Compsci 221 / ECE 259
Advanced Computer Architecture II
(Parallel Computer Architecture)

Introduction

Copyright 2012 Alvin R. Lebeck
Duke University

Slides are derived from work by
Sarita Adve (Illinois), Babak Falsafi (CMU),
Mark Hill (Wisconsin), Steve Reinhardt (Michigan), J. P.
Singh (Princeton), and Dan Sorin (Duke). Thanks!

General Course Information

• Professor: Alvin R. Lebeck
  – alvy@cs.duke.edu
  – http://www.cs.duke.edu/~alvy

• Course info
  – http://www.cs.duke.edu/courses/cps221/spring12/
  – All lecture notes will be posted here before class
  – All readings and assignments are posted here
  – Piazza for questions/discussion

• Office hours
  – By appointment

• TA: Jun Pang
Course Objectives

- Learn about parallel computer architecture
- Learn how to read/evaluate research papers
- Learn how to perform research
- Learn how to present research

Course Guidelines

- Students are responsible for:
  - Leading discussions of research papers - 15% of grade
  - Participating in class discussions - 15% of grade
  - Paper summaries - 15% of grade
  - A few assignments - 15% of grade
  - Individual or group project - 40% of grade
- We will read a lot of papers!
  - Often two papers per class
- Everything in your own words (you can grab graphs not words)
Papers

• How to lead a paper discussion
  – 20-25 minutes of talking (need time for discussion)
  – Summarize the paper (motivation, problem, solution, evaluation)
  – Handle questions from class & ask questions of class
  – Explain what (you think) is good about the paper
  – Explain what (you think) is bad or lacking or confusing
  – The presenter must email his/her slides (PPT) to me by 11:59pm the night before leading a discussion

• Summaries
  – Must complete 75% of the summaries and at least one per class
  – Three paragraphs
    » Summarize motivation and main ideas in paper
    » Critique of ideas (good, bad, etc.)
    » 1-3 questions and/or observations for discussion in class
  – Submit via Sakai

Project

• The project is a semester-long assignment that should reflect the goal of being no more than "a stone's throw" away from a research paper.
  – Written proposal (no more than 3 pages), due Wed, Feb 28
  – Maybe: Oral presentation of proposal (in class), Wed, Feb 27
  – Written progress report (<= 3 pages), Mon, Mar 23
  – Final document in conference/journal format (<= 12 pages, 11pt, single spaced), Apr 27
  – Final presentation (in class) at end of semester

• Individual or group of 2

• Get started early! Talk to me about your project ideas.
Academic Misconduct

• I will not tolerate academically dishonest work. This includes cheating on the final exam and plagiarism on the project, summaries, or presentations.

• Be careful on the project to cite prior work and to give proper credit to others' research.

• Ask me if you have any questions. Not knowing the rules does not make misconduct OK.

Course Topics

• Parallel programming
• Machine organizations
• Scalable, non-coherent machines
• Cache-coherent shared memory machines
• Memory consistency models
• Interconnection networks
• Evaluation tools and methodology
• Availability
• Novel architectures (vectors, dataflow, grid, etc.)
• Interactions with microprocessors and I/O
• Impact of new technology
Outline for Intro to Multiprocessing

• Motivation & Applications

• Programming Models
  – Shared Memory, Message Passing, Data Parallel

• Issues in Programming Models
  – Function: naming, operations, & ordering
  – Performance: latency, bandwidth, etc.

Motivation

• Compsci 220 / ECE 252: mostly uniprocessors
• This course uses $N$ processors in a computer to
  – Increase Throughput via many jobs in parallel
  – Improve Cost-Effectiveness (e.g., adding 3 processors may yield 4X throughput for 2X system cost)
  – Reduce Latency for shrink-wrapped software (e.g., databases and web servers)
  – Reduce latency through Parallelization of your application (but this is hard)
  – Avoid Melting the chip

• Need more performance than today’s processor?
  – Wait for tomorrow’s processor (not anymore…)
  – Use many processors in parallel
Applications: Science and Engineering

• Examples
  – Weather prediction
  – Evolution of galaxies
  – Oil reservoir simulation
  – Automobile crash tests
  – Drug development
  – VLSI CAD
  – Nuclear bomb simulation

• Typically model physical systems or phenomena
• Problems are 2D or 3D
• Usually require “number crunching”
• Involve “true” parallelism

Applications: Commercial

• Examples
  – On-line transaction processing (OLTP)
  – Decision support systems (DSS)
  – “Application servers” or “middleware” (WebSphere)

• Involves data movement, not much number crunching
  – OTLP has many small queries
  – DSS has fewer but larger queries

• Involves throughput parallelism
  – Inter-query parallelism for OLTP
  – Intra-query parallelism for DSS
Applications: Multi-media/home

• **Examples**
  – Speech recognition
  – Audio/video
  – Data compression/decompression
  – 3D graphics

• **Involves everything (crunching, data movement, true parallelism, and throughput parallelism)**

Rise, Fall and Rise of MP Research

• **Where is MP research presented?**
  – ISCA = International Symposium on Computer Architecture
  – ASPLOS = Arch. Support for Programming Languages and OS
  – MICRO = International Symposium on Microarchitecture (!!!)
  – HPCA = High Performance Computer Architecture
  – SPAA = Symposium on Parallel Algorithms and Architecture
  – ICS = International Conference on Supercomputing
  – PACT = Parallel Architectures and Compilation Techniques
  – Etc.

• #of MP decreased for about a decade.
• Multicore => # of MP papers increasing.
Outline

- Motivation & Applications
- Programming Models & A Generic Parallel Machine
- Issues in Programming Models

In Theory

- Sequential
  - Time to sum $n$ numbers? $O(n)$
  - Time to sort $n$ numbers? $O(n \log n)$
  - What model? RAM

- Parallel
  - Time to sum? Tree for $O(\log n)$
  - Time to sort? Non-trivially $O(\log n)$
  - What model?
    » PRAM [Fortune, Willie STOC78]
    » $P$ processors in lock-step
    » One memory (e.g., CREW for concurrent read exclusive write)

2<->3
1<->2
3<->4
2<->3
1<->2
3<->4
But in Practice, How Do You

- Name a datum across processors?
- Communicate values?
- Coordinate and synchronize?
- Select processing node size (few-bit ALU to a PC)?
- Select number of nodes in system?

Programming Model

- Provides a communication abstraction that is a contract between hardware and software (a la ISA)
- Programming model != programming language

Current Programming Models
1) Shared Memory
2) Message Passing
3) Data Parallel (Shared Address Space)
4) (Dataflow)
Programming Model

- How does the programmer view the system?
  - Which is NOT the same as how the system actually behaves!!

- **Shared memory**: processors execute instructions and communicate by reading/writing a globally shared memory

- **Message passing**: processors execute instructions and communicate by explicitly sending messages

- **Data parallel**: processors do the same instructions at the same time, but on different data

Historical View

- Historically: system architecture and programming model were tied together

Join At: I/O (Network)  Memory  Processor

Program With: Message Passing  Shared Memory  Data Parallel
- Single-Instruction
- Multiple-Data (SIMD)
Historical View, cont.

• Architecture → Programming Model
  – Join at network → program with message passing model
  – Join at memory → program with shared memory model
  – Join at processor → program with SIMD or data parallel

• Programming Model → Architecture
  – Message-passing programs on message-passing arch
  – Shared-memory programs on shared-memory arch
  – SIMD/data-parallel programs on SIMD/data-parallel arch
  – Slippery slope that led to LISP machines …

• But
  – Isn’t hardware basically the same? Processors, memory, & I/O?
  – Realization that most programming models could be supported on any hardware

1990’s Parallel Computer Architecture

• Extension of traditional computer architecture to support communication and cooperation
  – Communications architecture

(C) 2012 Alvin R. Lebeck from Adve, Falsafi, Hill, Reinhardt, Singh, Sorin
CompSci 221 / ECE 259
21

(C) 2012 Alvin R. Lebeck from Adve, Falsafi, Hill, Reinhardt, Singh, Sorin
CompSci 221 / ECE 259
22
Today’s Parallel Computer Architecture

- Multicore
  - Processors on one chip (not nodes)
- Graphics Cards (ADM & nVidia)
- Big Clusters (of multicore processors)
- Special purpose (IBM Blue Gene, etc.)

Simple Problem

```
for i = 1 to N
    A[i] = (A[i] + B[i]) * C[i]
    sum = sum + A[i]
```

- How do I make this parallel?
Simple Problem

for $i = 1$ to $N$
    $A[i] = (A[i] + B[i]) \times C[i]$
    sum = sum + $A[i]$

- Split the loops
  - Independent iterations
    for $i = 1$ to $N$
        $A[i] = (A[i] + B[i]) \times C[i]$
    for $i = 1$ to $N$
        sum = sum + $A[i]$

- Data flow graph?

Data Flow Graph

2 + N-1 cycles to execute on N processors
But with what assumptions?
Shared Memory Architectures

- Communication, sharing, and synchronization with loads/stores on shared variables
- Must map virtual pages to physical page frames
- Consider OS support for good mapping
- Examples: most of the servers from Sun, IBM, Intel, Compaq/HP, etc.

Shared Portion of Address Space

Private Portion of Address Space

Machine Physical Address Space
Return of the Simple Problem (Shared Memory)

```java
private int i, my_start, my_end, mynode;
shared float A[N], B[N], C[N], sum;
for i = my_start to my_end
    A[i] = (A[i] + B[i]) * C[i]
GLOBAL_SYNCH;
if (mynode == 0)
    for i = 1 to N
        sum = sum + A[i]
```

- Can run this pseudocode on any machine that supports shared memory

Message Passing Programming Model

- User-level send/receive abstraction
  - local buffer (x,y), process (P,Q), and tag (t)
  - naming and synchronization
int i, my_start, my_end, mynode;
float A[N/P], B[N/P], C[N/P], sum;
for i = 1 to N/P
    A[i] = (A[i] + B[i]) * C[i]
    sum = sum + A[i]
if (mynode != 0)
    send (sum, 0);
if (mynode == 0)
    for i = 1 to P-1
        recv(tmp, i)
        sum = sum + tmp

• Send/Recv communicates and synchronizes
• P processors

Data Parallel

• Programming Model
  – Operations are performed on each element of a large (regular) data structure in a single step
  – Arithmetic, global data transfer
• Processor is logically associated with each data element
  – SIMD architecture: Single instruction, multiple data
• Early architectures mirrored programming model
  – Many bit-serial processors
  – Today most architectures have data parallel instrs (e.g., Intel has MMX/SSE, SPARC has VIS)
• Can support data parallel model on shared memory or message passing architecture
The Simple Problem Strikes Back

Assuming we have N processors

\[
A = (A + B) \times C \\
\text{sum} = \text{global\_sum} (A)
\]

- Language supports array assignment
- Special HW support for global operations
- CM-2 bit-serial
- CM-5 32-bit SPARC processors
  - Message Passing and Data Parallel models
  - Special control network

Aside -- Single Program, Multiple Data

- SPMD Programming Model
  - Each processor executes the same program on different data
  - Many benchmarks are in SPMD model

for each molecule at this processor {
  simulate interactions with myproc+1 and myproc-1;
}

- Not connected to SIMD architecture model
  - Not lockstep instructions
  - Could execute different instructions on different processors
    » Data dependent branches cause divergent paths
Execute Data Flow Graph
No control sequencing

Data Flow Architectures

- Explicitly represent data dependencies (dataflow graph)
- No artificial constraints, like sequencing instructions!
  - Early machines had no registers or cache
- Instructions can “fire” when operands are ready
  - Remember Tomasulo’s algorithm
- How do we know when operands are ready?
- Matching store
  - Large associative search!
- Later machines moved to coarser grain (threads)
  - Allowed registers and cache for local computation
  - Introduced messages (with operations and operands)
Review: Separation of Model and Architecture

• Shared Memory
  – Single shared address space
  – Communicate, synchronize using load / store
  – Can support message passing

• Message Passing
  – Send / Receive
  – Communication + synchronization
  – Can support shared memory (w/ software)

• Data Parallel
  – Lock-step execution on regular data structures
  – Often requires global operations (sum, max, min...)
  – Can support on either shared memory or message passing

• Dataflow

Outline

• Motivation & Applications

• Programming Models

• Issues in Programming Models
  – Function: naming, operations, & ordering
  – Performance: latency, bandwidth, etc.
Programming Model Design Issues

• **Naming:** How is communicated data and/or partner node referenced?
• **Operations:** What operations are allowed on named data?
• **Ordering:** How can producers and consumers of data coordinate their activities?

• **Performance**
  – **Latency:** How long does it take to communicate in a protected fashion?
  – **Bandwidth:** How much data can be communicated per second? How many operations per second?

Issue: Naming

• **Single Global Linear-Address-Space** (shared memory)
• **Multiple Local Address/Name Spaces** (message passing)

• **Naming strategy affects**
  – Programmer / Software
  – Performance
  – Design complexity
Issue: Operations

• Uniprocessor RISC
  – Ld/St and atomic operations on memory
  – Arithmetic on registers
• Shared Memory Multiprocessor
  – Ld/St and atomic operations on local/global memory
  – Arithmetic on registers
• Message Passing Multiprocessor
  – Send/receive on local memory
  – Arithmetic on registers
  – Broadcast
• Data Parallel
  – Ld/St
  – Global operations (add, max, etc.)

Issue: Ordering

• Uniprocessor
  – Programmer sees order as program order
  – Out-of-order execution (Tomasulo’s algorithm) actually changes order
  – Write buffers
  – Important to maintain true (RAW) dependencies

• Multiprocessor
  – What is order among several threads accessing shared data?
  – What affect does this have on performance?
  – What if implicit order is insufficient?
  – Memory consistency model specifies rules for ordering
Issue: Order/Synchronization

• Coordination mainly takes three forms:
  – Mutual exclusion (e.g., spin-locks)
  – Event notification
    » Point-to-point (e.g., producer-consumer)
    » Global (e.g., end of phase indication, all or subset of processes)
  – Global operations (e.g., sum)

• Issues:
  – Synchronization name space (entire address space or portion)
  – Granularity (per byte, per word, ... → overhead)
  – Low latency, low serialization (hot spots)
  – Variety of approaches
    » Test&set, compare&swap, LoadLocked-StoreConditional
    » Full/Empty bits and traps
    » Queue-based locks, fetch&op with combining

Performance Issue: Latency

• Must deal with latency when using fast processors

• Options:
  – Reduce frequency of long latency events
    » Algorithmic changes, computation and data distribution
  – Reduce latency
    » Cache shared data, network interface design, network design
  – Tolerate latency
    » Message passing overlaps computation with communication
      (program controlled)
    » Shared memory overlaps access completion and computation
      using consistency model and prefetching
### Performance Issue: Bandwidth

- **Private and global bandwidth requirements**

- **Private bandwidth requirements can be supported by:**
  - Distributing main memory among PEs
  - Application changes, local caches, memory system design

- **Global bandwidth requirements can be supported by:**
  - Scalable interconnection network technology
  - Distributed main memory and caches
  - Efficient network interfaces
  - Avoiding contention (hot spots) through application changes

### Cost of Communication

**Cost = Frequency x (Overhead + Latency + Xfer size/BW - Overlap)**

- **Frequency** = number of communications per unit of work
  - Algorithm, placement, replication, bulk data transfer
- **Overhead** = processor cycles spent initiating or handling communication
  - Protection checks, status, buffer mgmt, copies, events
- **Latency** = time to move bits from source to dest
  - Communication assist, topology, routing, congestion
- **Transfer time** = time through bottleneck
  - Comm assist, links, congestions
- **Overlap** = portion overlapped with useful work
  - Comm assist, comm operations, processor design
Summary

- Motivation & Applications
- Programming Models & A Generic Parallel Machine
- Issues in Programming Models