CS184b: Computer Architecture (Abstractions and Optimizations)

Day 11: April 30, 2003
Multithreading

Today

- Multitasking/Multithreading model
- Fine-Grained Multithreading
- SMT (Symmetric Multi-Threading)

Problem

- Long latency of operations
  - IO or page-fault
  - Non-local memory fetch
    - Main memory, L3, remote node in distributed memory
  - Long latency operations (mpy, fp)
- Wastes processor cycles while stalled
- If processor stalls on return
  - Latency problem turns into a throughput (utilization) problem
  - CPU sits idle

Idea

- Run something else useful while stalled
- In particular, another process/thread
  - Another PC
- Use parallelism to “tolerate” latency

Old Idea

- Share expensive machine among multiple users (jobs)
- When one user task must wait on IO
  - Run another one
- Time multiplex machine among users

Mandatory Concurrency

- Some tasks must be run "concurrently" (interleaved) with user tasks
  - DRAM Refresh
  - IO
    - Keyboard, network, …
  - Window system (xclock…)
  - Autosave ☺
  - Clippy ☹

Other Useful Concurrency

- Print spooler
- Web browser
  - Download images in parallel
- Instant Messanger/Zephyr (Gale)
- biff/xbiff/xfaces…

Multitasking

- Single machine run multiple tasks
- Machine provides same ISA/sequential semantics to each task
  - Task believes it own machines
  - Same as if other tasks running on different machines
- Tasks isolated from one another
  - Cannot affect each other functionally
  - (may impact each other's performance)

Each task/process

- **Process** – virtualization of the CPU
  - Has own unique set of state:
    - PC
    - Registers
    - VM Page Table (hence memory image)

Sharing the CPU

- Save/Restore
  - PC/Registers/Page Table
- Virtual Memory Isolation
- Privileged system software
  - User/System mode execution
- Functionally, task not notice that it gave up the CPU for period of time

Threads

- **Threads** – separate PC, but shares and address space
- Has own processor state:
  - PC
  - Registers
- Shares
  - Memory
  - VM Page Table
- Process may have multiple threads

Multitasking/Multithreading

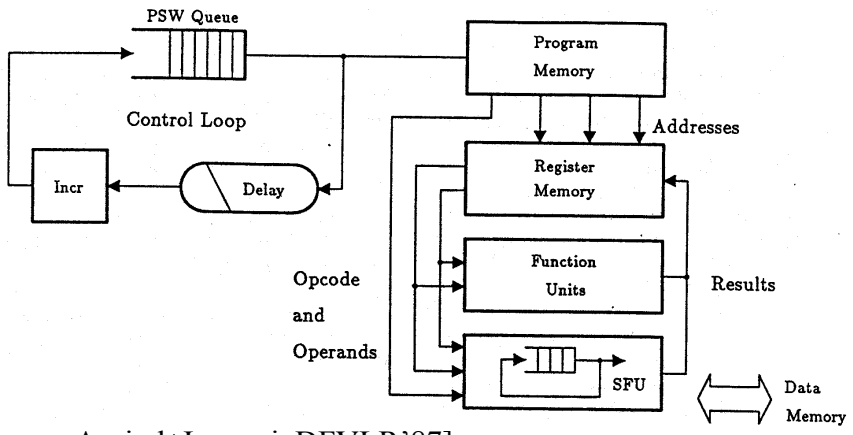
- Gives us an initial model for parallelism
- So far, parallelism of unrelated tasks
- Eventually, cooperating
  - Have to address concurrent memory model
  - (next time)

Fine Grained

HEP/μUnity/Tera

- Provide a number of contexts
  - Separate PCs, register files, …
- Number of contexts ≥ operation latency
  - Pipeline depth
  - Roundtrip time to main memory
- Run each context in round-robin fashion

HEP Pipeline



[figure: Arvind+Innucci, DFVLR'87]

Strict Interleaved Threading

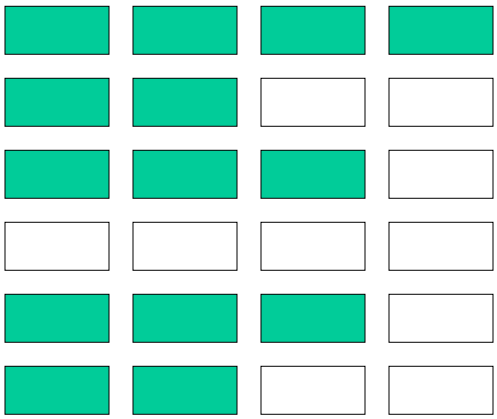
- Uses parallelism to get throughput
  - Avoid interlocking, bypass…
  - Cover memory latency
  - Essentially C-slow transformation of processor
- Potentially poor single-threaded performance
  - Increases end-to-end latency of thread

SMT

Superscalar and Multithreading?

- Do both?
- Issue from multiple threads into pipeline
- No worse than (super)scalar on single thread
- More throughput with multiple threads
  - Fill in what would have been empty issue slots with instructions from different threads

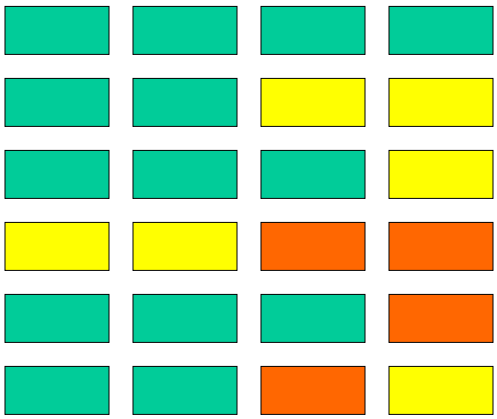
SuperScalar Inefficiency



Unused Slot

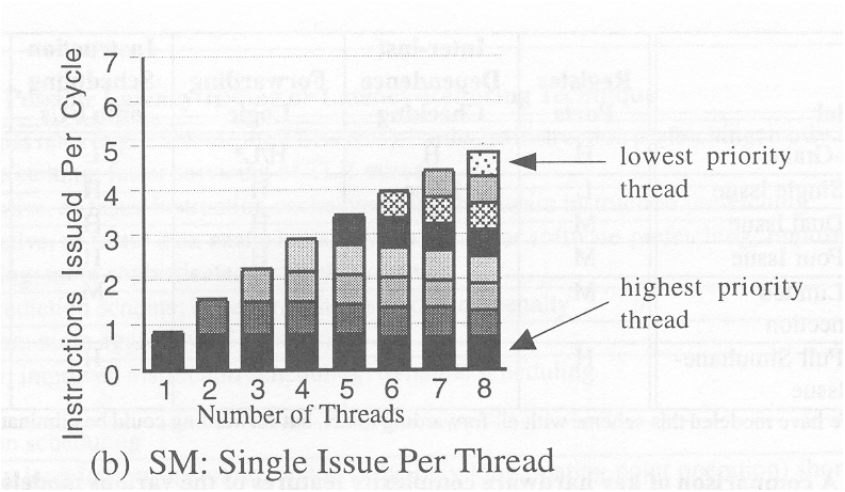
Recall: limited Scalar IPC

SMT Promise



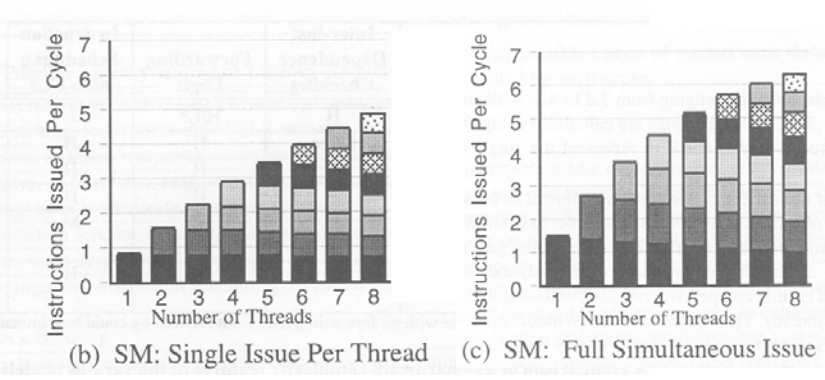
Fill in empty slots with other threads

SMT Estimates (ideal)



[Tullsen et. al. ISCA '95]

SMT Estimates (ideal)

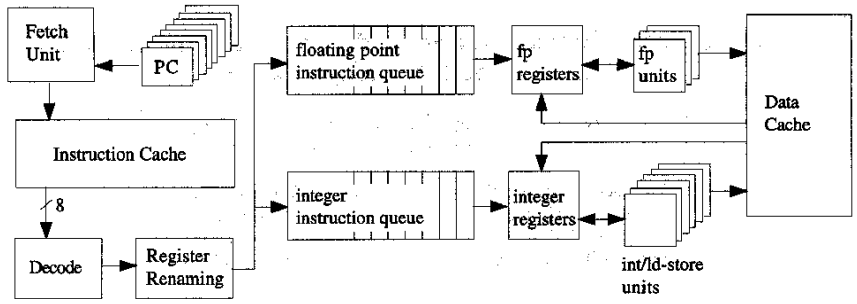


[Tullsen et. al. ISCA '95]

SMT uArch

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

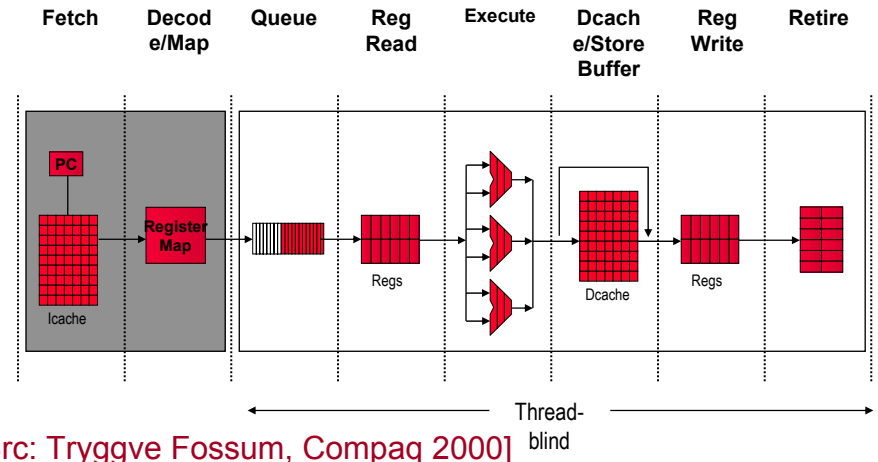
SMT uArch



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

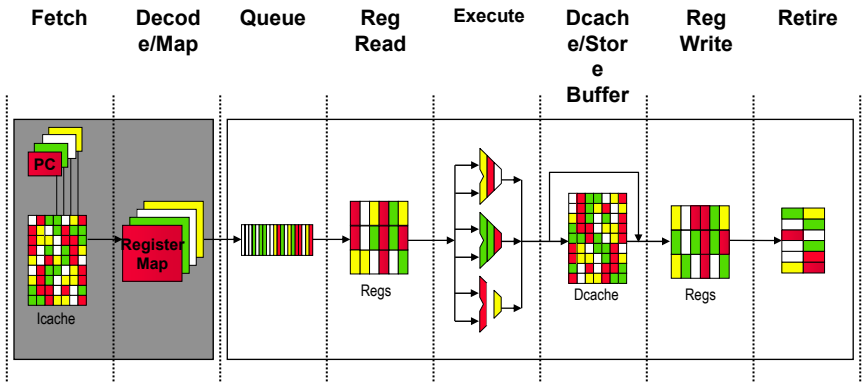
[Tullsen et. al. ISCA '96]

Alpha: Basic Out-of-order Pipeline



[Src: Tryggve Fossum, Compaq 2000]

Alpha SMT Pipeline

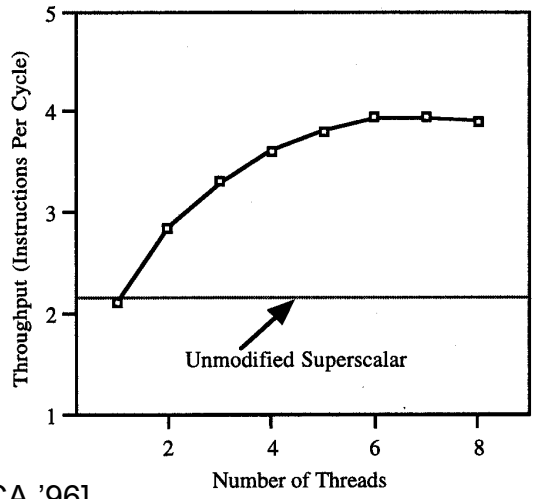


[Src: Tryggve Fossum, Compaq]

SMT uArch

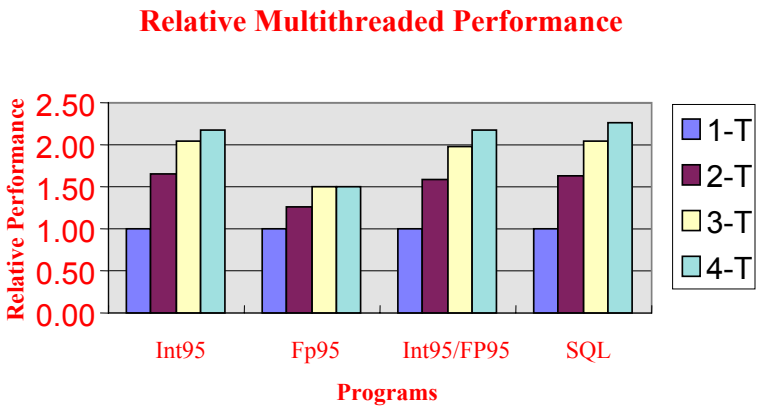
- Changes:
  - Multiple PCs
  - Control to decide how to fetch from
  - Separate return stacks per thread
  - Per-thread reorder/commit/flush/trap
  - Thread id w/ BTB
  - Larger register file
    - More things outstanding

Performance



[Tullsen et. al. ISCA '96]

Relative Performance (Alpha)



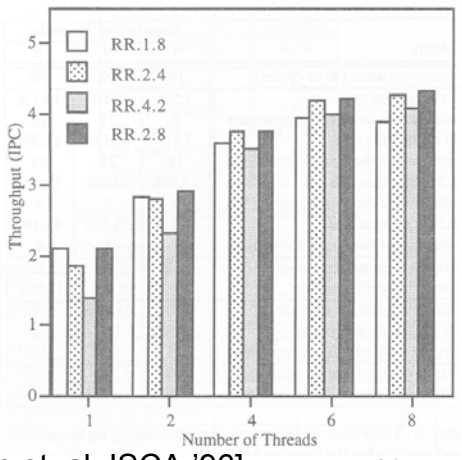
[Src: Tryggve Fossum, Compaq]

Alpha SMT

- Cost-effective Multiprocessing--increased throughput
- 4 X architectural registers
- 2 X performance gain with little additional cost and complexity

Optimizing: fetch freedom

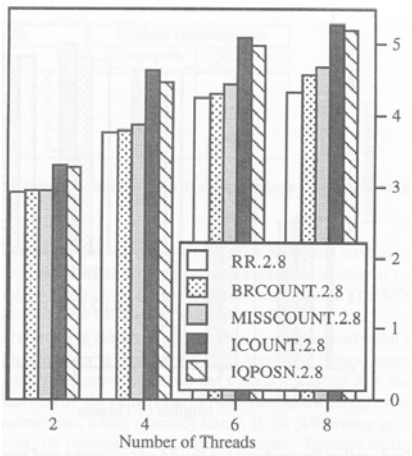
- RR=Round Robin
- RR.X.Y
  - X – threads do fetch in cycle
  - Y – instructions fetched/thread



[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

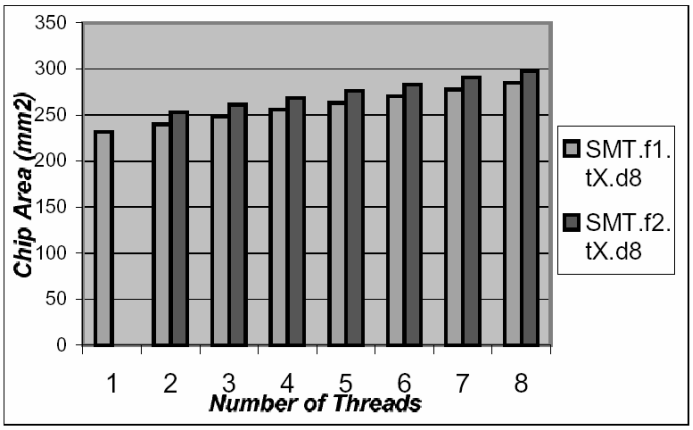
- 8-issue superscalar
  - Achieves little over 2 instructions per cycle
- Optimized SMT
  - Achieves 5.4 instructions per cycle on 8 threads
- 2.5x throughput increase

Costs

Function	Chip Block	MIPS R10K-2x in 0.18µ (mm^2)	MIPS R10K-2x (0.18µ) with SMT (mm^2)	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	68	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
	RPMUL	4.0	4.0				
	FPALU	8.3	8.3				
Miscellaneous	ExtInt	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			37.1	
	Chip w/ L2 cache	231.7	297.7				28.3

[Burns+Gaudiot HPCA'2000]

Costs



Single and double cache line fetches

[Burns+Gaudiot HPCA'2000]

Check on B5

Machines

- Pull machines off grid?
  - Matrix1?
  - Matrix37—39
    - Off for someone else, leave off and use if they finish in timely fashion?
  - Not necessary for single processor machine
    - Department grid cluster

Big Ideas

- 0, 1, Infinity → virtualize resources
  - Processes virtualize CPU
- Latency Hiding
  - Processes, Threads
  - Find something else useful to do while wait…