Multicore computers
Course Summary

Multicore, Ch 18 [Sta10]
Background
Organization
Intel Core Duo and Core i7
ARM11 MPCore

Why Multicore?
- Current trend by processor manufacturers, because older improvements are no longer that promising
- Clock frequency
- Pipeline, superscalar,
- Simultaneous multithreading, SMT (or hyperthreading)
- Enough transistors available on one chip to put two or more whole cores on the chip
- Symmetric multiprocessor on one chip only
- But ... diminishing returns
  - More complexity requires more logic
  - Increasing chip area for coordinating and signal transfer logic

Real Problem
- Moore’s Law: transistor density doubles every 1.5 years
- Processor speed doubles also
- True 1980-2003?
- Heat barrier: can not pack processors so thick
- No more faster processors
- Now: more processors per chip
  - Multicore CPU
  - Chip-level multiprocessor (CMP)

What is Multicore?

Processor Performance History
- Late 1980’s no parallelism yet — flat curve
- Steep rise of the curve with improvements in instruction-level parallelism
  - pipelines, superscalar, SMT
- Flat again around 2000 -> limit of instruction-level parallelism reached
Lecture 12: Multicore and Summary

Processor Power Consumption

- Power consumption of Intel processors
- Notice the power requirement has grown exponentially

How to Use All the Transistors Available?

- Reduce power intensity by increasing the ratio of memory transistors to logic transistors
- Memory transistors used mainly for cache
- Logic transistors used for everything else
- Increased complexity in logic follows Pollack’s rule
  - On a single core the increased complexity of structure means that more of the logic is needed just for coordination and signal transfer logic

Software Performance on Multicore

- Amdahl’s law: speedup is proportional to the fraction of time enhancement is used
- Thus, even a small portion of sequential code has noticeable impact with larger number of processors!
- Software improvements are not covered in this course

Overhead Effect on Multicore Efficiency

- Communication
- Distribution of work
- Cache coherence

Application Specific Multicore Efficiency

- Key difference: Cache
  - L1 always dedicated
  - L2 always dedicated if instructions and data
  - L2 shared or dedicated (or mixed)
  - Active research on this issue
  - L3 shared, if exists
  - Cache coherence
  - Varies

Comp. Org II, Spring 2012
Shared L2 Cache vs. Dedicated ones

- Constructive interference
  - One core may fetch a cache line that is soon needed by another code – already available in shared cache
- Single copy
  - Shared data is not replicated, so there is just one copy of it.
- Dynamic allocation
  - The thread that has less locality needs more cache and may occupy more of the cache area
- Shared cache – no cache coherence solution needed
  - The shared data element already in the shared cache. With dedicated caches, the shared data must be invalidated from other caches before using
- Slower access
  - Larger cache area is slower to access, small dedicated cache would be faster

Intel Core Duo and Core i7

Intel Core Duo, 2006

- Two x86 superscalar, shared L2 cache
- MESI support for L1 caches
- L2 data shared between local cores or external
- Thermal control unit per core
  - Manages chip heat dissipation
  - Maximizes performance within constraints
- Advanced Programmable Interrupt Controlled (APIC)
  - Inter process interrupts between cores
  - Routes interrupts to appropriate core
  - Includes timer so OS can interrupt core
- Power Management Logic
  - Adjusts voltage and power consumption
  - Can switch individual processor logic subsystems on and off

Intel Core i7, 2008

- Four x86 SMT processors each with two simultaneous threads
- Dedicated L2, shared L3 cache
- Speculative pre-fetch for caches
- On chip DDR3 memory controller
  - No front side bus (mem access through cache)
- QuickPath Interconnection
  - Cache coherent point-to-point link
  - High speed communications between processor chips
  - Dedicated bi directional pairs

ARM11 MPCore

- Up to 4 processors per chip
- Distributed interrupt controller
- Timer per CPU
- Watchdog
  - Warning alerts for software failures
  - Counts down from predetermined values, issues warning at zero
- CPU interface
  - Interrupt acknowledgement, masking and completion acknowledgement
- MP11 – Single ARM11 core (CPU)
- Vector floating-point (VFP) co-processor
- Dedicated split snoopy L1 cache
- Shared unified L2 cache, off-chip

ARM11 MPCore

- Up to 4 processors per chip
- Distributed interrupt controller
- Timer per CPU
- Watchdog
  - Warning alerts for software failures
  - Counts down from predetermined values, issues warning at zero
- CPU interface
  - Interrupt acknowledgement, masking and completion acknowledgement
- MP11 – Single ARM11 core (CPU)
- Vector floating-point (VFP) co-processor
- Dedicated split snoopy L1 cache
- Shared unified L2 cache, off-chip
ARM11 MPCore Interrupt Control

- Distributed Interrupt Controller (DIC)
- Collates interrupts from many sources
- Masking, prioritization
- Distribution to target MP11 CPUs
- Status tracking (Interrupt states: pending, active, inactive)
- Software interrupt generation
- Number of interrupts independent of MP11 CPU design
- Accessed by CPUs via private interface through SCU
- Can route interrupts to single or multiple CPUs
- OS can generate interrupts: all-but-self, self, or specific CPU
- Provides inter-process communication (16 intr. ids)
- Thread on one CPU can cause activity by thread on another CPU

ARM11 MPCore L1 Cache Coherency

- (modified) MESI
- Direct Data Intervention (DDI)
- Copy clean cache lines directly between caches
- Duplicated tag RAM (tag fields)
- Copies of tag RAM in many CPU's
- Cache knows who has the data needed
- Migratory lines
  - Copy dirty cache lines directly to other caches
  - No need to go to L2 cache, or to memory
  - "Invalid" MESI-state to old location

Course Summary and exam information

Two previous course exams from year 2010 distributed on paper during the lecture.

Course Exam

Wed 29.02.2012 16:00-19:00 (B123)

- 2.5 hours – three or four questions
- Questions are in English
- You may answer in English, Finnish, or Swedish
- Question try to assess your deeper understanding of relevant topics, not superficial facts
- More of applying what you have learned
- Some of understanding relevant concepts
- Less of rephrasing topics from text book or lectures
- No details on example architectures
- You can write on all answers on the same paper using pencil or pen
- No need to write answers to each question to separate sheet
- There is no need for a calculator, but a simple one is allowed
- If there is math needed, you can just write the formula and you do not need to write the result number without a calculator

For the Exam

- Go through the exercises
- If you did all homeworks and understand them well, you should do fine in the exam
- Read the book and lecture slides
- If there is nothing on the slides about the subsection, then there very probably is not a question in the exam
- The review questions in the slides are good hints!
- Old exams are in web
- Many exams only in Finnish
- See https://www.cs.helsinki.fi/courses/581365/2010/sk/1
  - "Basic Information" sub-page (tab)
  - "Kerola’s CO-II home page", and “Previous Exams” there
- Exam questions have high temporal locality!
Example question from year 2006

One SPARC processor model has 136 physical registers, but only 32 registers are usable in the instruction at a given time, since there is only 5 bits for the register number in the instruction. These 32 registers form a register window. Processor has a management register CWP (current window pointer) to indicate the active register window of the currently executing subroutine.

What are the benefits of using register window instead of a more traditional subroutine call mechanism (e.g. activation record)?

How does the calling code pass the parameters to the subroutine? How does the subroutine refer to global variables, input parameters, and local variables?

How many registers windows does the processor model have?

How does the SPARC processor change the register number of the instruction to the physical register number? Give an example!

How does it handle a situation where all register windows are already in use and a new subroutine call is made?

---

Example question from year 2006: Dependencies

A) What types of dependencies between machine instructions appear in a normal (not super-scalar) pipeline? Give a concrete (machine language level) example on each dependency type.

B) How can one prevent the possible problems caused by these (part a) dependencies? How can one reduce the performance deficiencies caused by them?

C) What types of dependencies between machine instructions appear in a super-scalar pipeline (dependencies not mentioned above)? Give a concrete (machine language level) example on each dependency type.

D) How can one prevent the possible problems caused by these (part c) dependencies? How can one reduce the performance deficiencies caused by them?

---

Example question from year 2003: Cache

Let us assume that (in an unrealistically small memory addressing) an 8-bit byte address of a 4-byte word is 0xA4. The cache line size is 32 bytes. Draw the cache structure and explain in details, how the word is found from the cache, when the cache is using

A) direct mapping.
B) associative or fully associative mapping.
C) 4-way set associative mapping.

You may assume that the word is found from the cache.

How many bytes can the cache contain? What are the field names and sizes used by each mapping?

---

Course Structure

Week 1
- Overview (Ch 1 – 8)
- Digital logic (Ch 20)
- Bus (Ch 3)

Week 2
- Memory, Cache (Ch 4, 5)
- Virtual memory (Ch 8)

Week 3
- Computer arithmetic (Ch 9)
- Instruction set (Ch 10, 11)

Week 4
- CPU struct & func. (Ch 12)
- RISC-architecture (Ch 13)

Week 5
- Instruction-level parallelism, Superscalar Processor (Ch 14)
- Control Unit (Ch 15-16)

Week 6
- Parallel Processing (Ch 17)
- Multicore (Ch 18)
- Summary

---

Digital logic (Ch 20)

What is the problem and how is it solved?
Boolean algebra, gates and flip-flops
Basic ideas on optimization, no Carnaugh maps
Circuit description with Boolean tables, gates, and graphs
Flip-flops and basic circuits, basic functionality
Understand, how S-R flip-flop works
Combination circuits vs. sequential circuits
How to implement memory?
How to implement functions?
How to implement 32-bit add?

---

Bus (Ch 3)

What is the problem and how is it solved?
Instruction cycle, interrupts
Bus characteristics
Speed, width, asynchronous/synchronous timing,
Signaling, centralized/distributed arbitration,
Events or transactions
PCI
Arbitration, read & write sequences
Can read and explain timing diagrams
Lecture 12: Multicore and Summary

Cache (Ch 4)
- What is the problem? How is it solved?
- Principle of locality, temporal & spatial locality
- Design features
  - Size, line size, split/unified, levels (L1, L2, L3)
  - Mapping: direct mapping, fully-associative, set-associative
  - Replacement policy
  - Write policy: write-through, write-back, write-once
- Cache coherency problem for multiprocessors

Main Memory (Ch 5)
- Basic ideas, no details
- DRAM implementation principles
  - Memory address split row and column access select fields
- How to build larger memory from smaller chips

Memory Management (Ch 8)
- Focus on (hw support for) virtual memory
  - What problem is solved? How is it solved?
  - Solution is based on locality
  - Solve protection problems at the same time?
- Virtual memory organization
  - page table, inverted page table, segment table
  - hierarchical tables
  - Disk organization to support VM
- Address translation
  - What is the problem, what is the solution, how is it done?
  - TLB, how does it work, how is it implemented?
  - TLB and cache, how do they work together
  - How do you locate referenced data (in cache or in memory)

Computer Arithmetic (Ch 9)
- What is the problem and how is it solved?
  - Integers
    - Representation
    - Add & subtract, multiply, divide
    - Booth’s algorithm for multiplication
  - Floating-point
    - IEEE representation, unnormalized, NaN
  - Principles of add, sub, mul, div – overflows/underflows
  - Accuracy
    - In representation
    - In computation
    - Loss of accuracy in certain math ops

Instruction Sets (Ch 10, 11)
- What is the problem and how is it solved?
- Characteristics
  - Data types, register sets
  - Addressing modes
- Architecture types
  - Accumulator, stack, register, load-store
- Instruction formats
  - Pentium cisc vs. Arm risc
  - Can explain basic differences
  - No need to study details

CPU Structure and Function (Ch 12)
- What is the problem and how is it solved?
  - Structural elements: registers, internal registers, pws
  - Pipelined implementation of fetch-execute cycle
- What is the problem and how is it solved?
  - Performance gains: when and how much?
  - Hazards & dependencies
    - Types: structural, control, data
    - Solution methods: bubbles, compiler, more HW
  - How to solve RAW data dependency problems?
    - Bubble (hw), NOP (sw bubble), instr order (sw), by-pass circuits (hw)
- How to solve control dependencies?
  - Clear pipeline (hw), delay slots (sw), mult. conditional instr. streams
  - Prefetch target buffer, loop buffer
  - Static and dynamic branch prediction, branch history table, ...
RISC (Ch 13)
- What is the problem and how is it solved?
- What is CISC, what is RISC?
- RISC vs. CISC
- RISC features
  - Lots of registers, few data types
  - Few operands and memory addressing types
  - Simple instructions optimized for pipeline
  - Load/Store architecture
- Register files
  - What is the problem and how is it solved?
  - Registers windows, register optimization
- Register allocation problem
  - What is the problem and how is it solved (graph coloring)?

Superscalar (Ch 14)
- What is the problem and how is it solved?
- Implementation strategies
  - In-order or out-of-order issue
  - In-order or out-of-order complete
  - Instruction selection window, window of execution
- Name dependencies
  - What is the problem and how is it solved?
  - New dependency types to worry about: WAR, WAW
  - Register renaming
- Hyperthreading or multithreading
  - What is the problem and how is it solved?
  - Use larger register set to better utilize pipelines

Control-Unit (Ch 15, 16)
- What is the problem and how is it solved?
- Micro-ops
  - Micro-op sequences in different phases of the execution cycle
- How control signals make things happen?
  - Control signal state machine
- Hardwired control
  - Direct implementation of control state machine
  - Requires lots of optimization to reduce state space
- Microprogrammed control
  - Structure: control memory, control address, control buffer
  - Horizontal or vertical (functional & resource encoding)
  - Sequencing, i.e., which microinstruction next?

Parallel Processing (Ch 17)
- What is the problem and how is it solved?
- Classification, SIMD, SMP, etc
- Cache coherency
  - Snoopy-cache
  - MESI
- Clusters
  - NUMA, CC-NUMA
- Vector computation
  - Vector operations and their implementation

Multicore (Ch 18)
- What is the problem and how is it solved?
- Multicore vs. SMP
- Different multicore organizations
  - What type of cache

-- The End --