EEL 5764 Graduate Computer Architecture

Chapter 3 – Limits to ILP and Simultaneous Multithreading

Ann Gordon-Ross
Electrical and Computer Engineering
University of Florida

http://www.ann.ece.ufl.edu/

These slides are provided by:
David Patterson
Electrical Engineering and Computer Sciences, University of California, Berkeley
Modifications/additions have been made from the originals

Outline

- **Limits to ILP**
- **Thread Level Parallelism**
- **Multithreading**
- **Simultaneous Multithreading**

Limits to ILP

- **Study conclusions conflict on amount available**
  - **Different benchmarks (vectorized Fortran FP vs. integer C programs)**
  - **Assumptions are made**
    - » **Hardware sophistication**
    - » **Compiler sophistication**
- **How much ILP can we expect using existing mechanisms with increasing HW budgets?**
- **Do we need to invent new HW/SW mechanisms to keep on the processor performance curve?**

Overcoming Limits - What do we need??

- **Advances in compiler technology + significantly new and different hardware techniques *may* be able to overcome limitations assumed in studies**
- **However, unlikely such advances when coupled *with realistic hardware* will overcome these limits in near future**

Limits to ILP

- Determine maximum limit on ILP given an ideal/perfect machine. Look at how each effect maximum ILP
  - Assumptions:

1. Register renaming
   - infinite virtual registers => all register WAW & WAR hazards are avoided
2. Branch prediction
   - perfect; no mispredictions
3. Jump prediction
   - all jumps perfectly predicted (returns, case statements)
4. Memory-address alias analysis
   - addresses known & a load can be moved before a store provided addresses not equal
5. Perfect caches
6. 1 cycle latency for all instructions (FP *,/)
7. unlimited instructions issued/clock cycle;

- 2 & 3 ⇒ no control dependencies; perfect speculation & an unbounded buffer of instructions available
- 1&4 eliminates all but RAW

Limits to ILP HW Model comparison

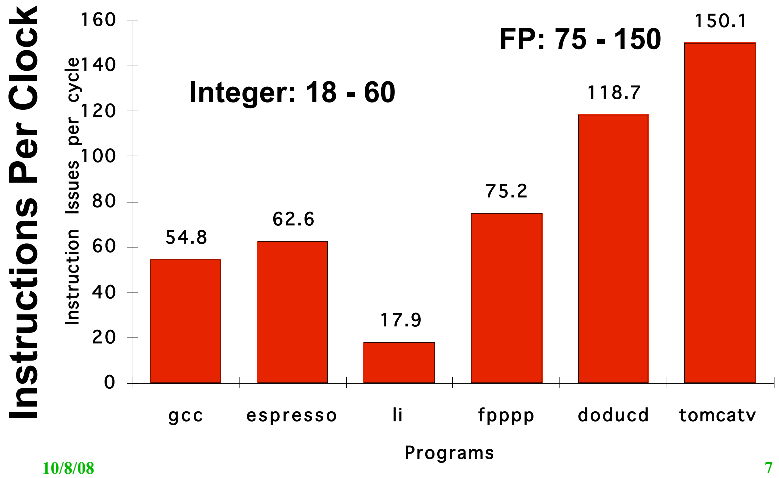
Comparison of two machines

	Model	Power 5
Instructions Issued per clock	Infinite	4
Instruction Window Size	Infinite	200
Renaming Registers	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias Analysis	Perfect	??

Upper Limit to ILP: Ideal Machine

(Figure 3.1)

Everything infinite or perfect



Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	Infinite	Infinite	4
(Vary) Instruction Window Size	Infinite, 2K, 512, 128, 32	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	Perfect	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect	Perfect	??

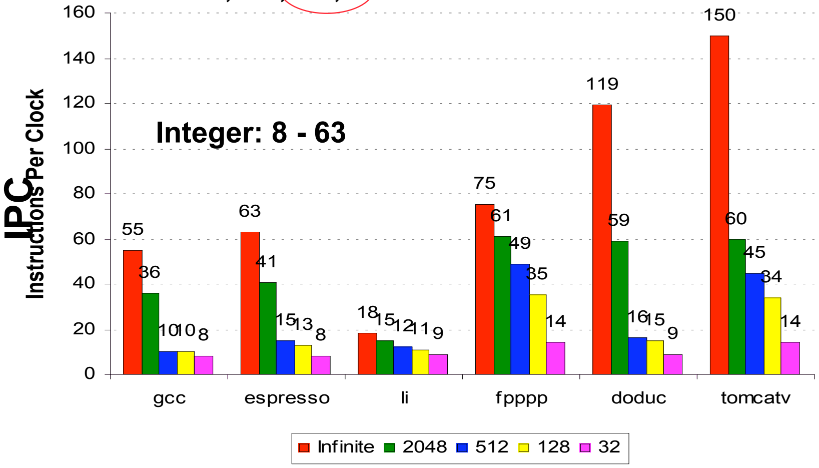
More Realistic HW: Window Impact

Figure 3.2

Change from Infinite window 2048, 512, 128, 32

Realistic

FP: 9 - 150



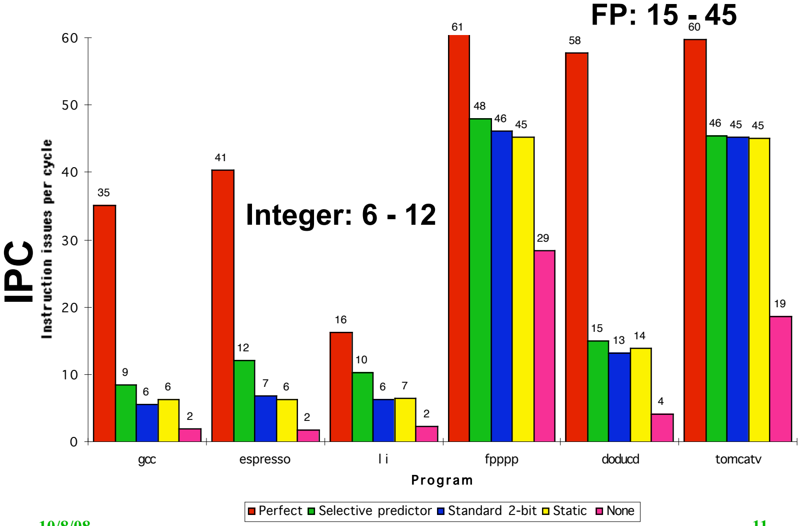
Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	Infinite	Infinite	48 integer + 40 Fl. Pt.
(Vary) Branch Prediction	Perfect vs. 8K Tournament vs. 512 2-bit vs. profile vs. none	Perfect	2% to 6% misprediction (Tournament Branch Predictor)
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect	Perfect	??

More Realistic HW: Branch Impact

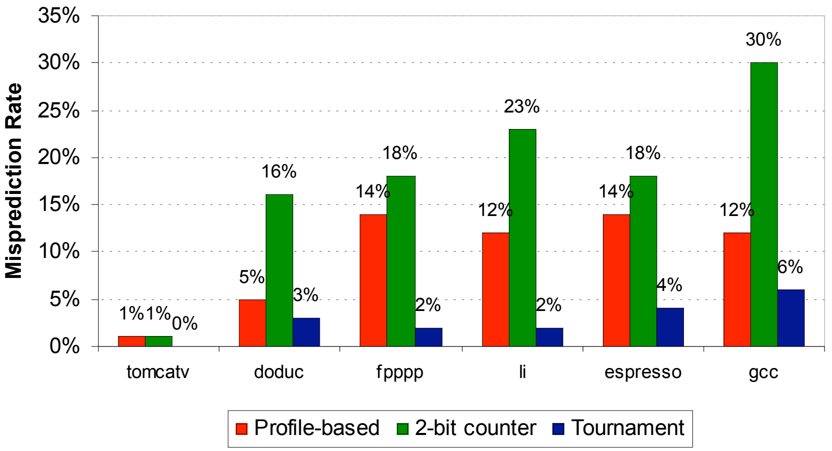
Figure 3.3

FP: 15 - 45



Perfect Tournament BHT (512) Profile No prediction

Branch Misprediction Rates

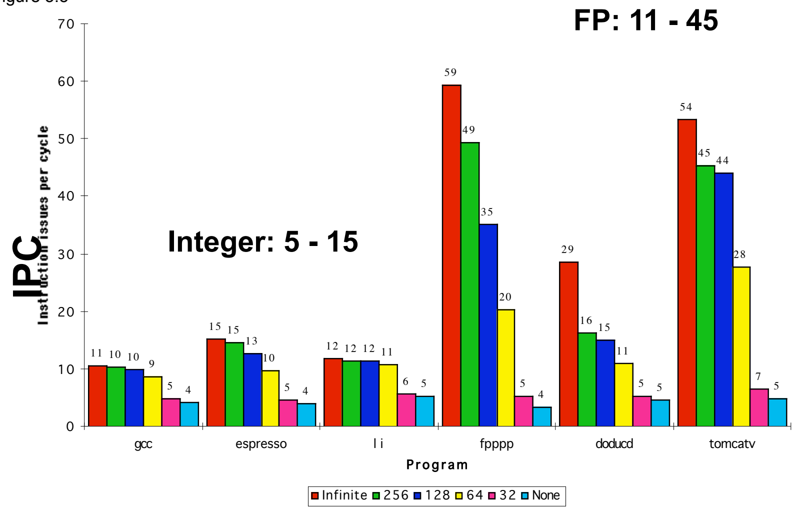


Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
(Vary) Renaming Registers	Infinite v. 256, 128, 64, 32, none	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	8K 2-bit	Perfect	Tournament Branch Predictor
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	Perfect	Perfect	Perfect

More Realistic HW: Renaming Register Impact (N int + N fp)

Figure 3.5



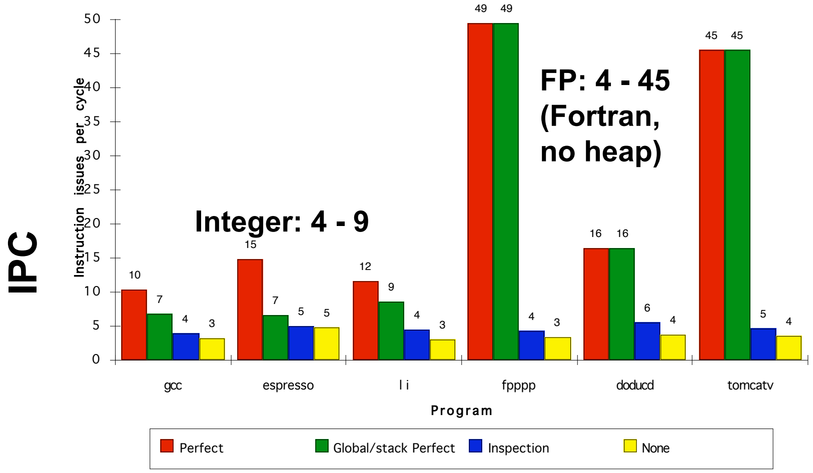
Infinite 256 128 64 32 None

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64	Infinite	4
Instruction Window Size	2048	Infinite	200
Renaming Registers	256 Int + 256 FP	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	8K 2-bit	Perfect	Tournament
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
(Vary) Memory Alias	Perfect v. Stack v. Inspect v. none	Perfect	Perfect

More Realistic HW: Memory Address Alias Impact

Figure 3.6



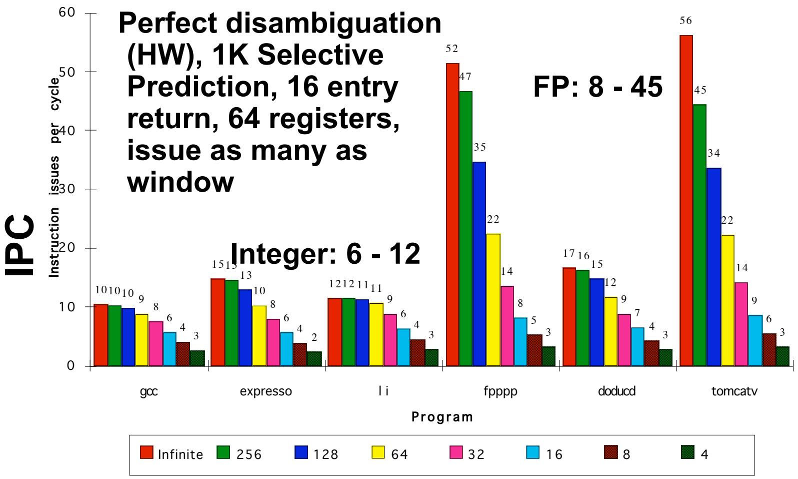
Perfect Global/Stack perf; (heap not considered) Inspec. Assem. None

Limits to ILP HW Model comparison

	New Model	Model	Power 5
Instructions Issued per clock	64 (no restrictions)	Infinite	4
Instruction Window Size	Infinite vs. 256, 128, 64, 32	Infinite	200
Renaming Registers	64 Int + 64 FP	Infinite	48 integer + 40 Fl. Pt.
Branch Prediction	1K 2-bit	Perfect	Tournament
Cache	Perfect	Perfect	64KI, 32KD, 1.92MB L2, 36 MB L3
Memory Alias	HW disambiguation	Perfect	Perfect

Realistic HW: Window Impact

(Figure 3.7)



Perfect disambiguation (HW), 1K Selective Prediction, 16 entry return, 64 registers, issue as many as window

Integer: 6 - 12

FP: 8 - 45

Infinite 256 128 64 32 16 8 4

Outline

- Limits to ILP
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading

How to Exceed ILP Limits of this study?

- These were practical limits for modern computers
  - These are not laws of physics
  - Perhaps overcome via research
- Compiler and ISA advances could change results
- Memory aliasing - WAR and WAW hazards through memory
  - eliminated WAW and WAR hazards through register renaming, but not in memory usage
    - » Research on predicting address conflicts should help
  - Can get conflicts via allocation of stack frames as a called procedure reuses the memory addresses of a previous frame on the stack

Which is better for increasing ILP: HW vs. SW

- Memory disambiguation:
  - HW best
  - Compile time pointer analysis is hard
- Speculation:
  - HW best when dynamic branch prediction better than compile time prediction
    - » Profiling is not good enough
  - Exceptions easier for HW
    - » HW doesn't need bookkeeping code or compensation code
  - Speculation is very complicated to get right
    - » Execution is hard enough to get right without speculation
    - » Speculation leads to many special cases
    - » Hard to get right
- Scheduling
  - SW can look ahead to schedule better, look beyond current PC
- Advantage for HW based:
  - Compiler independence: does not require new compiler, recompilation to run well

Performance beyond single thread ILP - How do we progress?

- Some applications have high natural parallelism
  - Database, searching
- Explicit Thread Level Parallelism or Data Level Parallelism
  - Thread: process with own instructions and data
    - » Part of parallel program (same address space) 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
    - » Graphics processing
    - » ATI - 130,000+ threads

Thread Level Parallelism (TLP)

- ILP vs. 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
- TLP Goal: Use multiple instruction streams to improve…
  - Throughput of computers that run many programs
  - Execution time of multi-threaded programs
- TLP could be more cost-effective to exploit than ILP

Outline

- Limits to ILP
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading

New Approach: Mulithreaded Execution

- Attempt better performance while reusing a lot of existing hardware
- Multithreading: multiple threads to share the functional units of 1 processor via overlapping
- What do we need to support it:
  - duplicate independent state of each thread
    - » a separate copy of register file
    - » a separate PC
    - » a separate page table (if separate programs)
  - Memory shared through the virtual memory mechanisms, which already support multiple processes
  - HW for fast thread switch
    - » Needs to be faster than full process switch ≈ 100s to 1000s of clocks

New Approach: Mulithreaded Execution

- When to switch?
  - Fine grained
    - » Alternate instruction per thread - switch on each clock cycle
  - Coarse grained
    - » When a thread is stalled, perhaps for a cache miss, another thread can be executed
- Both switching methods allow stalls to be hidden by doing work for another thread

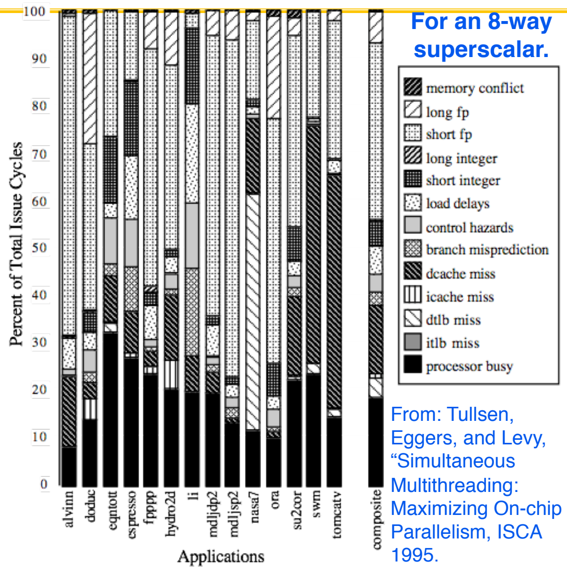
Fine-Grained Multithreading

- Switches between threads on each instruction, causing the execution of multiple threads to be interleaved
- CPU must be able to switch threads every clock
- Usually done in a round-robin fashion, skipping any stalled threads
- Advantage
  - can hide both short and long stalls, since instructions from other threads executed when one thread stalls
- Disadvantage
  - 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

- More conservative
  - Switches threads only on costly stalls, such as L2 cache misses
- Advantages
  - Relieves need to have very fast thread-switching
    - » Easier to build
  - Doesn't slow down thread, since instructions from other threads issued only when the thread encounters a costly stall
- Disadvantage
  - hard to overcome throughput losses from shorter stalls, due to pipeline start-up costs
    - » On switch, pipeline is emptied. Need to refill for new thread
  - Doesn't switch on short stalls, can't hide those
- Because of this start-up overhead, coarse-grained multithreading is better for reducing penalty of high cost stalls, where pipeline refill << stall time
- Used in IBM AS/400

For most apps, most execution units lie idle



For an 8-way superscalar.

From: Tullsen, Eggers, and Levy, “Simultaneous Multithreading: Maximizing On-chip Parallelism, ISCA 1995.

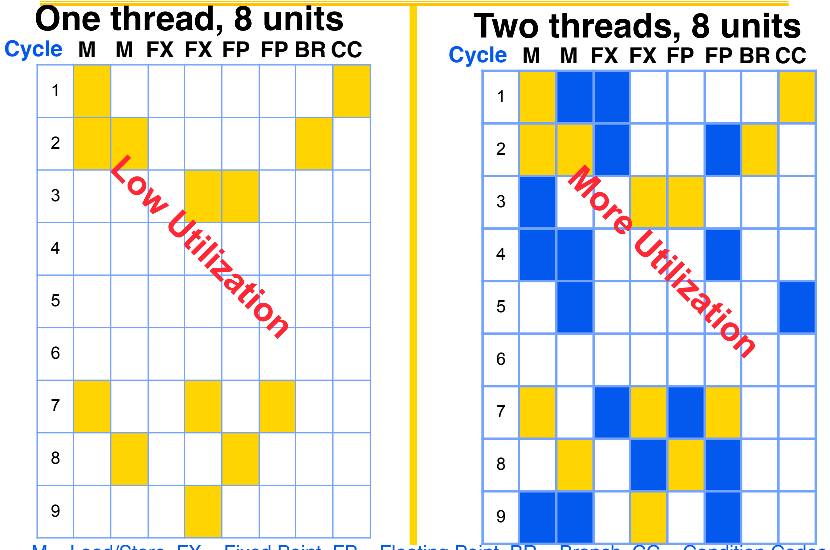
Do both ILP and TLP?

- TLP and ILP exploit two different kinds of parallel structure in a program
- Could a processor oriented for ILP be used 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?

Outline

- Limits to ILP
- Thread Level Parallelism
- Multithreading
- Simultaneous Multithreading

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

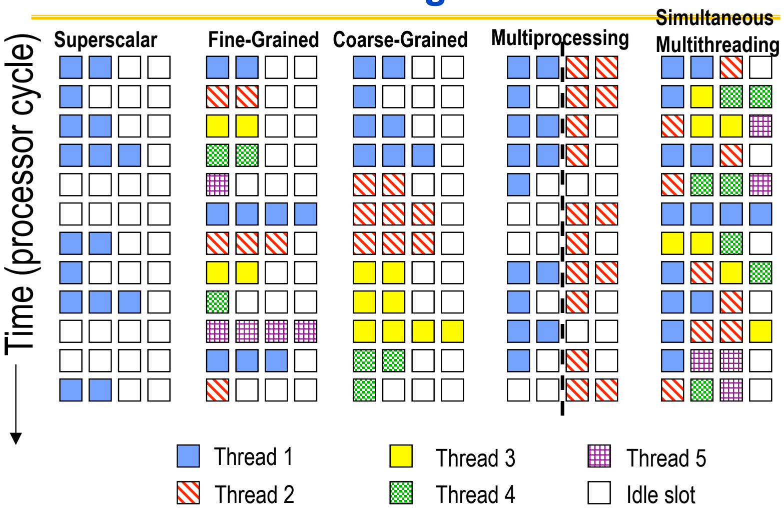
Cycle	M	M	FX	FX	FP	FP	BR	CC
1	🟨	🟦	🟦					🟨
2	🟨	🟨	🟦			🟦	🟨	
3	🟦			🟨	🟨			
4	🟦	🟦				🟦		
5		🟦						🟦
6								
7	🟨		🟦	🟨	🟦	🟨		
8		🟨		🟦	🟨	🟦		
9	🟦	🟦		🟨		🟦		

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

Simultaneous Multithreading (SMT)

- Simultaneous multithreading (SMT)
  - 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 execution allows the threads to execute out of order, and get better utilization of the HW
  - Added support
    - » Different threads can be scheduled together on same clock cycle
    - » Per thread renaming table
    - » Separate PCs
    - » Independent commitment can be supported by logically keeping a separate reorder buffer for each thread

Multithreaded Categories



Design Challenges in SMT

- Impact of fine-grained scheduling on single thread performance?
  - SMT makes sense only with fine-grained implementation
  - A preferred thread approach sacrifices neither throughput nor single-thread performance?
  - Unfortunately when a preferred thread stalls, the processor is likely to sacrifice some throughput,
- 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

And in conclusion …

- Limits to ILP (power efficiency, compilers, dependencies …) seem to limit to 3 to 6 issue for practical options
- Explicitly parallel (Data level parallelism or Thread level parallelism) is next step to performance
- Coarse grain vs. Fine grained multihreading
  - Only on big stall vs. every clock cycle
- Simultaneous Multithreading if fine grained multithreading based on superscalar microarchitecture
  - Instead of replicating registers, reuse rename registers
- Balance of ILP and TLP decided in marketplace