CS 258

Parallel Computer Architecture

Data Speculation Support for a Chip Multiprocessor (Hydra CMP)

Lance Hammond, Mark Willey and Kunle Olukotun

Presented:
May 7th, 2008
Ankit Jain

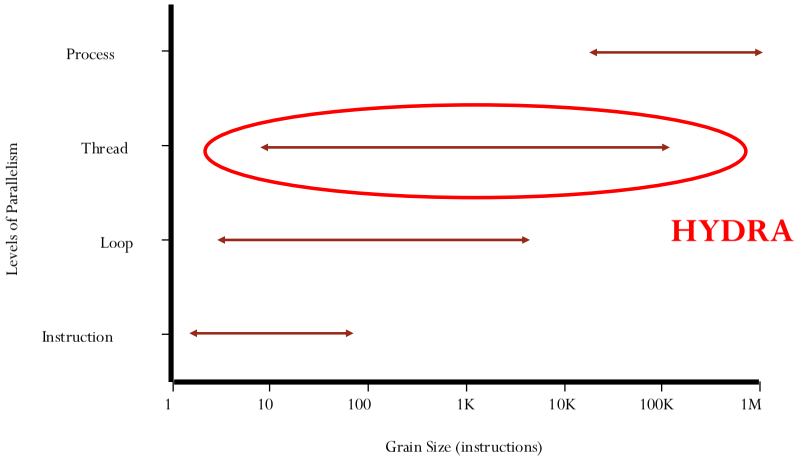
(Some slides have been adopted from Olukotun's talk to CS252 in 2000)

Outline

- The Hydra Approach
- Data Speculation
- Software Support for Speculation (Threads)
- Hardware Support for Speculation
- Results

The Hydra Approach

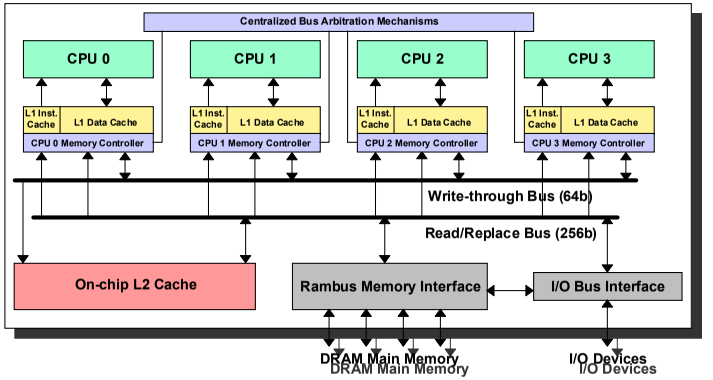
Exploiting Program Parallelism



Hydra Approach

- A single-chip multiprocessor architecture composed of simple fast processors
- Multiple threads of control
  - Exploits parallelism at all levels
- Memory renaming and thread-level speculation
  - Makes it easy to develop parallel programs
- Keep design simple by taking advantage of single chip implementation

The Base Hydra Design



- Single-chip multiprocessor
- Four processors
- Separate primary caches
- Write-through data caches to maintain coherence
- Shared 2nd-level cache
- Low latency interprocessor communication (10 cycles)
- Separate fully-pipelined read and write buses to maintain single-cycle occupancy for all accesses

Data Speculation

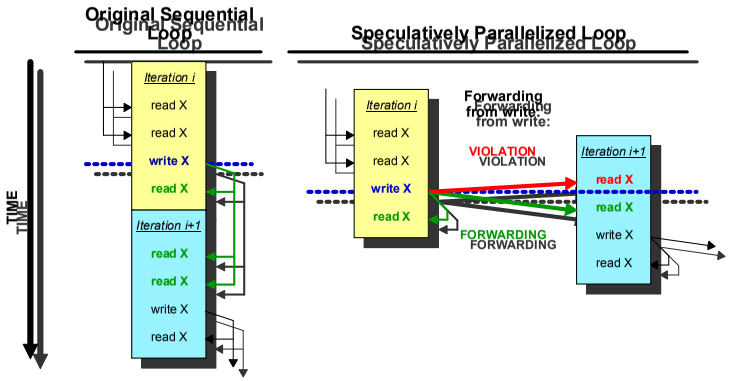
Problem: Parallel Software

- Parallel software is limited
  - Hand-parallelized applications
  - Auto-parallelized applications
- Traditional auto-parallelization of C-programs is very difficult
  - Threads have data dependencies ⇒ synchronization
  - Pointer disambiguation is difficult and expensive
  - Compile time analysis is too conservative
- How can hardware help?
  - Remove need for pointer disambiguation
  - Allow the compiler to be aggressive

Solution: Data Speculation

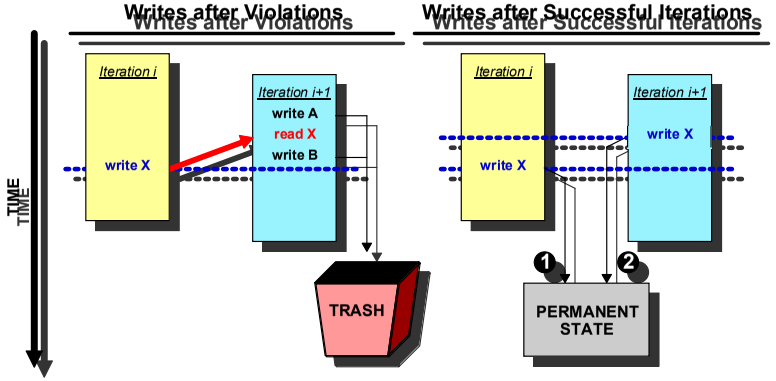
- Data speculation enables parallelization without regard for data-dependencies
  - Loads and stores follow original sequential semantics (committed in order using thread sequence number)
  - Speculation hardware ensures correctness
  - Add synchronization only for performance
  - Loop parallelization is now easily automated
- Other ways to parallelize code
  - Break code into arbitrary threads (e.g. speculative subroutines)
  - Parallel execution with sequential commits

Data Speculation Requirements I



- Forward data between parallel threads
- Detect violations when reads occur too early

Data Speculation Requirements II



- Safely discard bad state after violation
- Correctly retire speculative state
- Forward progress guarantee

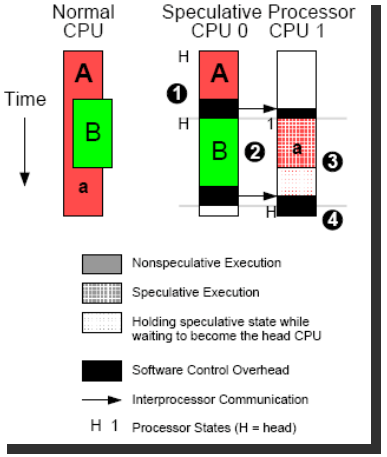
Data Speculation Requirements Summary

- Method for detecting true memory dependencies, in order to determine when a dependency has been violated.
- Method for backing up and re-executing speculative loads and any instructions that may be dependent upon them when the load causes a violation.
- Method for buffering any data written during a speculative region of a program so that it may be discarded when a violation occurs of permanently committed at the right time.

Software Support for Speculation

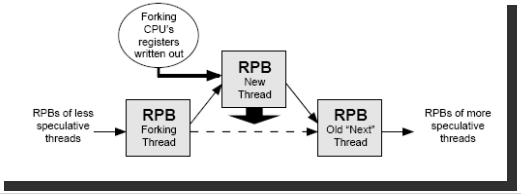
(Threads + Register Passing Buffers)

Thread Fork and Return



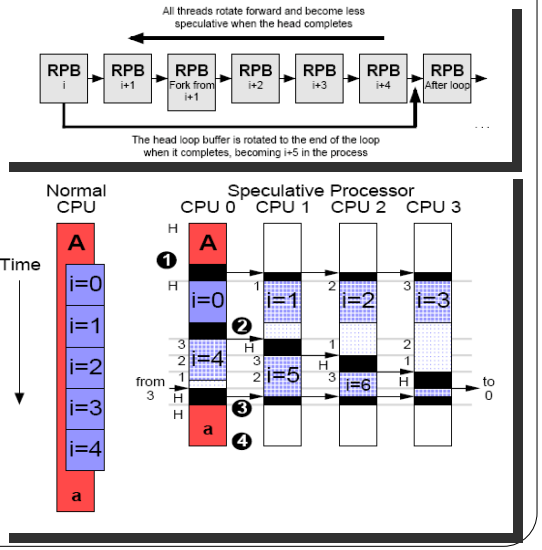
Register Passing Buffers (RPBs)

- Allocate one per thread
- Allocate once in memory at starting time so that can be loaded/re-loaded when thread is started/re-started
- Speculated values set using 'repeat last return value' prediction mechanism
- When a new RPB is allocated, it is added to 'active buffer list' from where free processors pick up the next-most-speculative thread



E.g.: Speculatively Executed Loop

- Termination Message sent from first processor that detects end-of-loop condition.
- Any speculative processors that executed iterations 'beyond the end of the loop' are cancelled and freed.
- Justifies need for precise exceptions
  - Operating system call or exception can only be called from a point that would be encountered in the sequential execution.
  - Thread is stalled until it becomes the head processor.

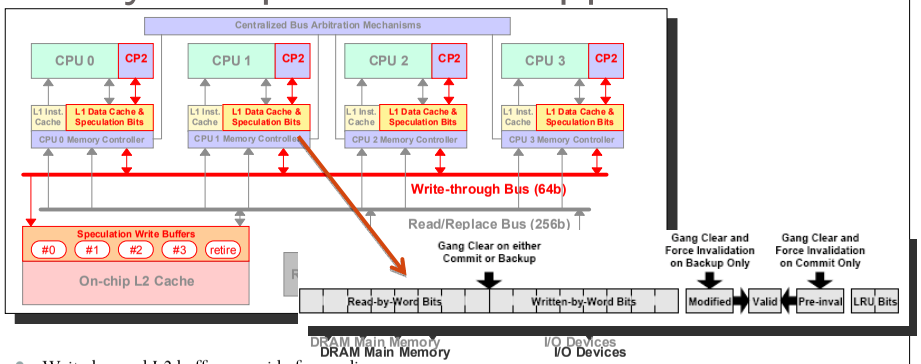


Miscellaneous Issues

- Thread Size
  - Limited Buffer Size
  - True dependencies
  - Restart length
  - Overhead
- Explicit Synchronization
  - Protects
  - Used to improve performance
  - Not needed for correctness
- Ability to dynamically turn off speculation when there are parallel threads in code (@ runtime)
- Ability to share threads with OS (speculative threads give up processors)

Hardware Support for Speculation

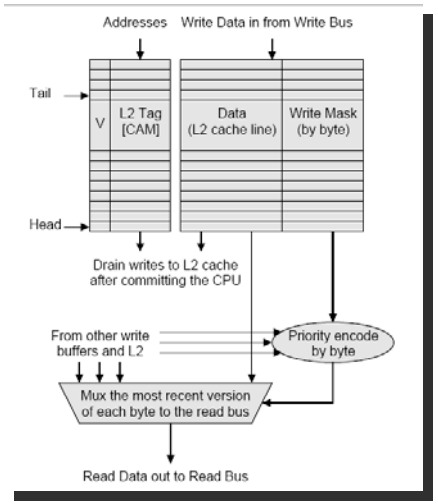
Hydra Speculation Support



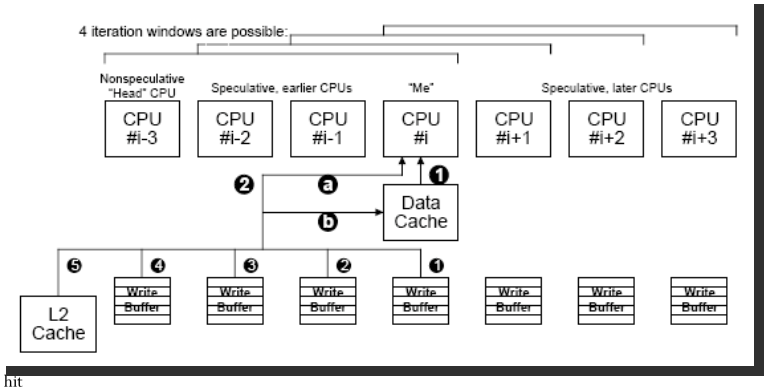
- Write bus and L2 buffers provide forwarding
- "Read" L1 tag bits detect violations
- "Dirty" L1 tag bits and write buffers provide backup
- Write buffers reorder and retire speculative state
- Separate L1 caches with pre-invalidation & smart L2 forwarding to provide "multiple views of memory"
- Speculation coprocessors to control threads

Secondary Cache Write Buffers

- Data forwarded to more speculative processors based on Write Masks (by byte)
- Drain only set bytes to L2 Cache on commit
- More buffers than processors in order allow execution to continue as draining happens
- Processor keeps tags of written lines in order to calculate when buffer will overflow and then halt process until it is the 'head processor'

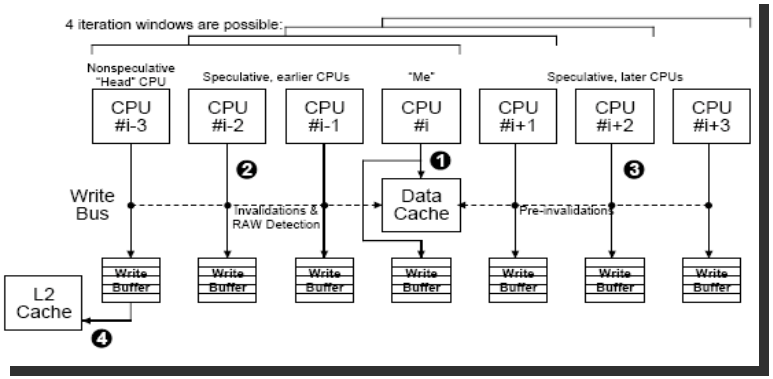


Speculative Loads (Reads)



- L1 hit
  - The read bits are set
- L1 miss
  - L2 and write buffers are checked in parallel
  - The newest bytes written to a line are pulled in by priority encoders on each byte (priority 1-5)
  - Read and modified bits for appropriate read bytes are set in L1

Speculative Stores (Writes)



- A CPU writes to its L1 cache & write buffer
- "Earlier" CPUs invalidate our L1 & cause RAW hazard checks
- "Later" CPUs just pre-invalidate our L1
- Non-speculative write buffer drains out into the L2

Results

Results (1/3)

Benchmark	Speedup	Uniprocessor **Data Cache Miss Rate (%)**	Speculative **Data Cache Miss Rate (%)**	% increase in **Loads**
wc	0.62	0.61	18.64	548.5
wc (w/ delay)	0.66		13.47	322.6
m88ksim	1.04	0.54	11.34	282.1
compress	1.00	4.35	15.31	-6.4
compress (w/ synchronization)	1.09		14.27	-10.9
ijpeg	1.51	0.69	5.76	63.4
vortex	0.58	1.59	13.06	38.3

Table 4. Benchmark performance summary.

Results (2/3)

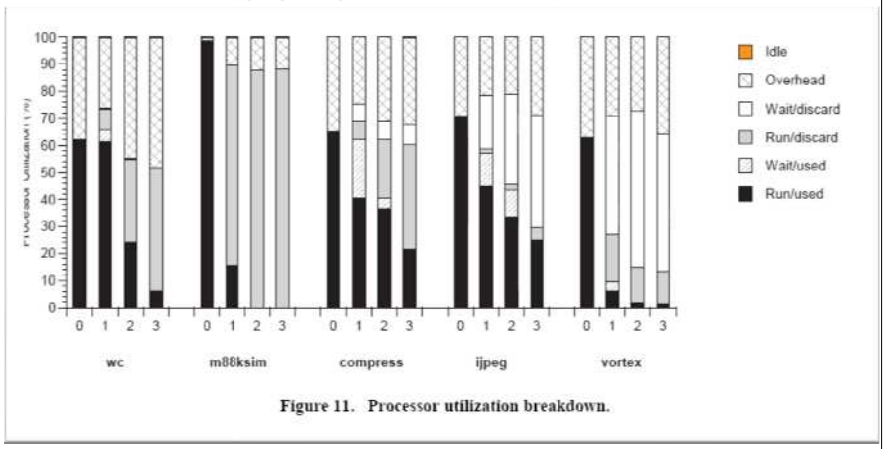


Figure 11. Processor utilization breakdown.

27 cycles

4000 cycles

140 cycles

occasional dependencies

too many dependencies

Results (3/3)

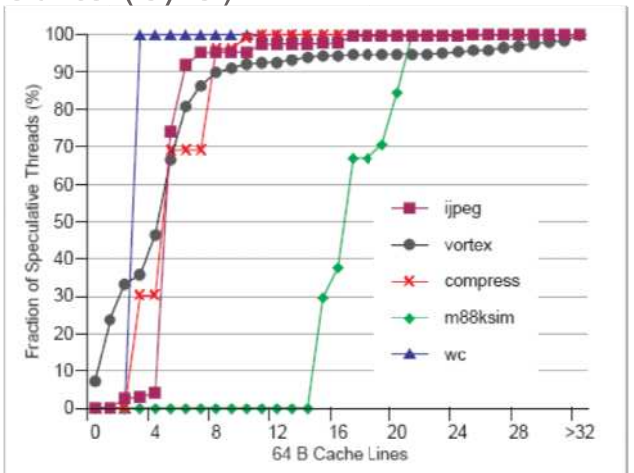


Figure 12. Speculative store buffer size requirements.

Conclusion

- Speculative support is only able to improve performance when there is a substantial amount of medium–grained loop-level parallelism in the application.
- When the granularity of parallelism is too small or there is little inherent parallelism in the application, the overhead of the software handlers overwhelms any potential performance benefits from speculative-thread parallelism.

Extra Slides

Tables and Charts

	L1 Cache	L2 Cache	Main Memory
Configuration	Separate I & D SRAM cache pairs for each CPU	Shared, on-chip SRAM cache	Off-chip DRAM
Capacity	16KB each	2 MB	128 MB
Bus Width	64-bit connection to CPU	256-bit read bus + 64-bit write bus	32-bits of Rambus (running at the full CPU speed)
Access Time	1 CPU cycle	5 CPU cycles	at least 50 cycles
Associativity	4-way	4-way	N/A
Line Size	32 bytes	32 bytes	4 KB pages
Write Policy	Writethrough, no allocate on write	Writeback, allocate on writes	"Writeback" (virtual memory)
Inclusion	N/A	Inclusion not enforced by L2 on L1 caches	Includes all cached data

Table 1. Hydra memory hierarchy characteristics.

First: CPU *i*	Then: CPU *i+1*	Action
R	R	—
R	W	Rename in *i+1*
W	R	Forward written data from *i* to *i+1*
W	W	Forward occurs, but then *i+1* renames and overwrites its copy of the data

(a)

First: CPU *i+1*	Then: CPU *i*	Action
R	R	—
W	R	Rename in *i+1*
R	W	RAW Hazard: *i+1* must restart
W	W	Rename in *i+1*, so later forwarding from *i* is ignored

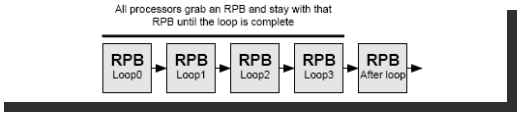
(b)

Table 2. Desired speculative memory behavior.

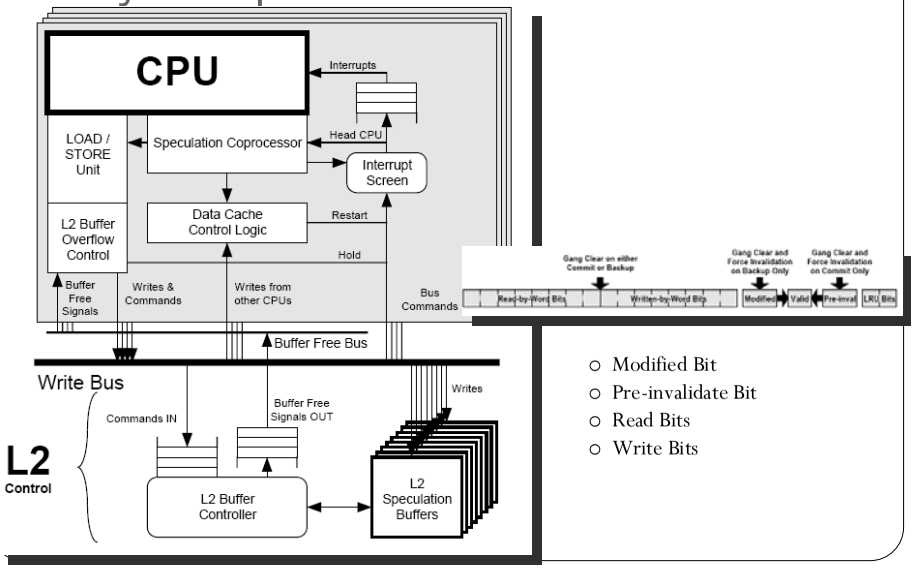
Group	Handler	Use	What it does	Overhead (instr.)
Subroutines	Fork	Exception, generated upon execution of a jump-to-subroutine (JALR) instruction	Allocates an RPB, saves necessary processor state into the RPB, and then sends a FORK command out to the most speculative (or free) processor.	~70
	Subroutine Completion	Routine returned to after a forked subroutine completes, instead of the normal return address	Waits to become the head, commits speculative data, checks the guessed return value from the last fork against the actual one, kills speculative processors if necessary, and then continues with a "Start Buffer" command to get a new thread.	~40 + full "Start Buffer"
	Start Buffer	Jumped to by completion commands or an exception triggered by a FORK or STARTLOOP	Selects one of the first 4 RPBs on the active list, deletes any old children generated by this thread during any restarted earlier executions, reads in its contents, and begins executing the subroutine or loop thread. If insufficient threads are available, the processor is freed.	Full: ~60 (loop) or ~75 (fork) Short: ~25 (loop) or ~40 (fork)
Loops	Start Speculative Loop	Called by spec_begin() in original program code	Sets up a ring of RPBs for "slow" loop operation, configuring their buffers correctly, then sends out a STARTLOOP bus command to other processors.	~75
	Start Speculative Loop (quick version)	Called by spec_begin_quick() in original program code	Sets up 4 RPBs for "quick" loop operation, configuring their buffers correctly, then sends out a STARTLOOP bus command to other processors.	~70
	End-of-Iteration / Terminate	Routine returned to after a speculative loop iteration completes	Waits to become the head, commits speculative data, moves its RPB to the end of the loop, making adjustments to it if it was used for a fork during the loop iteration, and claims the next available iteration with a COMPLETE command, or commits its speculative data, discards all of the RPBs within the entire loop, and kills all other loop iterations with a KILL command before picking up the buffer following the loop with a "Start Buffer" command.	~45 + short "Start Buffer"
	End-of-Iteration / Terminate (quick version)	Routine returned to after a speculative loop iteration completes	Same as slow version, except without RPB manipulation and adjustments during the end of every iteration.	16 (on EOI) or ~45 + short "Start Buffer" (on terminate)
Support	Violation: Initiate	Exception, initiated by a RAW hazard detection in the L1 cache speculation logic	Causes speculative data to be discarded and the iteration to be restarted, and sends out a KILL bus command to following CPUs.	~30 (20 if in a quick loop)
	Violation: Receive	Exception, initiated by KILL bus command	Causes speculative data to be discarded and the iteration to be restarted.	8 + full "Start Buffer" (11 if in a quick loop)
	Hold: Buffer Full	Exception, initiated by buffer-management hardware within each CPU, due to a full L2 buffer or the possible loss of read-bit information due to an L1 replacement	Causes the CPU to stop until it becomes the head CPU, when it is allowed to continue non-speculatively, bypassing the full buffer.	15
	Hold: Exception	Exception, initiated in place of a "normal" instruction exception or system call trap encountered during speculation	Causes the CPU to stop until it becomes the head CPU, when it is allowed to continue non-speculatively with the SYSCALL or instruction exception.	25 + OS time

Table 3. A summary of the software handlers required to support speculation

Quick Loops



Hydra Speculation Hardware



- Modified Bit
- Pre-invalidate Bit
- Read Bits
- Write Bits