Chip-level Multithreading and Multiprocessing

Introduction

- Performance of a single serial program is limited by its available ILP and long-latency operations
- Time-sharing
  - Multiprogramming workloads
  - Parallel applications
    - Synchronization
- Thread-level Parallelism
  - Increase overall instruction throughput of the processor

Thread

- A full program (single-threaded UNIX process)
- An operating system thread, e.g., a POSIX thread
- A compiler-generated thread, e.g. microthread
- A hardware-generated thread

Exploit Thread-level Parallelism

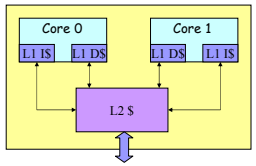
- Multiprocessor system
  - Shared memory
    - Cache coherence, memory consistency
  - Message passing
- Multithreaded processor
  - Explicitly
    - Interleave the execution of instructions of different user-defined threads (OS threads or processes)
    - Chip multiprocessors (CMP), fine-grained, coarse-grained, and simultaneous multithreading (SMT)
  - Implicitly
    - Dynamically generate threads from single-threaded programs and execute such speculative threads concurrent with the lead thread.
    - Multiscalar, dynamic multithreading, speculative multithreaded, ……

Explicitly Multithreading Processors

- Issuing instructions from multiple threads in a cycle
  - CMP
  - SMT
- Issuing instructions from a single thread in a cycle
  - Fine-grained (FGMT)
  - Coarse-grained (CGMT)

Chip Multiprocessing

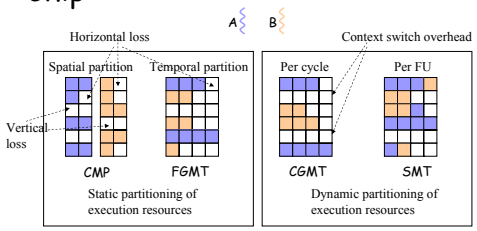
- Replicate an entire processor core for each thread to support multiple threads within a single processor chip



CMP

- Advantage
  - Reduce latencies for processor-to-processor communication and synchronization
- Drawback
  - More complicated uniprocessor vs. simple CMPs
  - Lower frequency

Running Multiple Threads on One Chip



Fine-grained Multithreading

- Provide two or more thread contexts on chip
- Switch from one thread to the next on a fixed, fine-grained schedule (e.g. every cycle)
- Example: Tera MTA machine
  - 128 threads (128 register contexts)
  - Switch threads on every clock cycle
  - Fully mask the 128-cycle memory access latency (no cache)
  - Drawback: sacrifice single-thread performance for overall throughput

Coarse-grained Multithreading

- Provide multiple thread contexts within the processor core
- The currently active thread is executed until it reaches a situation that triggers a context switch (e.g. stalls on a long-latency event, such as a cache miss)

Models of CGMT

- Static: context switch occurs each time the same instruction is executed
  - Explicit context switch instructions
  - Implicit-switch: switch-on-load, switch-on-store, switch-on-branch
  - Advantage: low context switching overhead (0 or 1 cycle)
  - Disadvantage: switching contexts more often than necessary

Models of CGMT

- Dynamic: context switch is triggered by a dynamic event
  - Switch-on-cache-miss, switch-on-signal, switch-on-use
  - Advantage: reduce unnecessary context switches
  - Disadvantage: higher context switching overhead

Cost of Thread Switches

- Dynamic events that trigger context switches may only be detected late in the pipeline
- Naïve implementation → several pipeline bubbles
- Replicate registers for each thread and save current state of pipeline at context switch → avoid switch penalty but increase complexity
- Which approach should be used?

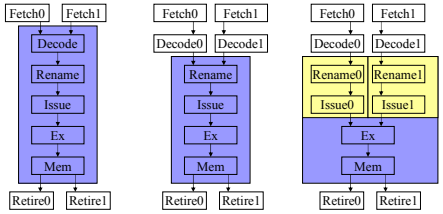
Fairness and Priority

- Fairness
  - Cache miss rate + OS-controlled context switch
  - Threads with low miss rates are preempted after a time slice expires
  - Threads are prevented from preemption for a minimum quantum
- Priority
  - Thread enters a critical section → increase priority
  - Thread leaves a critical section → reduce priority
  - Thread spins on a lock or enters an idle loop → reduce priority

Simultaneous Multithreading (SMT)

- Allow instructions from multiple active threads to be interleaved within and across pipeline stages
- Reduce both horizontal and vertical losses
- Maximize processor resource utilization

SMT Resource Sharing



SMT Sharing of Pipeline Stages

- Dedicated → low utilization
- Shared → complicated design, sometimes poor performance
- Fetch
  - Time-share an instruction cache port among multiple threads
  - Branch predictor
    - Time-sharing, but
    - RAS and global BHR are better to be dedicated

SMT Sharing of Pipeline Stages

- Decode
  - For RISC machines, major complexity is to resolve dependences ($O(n^2)$ complexity); thus partitioning would reduce complexity but could compromise single-thread performance
  - For CISC machines, determining instruction semantics and decomposing them can be very complex, time-sharing the decode stage may be more beneficial

SMT Sharing of Pipeline Stages

- Issue
  - Selection must involve multiple threads
  - Wakeup is limited to intra-thread
    - Partition instruction window?
- Execute
  - Sharing is straightforward
  - Design tradeoffs on bypass network

SMT Sharing of Pipeline Stages

- Memory
  - Sharing cache ports is straightforward
  - Design tradeoff of load/store queue
    - Sharing → potential consistency problem
    - Partitioning → simpler but lower utilization
- Retire
  - Partition or time-share

CMP vs. SMT

- CMP is easier to implement
- SMT can hide long latencies
- SMT has better resource utilization
- CMP + SMT?
  - IBM Power5

Comparisons between Multithreading Schemes

MT Approach	Resources Shared between Threads	Context Switch Mechanism
None	Everything	Explicit OS context switch
Fine-grained	Everything but register file and control logic/state	Switch every cycle
Coarse-grained	Everything but I-fetch buffers, RF, and control logic/state	Switch on pipeline stall
SMT	Everything but I-fetch buffers, RAS, ARF, control logic/state, ROB, SQ, …	All contexts concurrently active; no switch
CMP	L2 cache, system interconnect	All contexts concurrently active; no switch

Intel's Hyper-Threading Technology

- A single physical processor appear as two logical processors by applying a two-threaded SMT approach
- Each logical processor maintains a complete set of the architecture state (general-purpose registers, control registers, …)
- Logical processors share nearly all other resources, such as caches, execution units, branch predictors, control logic, and buses

Intel's Hyper-Threading Technology

- ROB entries, load and store buffer entries are statically partitioned among two threads
- Partitioned resources are recombined when only one thread is active
- Add less than 5% to the relative chip size
- Improve performance by 16% to 28% on server applications

Explicit vs. Implicit Multithreading

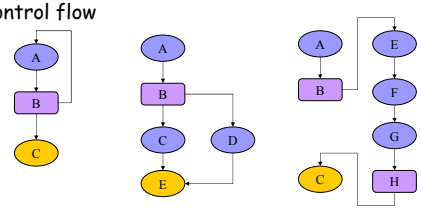
- Explicit
  - Improve instruction throughput
  - Programmer-created threads
- Implicit
  - Improve individual application's performance
  - Dynamically generated threads

Challenges in IMT Processor Designs

- Resolving control dependences
- Resolving register data dependences
- Resolving memory data dependences

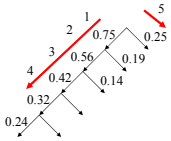
Resolving Control Dependences

- Spawn implicit future threads at subsequent control-independent points in the program's control flow



Disjoint Eager Execution

- Choose the branch path with the highest cumulative prediction rate



Resolving Register Data Dependences

- Dependences within a thread
  - Resolved with standard techniques
- Dependences across threads
  - Disallow interthread register data dependences, communicate all shared operands through memory with L/S
  - Compiler identify interthread dependences explicitly
  - Data dependence speculation

Resolving Memory Data Dependences

- Interthread false dependences (WAR and WAW)
  - Buffer writes from future threads and commit them when those threads retire
- Interthread true dependences (RAW)
  - Future threads assume no dependences violations + extensions to snoop-based cache coherence
  - Track L/S from each thread in separate per-thread L/S queues

Executing the Same Thread

- Execute the same instructions in multiple contexts
- Fault detection (transient errors)
- Prefetching
- Branch resolution

Real Processor: IBM Power5

- Each processor has two full-performance processor
- Each core supports two-way SMT
- Right picture: a Power5 MCM with four processor chips (16 logic CPUs)
- Each chip has 276M Xtors, size 389mm^2

POWER5 chief scientist Balaram Sinharoy holding a POWER5 MCM (Multi-chip Module)

Further Reading

1. Reference book, Chapter 11, "Executing Multiple Threads"
2. "A survey of processors with explicit multithreading", Theo Ungerer, Borut Robic and Jurij Silc, ACM Computing Surveys, Vol. 35, No. 1, March 2003, pages 29-63