CS184c: Computer Architecture [Parallel and Multithreaded]

Day 8: April 26, 2001

Simultaneous Multi-Threading (SMT)

Shared Memory Processing (SMP)

Note

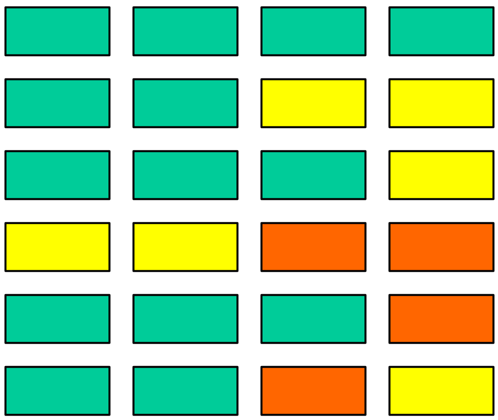
- No class Tuesday
  - Time to work on project
  - [andre at FCCM]
- Class on Thursday

Today

- SMT
- Shared Memory
  – Programming Model
  – Architectural Model
  – Shared-Bus Implementation

SMT

SMT Promise

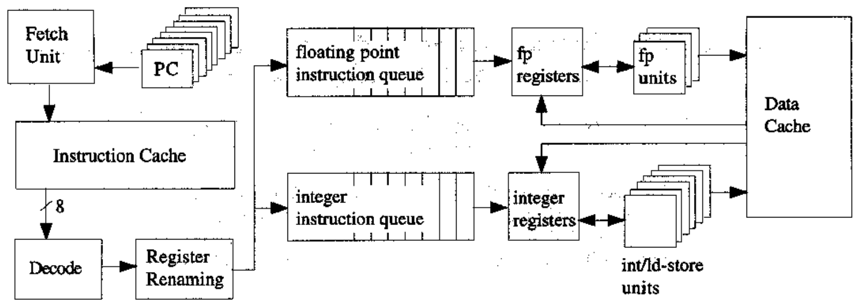


Fill in empty slots with other threads

SMT uArch

- Observation: exploit register renaming
  - Get small modifications to existing superscalar architecture

SMT uArch



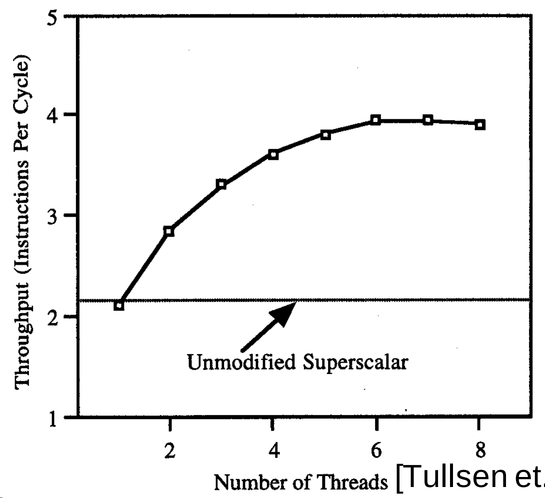
- N.B. remarkable thing is how similar superscalar core is

[Tullsen et. al. ISCA '96]

SMT uArch

- Changes:
  - Multiple PCs
  - Control to decide how to fetch from
  - Separate return stacks per thread
  - Per-thread reorder/commit/flush/trap
  - Thread id w/ BTB
  - Larger register file
    - More things outstanding

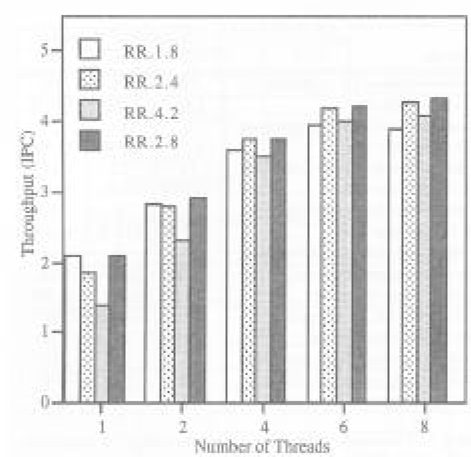
Performance



[Tullsen et. al. ISCA '96]

Optimizing: fetch freedom

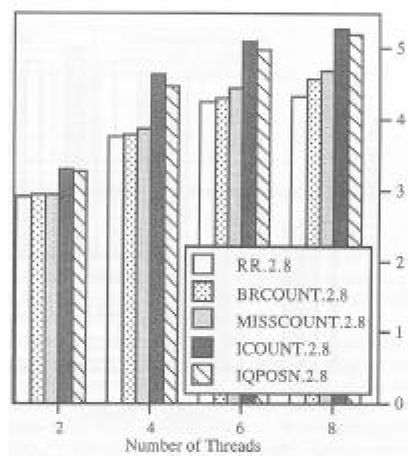
- RR=Round Robin
- RR.X.Y
  - X – threads do fetch in cycle
  - Y – instructions fetched/thread



[Tullsen et. al. ISCA '96]

Optimizing: Fetch Alg.

- ICOUNT – priority to thread w/ fewest pending instrs
- BRCOUNT
- MISSCOUNT
- IQPOSN – penalize threads w/ old instrs (at front of queues)



[Tullsen et. al. ISCA '96]

Throughput Improvement

- 8-issue superscalar
  - Achieves little over 2 instructions per cycle
- Optimized SMT
  - Achieves 5.4 instructions per cycle on 8 threads
- 2.5x throughput increase

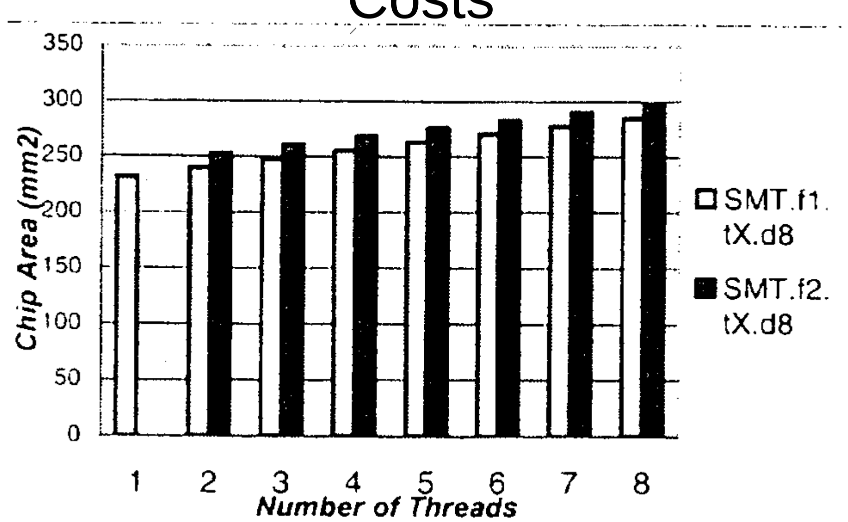
Costs

Table 6) Adding SMT, shows area increase relative to MIPS R10K-2x

Function	Chip Block	MIPS R10K-2x in 0.18μ (mm²)	MIPS R10K-2x (0.18μ) with SMT (mm²)	Relative area increase of adding SMT			
				% Increase of adding SMT to R10K-2x block	% increase versus core area	% increase versus chip area excluding L2 cache	% increase versus chip area including L2 cache
Dcache	Dcache	11.4	11.4	0		0.0	0.0
	Dtag	1.6	1.6				
Icache	Icache	9.1	13.7	50		2.9	2.2
	Itag	1.3	1.9				
TLB	TLB	4.4	5.7	30		0.7	0.6
Fetch	Fetch	1.0	4.6	157	5.6	4.1	3.1
	Bpred	3.6	7.2				
Decode	Decode	2.3	4.5	96	1.7	1.2	0.9
Out-of-Order execution	Remap-Logic	2.5	3.3	48	19.4	13.9	10.6
	Remap-tables	2.4	16.2				
	FreeList	2.3	2.9				
	IQ	7.0	8.8				
	LSQ	9.4	11.8				
	FPQ	6.3	8.0				
	Reorder	6.1	7.8				
	RAS	0.3	2.1				
Register Files	IntRF	5.7	18.8	231	20.0	14.3	10.9
	FP-RF	5.3	17.6				
Arithmetic Units	IntFU	7.6	7.6	0	0	0	0
	FPMUL	4.0	4.0				
	FPALU	8.3	8.3				
Miscellaneous	Earlin	5.2	5.2	0		0	0
	JTAG	0.9	0.9				
	Misc.	2.4	2.4				
	I/O	13.7	13.7				
Routing	Routing	52.7	52.7	0	0	0	0
256K L2 Cache		55	55	0			0
Total	core	126.7	188.8		46.7		
	Chip w/o L2 cache	176.7	242.7			37.3	
	Chip w/ L2 cache	231.7	297.7				28.3

[Burns+Gaudiot HPCA'99]

Costs



[Burns+Gaudiot HPCA'99]

Not Done, yet…

- Conventional SMT formulation is for coarse-grained threads
- Combine SMT w/ TAM ?
  - Fill pipeline from multiple runnable threads in activation frame
  - ?multiple activation frames?
  - Eliminate thread switch overhead?

Thought?

- SMT reduce need for split-phase operations?

Big Ideas

- Primitives
  - Parallel Assembly Language
  - Threads for control
  - Synchronization (post, full-empty)
- Latency Hiding
  - Threads, split-phase operation
- Exploit Locality
  - Create locality
    - Scheduling quanta

Shared Memory

Shared Memory Model

- Same model as multithreaded uniprocessor
  - Single, shared, global address space
  - Multiple threads (PCs)
  - Run in same address space
  - Communicate through memory
    - Memory appear identical between threads
    - Hidden from users (looks like memory op)

That's All?

- For correctness have to worry about synchronization
  - Otherwise non-deterministic behavior
  - Recall threads run asynchronously
  - Without additional/synchronization discipline
    - Cannot say anything about relative timing
    - [Dataflow had a synchronization model]

Day 6

Future/Side-Effect hazard

- (define (decrement! a b)
  - (set! a (- a b)))
- (print (* (future (decrement! c d))
- (future (decrement! d 2))))

Multithreaded Synchronization

- (define (decrement! a b)
  - (set! a (- a b)))
- (print (* (future (decrement! c d))
- (future (decrement! d 2))))
- Problem
  - Ordering matters
  - No synchronization to guarantee ordering

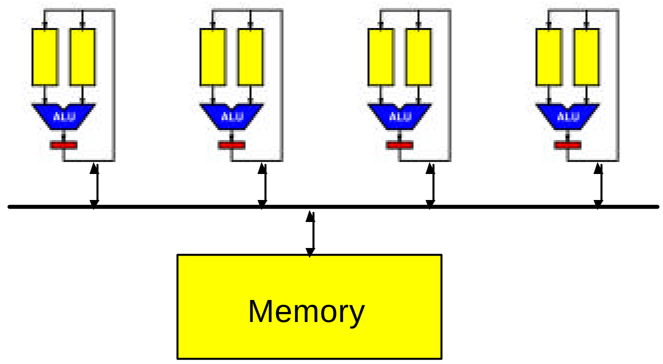
Synchronization

- Already seen
  - Data presence (full/empty)
- Barrier
  - Everything before barrier completes before anything after barrier begins
- Locking
  - One thread takes exclusive ownership
- …we'll have to talk more about synch.

Models

- **Conceptual model**:
  - Processor per thread
  - Single shared memory
- **Programming Model**:
  - Sequential language
  - Thread Package
  - Synchronization primitives
- **Architecture Model**: Multithreaded uniprocessor

Conceptual Model



Architecture Model Implications

- Coherent view of memory
  - Any processor reading at time X will see same value
  - All writes eventually effect memory
    - Until overwritten
  - Writes to memory seen in same order by all processors
- Sequentially Consistent Memory View

Sequential Consistency

- P1: A = 0
-
- A = 1
- L1: if (B==0)

- P2: B = 0
- B = 1
- L2: if (A==0)

Can both conditionals be true?

Coherence Alone

- Coherent view of memory
  - Any processor reading at time X will see same value
  - All writes eventually effect memory
    - Until overwritten
  - Writes to memory seen in same order by all processors
- Does not guarantee sequential consistency

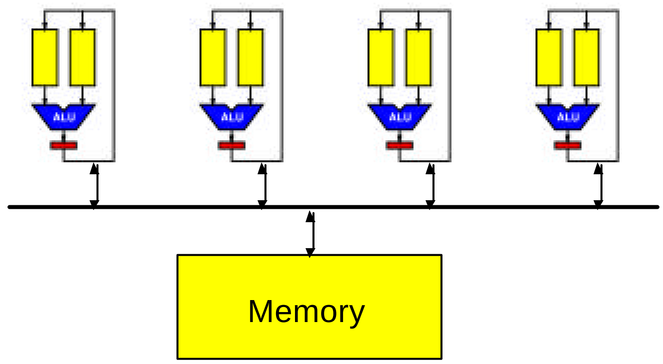
Consistency

- …there are less strict consistency models…

Implementation

Naïve

- What's wrong with naïve model?



What's Wrong?

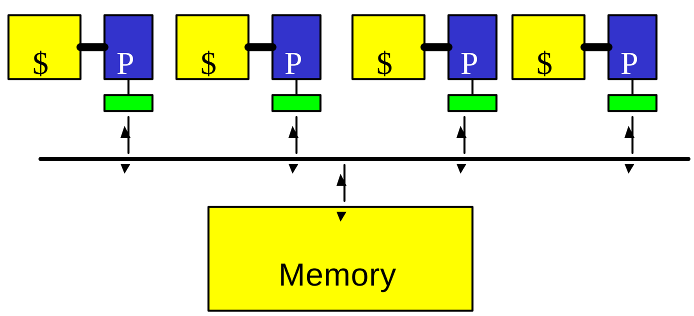
- Memory bandwidth
  - 1 instruction reference per instruction
  - 0.3 memory references per instruction
  - 1ns cycle
  - N*1.3 Gwords/s ?
- Interconnect
- Memory access latency

Optimizing

- How do we improve?

Naïve Caching

- What happens when add caches to processors?



Naïve Caching

- Cached answers may be stale
- Shadow the correct value

How have both?

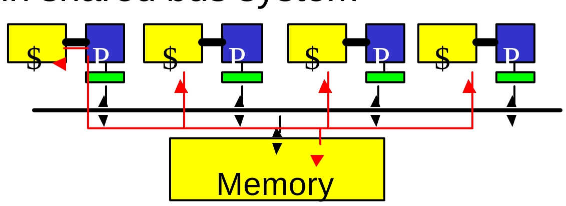
- Keep caching
  - Reduces main memory bandwidth
  - Reduces access latency
- Satisfied Model

Cache Coherence

- Make sure everyone sees same values
- Avoid having stale values in caches
- At end of write, all cached values should be the same

Idea

- Make sure everyone sees the new value
- Broadcast new value to everyone who needs it
  - Use bus in shared-bus system



Effects

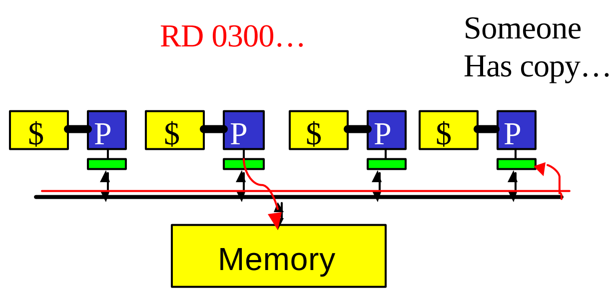
- Memory traffic is now just:
  - Cache misses
  - All writes

Additional Structure?

- Only necessary to write/broadcast a value if someone else has it cached
- Can write locally if know sole owner
  - Reduces main memory traffic
  - Reduces write latency

Idea

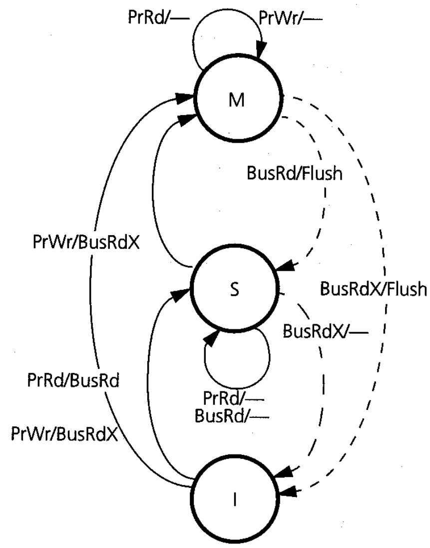
- Track usage in cache state
- “Snoop” on shared bus to detect changes in state



Cache State

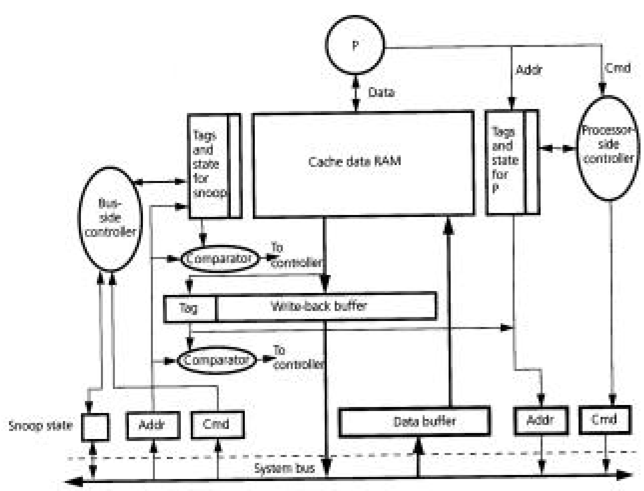
- Data in cache can be in one of several states
  - Not cached (not present)
  - Exclusive
    - Safe to write to
  - Shared
    - Must share writes with others
- Update state with each memory op

Cache Protocol



[Culler/Singh/Gupta 5.13]

Snoopy Cache Organization

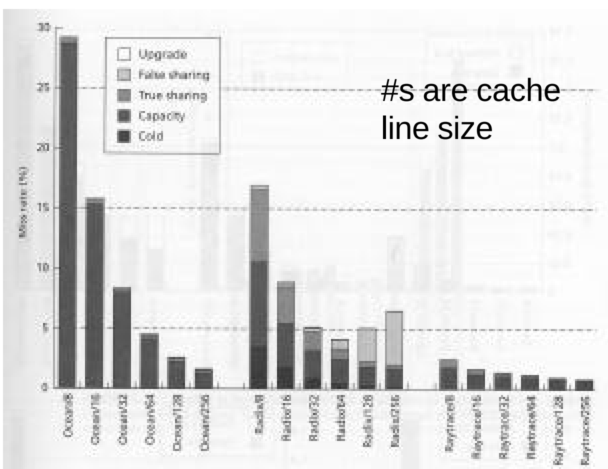


[Culler/Singh/Gupta 6.4]

Cache States

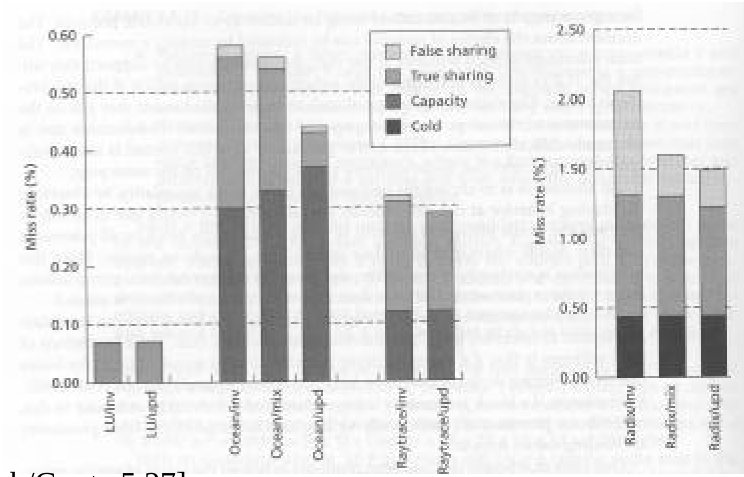
- Extra bits in cache
  - Like valid, dirty

Misses



[Culler/Singh/Gupta 5.23]

Misses



[Culler/Singh/Gupta 5.27]

Big Ideas

- Simple Model
  - Preserve model
  - While optimizing implementation
- Exploit Locality
  - Reduce bandwidth and latency