Parallelism

Chapter 17

Advanced Topics
Topics

- Introduction
- Characterizations of Parallelism
- Microscopic Vs Macroscopic
- Symmetric Vs. Asymmetric
- Explicit V.s. Implicit
- Fine Grain V.s Coarse-grain
- Parallel Architectures
- SISD
- SIMD
- SISD
Topics

• Distributed and Cluster Computers
• Redundant Parallel Architectures
• Programming Explicit and Implicit Parallel Computers
• Locks and Mutual Exclusion
• Consequence for Programmers
• Performance of Multiprocessors
• Communication, Coordination, Contention
• Symmetric Multiprocessors
• MIMD

Dr. Rajesh Subramaniam, 2005
Introduction

• Previous parts covered
  − processors
  − memory
  − I/O

• Now
  − use of parallelism to increase speed
  − advanced topics

Parallel and pipelined architectures

Parallel hardware, taxonomy, limitations etc.
Characterizations of Parallelism

- Explicit vs. Implicit
- Fine grain vs. coarse-grain
- Symmetric vs. asymmetric
- Microscopic vs. macroscopic
- Parallelism

The grey region in between describes amount and type of parallelism and non-parallel are the extremes
Microscopic Vs Macroscopic

- Microscopic
  - Parallelism in a computer remains hidden inside subcomponents

- Macroscopic
  - Parallelism is a basic premise around which the system is designed
  - Parallel hardware within specific component
  - Subcomponents
  - Parallelism in a computer remains hidden inside

Microscopic V/S Macroscopic
Examples

• Microscopic
  - Arithmetic
    - ALU, registers, physical memory, parallel bus
  - Architecture
    - Multiple dissimilar processors

• Macroscopic
  - Use parallelism across multiple large-scale components of computer systems, multiple identical processors
Symmetric Vs. Asymmetric

• Symmetric
  - replication of identical elements
  - e.g. dual processors

• Asymmetric
  - multiple elements that function at the same time but differ from each other
  - e.g. dual processors
  - replication of identical elements
Fine Grain Vs Coarse-grain

• Fine Grain
  − Parallelism at the level of individual instructions or individual data elements
  − e.g., graphics processor that uses 16 hardware units to parallelize at the same time

• Coarse-grain
  − Parallelism on the level of programs or large blocks of data
  − e.g., dual processor, using a processor each to print and compose an email simultaneously

Fine Grain V/s Coarse-grain
Explicit Vs. Implicit

- Implicit
  - Hardware handles parallelism automatically without requiring a programmer to initiate or control parallel execution

- Explicit
  - Programmer must control each parallel unit

Explicit Vs. Implicit
Parallel Architectures

- Parallel architectures: parallelism of the central features around which the entire system is designed
- Flynn's classification
  - Single Instruction Single Data stream (SISD)
  - Single Instruction Multiple Data streams (SIMD)
  - Multiple Instruction Multiple Data streams (MIMD)

Terminology used to characterize computers according to the amount and type of parallelism. Note, hybrids that span multiple types also exist.
SISD

· Single Instruction Single Data stream
· Also called sequential or uniprocessor architecture
· Does not support macroscopic parallelism, can use parallelism internally
· Follows Von Neumann's architecture
· Also called sequential or uniprocessor architecture

SISD
SIMD

- Single Instruction streams Multiple Data streams
- Each instruction specifies a single operation applied to many data items simultaneously

Algorithm for vector normalization in a sequential computer:

\[
\text{for } i \text{ from } 1 \text{ to } N \{ \\
V[i] \leftarrow V[i] \times O;
\}
\]

Same in a SIMD computer

- Vector, array processors
- Graphic processors
- Data items simultaneously
- Each instruction specifies a single operation applied to many streams

SIMD
MIMD

- Multiple Instruction streams
- Multiple Data streams

Each processor performs independent computations at the same time:

- CDC peripheral processors
- I/O processors
- Math & graphics coprocessors
- Asymmetric multiprocessors (AMP)
- Symmetric multiprocessors (SMP)

Example of MIMD

Each processor performs independent computations at the same time.
Symmetric multiprocessor with $N$ identical processors. Each processor has access to memory and I/O devices.
Performance of Multiprocessors

\[
\text{Speedup} = \frac{\text{execution time on a single processor}}{\text{execution time on a single processor/execution}}
\]

Illustration of the ideal and typical performance of a single processor versus a multiprocessor as the number of processors increases.

Values on the y-axis list the relative speedup compared to a single processor.
Reasons why multiprocessor architectures have not fulfilled the promise of scalable, high performance computing.

- OS bottlenecks
- Memory contention
- I/O bottlenecks

Performance of Multiprocessors
Consequence for Programmers

- Locks and mutual exclusion
- Programming explicit and implicit parallel computers
- Programming symmetric and asymmetric multiprocessors
- Consequence for Programmers
Locks and Mutual Exclusion

• Shared variables in multiprocessors face the following problem.
  − What happens if two processors attempt to increment at the same time?
  − Variable is incremented only once instead of the correct two times.

• One solution: locks

  • Two times.
  • Variable is incremented only once instead of the correct two times.
  • What happens if two processors attempt to increment at the same time?
  • Problem.

  • Shared variables in multiprocessors face the following problems.
Locks and Mutual Exclusion

An example sequence of machine instructions used to increment a variable in memory. In most architectures, increment entails a load and store.

loadx, R5
incrR5
storeR5, x
access to lock at a time.

Only one copy of lock exists, only one processor can gain

Illustration of the instructions used to guarantee exclusive access
to a variable. A separate lock is assigned to each shared item.

lock 17
load x, R5
incr R5
store R5, x
release 17

Locks
Locks

- Problems
  - Programmers forget to apply locks to shared variables
  - Errors due to simultaneous access depend on timing and
  - Locking can reduce performance
  - Locking adds overhead
  - Problems
From a programmer's point of view, a system that uses explicit parallelism is significantly more complex to program than a system that uses implicit parallelism. From a programmer's point of view, a system that uses explicit parallelism is significantly more complex to program than a system that uses implicit parallelism.
Redundant Parallel Architectures

- Use of parallel hardware to improve reliability and prevent failure
- Multiple copies of a hardware unit that operate in parallel to perform an operation (same data and same operation on all units)

What happens if redundant copies of hardware disagree?
- Votesc
- Error message

Multiple copies of a hardware unit that operate in parallel to improve reliability and prevent failure
Distributed and Cluster Computers

- Tightly coupled: parallel hardware units are located inside the same computer system, e.g., distributed architectures, cluster computer
- Loosely coupled: multiple computer systems that are interconnected by a communication mechanism
- Tightly coupled: parallel hardware units are located inside the same computer system

Distributed and Cluster Computers
Distributed and Cluster Computers

- Distributed architecture
  - set of computers connected by a computer network or Internet

- Cluster computer
  - set of independent computers connected by a high-speed computer network that are all dedicated to solving one problem at a time

- Cluster architecture

Distributed and Cluster Computers
Parallelism is one of the fundamental optimization techniques used to increase hardware performance. Explicit parallelism gives a programmer control over the use of parallel facilities; implicit parallelism handles parallelism automatically. SIMD architecture allows an instruction to operate on an array of values.

Vector processors –

Graphic processors –

Summary

Dr. Rajesh Subramaniam, 2005
and cluster architectures

Alternatives to SIMD and MIMD are redundant, distributed, symmetric multiprocessors that operate simultaneously and can each execute a separate program. SIMD employs multiple, independent processors that can each execute a separate program. Speedup

- symmetric multiprocessors

- theoretically machine with N processors should perform N times as fast as a single processor

In practice, speedup does not increase linearly with number of processors due to memory contention and communication overhead. MIMD employs multiple, independent processors that operate simultaneously and can each execute a separate program.

Summary