The Stanford Hydra CMP

Lance Hammond, Ben Hubbert, Michael Siu, Manohar Prabhu, Michael Chen, Maciek Kozyrczak*, and Kunle Olukotun

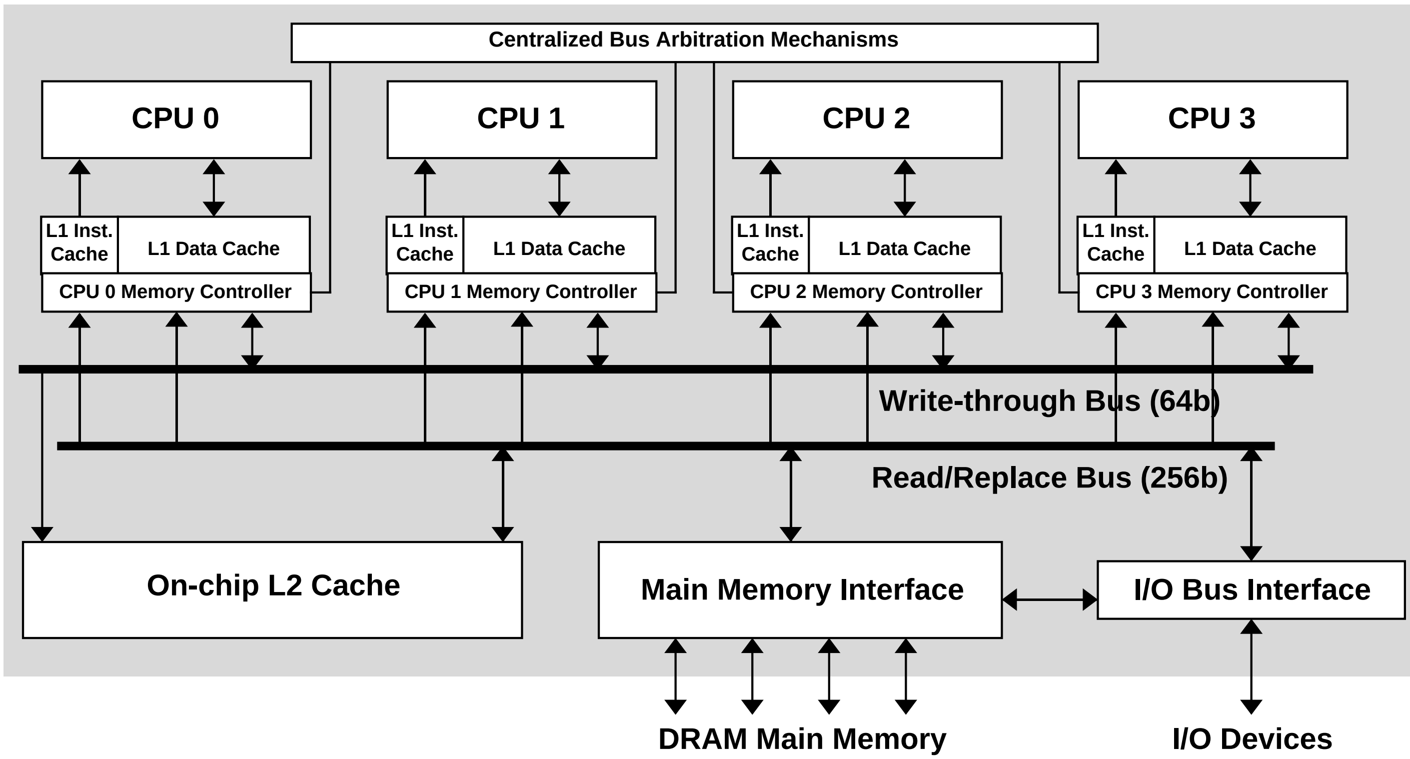
Computer Systems Laboratory
Stanford University
http://www-hydra.stanford.edu

* Integrated Device Technology, Inc.
RISC Microprocessor Division
http://www.idt.com

Hydra: A Chip Multiprocessor

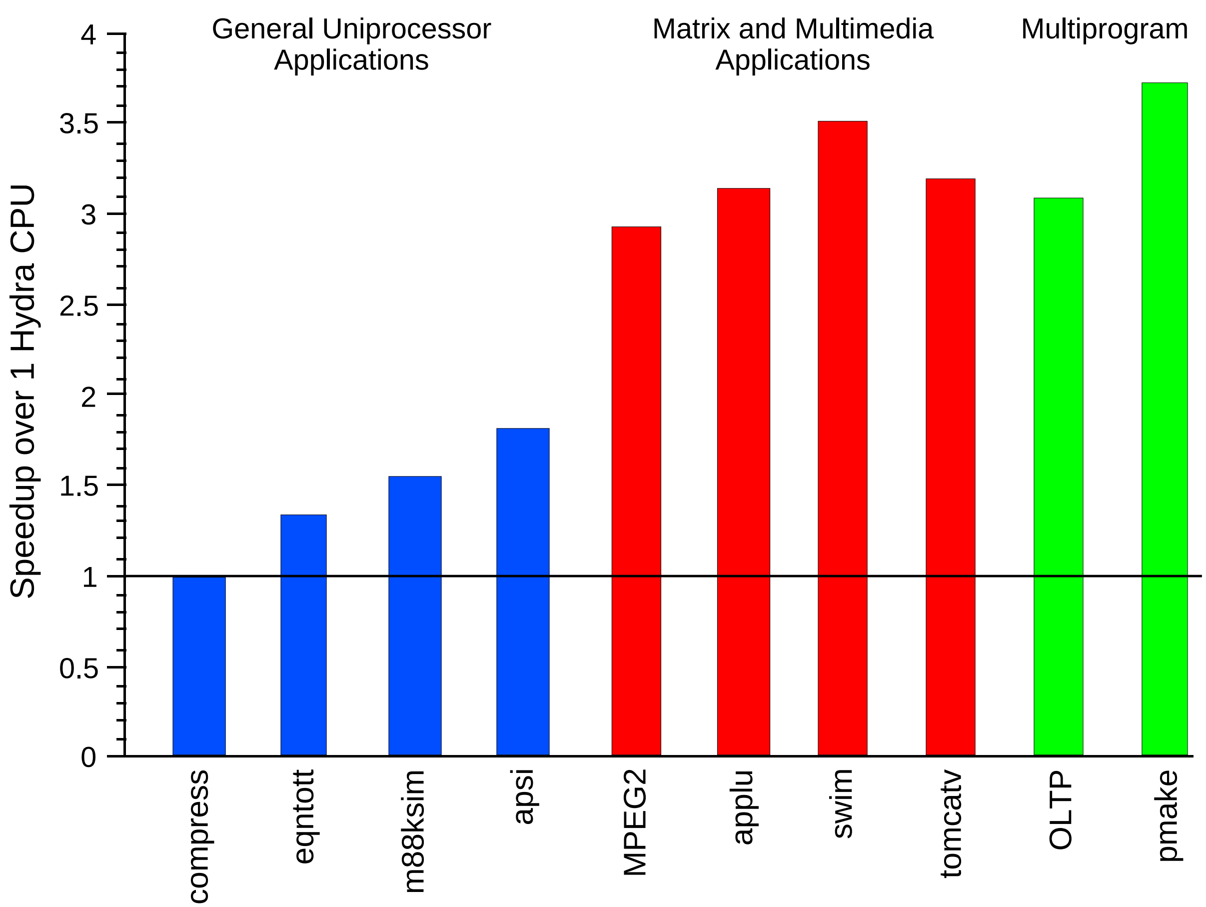
- A CMP offers implementation benefits
  - — High-speed signals are localized in individual CPUs
  - — A proven CPU design may be replicated across the die
- Overcomes diminishing performance/ transistor return problem in uniprocessors
  - — Transistors are used today mostly for ILP extraction
  - — MPs use transistors to run multiple *threads* . . .
    - – On parallelized programs
    - – With multiprogrammed workloads
- Fast inter-processor communication eases parallelization of code

The Basic Hydra CMP



- 4 processors and secondary cache on a chip
- 2 buses connect processors and memory
- Coherence: writes are broadcast on write bus

Parallel Performance



- Varying levels of performance
  - Multiprogrammed workloads work well
  - Very parallel apps (matrix-based FP and multimedia) are excellent
  - Acceptable only with a few less parallel (i.e. integer) apps

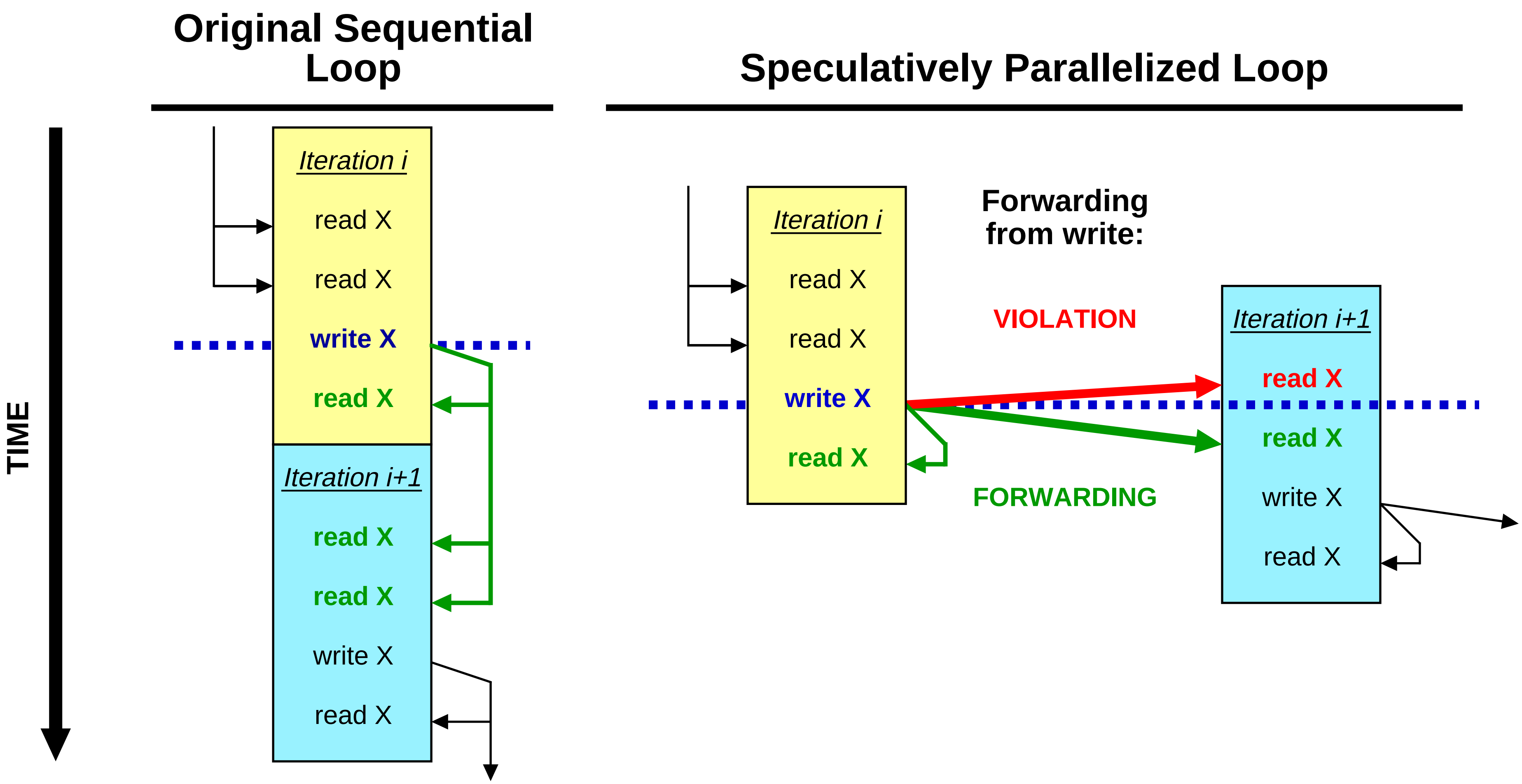
Problem: Parallel Software

- Current parallel software is limited
  — Some programs just don't have significant parallelism
  — Parallel compilers generally require dense matrix FORTRAN applications
- Many applications only hand-parallelizable
  — Parallelism may exist in algorithm, but code hides it
  — Compilers must *statically* verify parallelism
  — Data dependencies require synchronization
  — *Pointer disambiguation* is a major problem for this!
- Can hardware help the situation?

Solution: Data Speculation

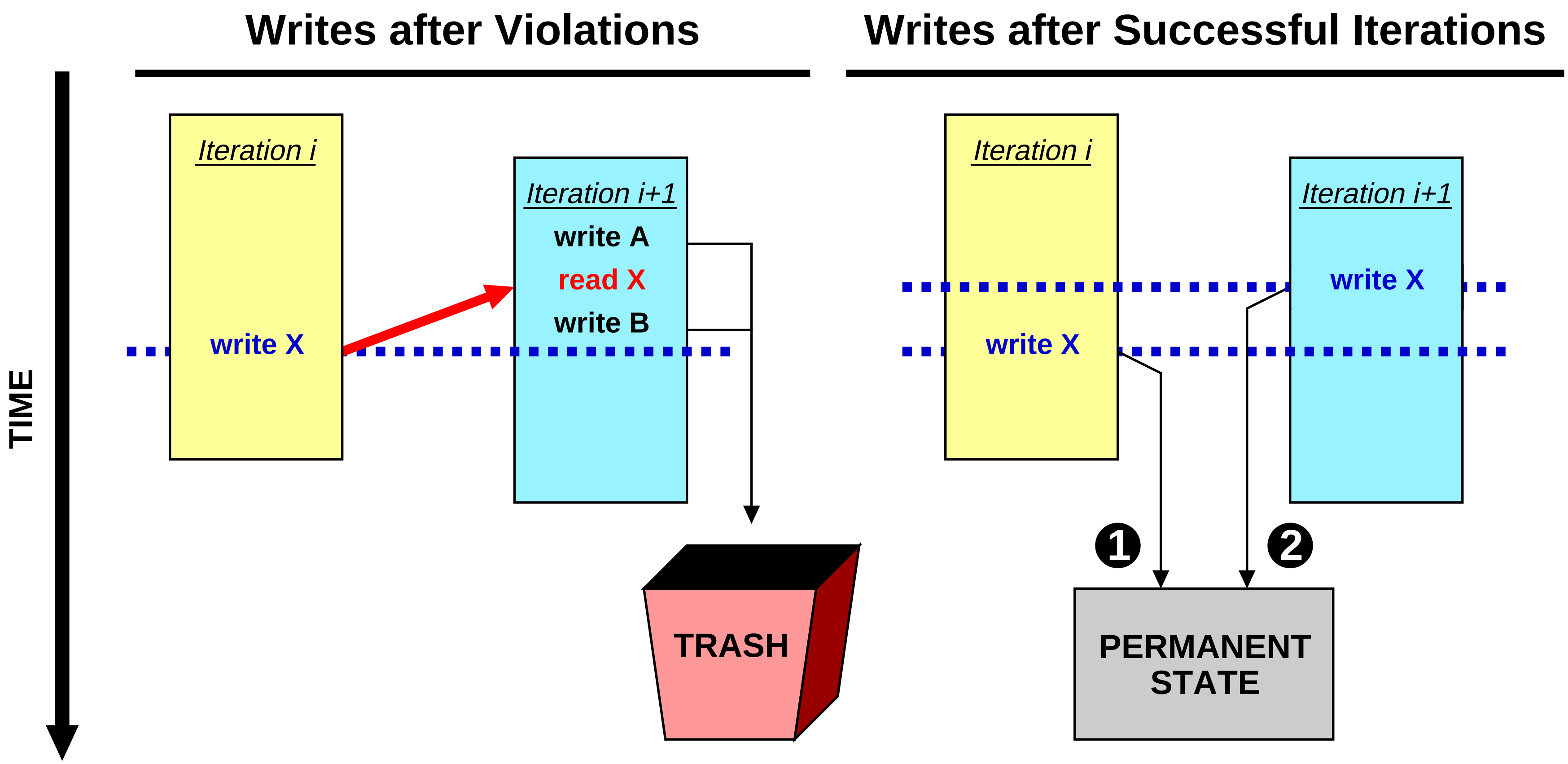
- Data speculation enables parallelization without regard for data dependencies
  - Normal sequential program is broken up into threads
  - Speculative threads are now run in parallel on CPUs
  - Speculation hardware ensures correctness
- Parallel software implications
  - Loop parallelization is now *easily automated*
  - More “arbitrary” threads are possible (subroutines)
  - Add synchronization only for performance
- Speculation support mechanisms
  - Speculative thread control mechanism
  - 5 memory system requirements

Memory Requirements I



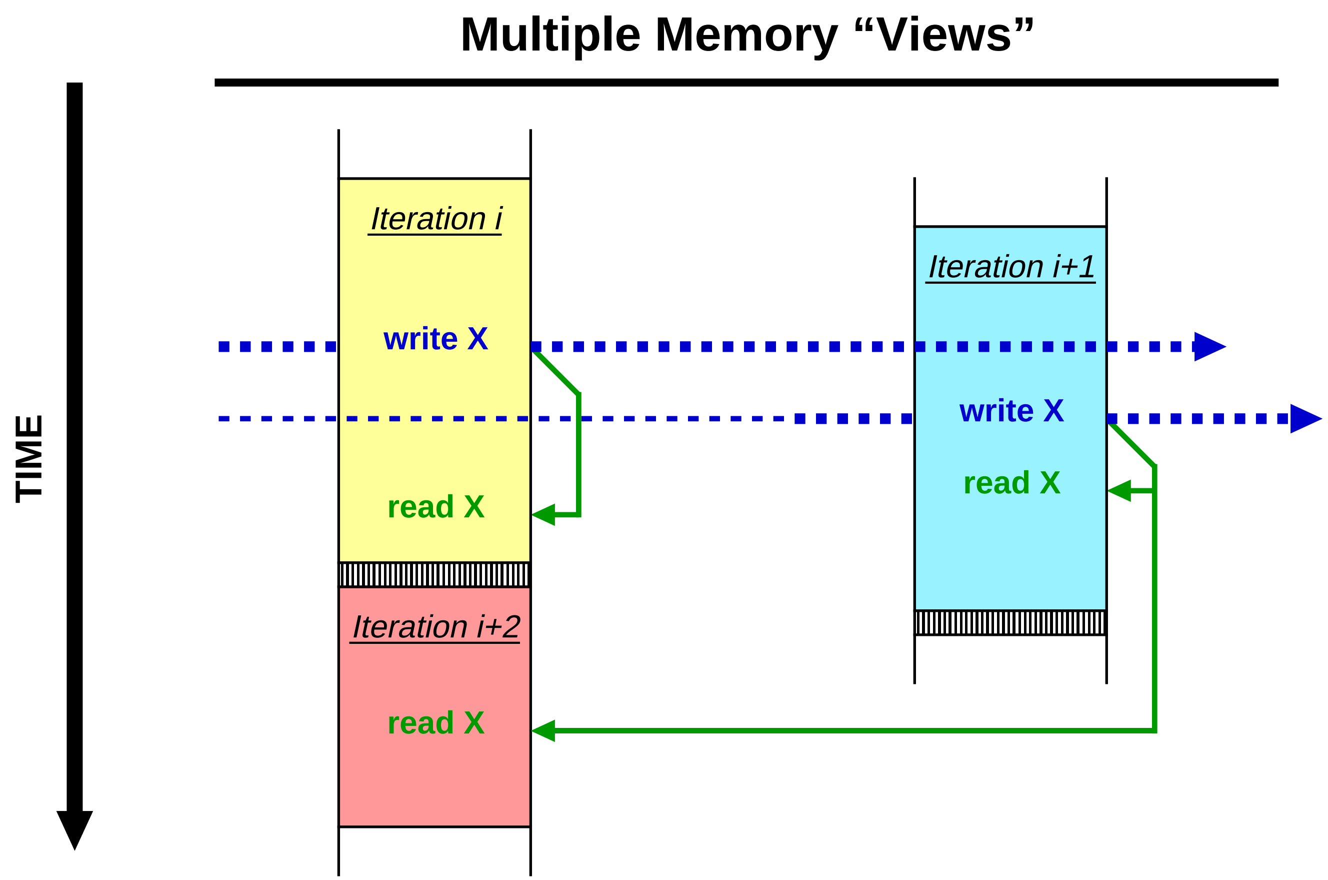
1. Forward data between parallel threads
2. Detect violations when reads occur too early

Memory Requirements II



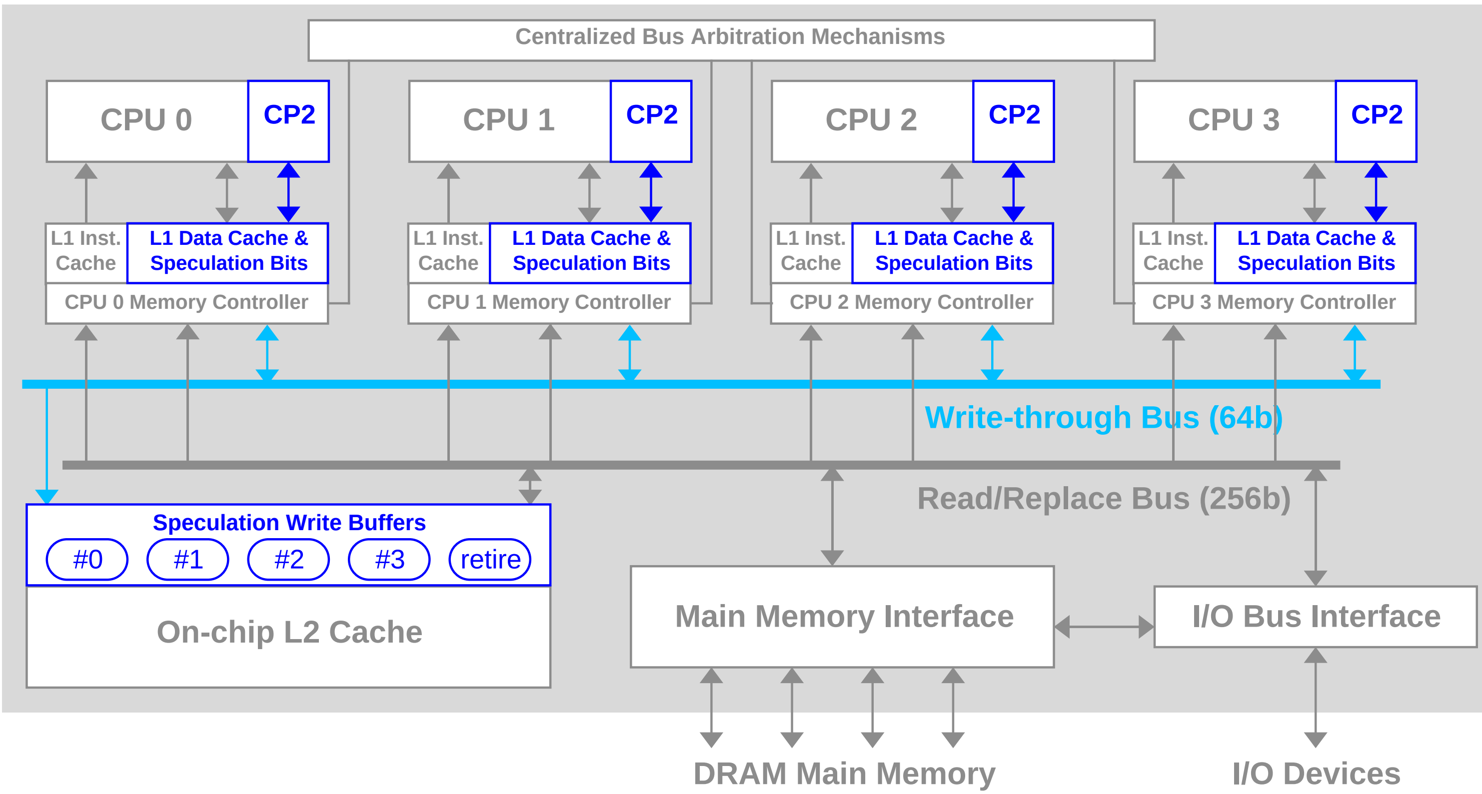
3. Safely discard bad state after violations
4. Retire speculative writes in the correct order

Memory Requirements III



5. Maintain multiple "views" of memory

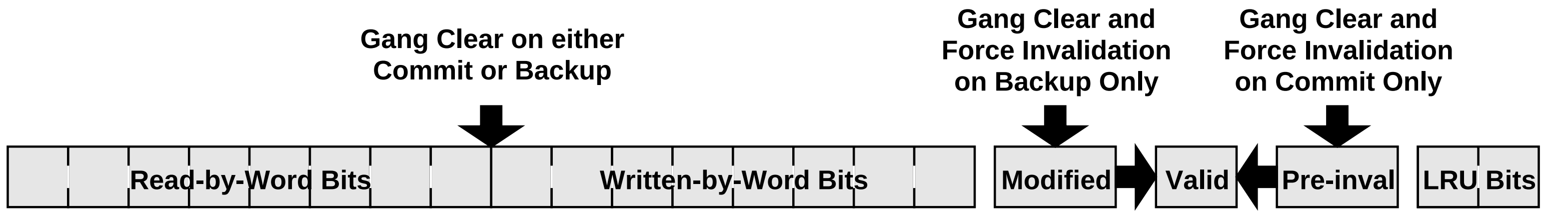
Hydra Speculation Support



1. Write bus & L2 buffers provide forwarding
2. “Read” L1 tag bits set and watching write bus to detect violations
3. “Dirty” L1 bits & clearable L2 buffers allow backup
4. L2 buffers reorder & retire speculative state
5. Separate L1 caches with pre-invalidation & smart L2 forwarding for “view”

— Speculation coprocessors to control threads

L1 Cache Tag Details

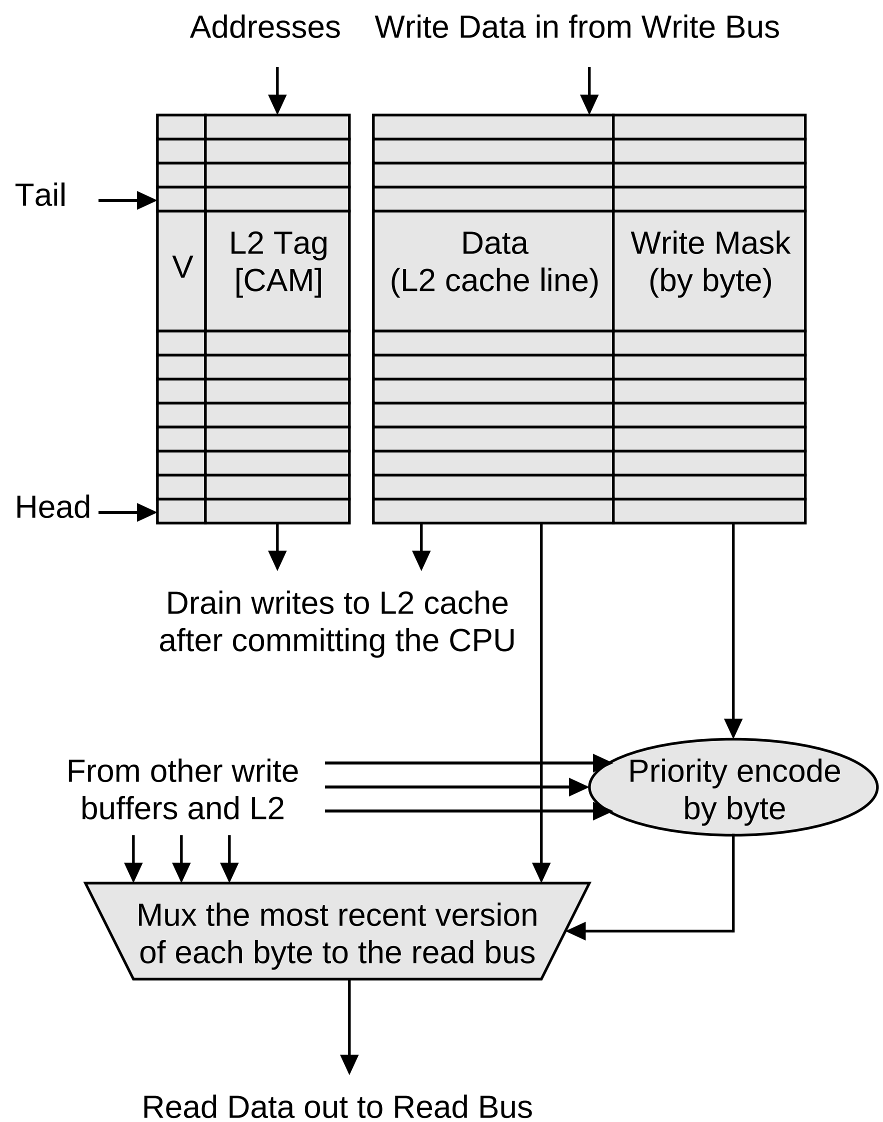


- Speculation requires 4 extra types of bits
  — **Read-by-word:** Allow violation detection
  — **Written-by-word:** Allow memory renaming
  — **Modified:** Allow us to back up after violations
  — **Pre-invalidation:** Allow us to commit and advance

- Special circuits are required in the array
  — Gang clear of all bits on commits and backups
  — Set modified bits cause valid bits to clear on backups
  — Set pre-inval bits cause valid bits to clear on commits

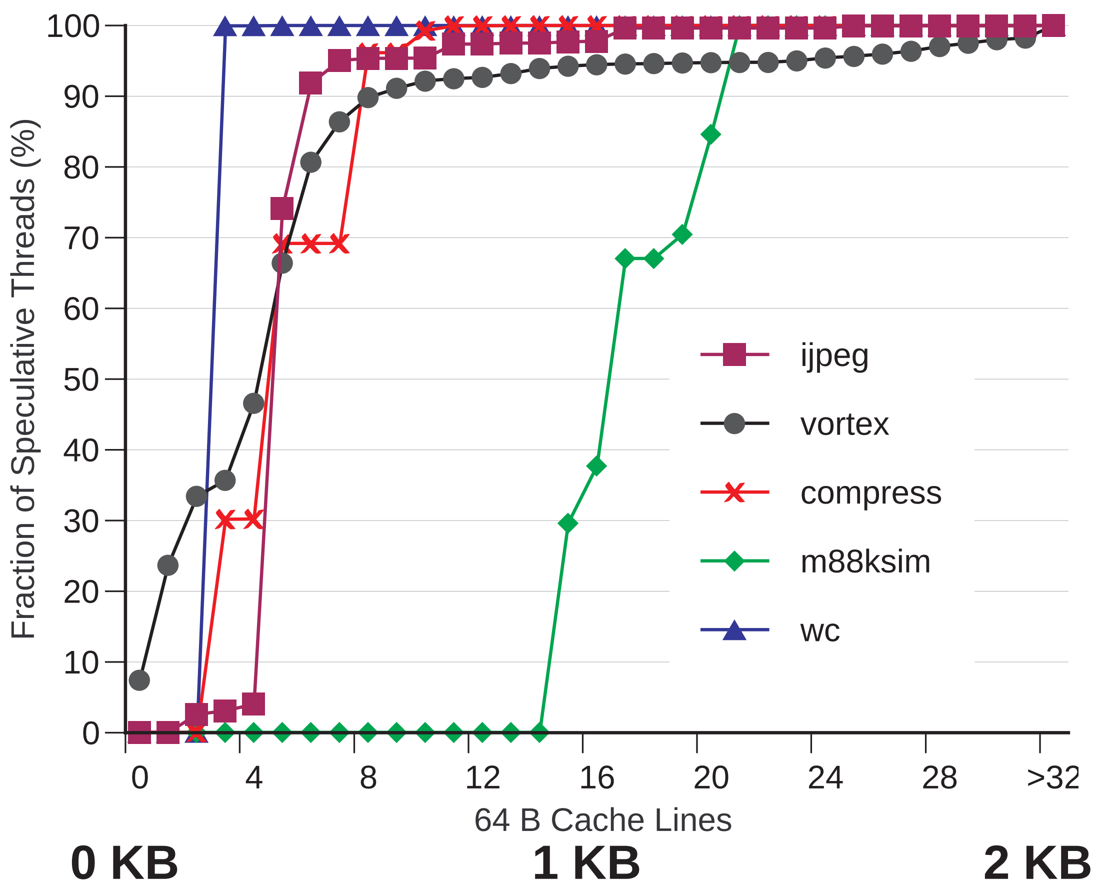
L2 Buffer Details

- Speculative writes are held here until commit time
  - Collected by cache line
  - CAM tag array, tail pointer
  - Byte write mask for each line
  - Drains into L2 when complete
  - Size of another pair of L1s
- Reads are tricky
  - Line read from L2 cache
  - Any data from “earlier” buffers is substituted, if present
  - Requires byte-by-byte priority encoding & muxing



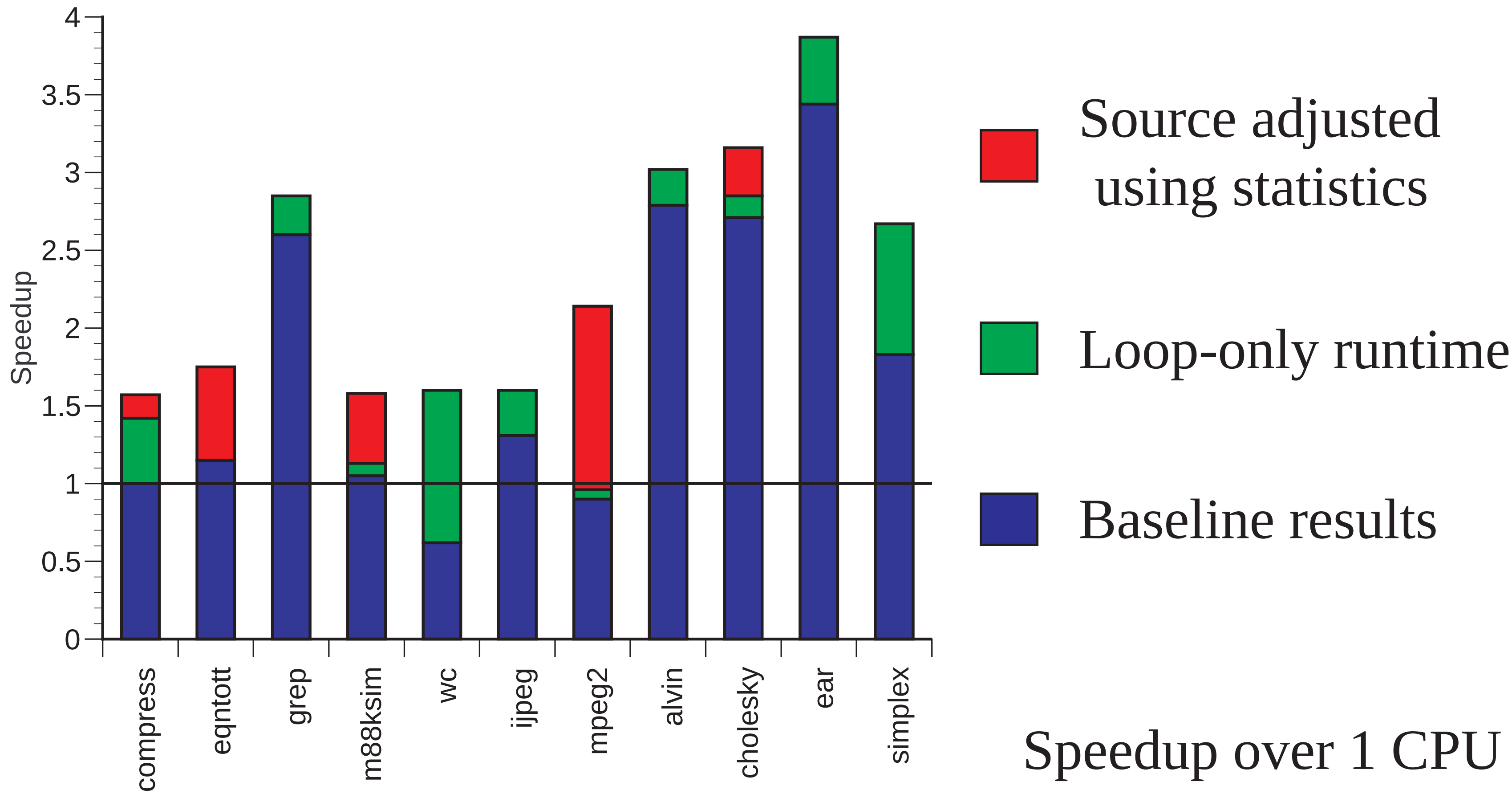
L2 Data Buffering

- Small buffers are sufficient
  — We used a fully associative line buffer
  — < 1 KB per thread captures most writes



Speculation Performance

— Results representative of entire uniprocessor applications
— Simulated with accurate modeling of Hydra's memory
— Enhanced performance versions were presented at ICS 99

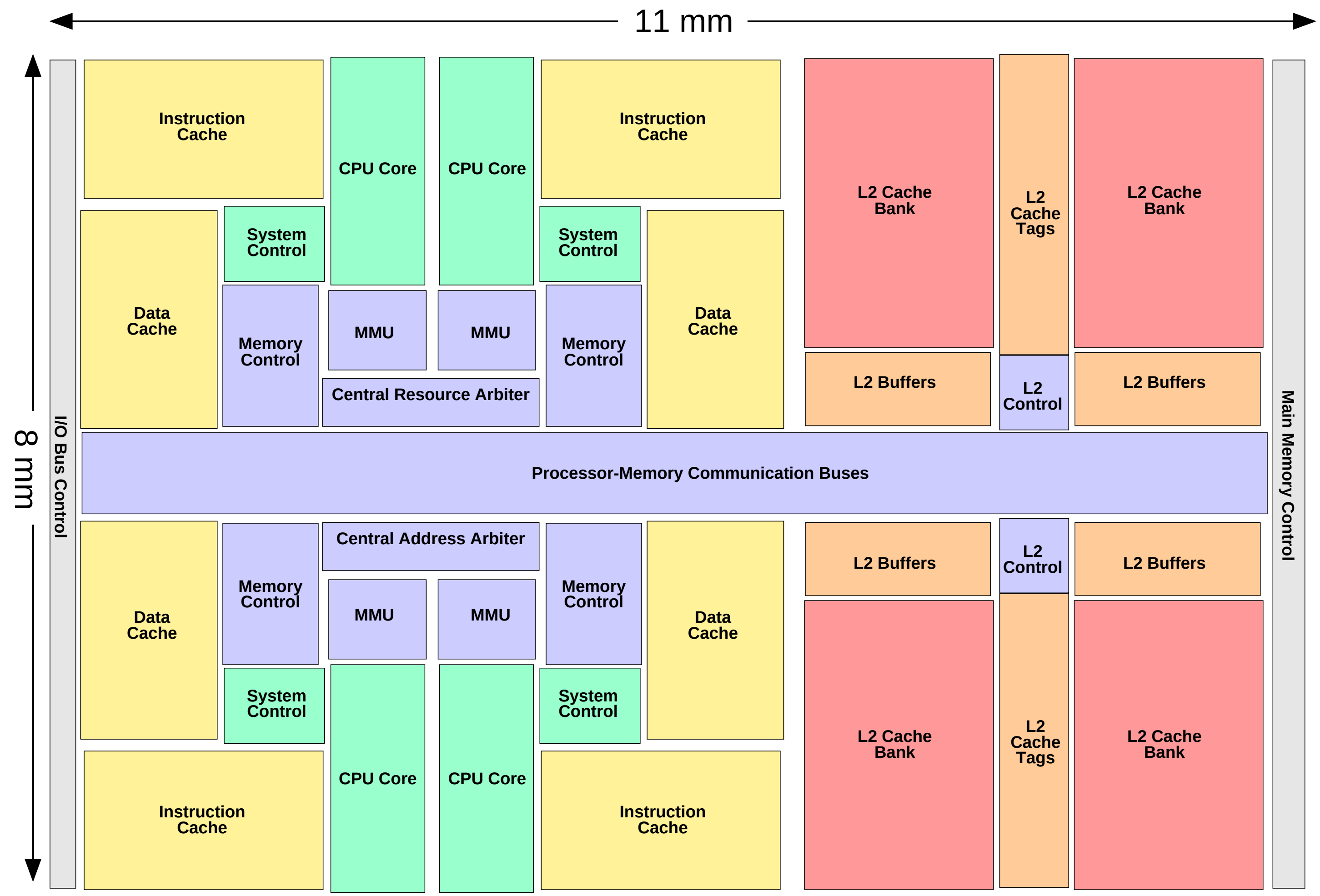


Speedup over 1 CPU

Prototype Overview

- CPU core and cache macros
- Speculative coprocessor
  — Speculative memory reference controller
  — Speculative interrupt screening mechanism
  — Statistics mechanisms for performance evaluation and to allow feedback for code tuning
- Memory system
  — Read and write buses
  — Controllers for all resources
  — On-chip L2 cache
  — Simple off-chip main memory controller
  — I/O and debugging interface

Prototype Layout



- Hardware design based on IDT RC32364
  - — 88 mm^2 in 0.25µm process with 8K I/8K D/~128K L2

Prototype Issues

- 250 MHz clock rate target
  - — Most critical parts are primarily in existing cores
  - — Pipelining in most of the memory system may change to meet timing requirements
- Central Bus Arbitration Mechanism
  - — High fan-in and fan-out gates
  - — Single cycle operation required here
- Drivers for long buses
- Road Map
  - — Finish synthesizable Verilog, this summer
  - — Finish circuit design and layout, H2 ’99
  - — Complete verification and tapeout, H1 ’00

Conclusions

- Hydra offers many advantages
  - Great performance on parallel applications
  - Good performance on most uniprocessor applications using data speculation mechanisms
  - Scalable, modular design
  - Speculative hardware does not add much to cost, yet greatly increases the number of parallel applications
- Prototype implementation
  - Will work out implementation details
  - Will allow us to validate our performance evaluations
  - Will provide a platform for application and compiler development
  - Will be the first implementation of a multiprocessor with speculative memory support