Token Coherence:
Enabling Faster Multiprocessors by Decoupling Performance and Correctness

Milo M. K. Martin, **Mark D. Hill**, David A. Wood

Wisconsin Multifacet Project
http://www.cs.wisc.edu/multifacet/
University of Wisconsin—Madison

Refs: ISCA'03, Martin Thesis, Martin Interview Talk

Overview

- Focus: **server hardware**
  - Database and web services
  - Computers with multiple processors
- Problem: **memory system**
  - Performance bottleneck
  - Complexity
- Solution: **decoupling**
  - **Correctness aspects** from
  - **Performance aspects**
- Result: **faster & cheaper server hardware**

Outline

- **Motivation & background** } **Background**
- Problem: resolving races
- Solution: Token Coherence (specific) } **Our Contributions**
- Evaluation
- Token Coherence (generalized)
- Conclusions

A World of Clients and Servers

- Clients
  - PCs with web browsers
  - Mobile, wireless, embedded
- Servers
  - **Database and web services**
  - E.g., e-commerce, financial/banking, travel reservations, inventory management, data mining
- Server Hardware
  - **Improve cost & real-world performance**
  - Long term: **enable new & emerging applications**

Server Workload Characteristics

- Parallel requests
  - From thousands or millions of clients
  - Throughput oriented
- Natural fit for multiprocessors
  - …and clusters of multiprocessors
- Large workloads → large hardware demands
  - Dozens of processors
  - 100+ GBs of memory
  - 100's of disks

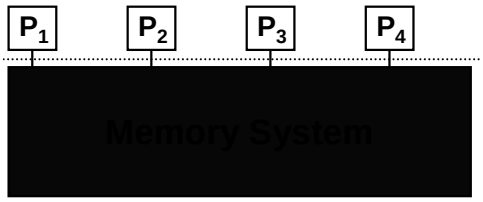
Aside: Simulating Commercial Workloads

- Computer architects use **simulation** to:
  - Characterize and understand workloads
  - Evaluate enhancements
- **Identified workloads** (OLTP, Java, web)
  - Tuned & scaled on a real 16-processor machine
- **Simulation & detailed characterization**
  - Simics full-system simulator (1 instruction = 1 cycle)
  - Added detailed timing simulation (multi-year effort)

Advanced workload and simulation environment
"Simulating $2M Server on a $2K PC"
[IEEE Computer, Feb 2003]

Multiprocessor Server Hardware

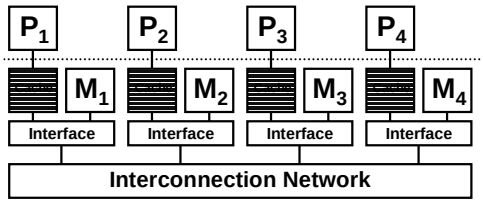
- **Provide a shared-memory abstraction**
  - Familiar and efficient for programmers



- **Hardware caches are critical for performance**
  - E.g., 4MB cache with 64 byte blocks
  - **100x difference** (1ns - 5ns vs. 100's of nanoseconds)

Multiprocessor Server Hardware

- **Provide a shared-memory abstraction**
  - Familiar and efficient for programmers



- **Hardware caches are critical for performance**
  - E.g., 4MB cache with 64 byte blocks
  - **100x difference** (1ns - 5ns vs. 100's of nanoseconds)

Hardware Cache-Coherence Problem

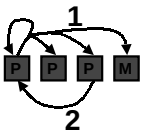
- Goal: provide a consistent view of memory
- Permissions in each cache per block
  - One read/write -or-
  - Many readers
- Cache coherence protocols
  - Distributed & complex
  - Correctness critical
  - Performance critical
- Races: the main source of complexity
  - Requests for the **same block** at the **same time**

Outline

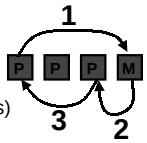
- Motivation & background
- **Problem: resolving races**
  - **Traditional solutions**
  - **Performance overheads**
- Solution: Token Coherence (specific)
- Evaluation
- Token Coherence (generalized)
- Conclusions

Two classes of multiprocessors

- Snooping multiprocessors
  - Uses broadcast
  - Ordered interconnect
  - \+ Directly locate data (2 hops)

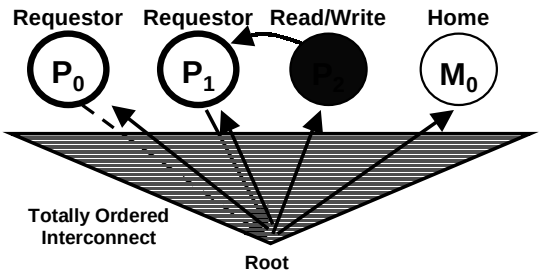


- Directory-based multiprocessors
  - Directory tracks writer or readers
  - \+ Avoids broadcast
  - \+ Avoids ordered interconnect
  - Indirection for cache-to-cache (3 hops)

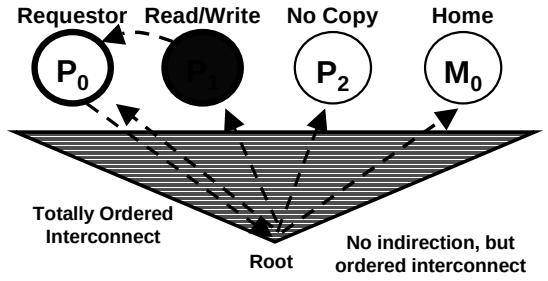


Method for ordering racing requests is key

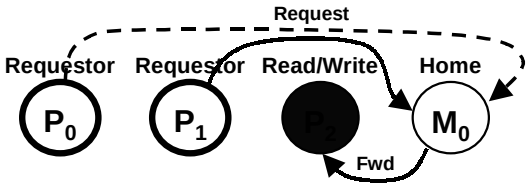
Snooping Example



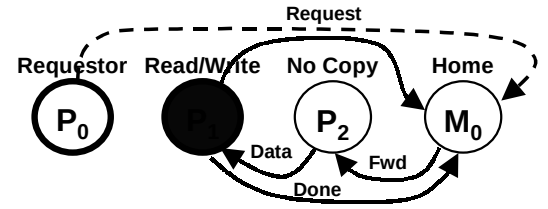
Snooping Example



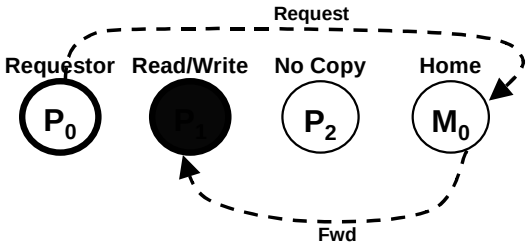
Directory Example



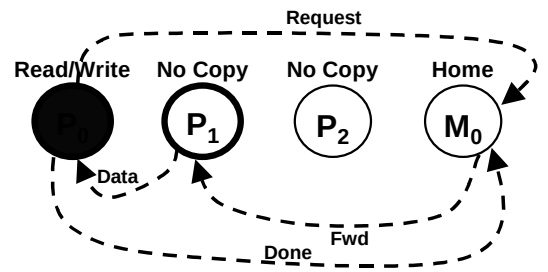
Directory Example



Directory Example



Directory Example



No ordered interconnect, but adds indirection

Snooping v. Directories

Which approach is "better"?

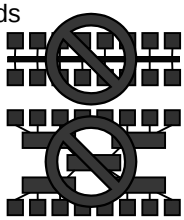
- Examine
  - Workload trends
  - Technology trends

Workload Trends

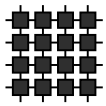
- Commercial workloads
  - Many cache-to-cache or sharing misses
- Clustering
  - Reduces demand for "scalable" systems
  - Build clusters of multiprocessors
- Goals:
  - **Low sharing-miss latency (2 hops)**
  - Moderate scalability

Workload trends → snooping protocols

Technology Trends

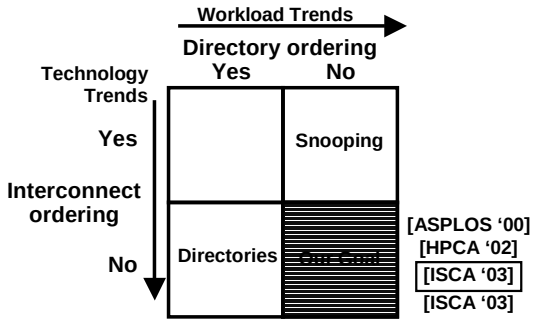


- High-speed point-to-point links
  - No (multi-drop) busses
- Increasing design integration
  - "Glueless" multiprocessors
  - Improve cost & latency
- Goal: unconstrained interconnect
  - Decouple interconnect & protocol
  - **No "virtual bus" ordering**



Technology trends → directory protocols

Multiprocessor Taxonomy



Outline

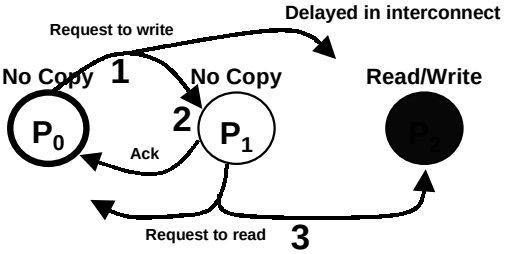
- Motivation & background
- Problem: resolving races
- **Solution: Token Coherence (specific)**
- Evaluation
- Token Coherence (generalized)
- Conclusions

Best of Both: Basic Approach

- Broadcast with direct responses (like snooping)
- Use unordered interconnect (like directory)
- Works fine with **no races**…

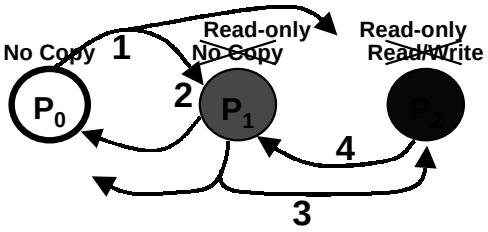
…but what happens in the **case of a race?**

Basic approach… but not yet correct



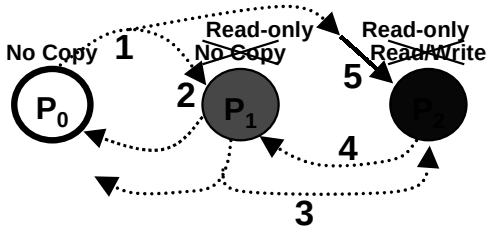
- P_0 issues a request to write (delayed to P_2)
- P_1 issues a request to read

Basic approach… but not yet correct



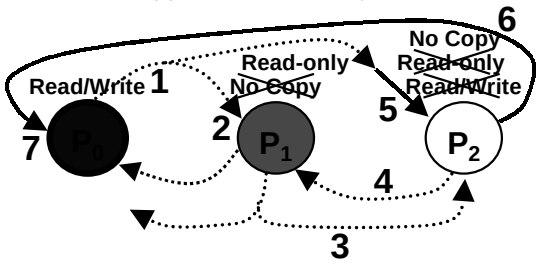
•P_2 responds with data to P_1

Basic approach… but not yet correct



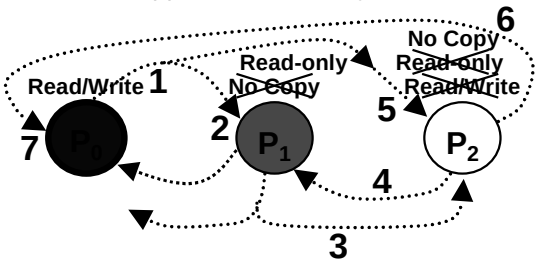
•P_0's delayed request arrives at P_2

Basic approach… but not yet correct



•P_2 responds to P_0

Basic approach… but not yet correct



Problem: P_0 and P_1 are in inconsistent states
(P_0 can write, P_1 can read)

Contribution #1: Token Counting

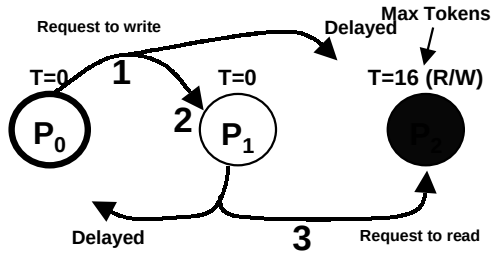
- Explicitly encode permissions with tokens
  - At all times, **all blocks have *T* tokens**
    E.g., one token per processor
  - Components exchange tokens & data
- Controls reading & writing of data
  - **One or more to read**
  - **All tokens to write**
- Tokens: in caches, memory, or in transit

Provides ***safety*** in all cases

Basic Approach (Revisited)

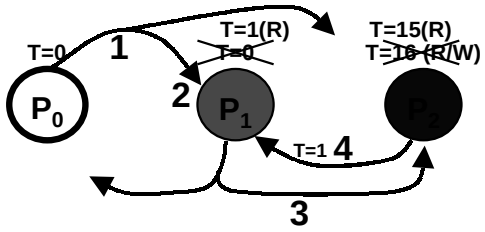
- As before:
  - Broadcast with direct responses (like snooping)
  - Use unordered interconnect (like directory)
- **Track tokens for safety**
- **Reissue requests as needed**
  - Needed due to racing requests (**uncommon**)
  - Timeout to detect failed completion
    - Wait twice average miss latency
    - Small hardware overhead
- (Ignore starvation for a moment)

Token Coherence Example



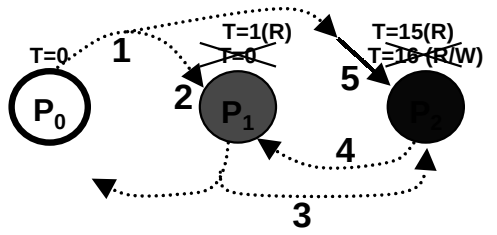
- P_0 issues a request to write (delayed to P_2)
- P_1 issues a request to read

Token Coherence Example



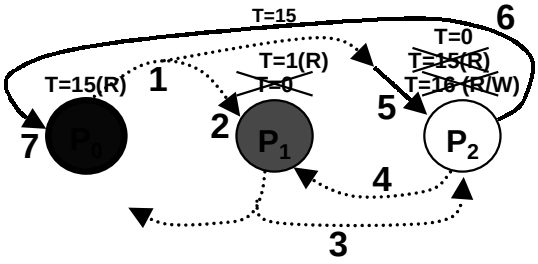
- P_2 responds with data to P_1

Token Coherence Example



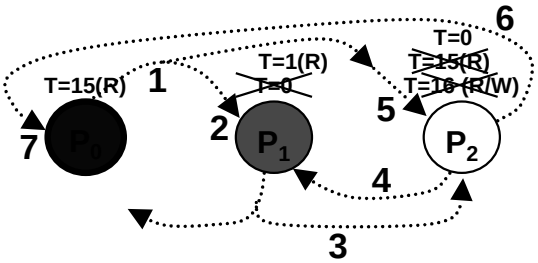
- P_0's delayed request arrives at P_2

Token Coherence Example

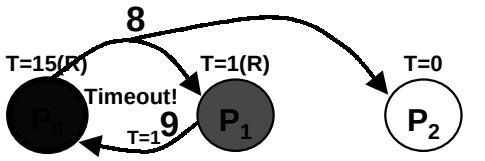


- P_2 responds to P_0

Token Coherence Example

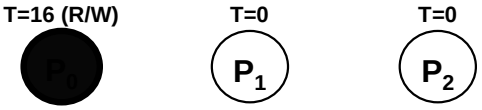


Token Coherence Example



- P_0 reissues request
- P_1 responds with a token

Token Coherence Example



- P_0's request completed

What about starvation (forward progress)?

Contribution #2: Guaranteeing Forward Progress

- Handle pathological cases
  - **Infrequently invoked**
  - Can be slow, inefficient, and simple
- When normal requests fail to succeed (4x)
  - Longer timeout and issue a **persistent request**
  - Request persists until satisfied
  - Table at each processor
  - "Deactivate" upon completion
- Implementation
  - Arbiter at memory orders persistent requests

More Depth in ISCA '03 & Martin Thesis

- Traffic optimizations
  - Transfer tokens without data
  - Add an "owner" token
  - Upgrade (Read-only to read/write)
  - "Exclusive Clean" State
- Note: no silent read-only replacements
  - Worst case: 10% traffic overhead
- Encoding tokens in memory
  - Using ECC bits
  - Reduce read/modify/writes with token cache

Outline

- Motivation & background
- Problem: resolving races
- Solution: Token Coherence (specific)
- **Evaluation**
- Token Coherence (generalized)
- Conclusions

Evaluation Goal

- Non-goal: exact speedup numbers
  - Many parameters and assumptions
  - Key parameter: **16 processors**
- Goal: Four Questions
  1. Are races rare?
  2. Can Token Coherence outperform Snooping?
  3. Can Token Coherence outperform Directories?
  4. Is broadcast overhead reasonable?

Quantitative evidence for qualitative behavior

Q1: Reissued Requests

(percent of all misses)

	OLTP	SPECjbb	Apache

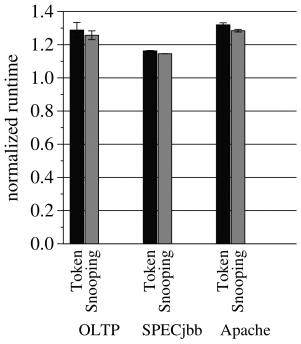
Q1: Reissued Requests

(percent of all misses)

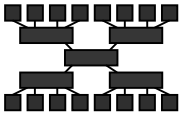
Outcome	OLTP	SPECjbb	Apache
Not Reissued	**98%**	**98%**	**96%**
Reissued Once	**2%**	**2%**	**3%**
Reissued > 1	**0.4%**	**0.3%**	**0.7%**
Persistent Requests (Reissued > 4)	**0.2%**	**0.1%**	**0.3%**

Yes; races are rare

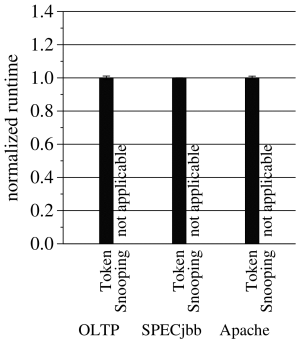
Q2: Runtime: Snooping vs. Token Coherence Hierarchical Switch Interconnect



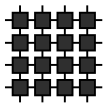
Similar performance on same interconnect



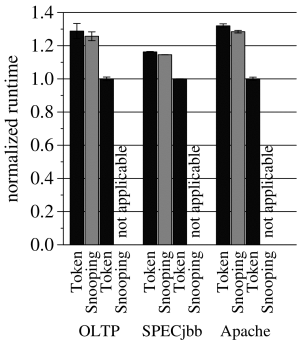
Q2: Runtime: Snooping vs. Token Coherence Direct Interconnect



Snooping not applicable



Q2: Runtime: Snooping vs. Token Coherence

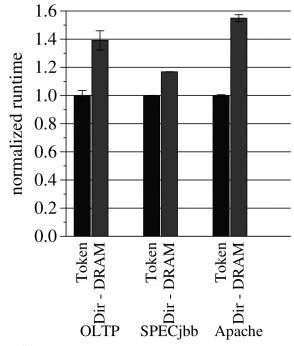


Yes; Token Coherence can outperform snooping

(15-28% faster)

Why? Lower-latency interconnect

Q3: Runtime: Directory vs. Token Coherence

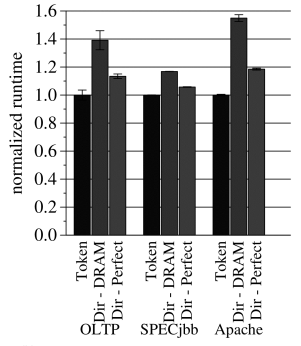


Yes; Token Coherence can outperform directories

(17-54% faster with slow directory)

Why? Direct "2-hop" sharing misses

Q3: Runtime: Directory vs. Token Coherence

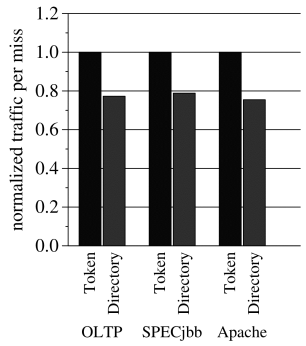


Yes; Token Coherence can outperform directories

(17-54% faster with slow directory)

Why? Direct "2-hop" sharing misses

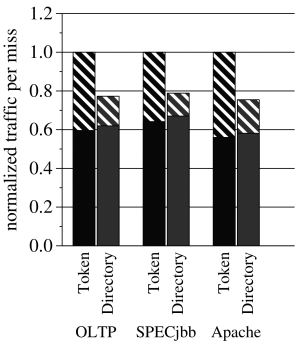
Q4: Traffic per Miss: Directory vs. TokenB



Yes; broadcast overheads reasonable for 16 processors

(directory uses 21-25% less bandwidth)

Q4: Traffic per Miss: Directory vs. TokenB



Requests & forwards

Responses

Yes; broadcast overheads reasonable for 16 processors

(directory uses 21-25% less bandwidth)

Why? Requests are smaller than data (8B v. 64B)

Evaluation Summary

1. Are races rare? **Yes**
2. Can Token Coherence outperform Snooping?
   Yes: Lower-latency interconnect
3. Can Token Coherence outperform Directories?
   Yes: Direct sharing-misses
4. Is broadcast overhead reasonable?
   Yes (Reasonable for 16 processors)

Outline

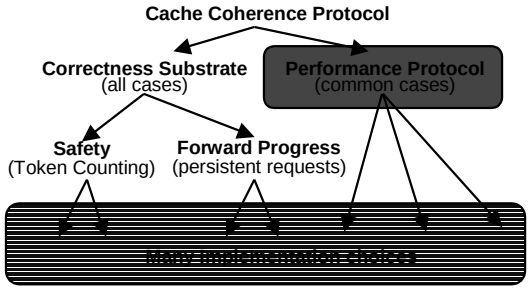
- Motivation & background
- Problem: resolving races
- Solution: Token Coherence (specific)
- Evaluation
- **Token Coherence (generalized)**
- Conclusions

Traditional v. Token Coherence

- **Traditional protocols**
  - Fixed point of ordering requests
    - Interconnect or directory
  - **Monolithic**
    - Intertwine correctness and performance
- **Token Coherence**
  - Unordered requests for common cases
  - Track tokens (safety)
  - Guarantee forward progress

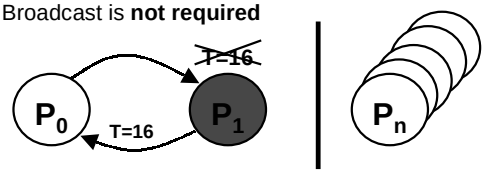
Separate Correctness and Performance

Contribution #3: Decoupled Coherence



Example Opportunity of Decoupling

- Broadcast is **not required**



- **Predict a *destination-set*** [ISCA '03]
  - Based on past history
  - Need not be correct (fall back on broadcast)
  - 90% of snooping's benefit, 1/3rd the traffic
- Enables **larger** or **more cost-effective** systems

Performance Protocols

- Opportunities
  - Aggressively target the common case
  - Requests are just **"hints"** to move data & tokens
- Correctness substrate may ignore hints
- **Robust**
  - **Can't cause "correctness" violations**
  - A null or random protocol is correct
  - Rely on correctness substrate

Verifiability & Complexity

- Divide and conquer complexity
  - Formal verification is future work
  - Difficult to quantify, but promising
  - E.g. simple replacements (no handshake)
- Strong invariants
  - Locally enforced with tokens
  - Response-centric; independent of requests
  - Prevent data-corruption bugs
- Explicit forward progress
  - Simple mechanism
- **Further innovation → no correctness worries**

Outline

- Motivation & background
- Problem: resolving races
- Solution: Token Coherence (specific)
- Evaluation
- Token Coherence (generalized)
- **Other research & future directions**
- **Conclusions**

Other Collaborative Research

- Coherence protocols
  - Destination-set prediction [ISCA '03]
  - Bandwidth adaptive coherence [HPCA '02]
  - Timestamp-based ordering [ASPLOS '00]
- Correctness & specification
  - Value prediction & consistency [Micro '01]
  - Protocol specification [TPDS '02]
  - SLICC: A domain specific language for protocol specification
- Improving hardware availability [ISCA '02]
- Compiler/hardware interaction [Micro '97]

Future Directions: Servers

- Tackling complexity through decoupling
  - Simpler, faster, cheaper, more robust
- Other hardware examples
  - Hierarchical multiprocessors (CMPs)
  - Multiprocessor interconnection networks (e.g., domain specific TCP/IP)
  - Simplify processor design with "correctness" checker
- Possible software example (for collaboration)
  - Content distribution networks, secure systems

Future Directions: Clients

- Mobile, wireless, embedded, sensors
  - Not PCs
- **Energy consumption**, cost, performance
  - Applying techniques from servers to clients
  - Exploit explicit parallelism
- Example: Intel Xscale
  - 1 GHz → 200 MHz reduces energy used by 30x
  - 5 x 200 MHz in parallel, use **1/6th the energy**
- Same research approach:
  - Identify emerging workloads, trends, problems
  - Build tools, characterize, **find innovate solutions**

Conclusions

- Token Coherence (generalization)
  - **Decouple correctness from performance**
  - Correctness substrate
    - Tokens for **safety**
    - Persistent requests for **forward progress**
  - Performance protocol for **performance**
- Token Coherence (broadcast version)
  - Low sharing-miss latency (no indirection)
  - Requires no interconnect ordering
  - Can be faster than current alternatives
- Enables further protocol innovation