Chapter 5: Thread-Level Parallelism – Part 1

1

What is a parallel or multiprocessor system?

Multiple processor units working together to solve the same problem

Key architectural issue: Communication model

2

Why parallel architectures?

Absolute performance

Technology and architecture trends

- Dennard scaling, ILP wall, Moore’s law

$\Rightarrow$ Multicore chips

Connect multicore together for even more parallelism

3

Performance Potential

Amdahl's Law is pessimistic
- Let s be the serial part
- Let p be the part that can be parallelized n ways

```
Serial:          SSPPPPPP
6 processors:    SSP
                   P
                   P
                   P
                   P
                   P
```

Speedup = 8/3 = 2.67

$T(n) = \frac{1}{s+p/n}$

As $n \rightarrow \infty$, $T(n) \rightarrow \frac{1}{s}$

Pessimistic

4

Performance Potential (Cont.)

Gustafson's Corollary

- Amdahl's law holds if run same problem size on larger machines
- But in practice, we run larger problems and "wait" the same time

5

Performance Potential (Cont.)

Gustafson's Corollary (Cont.)

- Assume for larger problem sizes
  - Serial time fixed (at s)
  - Parallel time proportional to problem size (truth more complicated)

```
Old Serial:     SSPPPPPP
6 processors:   SSPPPPPP
                  PPPPPP
                  PPPPPP
                  PPPPPP
                  PPPPPP
                  PPPPPP
Hypothetical Serial:
   SSPPPPPP PPPPPP PPPPPP PPPPPP PPPPPP PPPPPP
```

- Speedup = (8+5*6)/8 = 4.75
- $T'(n) = s + n*p$; $T'(\infty) \rightarrow \infty$!!!!

How does your algorithm "scale up"?

6

Flynn classification

Single-Instruction Single-Data (SISD)

Single-Instruction Multiple-Data (SIMD)

Multiple-Instruction Single-Data (MISD)

Multiple-Instruction Multiple-Data (MIMD)

7

Communication models

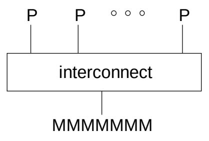
Shared-memory

Message passing

Data parallel

8

Communication Models: Shared-Memory



Each node a processor that runs a process

One shared memory

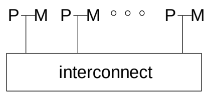
- Accessible by any processor
- The same address on two different processors refers to the same datum

Therefore, write and read memory to

- Store and recall data
- Communicate, Synchronize (coordinate)

9

Communication Models: Message Passing



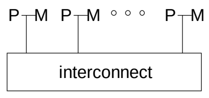
Each node a computer

- Processor – runs its own program (like SM)
- Memory – local to that node, unrelated to other memory

Add messages for internode communication, send and receive like mail

10

Communication Models: Data Parallel



Virtual processor per datum

Write sequential programs with "conceptual PC" and let parallelism be within the data (e.g., matrices)

C = A + B

Typically SIMD architecture, but MIMD can be as effective

11

Architectures

All mechanisms can usually be synthesized by all hardware

Key: which communication model does hardware support best?

Virtually all small-scale systems, multicores are shared-memory

12

Which is Best Communication Model to Support?

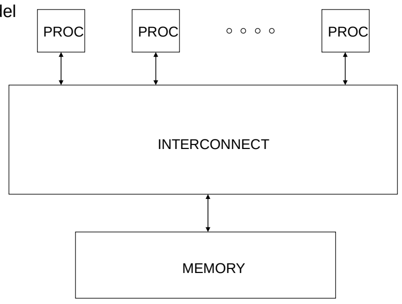
Shared-memory
- Used in small-scale systems
- Easier to program for dynamic data structures
- Lower overhead communication for small data
- Implicit movement of data with caching
- Hard to build?

Message-passing
- Communication explicit harder to program?
- Larger overheads in communication OS intervention?
- Easier to build?

13

Shared-Memory Architecture

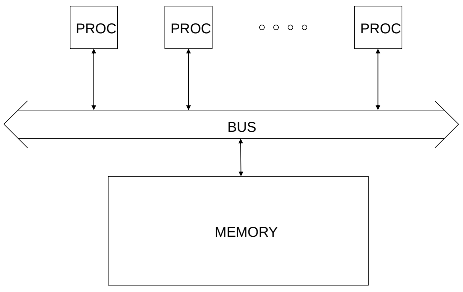
The model



For now, assume interconnect is a bus – *centralized architecture*

14

Centralized Shared-Memory Architecture



15

Centralized Shared-Memory Architecture (Cont.)

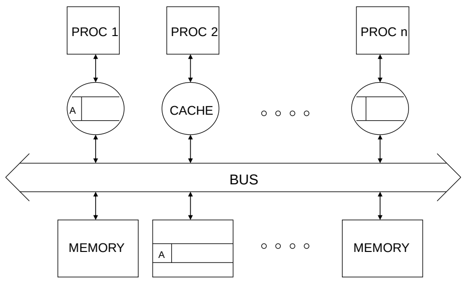
For higher bandwidth (throughput)

For lower latency

Problem?

16

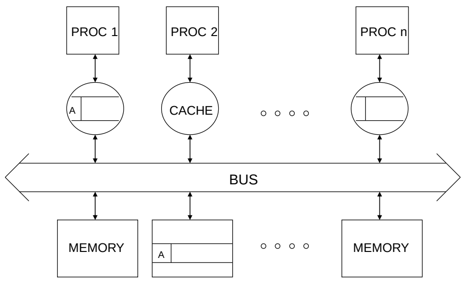
Cache Coherence Problem



17

Cache Coherence Solutions

Snooping

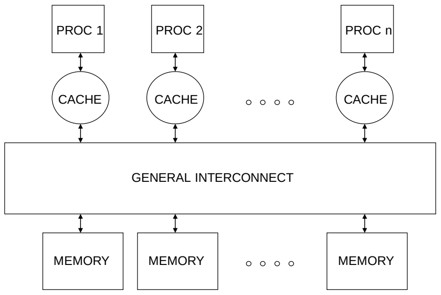


Problem with centralized architecture

18

Distributed Shared-Memory (DSM) Architecture

Use a higher bandwidth interconnection network



Uniform memory access architecture (UMA)

19

Distributed Shared-Memory (DSM) - Cont.

For lower latency: Non-Uniform Memory Access architecture (NUMA)

20

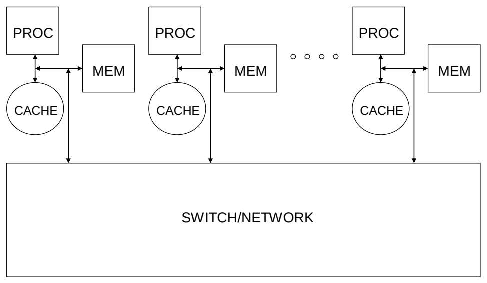
Non-Bus Interconnection Networks

Example interconnection networks

21

Distributed Shared-Memory - Coherence Problem

Directory scheme



Level of indirection!

22

Parallel Programming Example

Add two matrices: C = A + B

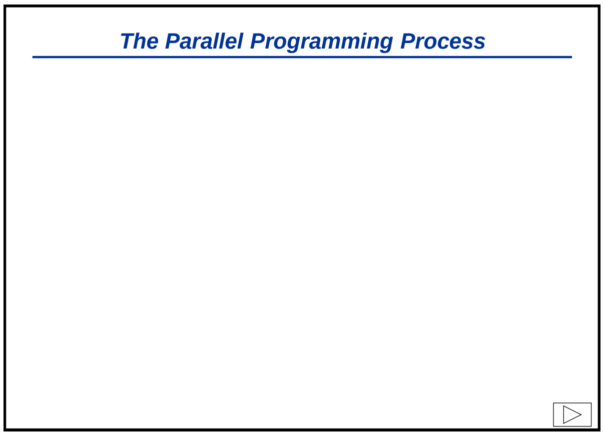
Sequential Program

```
main(argc, argv)
int argc; char *argv;
{
   Read(A);
   Read(B);
   for (i = 0; i ! N; i++)
        for (j = 0; j ! N; j++)
                C[i,j] = A[i,j] + B[i,j];
   Print(C);
}
```

23

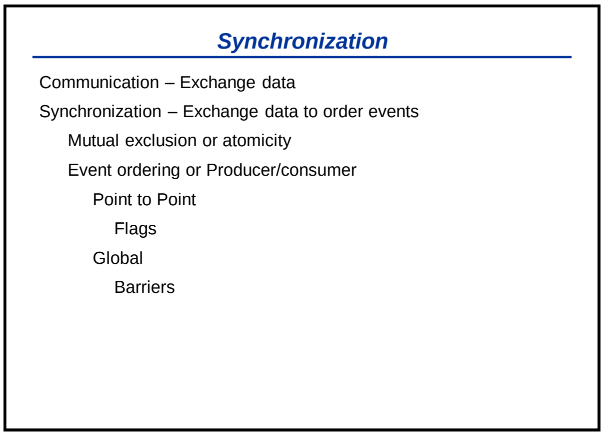
Parallel Program Example (Cont.)

24



The Parallel Programming Process

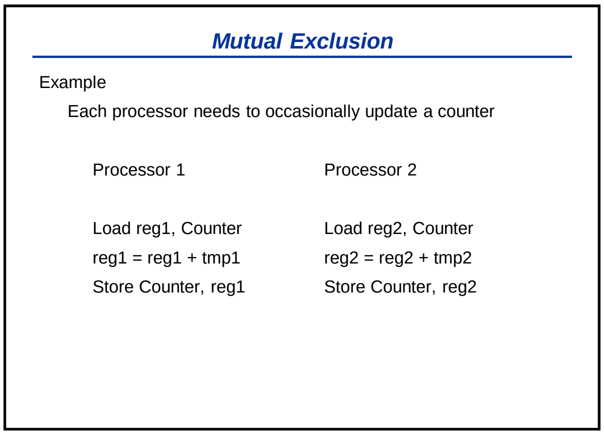
25



Synchronization

- Communication – Exchange data
- Synchronization – Exchange data to order events
  - Mutual exclusion or atomicity
  - Event ordering or Producer/consumer
    - Point to Point
      - Flags
    - Global
      - Barriers

26



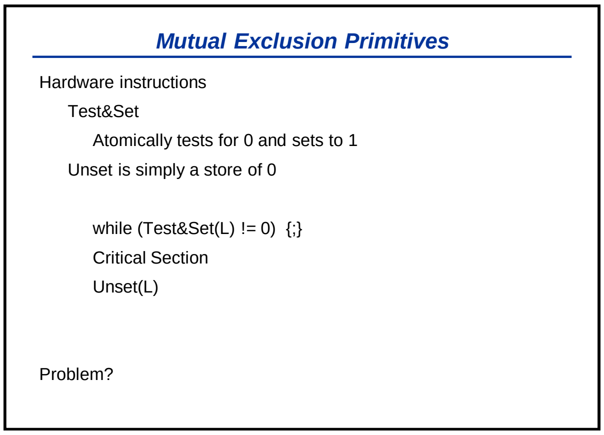
Mutual Exclusion

Example

Each processor needs to occasionally update a counter

Processor 1	Processor 2
Load reg1, Counter	Load reg2, Counter
reg1 = reg1 + tmp1	reg2 = reg2 + tmp2
Store Counter, reg1	Store Counter, reg2

27



Mutual Exclusion Primitives

- Hardware instructions
  - Test&Set
    - Atomically tests for 0 and sets to 1
  - Unset is simply a store of 0

```
while (Test&Set(L) != 0)  {;}
Critical Section
Unset(L)
```

Problem?

28

Mutual Exclusion Primitives – Alternative?

Test&Test&Set

29

Mutual Exclusion Primitives – Fetch&Add

```
Fetch&Add(var, data)
    { /* atomic action */
    temp = var
    var = temp + data
    }
    return temp
E.g., let X = 57
    P1: a = Fetch&Add(X,3)
    P2: b = Fetch&Add(X,5)
        If P1 before P2, ?
        If P2 before P1, ?
        If P1, P2 concurrent ?
```

30

Point to Point Event Ordering

Example

Producer wants to indicate to consumer that data is ready

```
Processor 1        Processor 2
A[1] = …           … = A[1]
A[2] = …           … = A[2]
  .                   .
  .                   .
A[n] = …           … = A[n]
```

31

Global Event Ordering – Barriers

Example

- All processors produce some data
- Want to tell all processors that it is ready
- In next phase, all processors consume data produced previously

Use barriers

32