ECE/CS 552: Parallel Processors

Instructor: Mikko H Lipasti

Fall 2010
University of Wisconsin-Madison

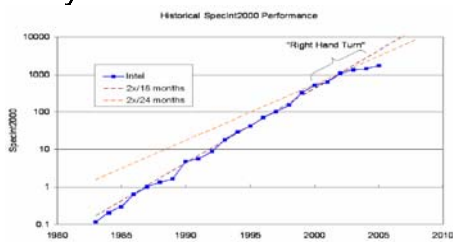
Parallel Processors

- Why multicore now?
- Thread-level parallelism
- Shared-memory multiprocessors
  - Coherence
  - Memory ordering
  - Split-transaction buses
- Multithreading
- Multicore processors

© Hill, Lipasti

2

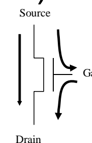
Why Multicore Now?



- Moore's Law for device integration
- Chip power consumption
- Single-thread performance trend

[source: Intel]

Leakage Power (Static/DC)



- Transistors aren't perfect on/off switches
- Even in static CMOS, transistors leak
  - Channel (source/drain) leakage
  - Gate leakage through insulator
    - High-K dielectric replacing SiO_2 helps
- Leakage compounded by
  - Low threshold voltage
    - Low V_{th} => fast switching, more leakage
    - High V_{th} => slow switching, less leakage
  - Higher temperature
    - Temperature increases with power
    - Power increases with C, V^2, A, f
- Rough approximation: leakage proportional to area
  - Transistors aren't free, unless they're turned off
- Controlling leakage
  - Power gating (turn off unused blocks)

Dynamic Power

$$P_{dyn} \approx kCV^2Af$$

- Aka AC power, switching power
- Static CMOS: current flows when transistors turn on/off
  - Combinational logic evaluates
  - Sequential logic (flip-flop, latch) captures new value (clock edge)
- Terms
  - C: capacitance of circuit (wire length, no. & size of transistors)
  - V: supply voltage
  - A: activity factor
  - f: frequency
- Moore's Law: which terms increase, which decrease?
  - Historically voltage scaling has saved us, but not any more

Reducing Dynamic Power

- Reduce capacitance
  - Simpler, smaller design
  - Reduced IPC
- Reduce activity
  - Smarter design
  - Reduced IPC
- Reduce frequency
  - Often in conjunction with reduced voltage
- Reduce voltage
  - Biggest hammer due to quadratic effect, widely employed
  - However, reduces max frequency, hence performance
  - Dynamic (power modes)
    - E.g. Transmeta Long Run, AMD PowerNow, Intel Speedstep

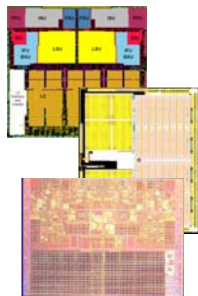
Frequency/Voltage relationship

- Lower voltage implies lower frequency
  - Lower V_{th} increases delay to sense/latch 0/1
- Conversely, higher voltage enables higher frequency
  - Overclocking
- Sorting/binning and setting various V_{dd} & V_{th}
  - Characterize device, circuit, chip under varying stress conditions
  - Black art – very empirical & closely guarded trade secret
  - Implications on reliability
    - Safety margins, product lifetime
    - This is why *overclocking* is possible

Frequency/Voltage Scaling

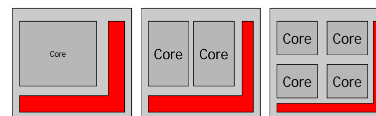
- Voltage/frequency scaling rule of thumb:
  - +/- 1% performance buys -/+ 3% power (3:1 rule)
- Hence, any power-saving technique that saves less than 3x power over performance loss is uninteresting
- Example 1:
  - New technique saves 12% power
  - However, performance degrades 5%
  - Useless, since 12 < 3 x 5
  - Instead, reduce f by 5% (also V), and get 15% power savings
- Example 2:
  - New technique saves 5% power
  - Performance degrades 1%
  - Useful, since 5 > 3 x 1
- Does this rule always hold?

Multicore Mania



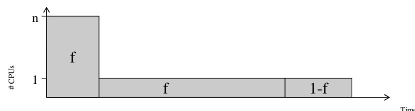
- First, servers
  - IBM Power4, 2001
- Then desktops
  - AMD Athlon X2, 2005
- Then laptops
  - Intel Core Duo, 2006
- Your cellphone
  - Baseband/DSP/application/graphics

Why Multicore



	Single Core	Dual Core	Quad Core
Core area	A	~A/2	~A/4
Core power	W	~W/2	~W/4
Chip power	W + O	W + O'	W + O''
Core performance	P	0.9P	0.8P
Chip performance	P	1.8P	3.2P

Amdahl's Law



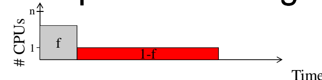
f – fraction that can run in parallel

1-f – fraction that must run serially

$$Speedup = \frac{1}{(1-f) + \frac{f}{n}}$$

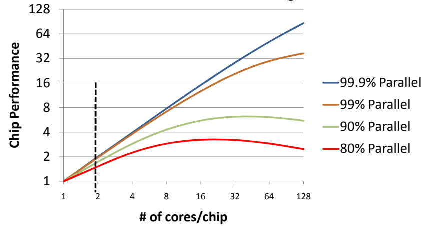
$$\lim_{n\to\infty} \frac{1}{1-f+\frac{f}{n}} = \frac{1}{1-f}$$

Fixed Chip Power Budget



- Amdahl's Law
  - Ignores (power) cost of n cores
- Revised Amdahl's Law
  - More cores → each core is slower
  - Parallel speedup < n
  - Serial portion (1-f) takes longer
  - Also, interconnect and scaling overhead

Fixed Power Scaling

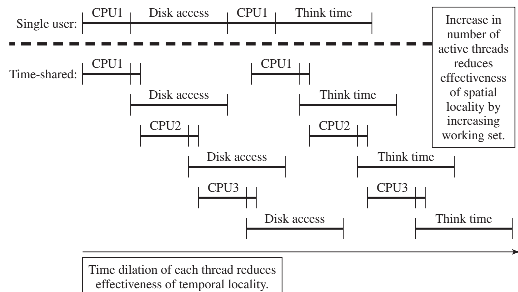


- Fixed power budget forces slow cores
- Serial code quickly dominates

Multicores Exploit Thread-level Parallelism

- Instruction-level parallelism
  - Reaps performance by finding independent work in a single thread
- Thread-level parallelism
  - Reaps performance by finding independent work across multiple threads
- Historically, requires explicitly parallel workloads
  - Originate from mainframe time-sharing workloads
  - Even then, CPU speed >> I/O speed
  - Had to overlap I/O latency with "something else" for the CPU to do
  - Hence, operating system would schedule other tasks/processes/threads that were "time-sharing" the CPU

Thread-level Parallelism



- Reduces effectiveness of temporal and spatial locality

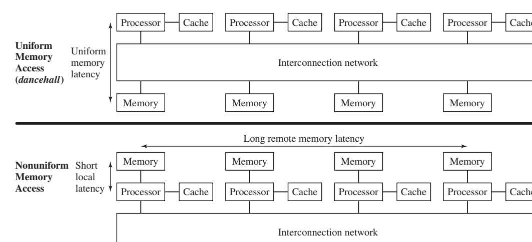
Thread-level Parallelism

- Motivated by time-sharing of single CPU
  - OS, applications written to be multithreaded
- Quickly led to adoption of multiple CPUs in a single system
  - Enabled scalable product line from entry-level single-CPU systems to high-end multiple-CPU systems
  - Same applications, OS, run seamlessly
  - Adding CPUs increases throughput (performance)
- More recently:
  - Multiple threads per processor core
    - Coarse-grained multithreading
    - Fine-grained multithreading
    - Simultaneous multithreading
  - Multiple processor cores per die
    - Chip multiprocessors (CMP)
    - Chip multithreading (CMT)

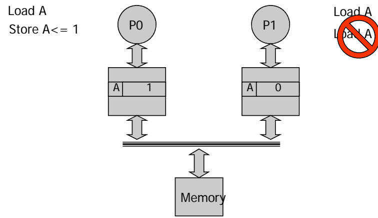
Multicore and Multiprocessor Systems

- Focus on shared-memory symmetric multiprocessors
  - Many other types of parallel processor systems have been proposed and built
  - Key attributes are:
    - Shared memory: all physical memory is accessible to all CPUs
    - Symmetric processors: all CPUs are alike
  - Other parallel processors may:
    - Share some memory, share disks, share nothing
      - E.g. GPGPU unit in the textbook
    - May have asymmetric processing units or noncoherent caches
- Shared memory idealisms
  - Fully shared memory: *usually nonuniform latency*
  - Unit latency: *approximate with caches*
  - Lack of contention: *approximate with caches*
  - Instantaneous propagation of writes: *coherence required*

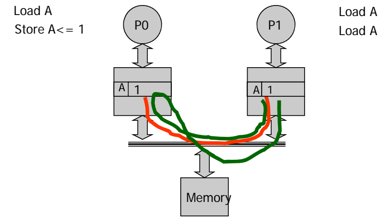
UMA vs. NUMA



Cache Coherence Problem



Cache Coherence Problem



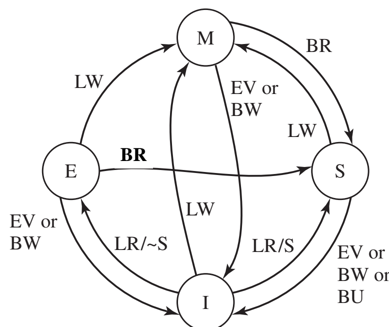
Invalidate Protocol

- Basic idea: maintain **single writer** property
  - Only one processor has write permission at any point in time
- Write handling
  - On write, invalidate all other copies of data
  - Make data private to the writer
  - Allow writes to occur until data is requested
  - Supply modified data to requestor directly or through memory
- Minimal set of states per cache line:
  - Invalid (not present)
  - Modified (private to this cache)
- State transitions:
  - Local read or write: I->M, fetch modified
  - Remote read or write: M->I, transmit data (directly or through memory)
  - Writeback: M->I, write data to memory

Invalidate Protocol Optimizations

- Observation: data can be *read-shared*
  - Add S (shared) state to protocol: MSI
- State transitions:
  - Local read: I->S, fetch shared
  - Local write: I->M, fetch modified; S->M, invalidate other copies
  - Remote read: M->S, supply data
  - Remote write: M->I, supply data; S->I, invalidate local copy
- Observation: data can be write-private (e.g. stack frame)
  - Avoid invalidate messages in that case
  - Add E (exclusive) state to protocol: MESI
- State transitions:
  - Local read: I->E if only copy, I->S if other copies exist
  - Local write: E->M silently, S->M, invalidate other copies

Sample Invalidate Protocol (MESI)



Sample Invalidate Protocol (MESI)

Current State s	Event and Local Coherence Controller Responses and Actions (s' refers to next state)					
	Local Read (LR)	**Local Write (LW)**	**Local Eviction (EV)**	**Bus Read (BR)**	**Bus Write (BW)**	**Bus Upgrade (BU)**
Invalid (I)	Issue bus read if no sharers then s' = E else s' = S	Issue bus write s' = M	s' = I	Do nothing	Do nothing	Do nothing
Shared (S)	Do nothing	Issue bus upgrade s' = M	s' = I	Respond shared	s' = I	s' = I
Exclusive (E)	Do nothing	s' = M	s' = I	Respond shared s' = S	s' = I	Error
Modified (M)	Do nothing	Do nothing	Write data back; s' = I	Respond dirty; Write data back; s' = S	Respond dirty; Write data back; s' = I	Error

Snoopy Cache Coherence

- Snooping implementation
  - Origins in shared-memory-bus systems
  - All CPUs could observe all other CPUs requests on the bus; hence "snooping"
    - Bus Read, Bus Write, Bus Upgrade
  - React appropriately to snooped commands
    - Invalidate shared copies
    - Provide up-to-date copies of dirty lines
      - Flush (writeback) to memory, or
      - Direct intervention (*modified intervention* or *dirty miss*)

Directory Cache Coherence

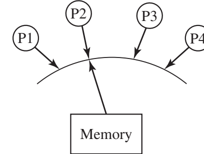
- Directory implementation
  - Extra bits stored in memory (directory) record MSI state of line
  - Memory controller maintains coherence based on the current state
  - Other CPUs' commands are not snooped, instead:
    - Directory forwards relevant commands
  - Ideal filtering: only observe commands that you need to observe
  - Meanwhile, bandwidth at directory scales by adding memory controllers as you increase size of the system
    - Leads to very scalable designs (100s to 1000s of CPUs)
- Can provide both snooping & directory
  - AMD Opteron switches based on socket count

Memory Consistency

```
Reorder    Proc0                    Proc1
load
before     st A=1                   st B=1
store      if (load B==0) {         if (load A==0) {
             ...critical section      ...critical section
           }                        }
```

- How are memory references from different processors interleaved?
- If this is not well-specified, synchronization becomes difficult or even impossible
  - ISA must specify consistency model
- Common example using Dekker's algorithm for synchronization
  - If load reordered ahead of store (as we assume for an OOO CPU)
  - Both Proc0 and Proc1 enter critical section, since both observe that other's lock variable (A/B) is not set
- If consistency model allows loads to execute ahead of stores, Dekker's algorithm no longer works
  - Common ISAs allow this: IA-32, PowerPC, SPARC, Alpha

Sequential Consistency [Lamport 1979]

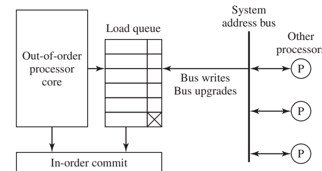


- Processors treated as if they are interleaved processes on a single time-shared CPU
- All references must fit into a total global order or interleaving that does not violate any CPUs program order
  - Otherwise sequential consistency not maintained
- Now Dekker's algorithm will work
- Appears to preclude any OOO memory references
  - Hence precludes any real benefit from OOO CPUs

High-Performance Sequential Consistency

- Coherent caches isolate CPUs if no sharing is occurring
  - Absence of coherence activity means CPU is free to reorder references
- Still have to order references with respect to misses and other coherence activity (snoops)
- Key: use speculation
  - Reorder references speculatively
  - Track which addresses were touched speculatively
  - Force replay (in order execution) of such references that collide with coherence activity (snoops)

High-Performance Sequential Consistency



- Load queue records all speculative loads
- Bus writes/upgrades are checked against LQ
- Any matching load gets marked for replay
- At commit, loads are checked and replayed if necessary
  - Results in machine flush, since load-dependent ops must also replay
- Practically, conflicts are rare, so expensive flush is OK

Multithreading

- Basic idea: CPU resources are expensive and should not be left idle
- 1960's: Virtual memory and multiprogramming
  - VM/MP invented to tolerate latency to secondary storage (disk/tape/etc.)
  - Processor:secondary storage cycle-time ratio: microseconds to tens of milliseconds (1:10000 or more)
  - OS context switch used to bring in other useful work while waiting for page fault or explicit read/write
  - Cost of context switch must be much less than I/O latency (easy)
- 1990's: Memory wall and multithreading
  - Processor: non-cache storage cycle-time ratio: nanosecond to fractions of a microsecond (1:500 or worse)
  - H/W task switch used to bring in other useful work while waiting for cache miss
  - Cost of context switch must be much less than cache miss latency
- Very attractive for applications with abundant thread-level parallelism
  - Commercial multi-user workloads

Approaches to Multithreading

- Fine-grained multithreading
  - Switch contexts at fixed fine-grain interval (e.g. every cycle)
  - Need enough thread contexts to cover stalls
  - Example: Tera MTA, 128 contexts, no data caches
- Benefits:
  - Conceptually simple, high throughput, deterministic behavior
- Drawback:
  - Very poor single-thread performance

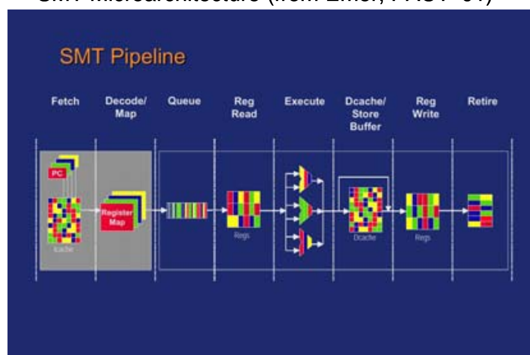
Approaches to Multithreading

- Coarse-grained multithreading
  - Switch contexts on long-latency events (e.g. cache misses)
  - Need a handful of contexts (2-4) for most benefit
- Example: IBM Northstar, 2 contexts
- Benefits:
  - Simple, improved throughput (~30%), low cost
  - Thread priorities mostly avoid single-thread slowdown
- Drawback:
  - Nondeterministic, conflicts in shared caches
  - Not suitable for out-of-order processors

Approaches to Multithreading

- Simultaneous multithreading
  - Multiple concurrent active threads (no notion of thread switching)
  - Need a handful of contexts for most benefit (2-8)
- Example: Intel P4, Core i7, IBM Power 5/6/7,
- Benefits:
  - Natural fit for OOO superscalar
  - Improved throughput
  - Low incremental cost
- Drawbacks:
  - Additional complexity over OOO superscalar
  - Cache conflicts

SMT Microarchitecture (from Emer, PACT '01)



Multithreading with Multicore

- Chip Multiprocessors (CMP)
- Share nothing in the core:
  - Implement multiple cores on die
  - Perhaps share L2, system interconnect (memory and I/O bus)
- Example: IBM Power4, 2 cores per die, shared L2
- Benefits:
  - Simple replication
  - Packaging density
  - Low interprocessor latency
  - ~2x throughput
- Drawbacks:
  - L2 shared – conflicts
  - Mem bandwidth shared – could become bottleneck

Approaches to Multithreading

- Chip Multiprocessors (CMP)
- Becoming very popular

Processor	Cores/ chip	Multi-threaded?	Resources shared
IBM Power 4	2	No	L2/L3, system interface
IBM Power 5	2	Yes (2T)	Core, L2/L3, system interface
Sun Ultrasparc	2	No	System interface
Sun Niagara	8	Yes (4T)	Everything
Intel Pentium D	2	Yes (2T)	Core, nothing else
Intel Core i7	4	Yes	L3, DRAM, system interface
AMD Opteron	2, 4, 6, 12	No	L3, DRAM, system interface

Approaches to Multithreading

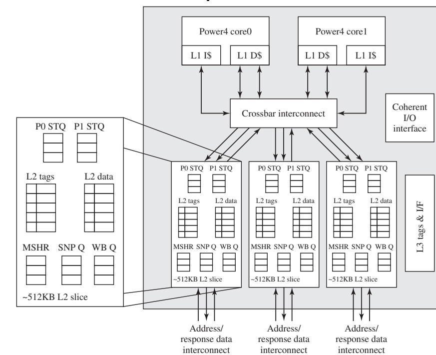
- Chip Multithreading (CMT)
  - Similar to CMP
- Share something in the core:
  - Expensive resource, e.g. floating-point unit (FPU)
  - Also share L2, system interconnect (memory and I/O bus)
- Example:
  - Sun Niagara, 8 cores, one FPU
  - AMD Bulldozer, FPU shared by two adjacent cores
- Benefit: amortize cost of expensive resource
- Drawbacks:
  - Shared resource may become bottleneck
  - Next generation Niagara does **not** share FPU

Multithreaded Processors

MT Approach	Resources shared between threads	Context Switch Mechanism
None	Everything	Explicit operating system context switch
Fine-grained	Everything but register file and control logic/state	Switch every cycle
Coarse-grained	Everything but I-fetch buffers, register file and con trol logic/state	Switch on pipeline stall
SMT	Everything but instruction fetch buffers, return address stack, architected register file, control logic/state, reorder buffer, store queue, etc.	All contexts concurrently active; no switching
CMT	Various core components (e.g. FPU), secondary cache, system interconnect	All contexts concurrently active; no switching
CMP	Secondary cache, system interconnect	All contexts concurrently active; no switching

- Many approaches for executing multiple threads on a single die
  - Mix-and-match: IBM Power7 8-core CMP x 4-way SMT

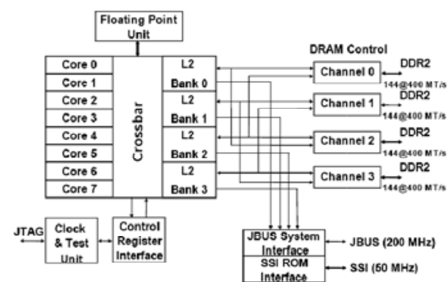
IBM Power4: Example CMP



Niagara Case Study

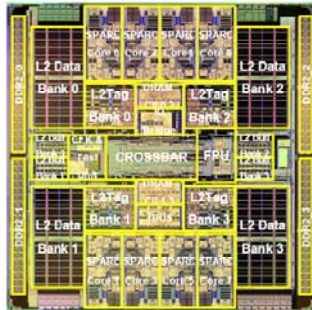
- Targeted application: web servers
  - Memory intensive (many cache misses)
  - ILP limited by memory behavior
  - TLP: Lots of available threads (one per client)
- Design goal: maximize throughput (/watt)
- Results:
  - Pack many cores on die (8)
  - Keep cores simple to fit 8 on a die, share FPU
  - Use multithreading to cover pipeline stalls
  - Modest frequency target (1.2 GHz)

Niagara Block Diagram [Source: J. Laudon]



- 8 in-order cores, 4 threads each
- 4 L2 banks, 4 DDR2 memory controllers

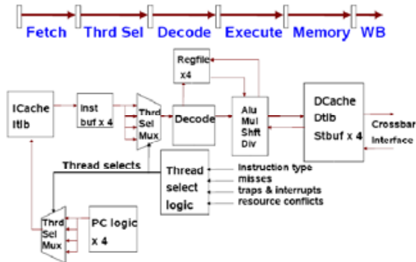
Ultrasparc T1 Die Photo [Source: J. Laudon]



Features:
- 8 64-bit Multithreaded SPARC Cores
- Shared 3 MB, 12-way 64B line writeback L2 Cache
- 16 KB, 4-way 32B line ICache per Core
- 8 KB, 4-way 16B line write-through DCache per Core
- 4 144-bit DDR-2 channels
- 3.2 GB/sec JBUS I/O

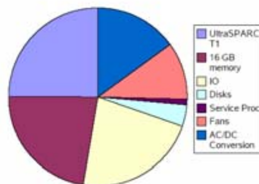
Technology:
- TI's 90nm CMOS Process
- 9LM Cu Interconnect
- 63 Watts @ 1.2GHz/1.2V
- Die Size: $379 mm^2$
- 279M Transistors
- Flip-chip ceramic LGA

Niagara Pipeline [Source: J. Laudon]



- Shallow 6-stage pipeline
- Fine-grained multithreading

T2000 System Power



- 271W running SpecJBB2000
- Processor is only 25% of total
- DRAM & I/O next, then conversion losses

Niagara Summary

- Example of *application-specific* system optimization
  - Exploit application behavior (e.g. TLP, cache misses, low ILP)
  - Build very efficient solution
- Downsides
  - Loss of *general-purpose* suitability
  - E.g. poorly suited for software development (parallel make, gcc)
  - Very poor FP performance (fixed in Niagara 2)

Summary

- Why multicore now?
- Thread-level parallelism
- Shared-memory multiprocessors
  - Coherence
  - Memory ordering
  - Split-transaction buses
- Multithreading
- Multicore processors

© Hill, Lipasti