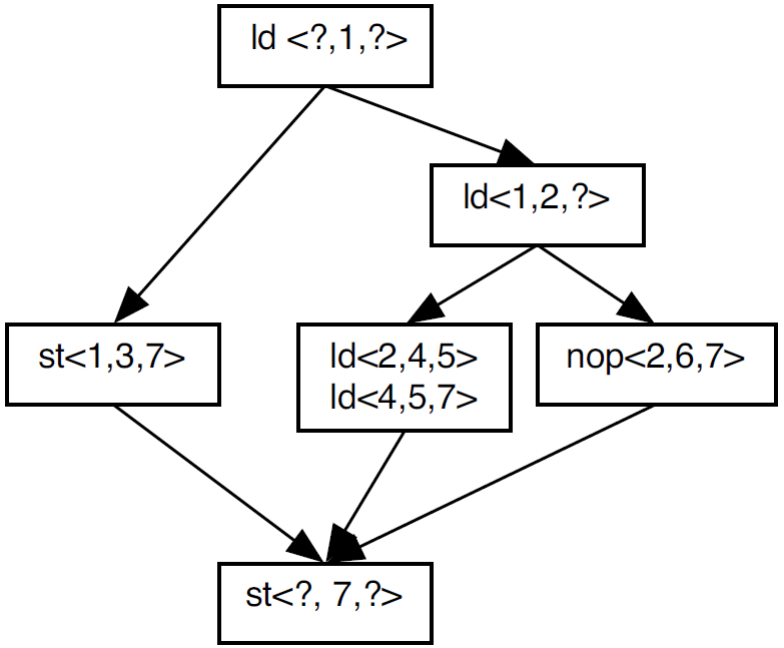


(a) (b)

WaveScalar Features

- **Outperforms Superscalar baseline by 3.1x**
- **Wave-ordered memory**
  - Wave number + BFS numbering
  - Enables load-store ordering: "traditional memory semantics"
- **WaveCache**





Hash table in main memory maps wave numbers to store buffers

Discussion: Summary Question #1

What Did the Paper Get Right?

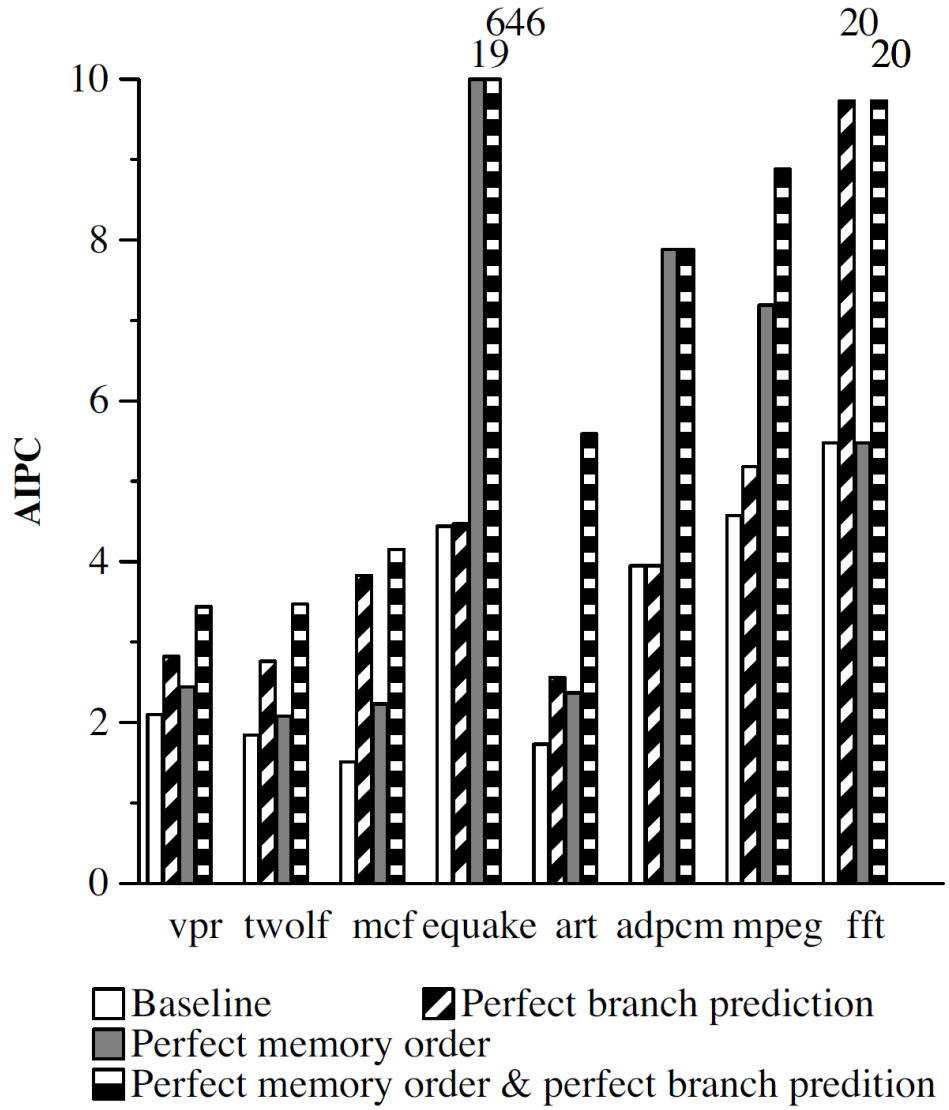
State the 3 most important things the paper says.

These could be some combination of the motivations, observations, interesting parts of the design, or clever parts of the implementation.

WaveScalar Features

- No need for centralized tag structure
- 16 element clusters are the sweet spot
- Instruction misses are costly, since must load all instruction state
- Input queue for a load can overflow
- On termination, must forcibly flush WaveCache
- Dynamic instruction placement

WaveCache Speculation



Discussion: Summary Question #2

What Did the Paper Get Wrong?

Describe the paper's single most glaring deficiency.

Every paper has some fault. Perhaps an experiment was poorly designed or the main idea had a narrow scope or applicability.

Retrospective by Mark Oskin

The project itself was extremely exciting to do, but in the end I learned something about ILP that has stuck with me to this day: there are two points of sequentialization in a superscalar processor, instruction fetch and memory reads/writes. WaveScalar parallelized instruction fetch to the absolute limit, but the memory interface was only parallelized to the limit of what a compiler could express to the hardware statically. This ultimately constrained performance. The challenge is still out there for another day: unlock the parallelism at the memory side.

To Read for Wednesday

"Pipestitch: An Energy-Minimal Dataflow Architecture with Lightweight Threads"

Nathan Serafin, Souradip Ghosh, Harsh Desai, Nathan Beckmann, Brandon Lucia 2023

Optional Further Reading:

"The TYR Dataflow Architecture: Improving Locality by Taming Parallelism"

Nikhil Agarwal, Mitchell Fream, Souradip Ghosh, Brian C. Schwedock, Nathan Beckman 2024