Multithreaded Architectures

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-739: Processor Design

Lecture 12 (19 Feb 2015)

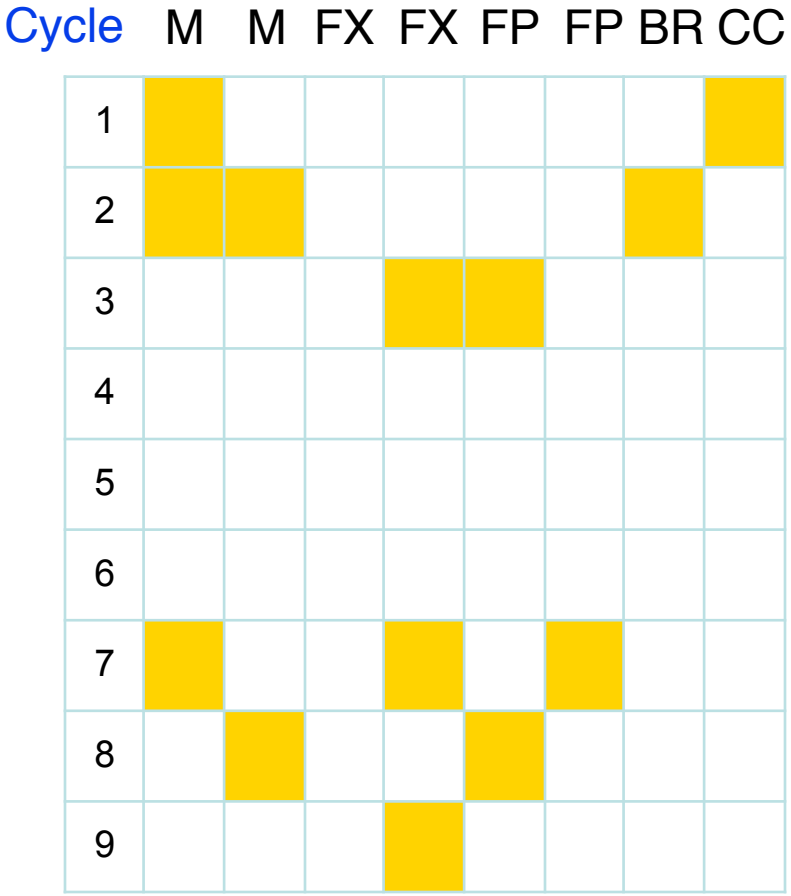
CADSL

Recap: Do both ILP and TLP?

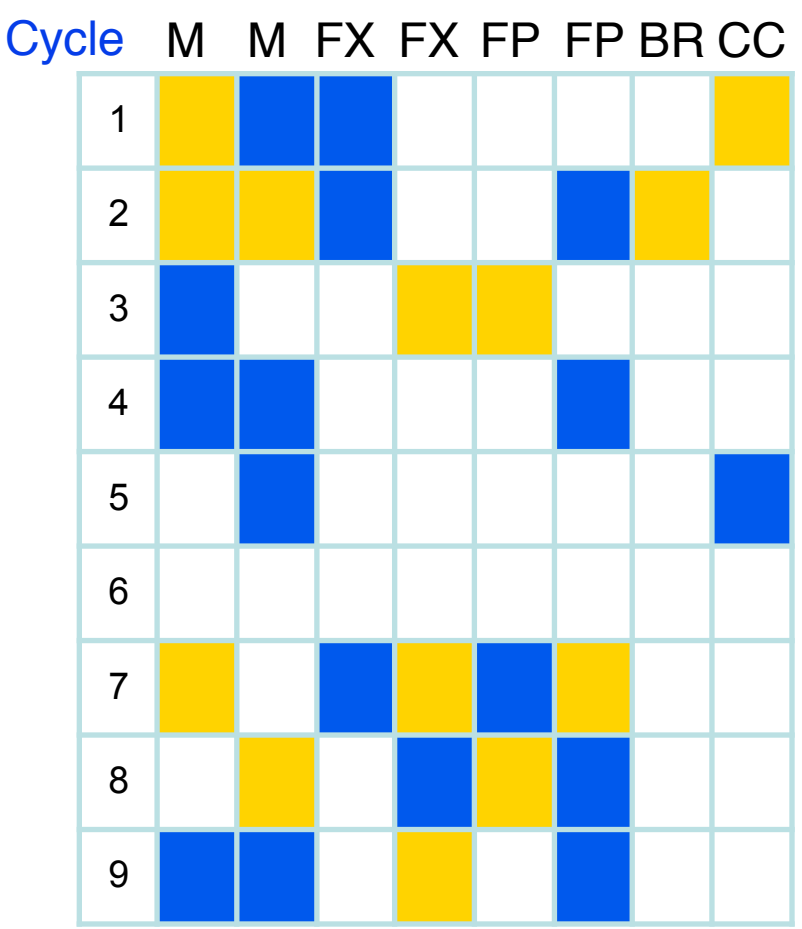
- TLP and ILP exploit two different kinds of parallel structure in a program
- Could a processor oriented at ILP to exploit TLP?
  - functional units are often idle in data path designed for ILP because of either stalls or dependences in the code
- Could the TLP be used as a source of independent instructions that might keep the processor busy during stalls?
- Could TLP be used to employ the functional units that would otherwise lie idle when insufficient ILP exists?

Recap: Simultaneous Multi-threading ...

One thread, 8 units



Two threads, 8 units



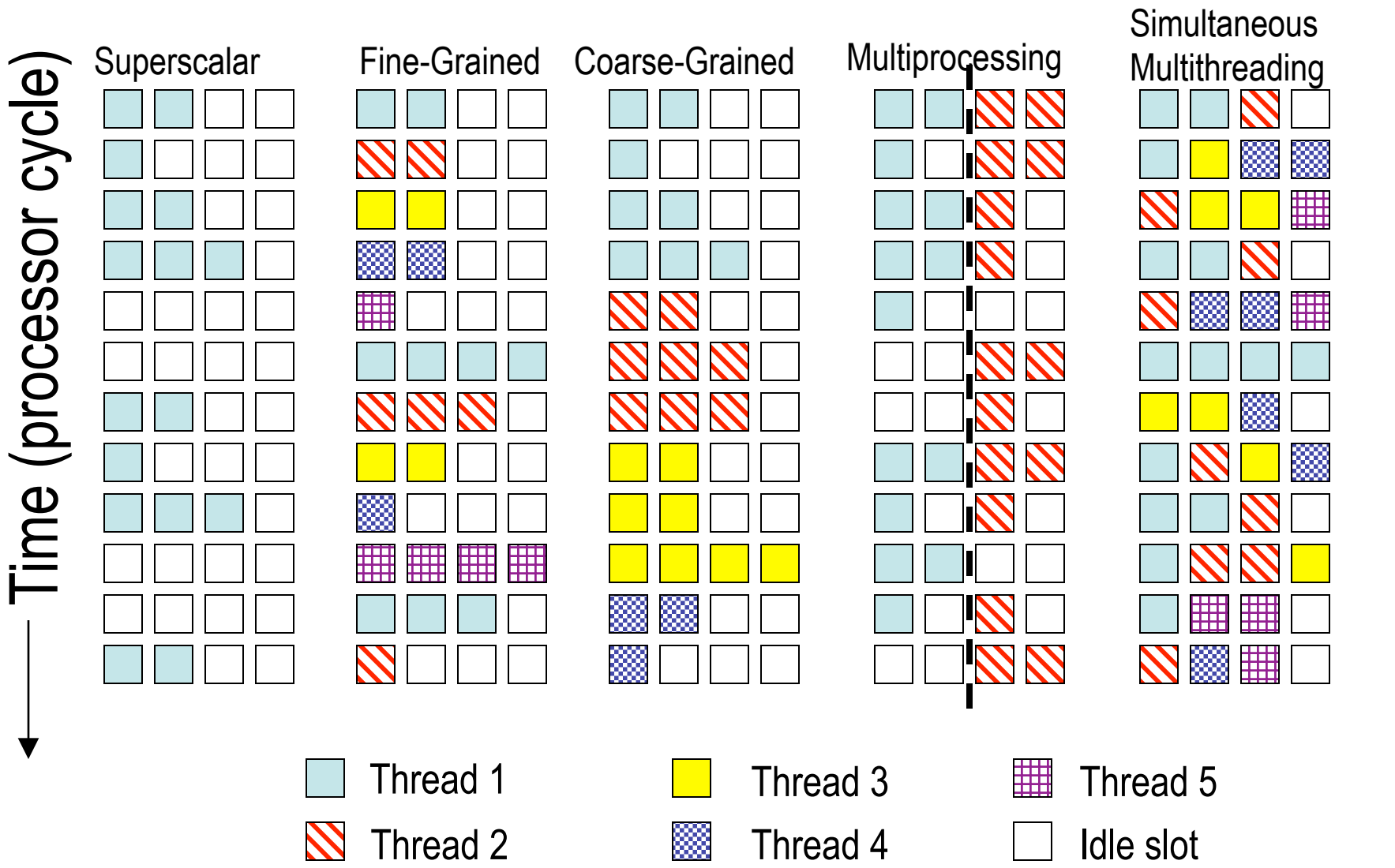
M = Load/Store, FX = Fixed Point, FP = Floating Point, BR = Branch, CC = Condition Codes

Simultaneous Multithreading (SMT)

- Simultaneous multithreading (SMT): insight that dynamically scheduled processor already has many HW mechanisms to support multithreading
  - Large set of virtual registers that can be used to hold the register sets of independent threads
  - Register renaming provides unique register identifiers, so instructions from multiple threads can be mixed in datapath without confusing sources and destinations across threads
  - Out-of-order completion allows the threads to execute out of order, and get better utilization of the HW
- Just adding a per thread renaming table and keeping separate PCs
  - Independent commitment can be supported by logically keeping a separate reorder buffer for each thread

Source: Micrprocessor Report, December 6, 1999
"Compaq Chooses SMT for Alpha"

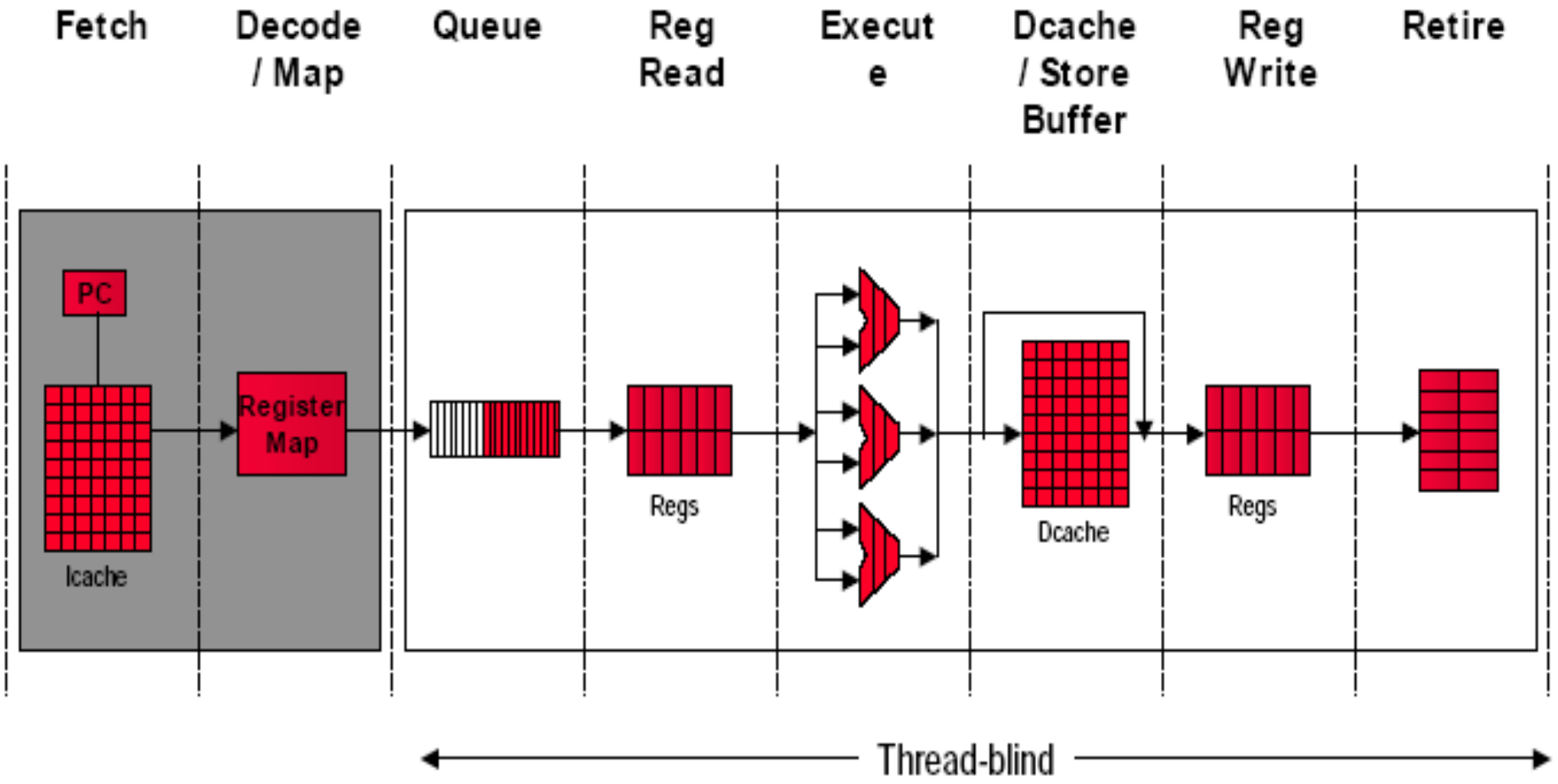
Recap: Multithreaded Categories



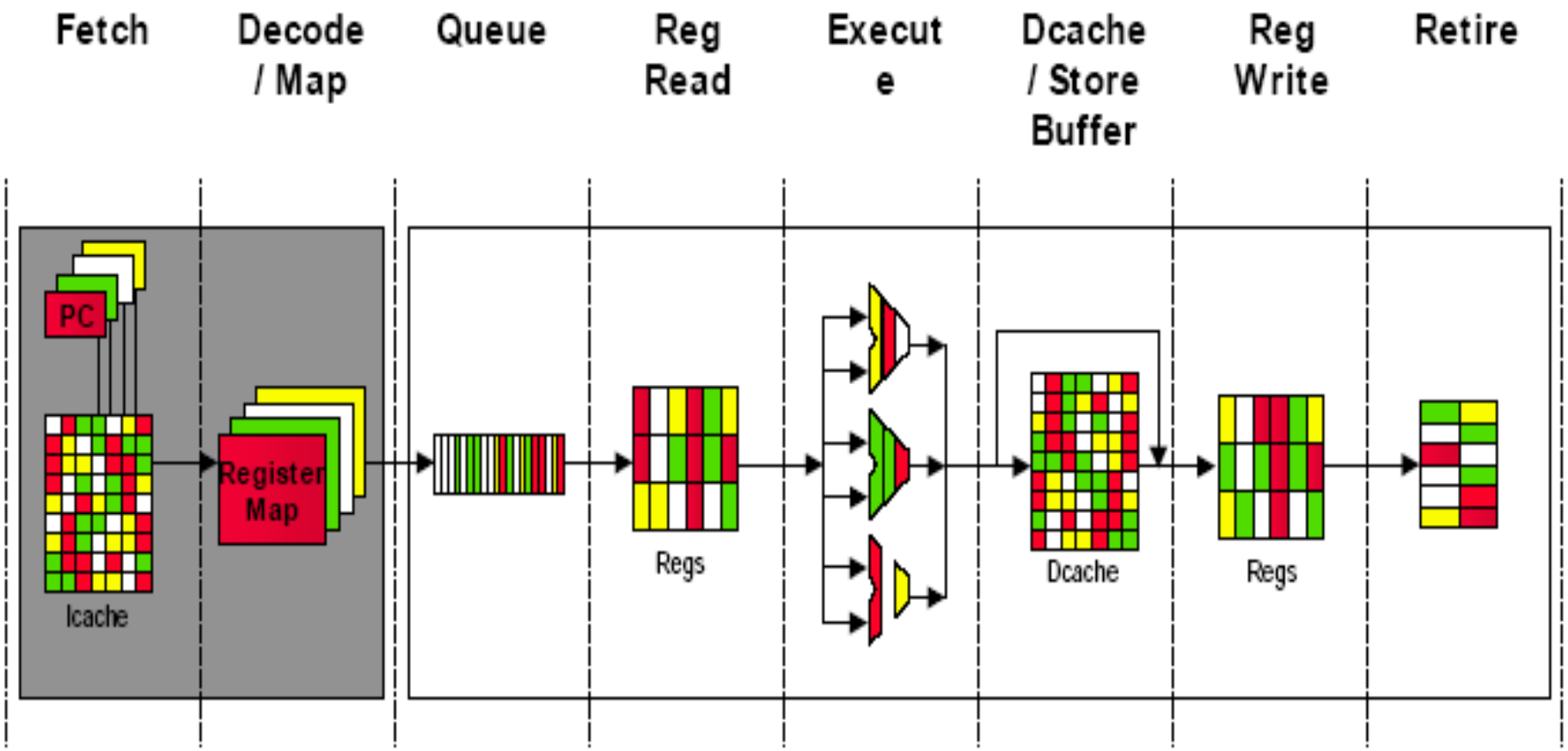
Design Challenges in SMT

- Since SMT makes sense only with fine-grained implementation, impact of fine-grained scheduling on single thread performance?
  - A preferred thread approach sacrifices neither throughput nor single-thread performance?
  - Unfortunately, with a preferred thread, the processor is likely to sacrifice some throughput, when preferred thread stalls
- Larger register file needed to hold multiple contexts
- Not affecting clock cycle time, especially in
  - Instruction issue - more candidate instructions need to be considered
  - Instruction completion - choosing which instructions to commit may be challenging
- Ensuring that cache and TLB conflicts generated by SMT do not degrade performance

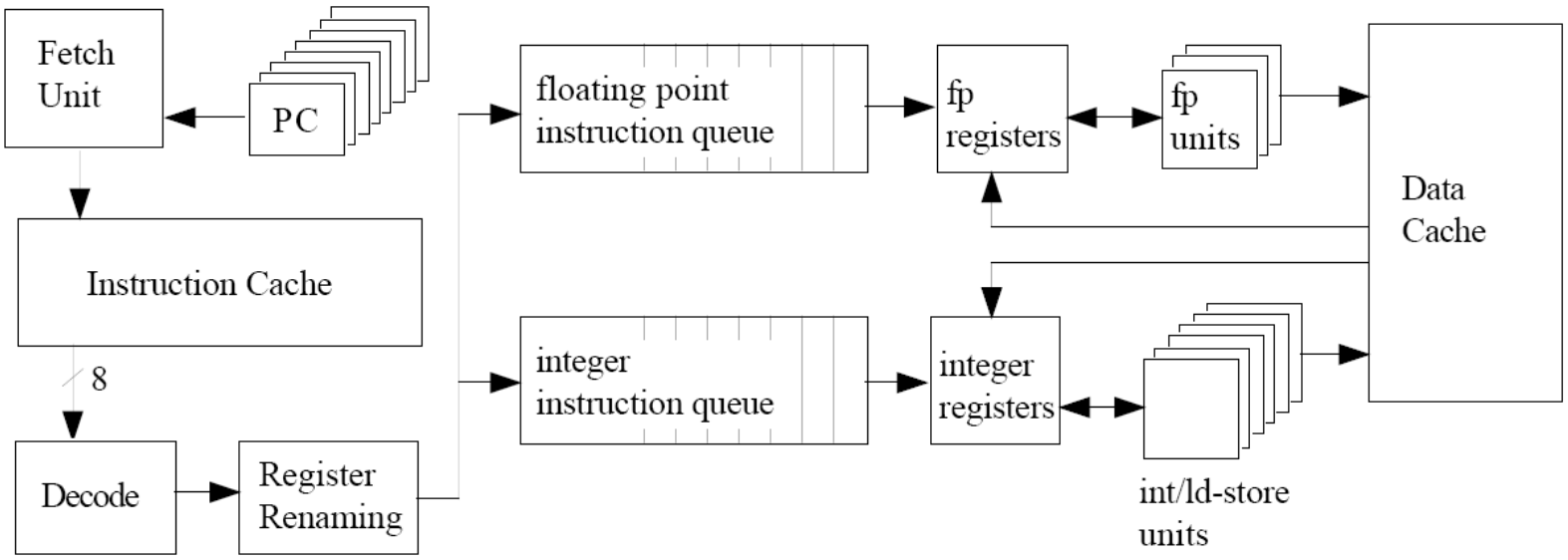
Basic Out-of-order Pipeline



SMT Pipeline



Simultaneous Multithreading



SMT Architecture

- Straightforward extension to conventional superscalar design.
  - multiple program counters and some mechanism by which the fetch unit selects one each cycle,
  - a separate return stack for each thread for predicting subroutine return destinations,
  - per-thread instruction retirement, instruction queue flush, and trap mechanisms,
  - a thread id with each branch target buffer entry to avoid predicting phantom branches, and
  - a larger register file, to support logical registers for all threads plus additional registers for register renaming.
    - The size of the register file affects the pipeline and the scheduling of load-dependent instructions.

Implementing SMT

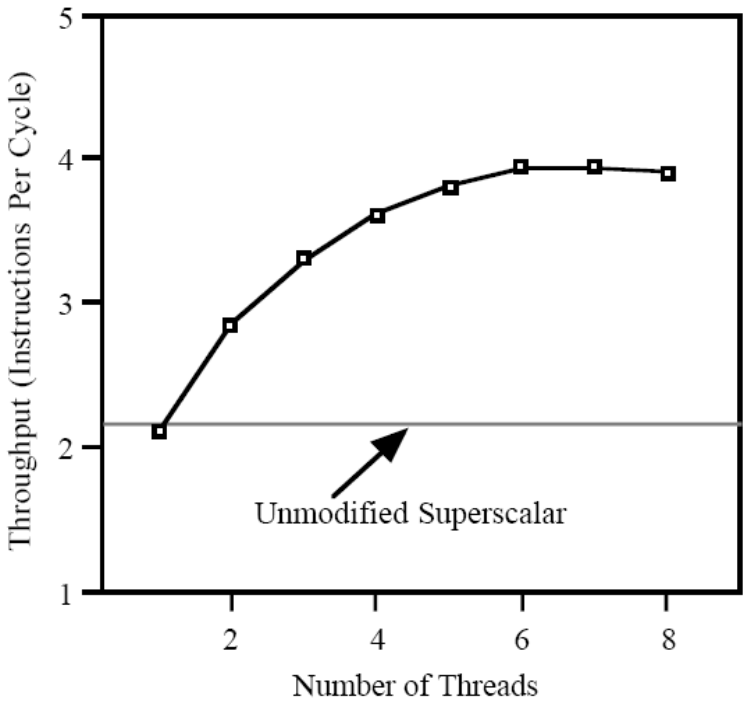
Can **use as is** most hardware on current out-or-order processors

Out-of-order renaming & instruction scheduling mechanisms

- physical register pool model
- renaming hardware eliminates false dependences both within a thread (just like a superscalar) & between threads
- map thread-specific architectural registers onto a pool of thread-independent physical registers
- operands are thereafter called by their physical names
- an instruction is issued when its operands become available & a functional unit is free
- instruction scheduler not consider thread IDs when dispatching instructions to functional units
  (unless threads have different priorities)

SMT Performance

Tullsen '96



Metric	Number of Threads		
	1	4	8
out-of-registers (% of cycles)	3%	7%	3%
I cache miss rate	2.5%	7.8%	14.1%
-misses per thousand instructions	6	17	29
D cache miss rate	3.1%	6.5%	11.3%
-misses per thousand instructions	12	25	43
L2 cache miss rate	17.6%	15.0%	12.5%
-misses per thousand instructions	3	5	9
L3 cache miss rate	55.1%	33.6%	45.4%
-misses per thousand instructions	1	3	4
branch misprediction rate	5.0%	7.4%	9.1%
jump misprediction rate	2.2%	6.4%	12.9%
integer IQ-full (% of cycles)	7%	10%	9%
fp IQ-full (% of cycles)	14%	9%	3%
avg (combined) queue population	25	25	27
wrong-path instructions fetched	24%	7%	7%
wrong-path instructions issued	9%	4%	3%

From Superscalar to SMT

Extra pipeline stages for accessing thread-shared register files

- 8 threads * 32 registers + renaming registers

SMT instruction fetcher (ICOUNT)

- fetch from 2 threads each cycle
  - count the number of instructions for each thread in the pre-execution stages
  - pick the 2 threads with the lowest number
- in essence fetching from the two highest throughput threads

From Superscalar to SMT

Per-thread hardware

- small stuff
- all part of current out-of-order processors
- none endangers the cycle time
- other per-thread processor state, e.g.,
  - program counters
  - return stacks
  - thread identifiers, e.g., with BTB entries, TLB entries
- per-thread bookkeeping for
  - instruction retirement
  - instruction queue flush

This is why there is only a 10% increase to Alpha 21464 chip area.

Implementing SMT

Thread-shared hardware:

- fetch buffers
- branch prediction structures
- instruction queues
- functional units
- active list
- all caches & TLBs
- MSHRs
- store buffers

This is why there is little single-thread performance degradation (~1.5%).

Design Challenges in SMT- Fetch

- Most expensive resources
  - Cache port
  - Limited to accessing the contiguous memory locations
  - Less likely that multiple thread from contiguous or even spatially local addresses
- Either provide dedicated fetch stage per thread
- Or time share a single port in fine grain or coarse grain manner
- Cost of dual porting cache is quite high
  - Time sharing is feasible solution

Design Challenges in SMT- Fetch

- Other expensive resource is → Branch Predictor
  - Multi-porting branch predictor is equivalent to halving its effective size
  - Time sharing makes more sense

- Certain element of BP rely on serial semantics and may not perform well for multi-thread
  - Return address stack rely on FIFO behaviour
  - Global BHR may not perform well
  - BHR needs to be replicated

Inter-thread Cache Interference

- Because the share the cache, so more threads, lower hit-rate.
- Two reasons why this is not a significant problem:
  1. The L1 Cache miss can almost be entirely covered by the 4-way set associative L2 cache.
  2. Out-of-order execution, write buffering and the use of multiple threads allow SMT to hide the small increases of additional memory latency.

0.1% speed up without interthread cache miss.

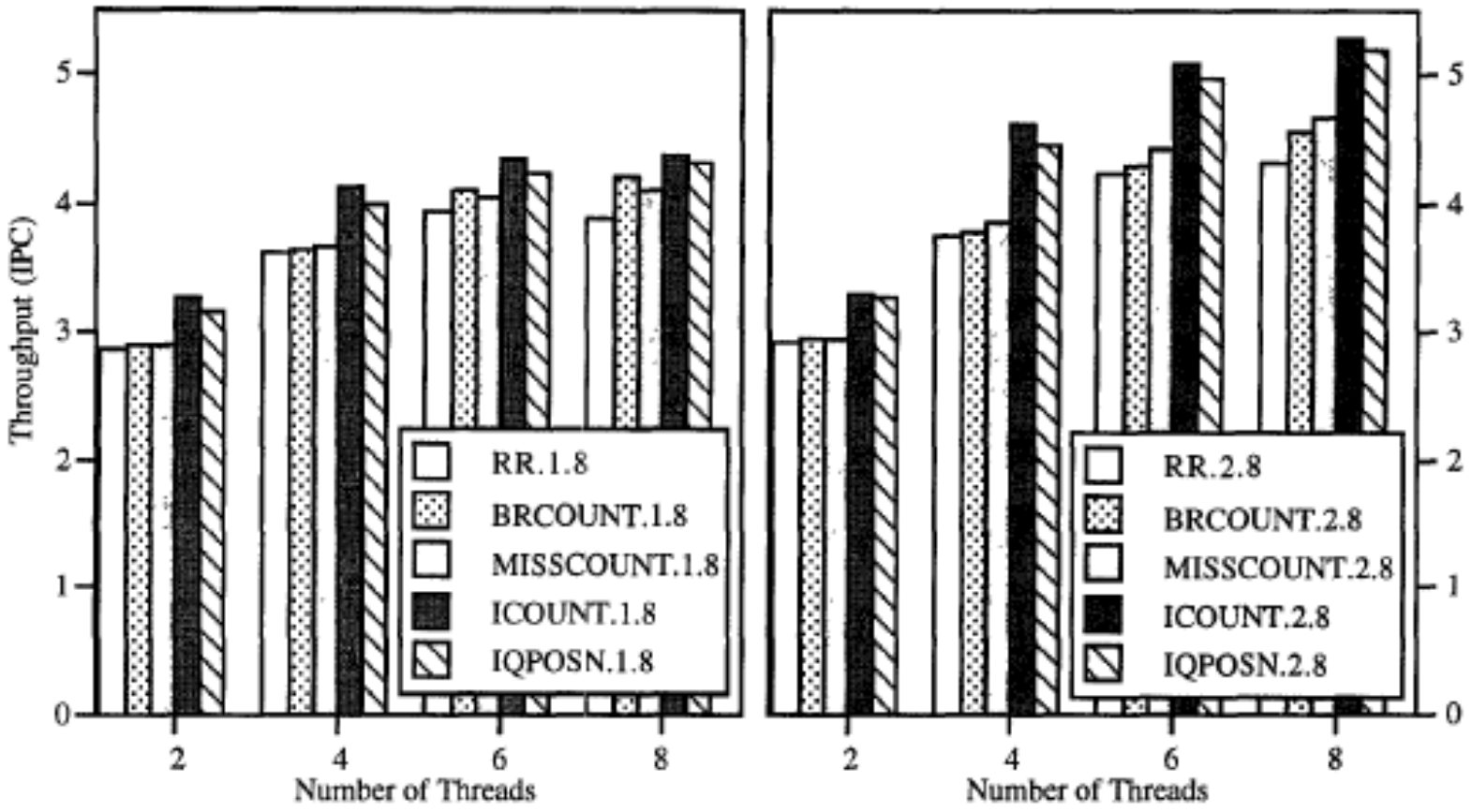
Increase in Memory Requirement

- More threads are used, more memory references per cycle.
- Bank conflicts in L1 cache account for the most part of the memory accesses.
- It is ignorable:
  1. For longer cache line: gains due to better spatial locality outweighted the costs of L1 bank contention
  2. 3.4% speedup if no interthread contentions.

Fetch Policies

- Basic: Round-robin: RR.2.8 fetching scheme, i.e., in each cycle, two times 8 instructions are fetched in round-robin policy from two different 2 threads,
  - superior to different other schemes like RR.1.8, RR.4.2, and RR.2.4
- Other fetch policies:
  - BRCOUNT scheme gives highest priority to those threads that are least likely to be on a wrong path,
  - MISSCOUNT scheme gives priority to the threads that have the fewest outstanding D-cache misses
  - IQPOSN policy gives lowest priority to the oldest instructions by penalizing those threads with instructions closest to the head of either the integer or the floating-point queue
  - ICOUNT feedback technique gives highest fetch priority to the threads with the fewest instructions in the decode, renaming, and queue pipeline stages

Fetch Policy



Dean Tullsen, 1996

Fetch Policies

- The ICOUNT policy proved as superior!
- The ICOUNT.2.8 fetching strategy reached a IPC of about 5.4 (the RR.2.8 reached about 4.2 only).
- Most interesting is that neither mispredicted branches nor blocking due to cache misses, but a mix of both and perhaps some other effects showed as the best fetching strategy.

- Simultaneous multithreading has been evaluated with
  - SPEC95,
  - database workloads,
  - and multimedia workloads.
- Both achieving roughly a 3-fold IPC increase with an eight-threaded SMT over a single-threaded superscalar with similar resources.

Design Challenges in SMT- Decode

- Primary tasks
  - Identify source operands and destination
  - Resolve dependency
- Instructions from different threads are not dependent
- Tradeoff → Single thread performance

Design Challenges in SMT- Rename

- Allocate physical register
- Map AR to PR
- Makes sense to share logic which maintain the free list of registers
- AR numbers are disjoint across the threads, hence can be partitioned
  - High bandwidth al low cost than multi-porting
- Limits the single thread performance

Design Challenges in SMT- Issue

- Tomasulo's algorithm
- Wakeup and select
- Clearly improve the performance
- Selection
  - Dependent on the instruction from multiple threads
- Wakeup
  - Limited to intrathread interaction
  - Make sense to partition the issue window
- Limit the performance of single thread

Design Challenges in SMT- Execute

- Clearly improve the performance
- Bypass network
- Memory
  - Separate LS queue

Commercial Machines w/ MT Support

- Intel Hyperthreding (HT)
  - Dual threads
  - Pentium 4, XEON
- Sun CoolThreads
  - UltraSPARC T1
  - 4-threads per core
- IBM
  - POWER5

Thank You