

MSc Informatics Eng.

2012/13

A.J.Proença

From ILP to Multithreading and Shared Cache (most slides are borrowed)

AJProença, Computer Systems & Performance, MEI, UMinho, 2012/13

1

Multiple Issue and Static Scheduling

- To achieve $CPI < 1$, need to complete multiple instructions per clock
- Solutions:
 - statically scheduled superscalar processors
 - VLIW (very long instruction word) processors
 - dynamically scheduled superscalar processors

Multiple Issue and Static Scheduling

- When exploiting ILP, goal is to minimize CPI
 - Pipeline CPI =>
 - Ideal pipeline CPI + ✓
 - Structural stalls + ✓
 - Data hazard stalls + ✓
 - Control stalls + ✓
 - Memory stalls ... *cache techniques ...*
 - Multiple issue =>
 - find enough parallelism to keep pipeline(s) occupied
 - Multithreading =>
 - find ways to keep pipeline(s) occupied
- Insert data parallelism features (*next set of slides*)

AJProença, Computer Systems & Performance, MEI, UMinho, 2012/13

2

Multiple Issue

Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Cortex A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Multiple Issue and Static Scheduling

Multithreading

- Performing multiple threads of execution in parallel
 - Replicate registers, PC, etc.
 - Fast switching between threads
- Fine-grain multithreading
 - Switch threads after each cycle
 - Interleave instruction execution
 - If one thread stalls, others are executed
- Coarse-grain multithreading
 - Only switch on long stall (e.g., L2-cache miss)
 - Simplifies hardware, but doesn't hide short stalls (eg, data hazards)

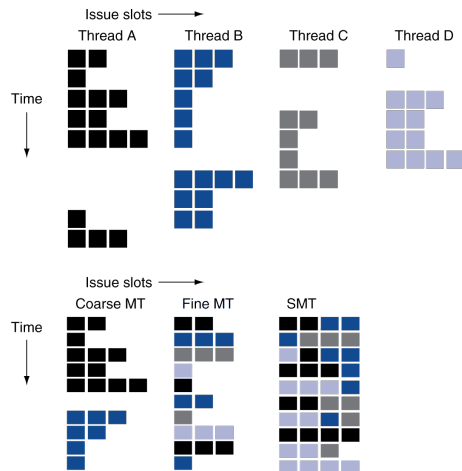


Simultaneous Multithreading

- In multiple-issue dynamically scheduled processor
 - Schedule instructions from multiple threads
 - Instructions from independent threads execute when function units are available
 - Within threads, dependencies handled by scheduling and register renaming
- Example: Intel Pentium-4 HT
 - Two threads: duplicated registers, shared function units and caches



Multithreading Example



Instruction and Data Streams

- An alternate classification

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345

- SPMD: Single Program Multiple Data
 - A parallel program on a MIMD computer
 - Conditional code for different processors

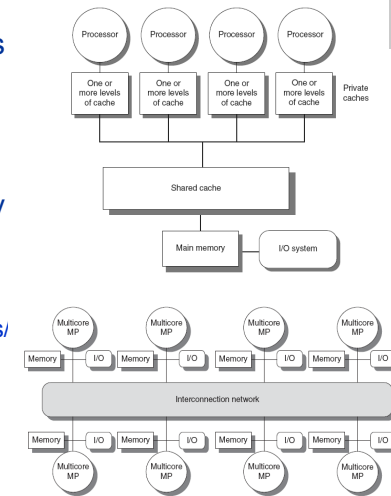


Introduction to multithreading

- Thread-Level parallelism
 - Have multiple program counters
 - Uses MIMD model
 - Targeted for tightly-coupled shared-memory multiprocessors
- For n processors, need n threads
- Amount of computation assigned to each thread = grain size
 - Threads can be used for data-level parallelism, but the overheads may outweigh the benefit

Types

- Symmetric multiprocessors (SMP)
 - Small number of cores
 - Share single memory with uniform memory latency
- Distributed shared memory (DSM)
 - Memory distributed among processors
 - Non-uniform memory access/latency (NUMA)
 - Processors connected via direct (switched) and non-direct (multi-hop) interconnection networks



Cache Coherence Problem

- Suppose two CPU cores share a physical address space
 - Write-through caches

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

Cache Coherence

- Coherence
 - All reads by any processor must return the most recently written value
 - Writes to the same location by any two processors are seen in the same order by all processors
- Consistency
 - When a written value will be returned by a read
 - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A

Cache Coherence Protocols

- Operations performed by caches in multiprocessors to ensure coherence
 - Migration of data to local caches
 - Reduces bandwidth for shared memory
 - Replication of read-shared data
 - Reduces contention for access
- Snooping protocols
 - Each cache monitors bus reads/writes
- Directory-based protocols
 - Caches and memory record sharing status of blocks in a directory



Snoopy Coherence Protocols

- Write invalidate
 - On write, invalidate all other copies
 - Use bus itself to serialize
 - Write cannot complete until bus access is obtained

Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

- Write update
 - On write, update all copies



Snoopy Coherence Protocols

- Locating an item when a read miss occurs
 - In write-back cache, the updated value must be sent to the requesting processor
- Cache lines marked as shared or exclusive/modified
 - Only writes to shared lines need an invalidate broadcast
 - After this, the line is marked as exclusive

Invalidating Snooping Protocols

- Cache gets exclusive access to a block when it is to be written
 - Broadcasts an invalidate message on the bus
 - Subsequent read in another cache misses
 - Owning cache supplies updated value

CPU activity	Bus activity	CPU A's cache	CPU B's cache	Memory
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidate for X	1		0
CPU B read X	Cache miss for X	1	1	1



