# 18-447: Computer Architecture Lecture 24: Runahead and Multiprocessing

Prof. Onur Mutlu Carnegie Mellon University Spring 2012, 4/23/2012

# Reminder: Lab Assignment 6

#### Due Today

- Implementing a more realistic memory hierarchy
  - L2 cache model
  - DRAM, memory controller models
  - MSHRs, multiple outstanding misses
- Extra credit: Prefetching

## Reminder: Lab Assignment 7

- Cache coherence in multi-core systems
  - MESI cache coherence protocol
- Due May 4
- Extra credit: Improve the protocol (open-ended)

#### Midterm II

- Midterms will be distributed today
- Please give me a 15-minute warning!

#### Final Exam

#### May 10

- Comprehensive (over all topics in course)
- Three cheat sheets allowed
- We will have a review session (stay tuned)
- Remember this is 30% of your grade
  - □ I will take into account your improvement over the course
  - □ Know the previous midterm concepts by heart

# A Belated Note on Course Feedback

- We have taken into account your feedback
- Some feedback was contradictory
  - Pace of course
    - Fast or slow?
    - Videos help
  - Homeworks
    - Love or hate?
  - Workload
    - Too little/easy
    - Too heavy
  - Many of you indicated you are learning a whole lot

# Readings for Today

Required

- Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.
- Hill, Jouppi, Sohi, "Multiprocessors and Multicomputers," pp. 551-560 in Readings in Computer Architecture.
- Hill, Jouppi, Sohi, "Dataflow and Multithreading," pp. 309-314 in Readings in Computer Architecture.
- Recommended
  - Historical: Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

## Readings for Wednesday

- Required cache coherence readings:
  - Culler and Singh, *Parallel Computer Architecture* 
    - Chapter 5.1 (pp 269 283), Chapter 5.3 (pp 291 305)
  - □ P&H, Computer Organization and Design
    - Chapter 5.8 (pp 534 538 in 4<sup>th</sup> and 4<sup>th</sup> revised eds.)
- Recommended:
  - Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," IEEE Transactions on Computers, 1979
  - Papamarcos and Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," ISCA 1984.

#### Last Lecture

- Wrap up prefetching
  - Markov prefetching
  - Content-directed prefetching
  - Execution-based prefetching
- Runahead execution

## Today

- Wrap-up runahead execution
- Multiprocessing fundamentals
- The cache coherence problem

Memory Latency Tolerance and Runahead Execution

#### How Do We Tolerate Stalls Due to Memory?

- Two major approaches
  - Reduce/eliminate stalls
  - □ Tolerate the effect of a stall when it happens
- Four fundamental techniques to achieve these
  - Caching
  - Prefetching
  - Multithreading
  - Out-of-order execution
- Many techniques have been developed to make these four fundamental techniques more effective in tolerating memory latency

#### Review: Execution-based Prefetchers

- Idea: Pre-execute a piece of the (pruned) program solely for prefetching data
  - Only need to distill pieces that lead to cache misses
  - Speculative thread: Pre-executed program piece can be considered a "thread"
  - Speculative thread can be executed
  - On a separate processor/core
  - On a separate hardware thread context (think fine-grained multithreading)
  - On the same thread context in idle cycles (during cache misses)

#### Review: Thread-Based Pre-Execution



- Dubois and Song, "Assisted Execution," USC Tech Report 1998.
- Chappell et al.,
  "Simultaneous Subordinate Microthreading (SSMT)," ISCA 1999.
- Zilles and Sohi, "Executionbased Prediction Using Speculative Slices", ISCA 2001.

#### Review: Runahead Execution

Perfect Caches:



#### Review: Runahead Execution Pros and Cons

#### Advantages:

- + Very accurate prefetches for data/instructions (all cache levels)
  - + Follows the program path
- + Simple to implement, most of the hardware is already built in
- + Versus other pre-execution based prefetching mechanisms:
  - + Uses the same thread context as main thread, no waste of context
  - + No need to construct a pre-execution thread

#### Disadvantages/Limitations:

- -- Extra executed instructions
- -- Limited by branch prediction accuracy
- -- Cannot prefetch dependent cache misses. Solution?
- -- Effectiveness limited by available "memory-level parallelism" (MLP)
- -- Prefetch distance limited by memory latency

Implemented in IBM POWER6, Sun "Rock"

#### Execution-based Prefetchers Pros and Cons

- + Can prefetch pretty much any access pattern
- + Can be very low cost (e.g., runahead execution)
  - + Especially if it uses the same hardware context
  - + Why? The processsor is equipped to execute the program anyway
- + Can be bandwidth-efficient (e.g., runahead execution)
- -- Depend on branch prediction and possibly value prediction accuracy
  - Mispredicted branches dependent on missing data throw the thread off the correct execution path
- -- Can be wasteful
  - -- speculatively execute many instructions
  - -- can occupy a separate thread context

#### Performance of Runahead Execution



#### Runahead Execution vs. Large Windows



#### Runahead on In-order vs. Out-of-order



#### Effect of Runahead in Sun ROCK

Shailender Chaudhry talk, Aug 2008.



#### Limitations of the Baseline Runahead Mechanism

#### Energy Inefficiency

- A large number of instructions are speculatively executed
- Efficient Runahead Execution [ISCA'05, IEEE Micro Top Picks'06]
- Ineffectiveness for pointer-intensive applications
  - Runahead cannot parallelize dependent L2 cache misses
  - Address-Value Delta (AVD) Prediction [MICRO'05]
- Irresolvable branch mispredictions in runahead mode
  - Cannot recover from a mispredicted L2-miss dependent branch
  - Wrong Path Events [MICRO'04]

#### The Efficiency Problem



#### Causes of Inefficiency

Short runahead periods

Overlapping runahead periods

Useless runahead periods

 Mutlu et al., "Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance," ISCA 2005, IEEE Micro Top Picks 2006.

## Overall Impact on Executed Instructions



#### The Problem: Dependent Cache Misses

Runahead: Load 2 is dependent on Load 1



- Runahead execution cannot parallelize dependent misses
  - wasted opportunity to improve performance
  - wasted energy (useless pre-execution)
- Runahead performance would improve by 25% if this limitation were ideally overcome

#### Parallelizing Dependent Cache Misses

 Idea: Enable the parallelization of dependent L2 cache misses in runahead mode with a low-cost mechanism

- How: Predict the values of L2-miss address (pointer) loads
  - Address load: loads an address into its destination register, which is later used to calculate the address of another load
  - as opposed to data load
- Read:
  - Mutlu et al., "Address-Value Delta (AVD) Prediction," MICRO 2005.

#### Parallelizing Dependent Cache Misses



# Readings

- Required
  - Mutlu et al., "Runahead Execution", HPCA 2003.

#### Recommended

- Mutlu et al., "Efficient Runahead Execution: Power-Efficient Memory Latency Tolerance," ISCA 2005, IEEE Micro Top Picks 2006.
- Mutlu et al., "Address-Value Delta (AVD) Prediction," MICRO 2005.
- □ Armstrong et al., "Wrong Path Events," MICRO 2004.

Multiprocessors and Issues in Multiprocessing

# Readings for Today

- Required
  - Hill, Jouppi, Sohi, "Multiprocessors and Multicomputers," pp. 551-560 in Readings in Computer Architecture.
  - Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.
  - Hill, Jouppi, Sohi, "Dataflow and Multithreading," pp. 309-314 in Readings in Computer Architecture.
- Recommended
  - Historical: Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

## Readings for Wednesday

- Required cache coherence readings:
  - Culler and Singh, *Parallel Computer Architecture* 
    - Chapter 5.1 (pp 269 283), Chapter 5.3 (pp 291 305)
  - □ P&H, Computer Organization and Design
    - Chapter 5.8 (pp 534 538 in 4<sup>th</sup> and 4<sup>th</sup> revised eds.)
- Recommended:
  - Lamport, "How to Make a Multiprocessor Computer That Correctly Executes Multiprocess Programs," IEEE Transactions on Computers, 1979
  - Papamarcos and Patel, "A low-overhead coherence solution for multiprocessors with private cache memories," ISCA 1984.

#### Remember: Flynn's Taxonomy of Computers

- Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966
- SISD: Single instruction operates on single data element
- **SIMD:** Single instruction operates on multiple data elements
  - Array processor
  - Vector processor
- MISD: Multiple instructions operate on single data element
  - Closest form: systolic array processor, streaming processor
- MIMD: Multiple instructions operate on multiple data elements (multiple instruction streams)
  - Multiprocessor
  - Multithreaded processor

## Why Parallel Computers?

- Parallelism: Doing multiple things at a time
- Things: instructions, operations, tasks
- Main Goal
  - Improve performance (Execution time or task throughput)
    - Execution time of a program governed by Amdahl's Law

#### Other Goals

- Reduce power consumption
  - (4N units at freq F/4) consume less power than (N units at freq F)
  - Why?
- Improve cost efficiency and scalability, reduce complexity
  - Harder to design a single unit that performs as well as N simpler units
- Improve dependability: Redundant execution in space

# Types of Parallelism and How to Exploit

# Them Instruction Level Parallelism

- Different instructions within a stream can be executed in parallel
- Pipelining, out-of-order execution, speculative execution, VLIW
- Dataflow
- Data Parallelism
  - Different pieces of data can be operated on in parallel
  - SIMD: Vector processing, array processing
  - Systolic arrays, streaming processors
- Task Level Parallelism
  - Different "tasks/threads" can be executed in parallel
  - Multithreading
  - Multiprocessing (multi-core)

# Task-Level Parallelism: Creating Tasks

- Partition a single problem into multiple related tasks (threads)
  - Explicitly: Parallel programming
    - Easy when tasks are natural in the problem
      - Web/database queries
    - Difficult when natural task boundaries are unclear
  - Transparently/implicitly: Thread level speculation
    - Partition a single thread speculatively
- Run many independent tasks (processes) together
  - Easy when there are many processes
    - Batch simulations, different users, cloud computing workloads
  - Does not improve the performance of a single task
# MIMD Processing Overview

# MIMD Processing

- Loosely coupled multiprocessors
  - No shared global memory address space
  - Multicomputer network
    - Network-based multiprocessors
  - Usually programmed via message passing
    - Explicit calls (send, receive) for communication
- Tightly coupled multiprocessors
  - Shared global memory address space
  - Traditional multiprocessing: symmetric multiprocessing (SMP)
  - Existing multi-core processors, multithreaded processors
  - Programming model similar to uniprocessors (i.e., multitasking uniprocessor) except
    - Operations on shared data require synchronization

### Main Issues in Tightly-Coupled MP

- Shared memory synchronization
  - Locks, atomic operations
- Cache consistency
  - More commonly called cache coherence
- Ordering of memory operations
  - □ What should the programmer expect the hardware to provide?
- Resource sharing, contention, partitioning
- Communication: Interconnection networks
- Load imbalance

## Aside: Hardware-based Multithreading

### Coarse grained

- Quantum based
- Event based (switch-on-event multithreading)
- Fine grained
  - Cycle by cycle
  - □ Thornton, "CDC 6600: Design of a Computer," 1970.
  - Burton Smith, "A pipelined, shared resource MIMD computer," ICPP 1978.

### Simultaneous

- Can dispatch instructions from multiple threads at the same time
- Good for improving execution unit utilization

### Parallel Speedup Example

- $a4x^4 + a3x^3 + a2x^2 + a1x + a0$
- Assume each operation 1 cycle, no communication cost, each op can be executed in a different processor
- How fast is this with a single processor?
- How fast is this with 3 processors?

 $R = a_4 x^4 + a_5 x^3 + a_2 x^2 + a_1 x + a_0$ Single processor: 11 operations ( date flow graph ) α. \* 02 X 03 20 q4 \* 03X3 agxu Ozxi \* a,X' Q4X4+03X3 ao T1 = 11 cycles

42



### Speedup with 3 Processors

T3 = 5 cycles 11 Speedup with 3 processors = 22 T1 Is this a four composison?

Revisiting the Single-Processor Algorithm  
Revisit 
$$T_1$$
  
Better single-processor algorithm:  
 $R = a_4 x^4 + a_5 x^3 + a_2 x^2 + a_1 x + a_0$   
 $R = (((a_4 x + a_3) x + a_2) x + a_1) x + a_0$   
(Homer's method)

Horner, "A new method of solving numerical equations of all orders, by continuous approximation," Philosophical Transactions of the Royal Society, 1819.

X 04 to a3 \* a, T1 = 8 cycles X  $\frac{Speedup}{with} = \frac{T_1^{best}}{T_3^{best}} = \frac{8}{5} = \frac{1.6}{5}$ 3 proves. a 30 (not 2.2)

## Superlinear Speedup

- Can speedup be greater than P with P processing elements?
- Cache effects
- Working set effects
- Happens in two ways:
  - Unfair comparisons
  - Memory effects



### Utilization, Redundancy, Efficiency

- Traditional metrics
  - □ Assume all P processors are tied up for parallel computation
- Utilization: How much processing capability is used
   U = (# Operations in parallel version) / (processors x Time)
- Redundancy: how much extra work is done with parallel processing
  - R = (# of operations in parallel version) / (# operations in best single processor algorithm version)

### Efficiency

- E = (Time with 1 processor) / (processors x Time with P processors)
- □ E = U/R

### Utilization of Multiprocessor

Multiprocessor metrics. Uthizoton : How much processing apololity we use 10 operations (in publication) Tp X X 3 processors x 5 time units 15 PX Tp ()=

Redundary: How much extra work due to multipreasing best 10 R = Ops was proved. Ops with 1 proc. best R is always > 1 Efficiency How much resource we use compared to how much resurce we can get away with 1. Tobest (tyms up 1 proces Typ true units) (typing up p pre for Tp tme units) P. Tpest E=U

### Caveats of Parallelism (I)

Speedup ' Superlaneer restur Imeor spedup reality P(# of processors) (domashing rehms) Why the really? x. [1 (1-a). T1 To 3 non-parallelizeble par purallelizable purt/fractions of the single-processor program

### Amdahl's Law

Speedup p proc. To Speedup as p->00 butneck for probled Speed

Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.

### Amdahl's Law Implication 1



### Amdahl's Law Implication 2



### Caveats of Parallelism (II)

- Amdahl's Law
  - □ f: Parallelizable fraction of a program
  - N: Number of processors



- Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.
- Maximum speedup limited by serial portion: Serial bottleneck
- Parallel portion is usually not perfectly parallel
  - Synchronization overhead (e.g., updates to shared data)
  - Load imbalance overhead (imperfect parallelization)
  - Resource sharing overhead (contention among N processors)

# Midterm II Results

### Midterm II Scores

- Average: 148 / 315
- Minimum: 58 / 315
- Maximum: 258 / 315
- Std. Dev.: 52



#### **Midterm 2 Distribution**

## Midterm II Per-Question Statistics (I)







## Midterm II Per-Question Statistics (II)



We did not cover the following slides in lecture. These are for your preparation for the next lecture.

### Sequential Bottleneck



## Why the Sequential Bottleneck?



- Parallel machines have the sequential bottleneck
- Main cause: Non-parallelizable operations on data (e.g. nonparallelizable loops) for (i = 0; i < N; i++) A[i] = (A[i] + A[i-1]) / 2
- Single thread prepares data and spawns parallel tasks (usually sequential)

### Another Example of Sequential Bottleneck

InitPriorityQueue(PQ); SpawnThreads();	<b>LEGEND</b> A,E: Amdahl's serial part B: Parallel Portion C1,C2: Critical Sections D: Outside critical section
ForEach Thread:	
while (problem not solved)	
Solve(SubProblem); If(problem solved) break NewSubProblems = Par	; <b>D1 B</b> tition(SubProblem);
Lock(X) PQ.insert(NewSubProblem Unlock(X)	
PrintSolution(); E	

### Bottlenecks in Parallel Portion

- Synchronization: Operations manipulating shared data cannot be parallelized
  - Locks, mutual exclusion, barrier synchronization
  - Communication: Tasks may need values from each other
  - Causes thread serialization when shared data is contended

Load Imbalance: Parallel tasks may have different lengths

- Due to imperfect parallelization or microarchitectural effects
- Reduces speedup in parallel portion
- Resource Contention: Parallel tasks can share hardware resources, delaying each other
  - Replicating all resources (e.g., memory) expensive
  - Additional latency not present when each task runs alone

## Difficulty in Parallel Programming

- Little difficulty if parallelism is natural
  - "Embarrassingly parallel" applications
  - Multimedia, physical simulation, graphics
  - Large web servers, databases?
- Difficulty is in
  - Getting parallel programs to work correctly
  - Optimizing performance in the presence of bottlenecks
- Much of parallel computer architecture is about
  - Designing machines that overcome the sequential and parallel bottlenecks to achieve higher performance and efficiency
  - Making programmer's job easier in writing correct and highperformance parallel programs

# Cache Coherence

### Cache Coherence

Basic question: If multiple processors cache the same block, how do they ensure they all see a consistent state?











## Cache Coherence: Whose Responsibility?

### Software

- Can the programmer ensure coherence if caches are invisible to software?
- What if the ISA provided the following instruction?
  - FLUSH-LOCAL A: Flushes/invalidates the cache block containing address A from a processor's local cache
  - When does the programmer need to FLUSH-LOCAL an address?
- What if the ISA provided the following instruction?
  - FLUSH-GLOBAL A: Flushes/invalidates the cache block containing address A from all other processors' caches
  - When does the programmer need to FLUSH-GLOBAL an address?
- Hardware
  - Simplifies software's job
  - One idea: Invalidate all other copies of block A when a processor writes to it
## Snoopy Cache Coherence

- Caches "snoop" (observe) each other's write/read operations
- A simple protocol:



- Write-through, nowrite-allocate cache
- Actions: PrRd, PrWr, BusRd, BusWr