

Shared memory multiprocessor system

Any memory location can be accessible by any of the processors.

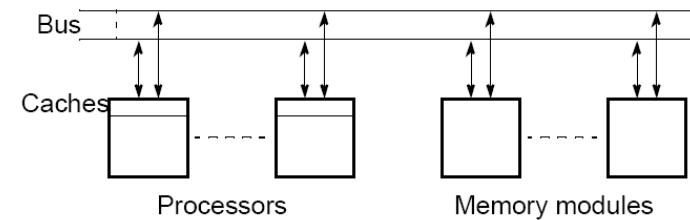
A *single address space* exists, meaning that each memory location is given a unique address within a single range of addresses.

Generally, shared memory programming more convenient although it does require access to shared data to be controlled by the programmer (using critical sections etc.)

1

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

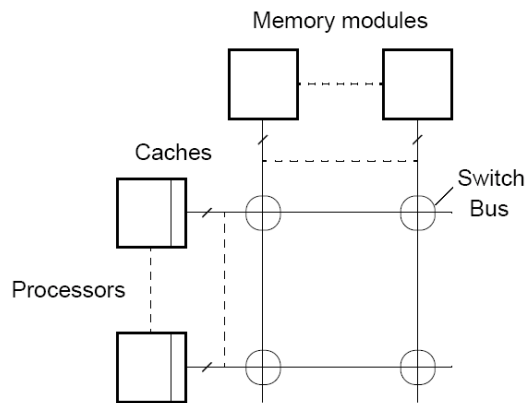
Shared memory multiprocessor using a single bus



2

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Shared memory multiprocessor using a crossbar switch



3

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Alternatives for Programming Shared Memory Multiprocessors:

- Using heavy weight processes.
- Using threads. Example Pthreads
- Using a completely new programming language for parallel programming - not popular. Example Ada.
- Using library routines with an existing sequential programming language.
- Modifying the syntax of an existing sequential programming language to create a parallel programming language. Example UPC
- Using an existing sequential programming language supplemented with compiler directives for specifying parallelism. Example OpenMP

4

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Language Constructs for Parallelism Shared Data

Shared memory variables might be declared as shared with, say,

```
shared int x;
```

5

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

par Construct

For specifying concurrent statements:

```
par {  
    S1;  
    S2;  
    .  
    .  
    Sn;  
}
```

6

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

forall Construct

To start multiple similar processes together:

```
forall (i = 0; i < n; i++) {  
    S1;  
    S2;  
    .  
    .  
    Sm;  
}
```

which generates n processes each consisting of the statements forming the body of the for loop, S1, S2, ..., Sm. Each process uses a different value of i .

7

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Example

```
forall (i = 0; i < 5; i++)  
    a[i] = 0;
```

clears **a[0]**, **a[1]**, **a[2]**, **a[3]**, and **a[4]** to zero concurrently.

8

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Dependency Analysis

To identify which processes could be executed together.

Example

Can see immediately in the code

```
forall (i = 0; i < 5; i++)  
  a[i] = 0;
```

that every instance of the body is independent of other instances and all instances can be executed simultaneously.

However, it may not be that obvious. Need algorithmic way of recognizing dependencies, for a *parallelizing compiler*.

9

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Bernstein's Conditions

Set of conditions sufficient to determine whether two processes can be executed simultaneously. Given:

I_i is the set of memory locations read (input) by process P_i .

O_j is the set of memory locations written (output) by process P_j .

For two processes P_1 and P_2 to be executed simultaneously, inputs to process P_1 must not be part of outputs of P_2 , and inputs of P_2 must not be part of outputs of P_1 ; i.e.,

$$I_1 \cap O_2 = \phi$$

$$I_2 \cap O_1 = \phi$$

where ϕ is an empty set. Set of outputs of each process must also be different; i.e.,

$$O_1 \cap O_2 = \phi$$

If the three conditions are all satisfied, the two processes can be executed concurrently.

10

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Example

Suppose the two statements are (in C)

```
a = x + y;  
b = x + z;
```

We have

```
I1 = (x, y)      O1 = (a)  
I2 = (x, z)      O2 = (b)
```

and the conditions

```
I1 ∩ O2 = φ  
I2 ∩ O1 = φ  
O1 ∩ O2 = φ
```

are satisfied. Hence, the statements $a = x + y$ and $b = x + z$ can be executed simultaneously.

11

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Shared Data in Systems with Caches

All modern computer systems have cache memory, high-speed memory closely attached to each processor for holding recently referenced data and code.

Cache coherence protocols

Update policy - copies of data in all caches are updated at the time one copy is altered.

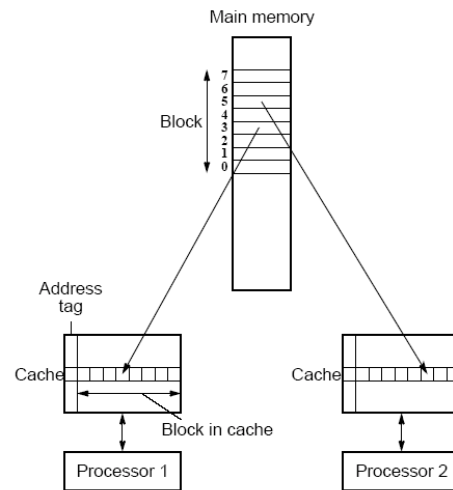
Invalidate policy - when one copy of data is altered, the same data in any other cache is invalidated (by resetting a valid bit in the cache). These copies are only updated when the associated processor makes reference for it.

12

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

False Sharing

Different parts of block required by different processors but not same bytes. If one processor writes to one part of the block, copies of the complete block in other caches must be updated or invalidated though the actual data is not shared.



13

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.

Solution for False Sharing

Compiler to alter the layout of the data stored in the main memory, separating data only altered by one processor into different blocks.

14

Slides for Parallel Programming Techniques & Applications Using Networked Workstations & Parallel Computers 2nd ed., by B. Wilkinson & M. Allen, © 2004 Pearson Education Inc. All rights reserved.