

ECE/CS 757: Advanced Computer Architecture II

Instructor: Mikko H Lipasti

Spring 2009

University of Wisconsin-Madison

Lecture notes based on slides created by John Shen, Mark Hill, David Wood, Guri Sohi, and Jim Smith, Natalie Enright Jerger, and probably others

Computer Architecture

- Instruction Set Architecture (IBM 360)
 - ... the attributes of a [computing] system as seen by the programmer. I.e. the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls, the logic design, and the physical implementation. -- Amdahl, Blaaw, & Brooks, 1964
- Machine Organization (microarchitecture)
 - ALUS, Buses, Caches, Memories, etc.
- Machine Implementation (realization)
 - Gates, cells, transistors, wires

757 In Context

- Prior courses
 - 352 – gates up to multiplexors and adders
 - 354 – high-level language down to machine language interface or [instruction set architecture \(ISA\)](#)
 - 552 – implement logic that provides ISA interface
 - CS 537 – provides OS background (co-req. OK)
- This course – 757 – covers parallel machines
 - Multiprocessor systems
 - Data parallel systems
 - Memory systems that exploit MLP
 - Etc.
- Additional courses
 - ECE 752 covers advanced uniprocessor design (not a prereq)
 - Will review key topics in optional lecture
 - ECE 755 covers VLSI design
 - CS 758 covers parallel programming

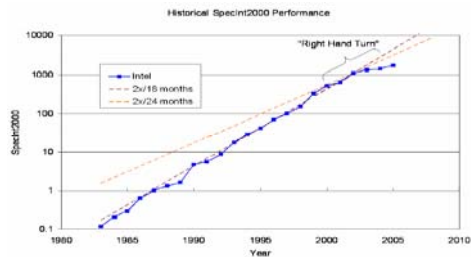
Why Take 757?

- To become a computer designer
 - Alumni of this class helped design your computer
- To learn what is *under the hood* of a computer
 - Innate curiosity
 - To better understand when things break
 - To write better code/applications
 - To write better system software (O/S, compiler, etc.)
- Because it is intellectually fascinating!
- Because multicore/parallel systems are ubiquitous

Computer Architecture

- Exercise in engineering tradeoff analysis
 - Find the fastest/cheapest/power-efficient/etc. solution
 - Optimization problem with 100s of variables
- All the variables are changing
 - At non-uniform rates
 - With inflection points
 - Only one guarantee: Today's right answer will be wrong tomorrow
- Two high-level effects:
 - **Technology push**
 - **Application Pull**

Trends



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

Mikko Lipasti-University of Wisconsin

[source: Intel]

Dynamic Power

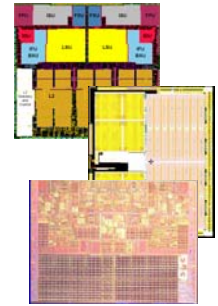
$$P_{dyn} \approx \sum_{i \in units} k_i C_i V^2 A_i f$$

- Static CMOS: current flows when active
 - Combinational logic evaluates new inputs
 - Flip-flop, latch captures new value (clock edge)
- Terms
 - C: capacitance of circuit
 - wire length, number and size of transistors
 - V: supply voltage
 - A: activity factor
 - f: frequency
- Future: Fundamentally power-constrained

Mikko Lipasti-University of Wisconsin

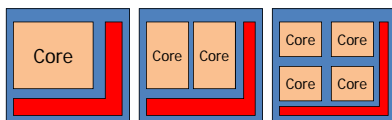
Multicore Mania

- First, servers
 - IBM Power4, 2001
- Then desktops
 - AMD Athlon X2, 2005
- Then laptops
 - Intel Core Duo, 2006
- Soon, your cellphone
 - ARM MPCore, prototypes for a while now



Mikko Lipasti-University of Wisconsin

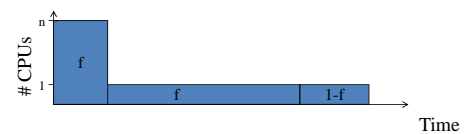
Why Multicore



	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

Mikko Lipasti-University of Wisconsin

Amdahl's Law



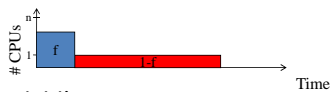
f – fraction that can run in parallel
 1-f – fraction that must run serially

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

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

Mikko Lipasti-University of Wisconsin

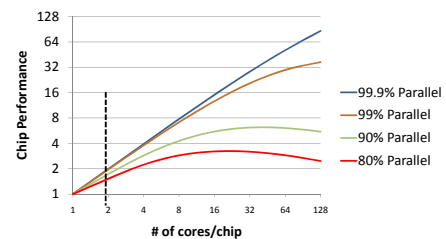
Fixed Chip Power Budget



- 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

Mikko Lipasti-University of Wisconsin

Fixed Power Scaling



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

Mikko Lipasti-University of Wisconsin

Challenges

- Parallel scaling limits *many-core*
 - >4 cores only for well-behaved programs
 - Optimistic about *new* applications
- Interconnect overhead
- Single-thread performance
 - Will degrade unless we innovate
- Parallel programming
 - Express/extract parallelism in new ways
 - Retrain programming workforce

Mikko Lipasti-University of Wisconsin

Finding Parallelism

1. Functional parallelism
 - Car: {engine, brakes, entertain, nav, ...}
 - Game: {physics, logic, UI, render, ...}
2. Automatic extraction
 - Decompose serial programs
3. Data parallelism
 - Vector, matrix, db table, pixels, ...
4. Request parallelism
 - Web, shared database, telephony, ...

Mikko Lipasti-University of Wisconsin

Balancing Work

- Amdahl's parallel phase f : all cores busy
- If not perfectly balanced
 - $(1-f)$ term grows (f not fully parallel)
 - Performance scaling suffers
- Manageable for data & request parallel apps
- Very difficult problem for other two:
 - Functional parallelism
 - Automatically extracted

Mikko Lipasti-University of Wisconsin

Coordinating Work

- Synchronization
 - Some data somewhere is shared
 - Coordinate/order updates and reads
 - Otherwise → chaos
- Traditionally: locks and mutual exclusion
 - Hard to get right, even harder to tune for perf.
- Research: Transactional Memory
 - Programmer: Declare potential conflict
 - Hardware and/or software: speculate & check
 - Commit or roll back and retry

Mikko Lipasti-University of Wisconsin

Single-thread Performance

- Still most attractive source of performance
 - Speeds up parallel and serial phases
 - Can use it to buy back power
- Must focus on *power consumption*
 - Performance benefit \geq Power cost
- Focus of 752; brief review in two weeks

Mikko Lipasti-University of Wisconsin

Focus of this Course

- How to minimize these overheads
 - Interconnect
 - Synchronization
 - Cache Coherence
 - Memory systems
- Also
 - How to write parallel programs (a little)
 - Non-cache coherent systems (clusters, MPP)
 - Data-parallel systems

Expected Background

- ECE/CS 552 or equivalent
 - Design simple uniprocessor
 - Simple instruction sets
 - Organization
 - Datapath design
 - Hardwired/microprogrammed control
 - Simple pipelining
 - Basic caches
 - Some 752 content (optional review)
- High-level programming experience
 - C/UNIX skills – modify simulators

About This Course

- Readings
 - Posted on website later this week
 - Make sure you keep up with these! Often discussed in depth in lecture, with required participation
 - Subset of papers must be reviewed in writing
- Lecture
 - Attendance required
- Homeworks
 - Due at beginning of class time
 - Must be individual effort

About This Course

- Exams
 - Midterm 1: W 3/11 in class
 - Midterm 2: Tue 5/12 10:05 AM (final exam time slot)
 - Keep up with reading list!
- Textbook
 - No textbook, use Culler&Singh for reference
 - Four beta book chapters from Jim Smith
 - PDF links on web page
 - Any demand for copy shop duplication?

About This Course

- Course Project
 - Research project
 - Replicate results from a paper
 - Or attempt something novel
 - Parallelize/characterize new application
- Final project includes a written report and an oral presentation
 - Reports due 5/8
 - Presentations during class time 5/4 and 5/6

About This Course

- Grading
 - Homework and Paper Reviews 20%
 - Midterm 1 25%
 - Midterm 2 25%
 - Project 30%
- Web Page (check regularly)
 - <http://ece757.ece.wisc.edu>

About This Course

- Office Hours
 - Prof. Lipasti: EH 4613, M 1-4, R 9-11, or by appt.
 - TAs: Mitch Hayenga, Andrew Nere, TBD
- Communication channels
 - E-mail to instructor, class e-mail list
 - ece757-1-s09@lists.wisc.edu
 - Web page
 - <http://ece757.ece.wisc.edu>
 - Office hours

About This Course

- Other Resources
 - Computer Architecture Colloquium – Tuesday 4-5PM, 1325 CSS
 - Computer Engineering Seminar – Friday 12-1PM, EH4610
 - Architecture mailing list: <http://lists.cs.wisc.edu/mailman/listinfo/architecture>
 - WWW Computer Architecture Page <http://www.cs.wisc.edu/~arch/www>

About This Course

- Lecture schedule:
 - MWF 11:00-11:50
- Proposed change:
 - MWF 11:00-12:15
 - Cancel 1 of 3 lectures (on average)
 - As long as no hard conflicts

26

Tentative Schedule

Week 1	Introduction
Week 2	MP Software and ISA
Week 3	More MP Software, optional 752 review
Week 4	Cores, Multithreading, Multicore
Week 5	Cores, Multithreading, Multicore
Week 6	Multiprocessor Memory Systems
Week 7	More memory, case studies
Week 8	Review, midterm 1 in class on 3/11
Week 9	Interconnects
Week 10	SIMD
Week 11	Dataflow
Week 12	Clusters
Week 13	MPP Systems
Week 14	Schedule slack
Week 15	Project talks, Course Evaluation, Final reports
Finals Week	Midterm 2 Tuesday 5/12 10:05am

Brief Introduction to Parallel Computing

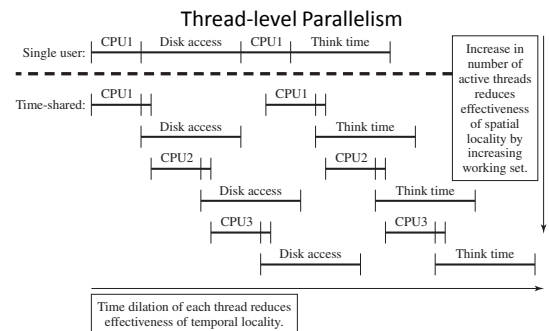
- Thread-level parallelism
- Multiprocessor Systems
- Cache Coherence
 - Snoopy
 - Scalable
- Flynn Taxonomy
- UMA vs. NUMA

28

Thread-level Parallelism

- Instruction-level parallelism (752 focus)
 - 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
 - Originates from mainframe time-sharing workloads
 - Even then, CPU speed \gg 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

29



- Reduces effectiveness of temporal and spatial locality

30

Thread-level Parallelism

- Initially 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 (aka “switch-on-event”)
 - Fine-grained multithreading
 - Simultaneous multithreading
 - Multiple processor cores per die
 - Chip multiprocessors (CMP)

31

Multiprocessor Systems

- Primary 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
 - Have asymmetric processing units
- Shared memory idealisms
 - Fully shared memory
 - Unit latency
 - Lack of contention
 - Instantaneous propagation of writes

32

Motivation

- So far: one processor in a system
- Why not use N processors
 - Higher throughput via parallel jobs
 - Cost-effective
 - Adding 3 CPUs may get 4x throughput at only 2x cost
 - Lower latency from multithreaded applications
 - Software vendor has done the work for you
 - E.g. database, web server
 - Lower latency through parallelized applications
 - Much harder than it sounds

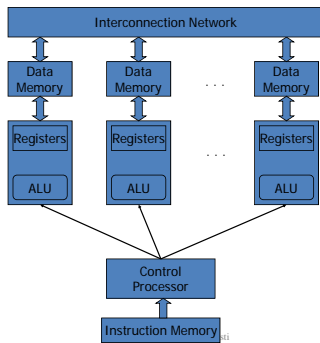
33

Where to Connect Processors?

- At processor?
 - Single-instruction multiple data (SIMD)
- At I/O system?
 - Clusters or multicomputers
- At memory system?
 - Shared memory multiprocessors
 - Focus on Symmetric Multiprocessors (SMP)

34

Connect at Processor (SIMD)



35

Connect at Processor

- SIMD Assessment
 - Amortizes cost of control unit over many datapaths
 - Enables efficient, wide datapaths
 - Programming model has limited flexibility
 - Regular control flow, data access patterns
- SIMD widely employed today
 - MMX, SSE, 3DNow vector extensions
 - Data elements are 8b multimedia operands

36

Connect at I/O

- Connect with standard network (e.g. Ethernet)
 - Called a cluster
 - Adequate bandwidth (GB Ethernet, going to 10GB)
 - Latency very high
 - Cheap, but “get what you pay for”
- Connect with custom network (e.g. IBM SP1, SP2, SP3)
 - Sometimes called a multicomputer
 - Higher cost than cluster
 - Poorer communication than multiprocessor
- Internet data centers built this way

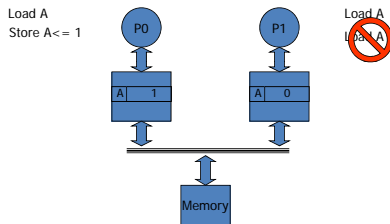
37

Connect at Memory: Multiprocessors

- Shared Memory Multiprocessors
 - All processors can address all physical memory
 - Demands evolutionary operating systems changes
 - Higher throughput with no application changes
 - Low latency, but requires parallelization with proper synchronization
- Most successful: Symmetric MP or SMP
 - 2-64 microprocessors on a bus
 - Still use cache memories

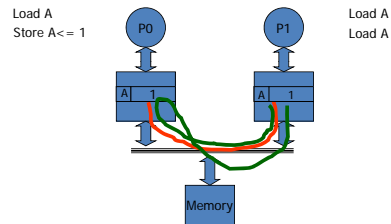
38

Cache Coherence Problem



39

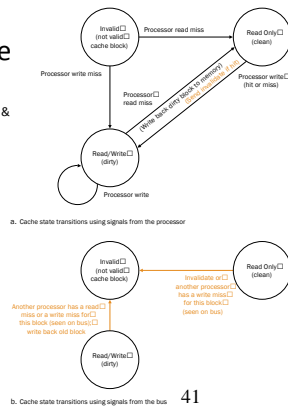
Cache Coherence Problem



40

Minimal Coherence Protocol FSM

[Source: Patterson/Hennessy, Comp. Org. & Design]



41

Snoopy Cache Coherence

- All requests broadcast on bus
- All processors and memory snoop and respond
- Cache blocks writeable at one processor or read-only at several
 - Single-writer protocol
- Snoops that hit dirty lines?
 - Flush modified data out of cache
 - Either write back to memory, then satisfy remote miss from memory, or
 - Provide dirty data directly to requestor
 - Big problem in MP systems
 - Dirty/coherence/sharing misses

42

Scaleable Cache Coherence

- Eschew physical bus but still snoop
 - Point-to-point tree structure
 - Root of tree provides ordering point
- Or, use level of indirection through directory
 - Directory at memory remembers:
 - Which processor is “single writer”
 - Forwards requests to it
 - Which processors are “shared readers”
 - Forwards write permission requests to them
 - Level of indirection has a price
 - Dirty misses require 3 hops instead of two
 - Snoop: Requestor->Owner->Requestor
 - Directory: Requestor->Directory->Owner->Requestor

43

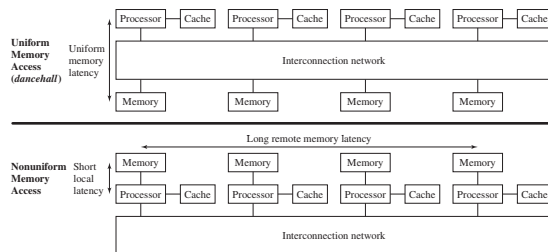
Taxonomies

Flynn (1966)	Single Data	Multiple Data
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

For Shared Memory	Uniform Memory	Nonuniform Memory
Cache Coherence	CC-UMA	CC-NUMA
No Cache Coherence	NCC-UMA	NCC-NUMA

44

UMA vs. NUMA



45

Example Commercial Systems

- CC-UMA (SMP)
 - Sun E10000: <http://doi.ieeecomputersociety.org/10.1109/40.653032>
- CC-NUMA
 - SGI Origin 2000: [The SGI Origin: A ccnuma Highly Scalable Server](http://www.sgi.com/origin2000/)
- NCC-NUMA
 - Cray T3E: http://www.cs.wisc.edu/~markhill/Misc/aspl096_t3e_comm.pdf
- Clusters
 - ASCI: <https://www.llnl.gov/str/Seager.html>

46

Summary

- Thread-level parallelism
- Multiprocessor Systems
- Cache Coherence
 - Snoopy
 - Scalable
- Flynn Taxonomy
- UMA vs. NUMA

47