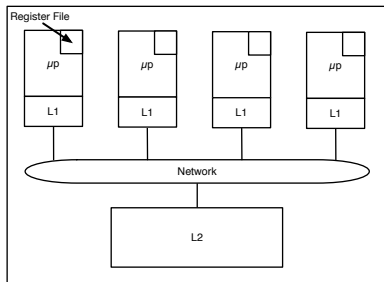


CS533: Speculation (I)

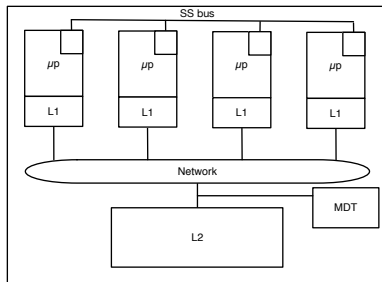
Josep Torrellas

University of Illinois in Urbana-Champaign

Architecture

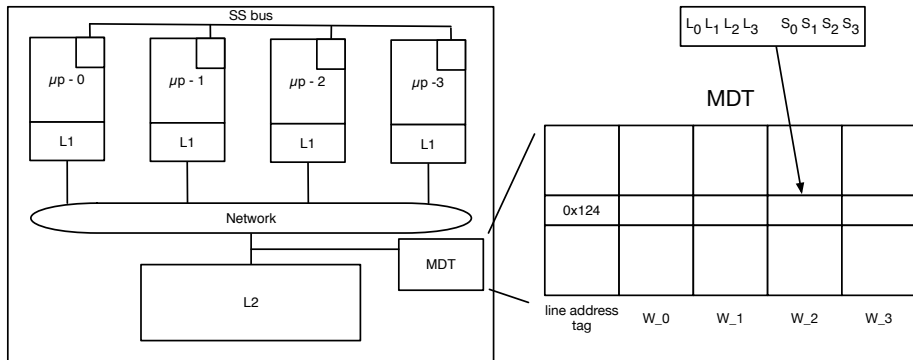


Traditional Chip Multiprocessor (CMP)



CMP + Speculation

MDT Architecture



- Round Robin
- No flush of dirty data

Transition

speculative $\rightarrow$ non-speculative $\rightarrow$ commit $\rightarrow$ speculative

- speculative:
 - no displacement of L1 Dirty lines; else stall
- non-speculative:
 - write-through. Displacement OK
- commit:
 - synch: flush all Dirty data to L2

In-order commit

Extra Bits per L1 Line

For now: per-word bits

Flush (F)	Invalid (I)	Dirty (D)	Safe Read (SR)	Safe Write (SW)
-----------	-------------	-----------	----------------	-----------------

- SR/SW: Allow ld/st without informing MDT
- F: Allow L1 lines to remain valid across thread initialization
- D: Data not consistent with L2
 - may be different in different L1's
- I: No data

Example

Valid	Address Tag	Ld bits (W_0)	St bits (W_0)
		L ₀ L ₁ L ₂ L ₃	S ₀ S ₁ S ₂ S ₃
1	0x1234	0 0 1 0	0 1 0 0
1	0x4321	0 0 1 1	0 1 0 0

Load: MDT Operation

if speculative **then**

/ access MDT to find who's got data and to record the LD */*

if no entry **then**

allocate one

end if

$L_i = 1$

Who has data: closest predecessor (including myself) w/ $S_i = 1$

/ Use mask bits */*

Forward request to right L1 */* maybe non-spec */*

/ if none, send it to L2 */*

end if

Store: High Level Idea

- Must be careful when performing a write and $SW \neq 0$
 - For predecessors, set $F=1$ (want to flush data for future iterations)
 - For successors, one may have to either invalidate cache lines or squash threads

Store: MDT Operation I

/ access MDT to squash successors with premature loads, send invalidations, and record the ST */*

if no entry found and speculative **then**

 allocate

end if

if speculative **then**

$S_i = 1$

end if

Evaluate the Squash Logic: “has any successor done an unsafe ld without any intervening thread performing a ST”?

e.g. if 0 non-speculative: $L_1 + \bar{S}_1(L_2 + \bar{S}_2 L_3)$

/ use mask bits complemented */*

Store: MDT Operation II

if True then

 squash all successors starting with the one with $L = 1$
 at same time: invalidate the entry from their caches

else

 invalidate up to (not including) the successor that redefines the word
($S=1, L=0$)

/ because of data from previous iteration */*

end if

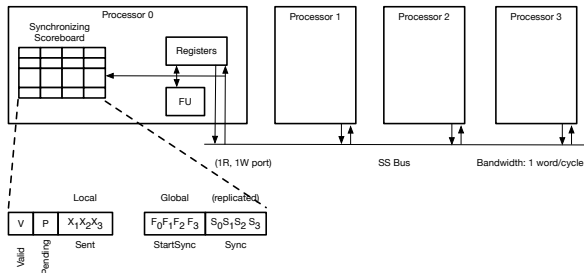
Send signal to all predecessors: $F=1$

- The MDT may overflow
- If so, thread trying to add entry (must be speculative) $\rightarrow$ stalls
- When non-speculative commits: clear L,S $\rightarrow$ may free up
- A small MDT is Ok

Details on MDT

- when a thread commits or gets squashed: its L,S are cleared
- MDT may be incorporated in L2
- Like a directory: Keeps track of who's sharing the data
- Overall, small size (see later)

Hardware Support → Enhanced Scoreboard



Synchronizing Scoreboard (SS)

- Local bits: V, P, X
- Rellicated Global bits: F, S
 - consistency easily maintained
 - F: when a thread is initiated
 - S: when value is written on the bus

Register Transfer Bus (SS bus)

- Transfer register values between processors
- Simple Broadcast bus
 - limited bandwidth: 1 word/cycle
 - only 1 read & 1 write port per processor
 - latency: 1-3 cycles + contention

Register Communication

- Producer initiated:
 - Thread clears S bit, puts reg on SS bus
 - Successors check own V, F of threads between producer and itself
 - if all $\emptyset$: load reg, $V = 1$
- Consumer initiated:
 - Check own V bit and F, S of predecessors
 - Available? e.g. $V_3 + \bar{S}_2(F_2 + \bar{S}_1(F_1 + S_0))$
 - If so, read and set $V = 1$. Else stall

Process to convert *Execution* to *Annotated Execution*

Step One

- Identify basic blocks
- Generate control flow graph
- Perform live variable and reaching definition analysis

Step Two

- Identify loops using dominator and back-edge information
- Annotate task initiation and termination points

Step Three

- Get *loopleftive* registers (i.e. registers live at loop entry/exits & also redefined in loop)
- Identify *loopleftive* reaching definitions at all exits
- Identify safe definitions & release points for all *loopleftive* reaching definitions
- Identify induction variables & move the induction updates that dominate all exists to entry point
- Annotate safe/release points

