

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

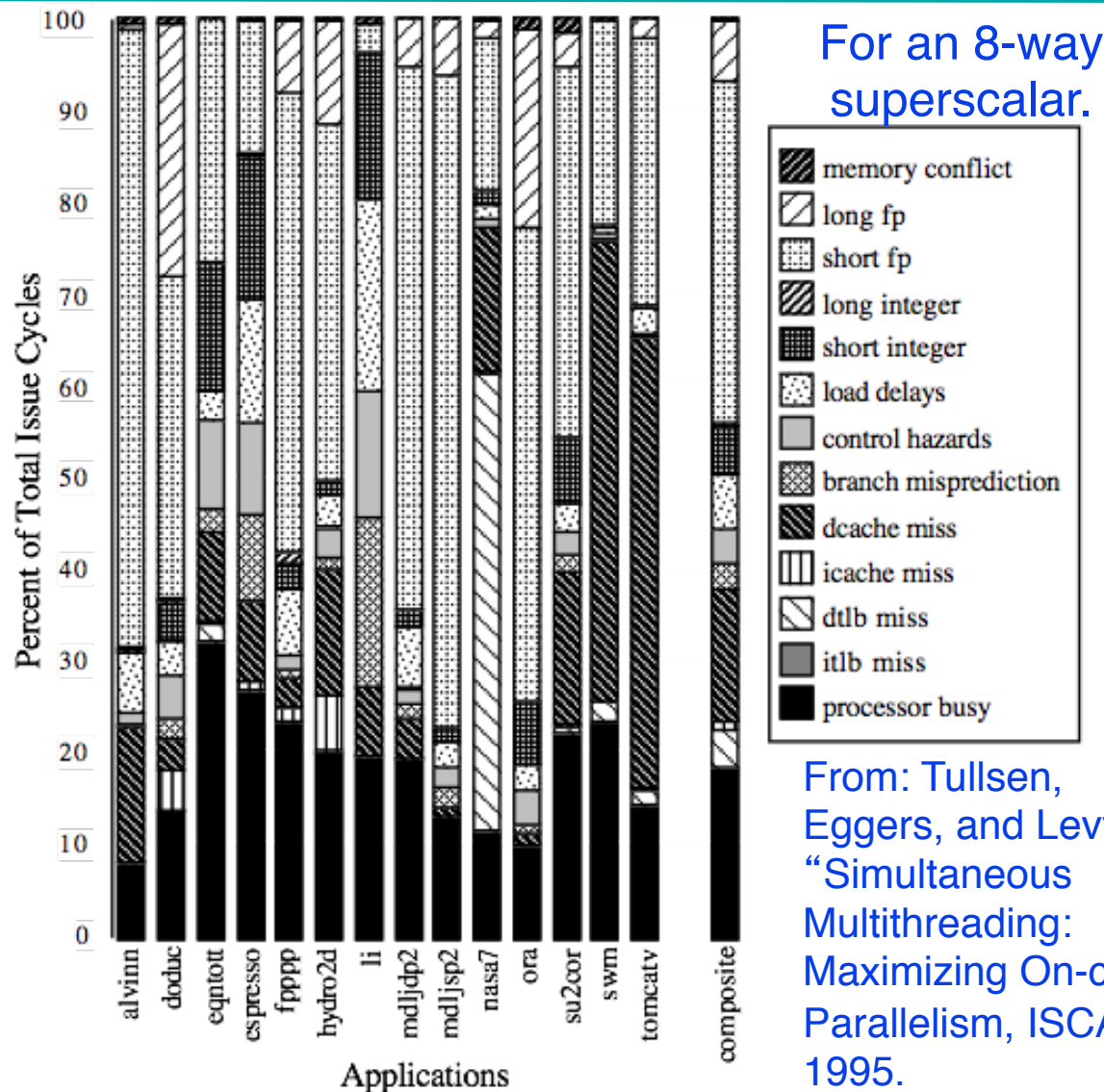
EE-739: Processor Design



Lecture 11 (18 Feb 2015)

CADSL

For most apps, most execution units lie idle



Superscalar Scenario

- Interest in **multiple-issue** because wanted to improve performance without affecting uniprocessor programming model
- Taking advantage of ILP is conceptually simple, but design problems are amazingly complex in practice
- Conservative in ideas, just faster clock and bigger
- Processors of last 15 years (Pentium 4, IBM Power 5, AMD Opteron) have the same basic structure and similar sustained issue rates (3 to 4 instructions per clock) as the 1st dynamically scheduled, multiple-issue processors announced in 1995
 - Clocks 10 to 20X faster, caches 4 to 8X bigger, 2 to 4X as many renaming registers, and 2X as many load-store units
→ performance 8 to 16X
- Peak v. delivered performance gap increasing



Performance Beyond Single Thread ILP

- There can be much higher natural parallelism in some applications (e.g., Database or Scientific codes)
- Explicit **Thread Level Parallelism** or **Data Level Parallelism**
- **Thread**: process with own instructions and data
 - thread may be a process part of a parallel program of multiple processes, or it may be an independent program
 - Each thread has all the state (instructions, data, PC, register state, and so on) necessary to allow it to execute
- **Data Level Parallelism**: Perform identical operations on data, and lots of data




Thread Level Parallelism (TLP)

- ILP exploits implicit parallel operations within a loop or straight-line code segment
- TLP explicitly represented by the use of multiple threads of execution that are inherently parallel
- **Goal:** Use multiple instruction streams to improve
 1. Throughput of computers that run many programs
 2. Execution time of multi-threaded programs
- TLP could be more cost-effective to exploit than ILP



New Approach: Multithreaded Execution

- **Multithreading:** multiple threads to share the functional units of one processor via overlapping
 - processor must duplicate independent state of each thread e.g., a separate copy of register file, a separate PC, and for running independent programs, a separate page table
 - memory shared through the virtual memory mechanisms, which already support multiple processes
 - HW for fast thread switch; much faster than full process switch 
100s to 1000s of clocks
- When switch?
 - Alternate instruction per thread (fine grain)
 - When a thread is stalled, perhaps for a cache miss, another thread can be executed (coarse grain)



Fine-Grained Multithreading

- Switches between threads on each instruction, causing the execution of multiples threads to be interleaved
- Usually done in a round-robin fashion, skipping any stalled threads
- CPU must be able to switch threads every clock
- **Advantage** is it can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- **Disadvantage** is it slows down execution of individual threads, since a thread ready to execute without stalls will be delayed by instructions from other threads
- Used on Sun' s Niagara



Course-Grained Multithreading

- Switches threads only on costly stalls, such as L2 cache misses
- Advantages
 - Relieves need to have very fast thread-switching
 - Doesn't slow down thread, since instructions from other threads issued only when the thread encounters a costly stall

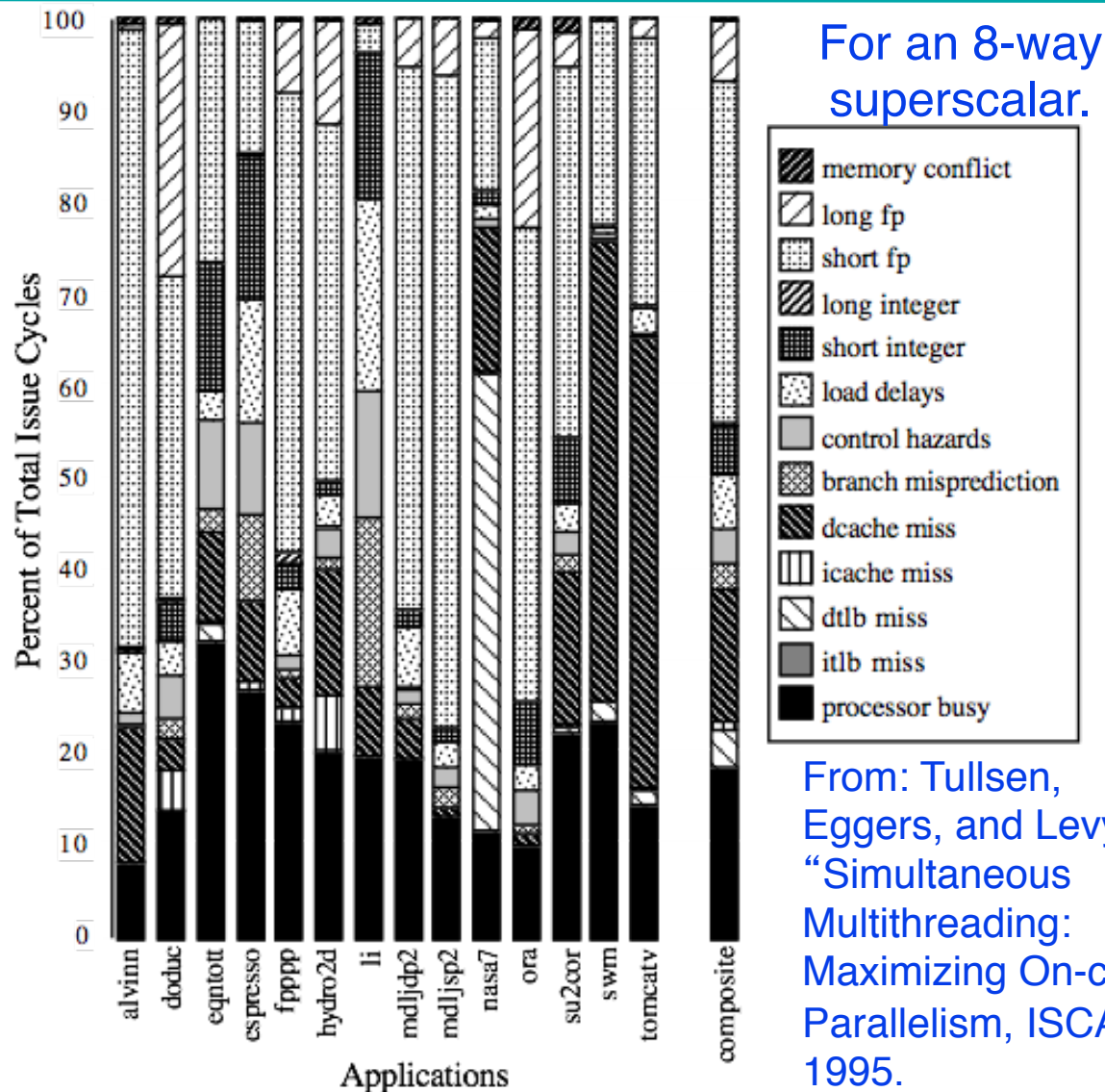


Course-Grained Multithreading

- Disadvantage is hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
 - Since CPU issues instructions from 1 thread, when a stall occurs, the pipeline must be emptied or frozen
 - New thread must fill pipeline before instructions can complete
- Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill \ll stall time
- Used in IBM AS/400



For most apps, most execution units lie idle



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?



Simultaneous Multi-threading ...

One thread, 8 units

Cycle M M FX FX FP FP BR CC

1	■							■
2	■	■					■	
3			■	■				
4								
5								
6								
7	■		■		■			
8		■		■				
9			■					

Two threads, 8 units

Cycle M M FX FX FP FP BR CC

1	■	■	■					■
2	■	■	■			■	■	
3	■			■	■			
4	■	■				■		
5		■						■
6								
7	■		■	■	■	■		
8		■		■	■	■		
9	■	■		■		■		

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



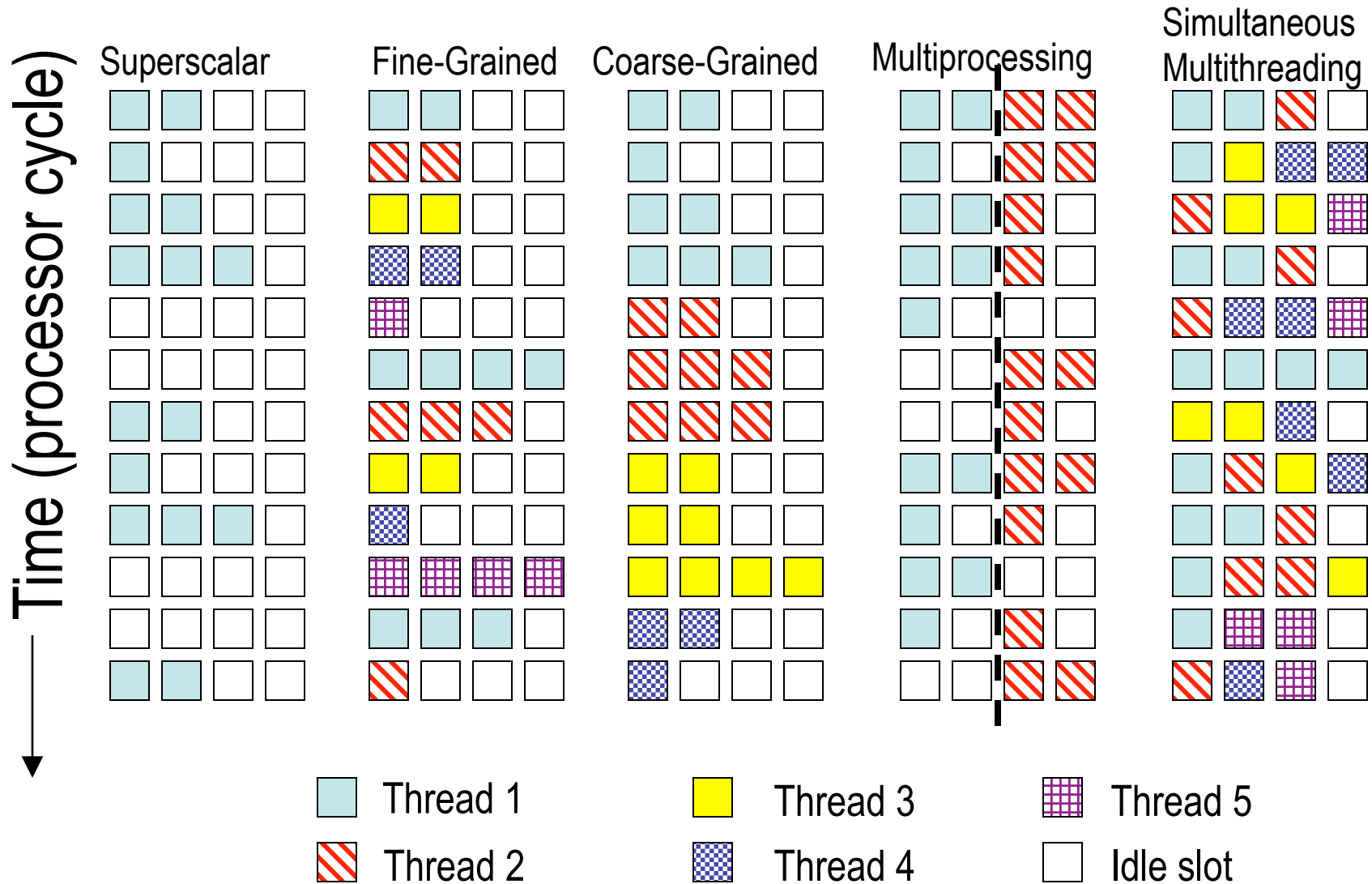
Simultaneous Multithreading (SMT)

- 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: Microprocessor Report, December 6, 1999
“Compaq Chooses SMT for Alpha”



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



Thank You

