CS 207 D

Computer Architecture

Lecture 9: Multiprocessors
Challenges of Parallel Processing

- First challenge is % of program inherently sequential
- • Second challenge is long latency to remote memory
  
 1. Application parallelism primarily via new algorithms that have better parallel performance
  
 2. Long remote latency impact both by architect and by the programmer
Forcing Challenges of Parallel Processing

- Long remote latency impact

For example, reduce frequency of remote accesses either by

1) Caching shared data (HW)
2) Restructuring the data layout to make more accesses local (SW)

More Challenges:

- Parallel Programming
- Complex Algorithms
- Memory issues
- Communication Costs
- Load Balancing
Shared-Memory Architectures

- From multiple boards on a shared bus to multiple processors inside a single chip
- **Caches both**
- **Private data** are used by a single processor
- **Shared data** are used by multiple processors
- **Caching shared data**
  - reduces latency to shared data, memory bandwidth for shared data, and interconnect bandwidth
- **cache coherence problem**
Example of Cache Coherence Problem

P1
Cache
U: 5

P2
Cache
U: 5

P3
Cache
U: 5

Memory
U: 5
Amdahl’s Law

- Architecture design is very bottleneck-driven
  1. make the common case fast
  2. do not waste resources on a component that has little impact on overall performance/power

- **Amdahl’s Law:**
  - performance improvements through an enhancement is limited by the fraction of time the enhancement comes into play

- Execution Time After Improvement =
  - Execution Time Unaffected + **Execution Time Affected**
  - Amount of Improvement

- Speedup =
  - Performance after improvement
  - Performance before improvement
Amdahl’s Law

\[ s = \frac{1}{f + \frac{1-f}{p}} \leq \min(p, \frac{1}{f}) \]

\( f \) = fraction unaffected

\( p \) = speedup of the rest

\( s \) = speedup

Enhancement factor \( (p) \)

Speedup \( (s) \)

Graph shows the relationship between speedup and enhancement factor for different fractions.
Flynn Classification of parallel processing architectures

- Flynn classification of parallel processing architectures
  - SISD, SIMD, MISD, MIMD

Basic multiprocessor architectures

<table>
<thead>
<tr>
<th>Single data stream</th>
<th>Multiple data streams</th>
</tr>
</thead>
<tbody>
<tr>
<td>SISD Uniprocessors</td>
<td>SIMD Array or vector processors</td>
</tr>
<tr>
<td>MISD Rarely used</td>
<td>MIMD Multiproc’s or multicomputers</td>
</tr>
</tbody>
</table>

Node’s expansion

- Shared variables
  - GMSV Shared-memory multiprocessors
  - GMMP Rarely used
- Message passing
  - DMSV Distributed shared memory
  - DMMP Distributed-memory multicomputers

Flynn’s categories
Flynn’s Taxonomy
• Flynn classified by data and control streams in 1966

<table>
<thead>
<tr>
<th>Single Instruction Single Data (SISD) or (Uniprocessor)</th>
<th>Single Instruction Multiple Data SIMD (single PC: Vector, CM-2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple Instruction Single Data (MISD) or (ASIP)</td>
<td>Multiple Instruction Multiple Data MIMD (Clusters, SMP servers)</td>
</tr>
</tbody>
</table>

**Computer Architectures**

- SISD
- SIMD
- MIMD
- MISD
SISD

No instruction parallelism
No data parallelism

• SISD processing architecture example
  — a personal computer processing instructions and data on single processor
SIMD

- Multiple data streams in parallel with a single instruction stream
- At one time, one instruction operates on many data
  - Data parallel architecture
  - Vector architecture has similar characteristics, but achieve the parallelism with pipelining.

- **Array processors**
  - a graphic processor processing instructions for translation or rotation or other operations are done on multiple data
  - An array or matrix is also processed in SIMD
MISD

Multiple instruction streams in parallel operating on single Data stream

- Processing architecture example—processing for critical controls of missiles where single data stream processed on different processors to handle faults if any during processing
MIMD

- Multiple processing streams in parallel processing on parallel data streams
- MIMD processing architecture example is super computer or distributed computing
- systems with distributed or single shared memory
Flynn Classification

- **SISD**: traditional sequential architecture
- **SIMD**: processor arrays, vector processor
  - Parallel computing on a budget – reduced control unit cost
  - Many early supercomputers
- **MIMD**: most general purpose parallel computer today
  - Clusters, MPP, data centers
- **MISD**: not a general purpose architecture.
Johnson’s expansion

MIMD Architectures

Control parallelism: executing several instruction streams in parallel

- GMSV: Shared global memory – symmetric multiprocessors
- DMSV: Shared distributed memory – asymmetric multiprocessors
- DMMP: Message passing – multicomputers

Diagram with processors, memory modules, and network connections illustrating centralized and distributed memory.
classification of Parallel Processor Architectures

Processor Organizations

- Single Instruction, Single Data Stream (SISD)
  - Uniprocessor
    - Vector Processor
    - Array Processor

- Single Instruction, Multiple Data Stream (SIMD)
  - Multiple Instruction, Single Data Stream (MISD)
  - Multiple Instruction, Multiple Data Stream (MIMD)

- Multiple Instruction, Multiple Data Stream (MIMD)
  - Distributed Memory (loosely coupled)
  - Shared Memory (tightly coupled)

  - Clusters
    - Symmetric Multiprocessor (SMP)
    - Nonuniform Memory Access (NUMA)
A parallel computer is a collection of processing elements that cooperate to solve large problems

- Most important new element: it is all about communication!!

• What does the programmer (or OS or compiler writer) think about?

1) Models of computation
   • Sequential consistency?

2) Resource allocation
   • What mechanisms must be in hardware
     – A high performance processor (SIMD, or Vector Processor)

3) Data access, Communication, and Synchronization
A parallel computer is a collection of processing elements that cooperate and communicate to solve large problems fast.

- **Parallel Architecture = Computer Architecture + Communication Architecture**

  1. **Centralized Memory Multiprocessor**
     - <few dozen cores
     - Small enough to share single, centralized memory with **Uniform Memory Access** latency (UMA)

  2. **Physically Distributed-Memory Multiprocessor**
     - Larger number chips and cores than the first.
     - BW demands Memory distributed among processors with **Non-Uniform Memory Access**/latency (NUMA)
Multicore: Mainstream Multiprocessors

Multicore chips

- **IBM Power5**
  - Two 2+GHz PowerPC cores
  - Shared 1.5 MB L2, L3 tags
- **AMD Quad Phenom**
  - Four 2+ GHz cores
  - Per-core 512KB L2 cache
  - Shared 2MB L3 cache
- **Intel Core 2 Quad**
  - Four cores, shared 4 MB L2
  - Two 4MB L2 caches
- **Sun Niagara**
  - 8 cores, each 4-way threaded
  - Shared 2MB L2, shared FP
- For servers, not desktop

All have same shared memory programming model
Definitions of Threads and Processes

- Instruction stream divided into smaller streams (threads)
- Parallel execution
- Thread in multithreaded processors may or may not be the same as software threads

Process:
- An instance of program running on computer

Thread:
- Dispatchable unit of work within process

Thread switch
- Switching processor between threads within the same process
- Typically less costly than process switch

A thread of execution is the smallest sequence of programmed instructions that can be managed independently.
But First, Uniprocessor Concurrency

- **Software “thread”:**
  - Independent flow of execution
- **Context state:** PC, registers
- **Threads generally share the same memory space**
- “Process” like a thread, but different memory space
- Java has thread support built in, C/C++ supports P-threads library
- Generally, system software (the O.S.) manages threads
- “Thread scheduling”, “context switching”: **PC and registers switching**
- All threads share the one processor
- Quickly swapping threads gives illusion of concurrent execution

Thread is a component of a process. Multiple threads can exist within the same process but different processes do not share these resources.
Hardware Multithreading (TLP) Coarse- grained multithreading

- Single thread runs until a costly stall
  - E.g. 2nd level cache miss
- Another thread starts during stall for first
  - Pipeline fill time requires several cycles!
- Does not cover short stalls
- Less likely to slow execution of a single thread (smaller latency)
- Needs hardware support
  - PC and register file for each thread
  - little other hardware
Coarse Multithreading

Stalls for A and C would be longer than indicated in previous slide. Assume long stalls at end of each thread indicated in previous slide.
Hardware Multithreading
Fine-grained multithreading

- Two or more threads interleave instructions
  - Round-ribbon fashion
  - Skip stalled threads
- Needs hardware support
  - Separate PC and register file for each thread
  - Hardware to control alternating pattern
- Naturally hides delays
  - Data hazards, Cache misses
  - Pipeline runs with rare stalls
- Does not make full use of multi-issue architecture
Fine Multithreading
Hardware Multithreading
Simultaneous Multithreading (SMT)

- Hyper-threading by Intel
- Instructions from multiple threads issued on same cycle
- Uses register renaming and dynamic scheduling facility of multi-issue architecture
- Needs more hardware support
  - Register files, PC’s for each thread
  - Temporary result registers before commit
  - Support to sort out which threads get results from which instructions
- Maximizes utilization of execution units
Simultaneous Multithreading
Hardware Multithreading

Hardware Multithreading (MT)

- Multiple threads dynamically share a single pipeline (caches)
- Replicate thread contexts: PC and register file
- Coarse-grain MT: switch on L2 misses Why?
- Fine grain MT: Does not make full use of multi-issue architecture
- Simultaneous MT: no explicit switching, it converts TLP into ILP
- **MT does not improve single-thread performance**
### Multithreading Categories

<table>
<thead>
<tr>
<th>Pipes: 1 2 3 4</th>
<th>Many Cyc/thread Coarse-Grained</th>
<th>New Thread/cyc Fine-Grained</th>
<th>Separate Jobs Multiprocessing</th>
<th>FUs: 1 2 3 4 Simultaneous Multithreading</th>
</tr>
</thead>
<tbody>
<tr>
<td>16/48 = 33.3%</td>
<td>27/48 = 56.3%</td>
<td>27/48 = 56.3%</td>
<td>29/48 = 60.4%</td>
<td>42/48 = 87.5%</td>
</tr>
</tbody>
</table>

- **Thread 1**: Orange
- **Thread 2**: Orange
- **Thread 3**: Yellow
- **Thread 4**: Light Blue
- **Thread 5**: Pink
- **Idle slot**: Empty
1990s: FPGAs

Programmable logic arrays were developed in the 1970s. PLAs provided cost-effective and flexible replacements for random logic or ROM/PROM.

The related programmable array logic devices came later. PALs were less flexible than PLAs, but more cost-effective.
Basic Principles

- Systems provided of reconfigurable logic are often called Reconfigurable Instruction Set Processors (RISP). The reconfigurable logic includes a set of programmable processing units, which can be reconfigured in the field to implement logic operations or functions, and programmable interconnections between them.

FPGAs Are Everywhere

Applications are found in virtually all industry segments:
- Aerospace and defense
- Medical electronics
- Automotive control
- Software-defined radio
- Encoding and decoding
Why FPGA Represents a Paradigm Shift

Modern FPGAs can implement any functionality

Initially used only for prototyping

Even a complete CPU needs a small fraction of an FPGA’s resources

FPGAs come with multipliers and IP cores (CPUs/SPs)
Reconfiguration Steps

To execute a program taking advantage of the reconfigurable logic, use the following Steps.

1. **Code Analysis**: the first thing to do is to identify parts of the code that can be transformed for execution on the reconfigurable logic.

   - The goal of this step is to find the best trade off considering performance and available resources.
Basic steps in a reconfigurable system

1. Code is analyzed. Hot spots are found
2. Hot spots are transformed to reconfigurable instructions
3. A reconfigurable instruction is found. Reconfiguration begins and the logic is set.
4. Input operands are fetched
5. The configuration is executed
6. Results are written back

Source Code

Lecture 1: Introduction 35
Reconfiguration Steps

2- Code transformation: Once the best candidate parts of code to be accelerated are found.

3- Reconfiguration After code transformation, it is time to send to the reconfigurable system.

4. Input Context Loading: To perform a given reconfigurable operation, a set of input operands are fetched.

5. Execution: After the reconfigurable unit is set and the proper input operands are fetched.

6. Write back: The results of the reconfigurable operation are saved back to the register file.
Why is reconfigurable system interesting?

1. Some applications are poorly suited to microprocessor.
2. VLSI “explosion” provides increasing resources.
3. Hardware/Software co-design
4. Relatively new research area.
5. Relatively the same computation
What is concurrency?

- What is a sequential program?
  A single thread of control that executes one instruction and when it is finished execute the next logical instruction

- What is a concurrent program?
  A collection of autonomous sequential threads, executing (logically) in parallel

- The implementation (i.e. execution) of a collection of threads can be:
  1. **Multiprogramming** – Threads multiplex their executions on a single processor
  2. **Multiprocessing** – Threads multiplex their executions on a multiprocessor or a multicore system
  3. **Distributed Processing** – Processes multiplex their executions on several different machines
Why use Concurrent Programming?

1) Natural Application Structure: The world is not sequential! Easier to program multiple independent and concurrent activities.

2) Increased application throughput and responsiveness

3) Not blocking the entire application due to blocking IO

4) Performance from multiprocessor/multicore hardware

5) Parallel execution

6) Distributed systems

7) Single application on multiple machines

8) Client/server type or peer-to-peer systems
Comparing Uniprocessing to Multiprocessing

- A Uniprocessor (UP) system has one processor.
- A driver in a UP system executes one thread at any given time.
- A Multiprocessor (MP) system has two or more processors.
- A driver in this system can execute multiple threads concurrently.