Chapter 12: Multiprocessor Architectures

Lesson 12:

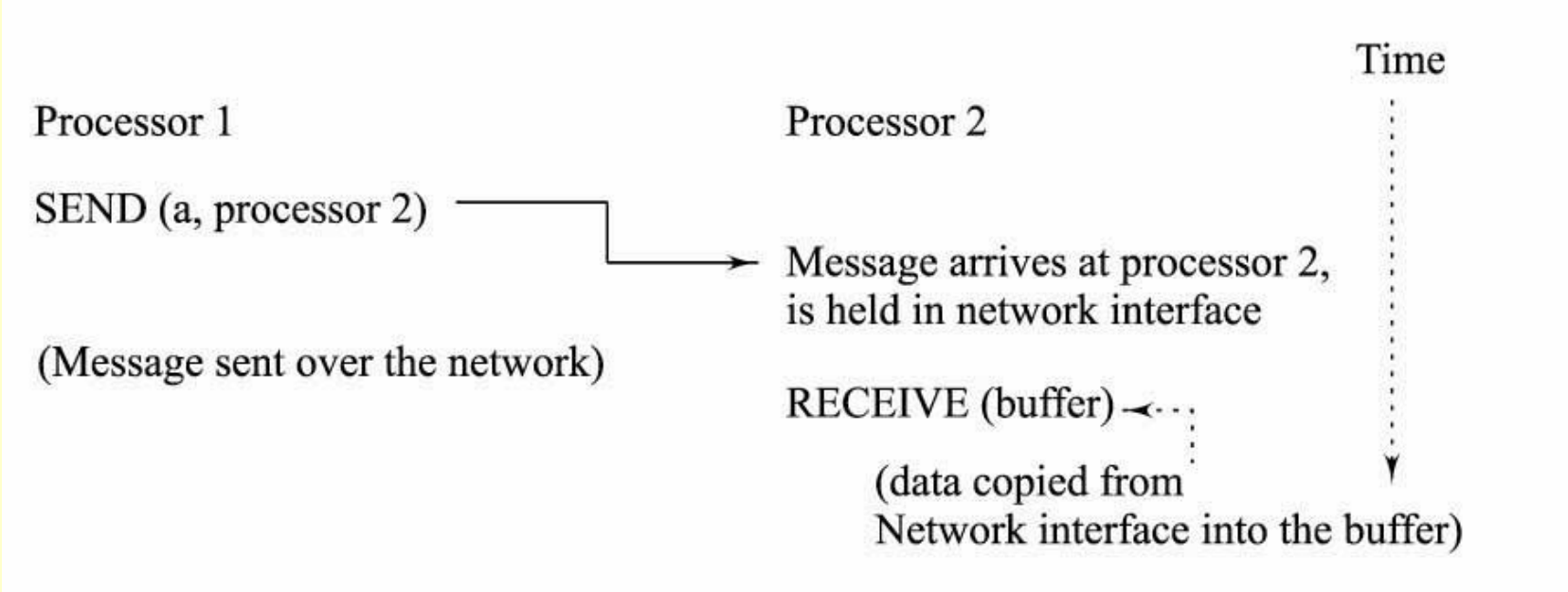
Message passing Systems and Comparison of Message Passing and Sharing

Objective

- Understand the message routing schemes and the difference between shared-memory and message-passing systems, and be able to compare these two approaches

Message passing procedure

Message passing procedure



SEND operation in Message Passing system

- To send a message, a processor executes an explicit SEND (*data*, *destination*) operation
- Instructs the hardware to send the specified data to the destination processor

RECEIVE operation in Message Passing system

- The destination processor executes a RECEIVE (*buffer*) operation
- Copies the sent data into the specified buffer
- Makes it available for use

Enforcing the order of SEND and RECEIVE operations

- If the processor responsible for sending the data has not executed its SEND operation before the RECEIVE is executed, the RECEIVE operation waits for the SEND to complete

Message Passing system

- Each processor has its own address space
- Processors cannot read or write data contained in another processor's address space

Message Passing system implemented in distributed-memory machines

- The latency benefits of associating a memory with each processor without the complexity of having to allow processors to access each other's memories

Comparing Message-Passing and Shared Memory

Most significant advantage of shared-memory systems

- That the computer handles communication, making it possible to write a parallel program without considering when data must be communicated from one processor to another

Most significant advantage of shared-memory systems

- However, to achieve good performance, the programmer must consider how data is used by the processors to minimize inter-processor communication (requests, invalidations, and updates)
- Particularly true on shared-memory systems with distributed memories, because the programmer must also think about which processor's memory should have the main copy of a piece of data

Disadvantage of shared-memory systems

- The programmer's ability to control inter-processor communication is limited
- All communication is handled by the system

Disadvantage of shared-memory systems

- On many systems, transferring a large block of data between processors more efficient if done as one communication, which is not possible on shared-memory systems, since the hardware controls the amount of data transferred at one time

Disadvantage of shared-memory systems

- The system also controls when communication happens, making it difficult to send data to a processor that will need it later in order to keep the processor from having to request the data and then wait for it

Message-passing systems

- Can achieve greater efficiency than shared-memory systems by allowing the programmer to control inter-processor communication, but this comes at the cost of requiring that the programmer explicitly specify all communication in the program

Message-passing systems

- Controlling communication can improve efficiency by allowing data to be communicated at the time it becomes available, instead of when it is needed, and by matching the amount of data communicated in each message to the needs of the application instead of the line size of the system that the application runs on

Message-passing systems

- Require fewer synchronization operations than shared-memory systems, because SEND and RECEIVE operations provide a great deal of the required synchronization by themselves

Message-passing systems

- In general, very attractive for many scientific computations
- The regular structure of these applications makes it easier to determine what data must be communicated between processors

Message-passing systems

- Very attractive for many scientific computations, because the regular structure of these applications makes it easier to determine what data must be communicated between processors

Shared-memory systems

- More attractive for irregular applications, because of the difficulty of determining what communication is required at the time the program is written

Supporting Both systems

Support for both message passing and shared-memory on the same system

- Allows programmers to choose the programming model that best suits their application
- It also allows *incremental parallelization*, in which a program is first written in a shared-memory style to reduce implementation time

Incremental parallelization

- Once the program is working correctly, time-critical communications are rewritten in message passing to optimize performance, allowing the programmer to trade off additional implementation effort against improved performance at a fine grain

Summary

We Learnt

- Message passing system procedure
- Comparison of shared and message passing systems
- Shared memory system more attractive for irregular applications
- Message passing system more attractive for regular applications, such as scientific computations
- Support to both systems

End of Lesson 12 on

Message passing Systems and Comparison of Message Passing and Sharing