# ADVANCED COMPUTER ARCHITECTURE AND PARALLEL PROCESSING

Hesham El-Rewini
Southern Methodist University

Mostafa Abd-El-Barr
Kuwait University


A JOHN WILEY & SONS, INC PUBLICATION

# ADVANCED COMPUTER ARCHITECTURE AND PARALLEL PROCESSING

*WILEY SERIES ON PARALLEL AND DISTRIBUTED COMPUTING*
SERIES EDITOR: Albert Y. Zomaya

**Parallel & Distributed Simulation Systems** / Richard Fujimoto

**Surviving the Design of Microprocessor and Multimicroprocessor Systems: Lessons Learned** / Veljko Milutinovic

**Mobile Processing in Distributed and Open Environments** / Peter Sapaty

**Introduction to Parallel Algorithms** / C. Xavier and S.S. Iyengar

**Solutions to Parallel and Distributed Computing Problems: Lessons from Biological Sciences** / Albert Y. Zomaya, Fikret Ercal, and Stephan Olariu (Editors)

**New Parallel Algorithms for Direct Solution of Linear Equations** / C. Siva Ram Murthy, K.N. Balasubramanya Murthy, and Srinivas Aluru

**Practical PRAM Programming** / Joerg Keller, Christoph Kessler, and Jesper Larsson Traeff

**Computational Collective Intelligence** / Tadeusz M. Szuba

**Parallel & Distributed Computing: A Survey of Models, Paradigms, and Approaches** / Claudia Leopold

**Fundamentals of Distributed Object Systems: A CORBA Perspective** / Zahir Tari and Omran Bukhres

**Pipelined Processor Farms: Structured Design for Embedded Parallel Systems** / Martin Fleury and Andrew Downton

**Handbook of Wireless Networks and Mobile Computing** / Ivan Stojmenoviic (Editor)

**Internet-Based Workflow Management: Toward a Semantic Web** / Dan C. Marinescu

**Parallel Computing on Heterogeneous Networks** / Alexey L. Lastovetsky

**Tools and Environments for Parallel and Distributed Computing Tools** / Salim Hariri and Manish Parashar

**Distributed Computing: Fundamentals, Simulations and Advanced Topics, Second Edition** / Hagit Attiya and Jennifer Welch

**Smart Environments: Technology, Protocols and Applications** / Diane J. Cook and Sajal K. Das (Editors)

**Fundamentals of Computer Organization and Architecture** / Mostafa Abd-El-Barr and Hesham El-Rewini

**Advanced Computer Architecture and Parallel Processing** / Hesham El-Rewini and Mostafa Abd-El-Barr

# ADVANCED COMPUTER ARCHITECTURE AND PARALLEL PROCESSING

Hesham El-Rewini
Southern Methodist University

Mostafa Abd-El-Barr
Kuwait University


A JOHN WILEY & SONS, INC PUBLICATION

This book is printed on acid-free paper. ♾



Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.



***Library of Congress Cataloging-in-Publication Data is available***

ISBN 0-471-46740-5

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

*To the memory of Abdel Wahab Motawe, who wiped away the tears of many people and cheered them up even when he was in immense pain. His inspiration and impact on my life and the lives of many others was enormous.*
*—Hesham El-Rewini*

*To my family members (Ebtesam, Muhammad, Abd-El-Rahman, Ibrahim, and Mai) for their support and love*
*—Mostafa Abd-El-Barr*

## CONTENTS

# PREFACE

Single processor supercomputers have achieved great speeds and have been pushing hardware technology to the physical limit of chip manufacturing. But soon this trend will come to an end, because there are physical and architectural bounds, which limit the computational power that can be achieved with a single processor system. In this book, we study advanced computer architectures that utilize parallelism via multiple processing units. While parallel computing, in the form of internally linked processors, was the main form of parallelism, advances in computer networks has created a new type of parallelism in the form of networked autonomous computers. Instead of putting everything in a single box and tightly couple processors to memory, the Internet achieved a kind of parallelism by loosely connecting everything outside of the box. To get the most out of a computer system with internal or external parallelism, designers and software developers must understand the interaction between hardware and software parts of the system. This is the reason we wrote this book. We want the reader to understand the power and limitations of multiprocessor systems. Our goal is to apprise the reader of both the beneficial and challenging aspects of advanced architecture and parallelism. The material in this book is organized in 10 chapters, as follows.

Chapter 1 is a survey of the field of computer architecture at an introductory level. We first study the evolution of computing and the changes that have led to obtaining high performance computing via parallelism. The popular Flynn's taxonomy of computer systems is provided. An introduction to single instruction multiple data (SIMD) and multiple instruction multiple data (MIMD) systems is also given. Both shared-memory and the message passing systems and their interconnection networks are introduced.

Chapter 2 navigates through a number of system configurations for multiprocessors. It discusses the different topologies used for interconnecting multiprocessors. Taxonomy for interconnection networks based on their topology is introduced. Dynamic and static interconnection schemes are also studied. The bus, crossbar, and multi-stage topology are introduced as dynamic interconnections. In the static interconnection scheme, three main mechanisms are covered. These are the hypercube topology, mesh topology, and $k$-ary $n$-cube topology. A number of performance aspects are introduced including cost, latency, diameter, node degree, and symmetry.

Chapter 3 is about performance. How should we characterize the performance of a computer system when, in effect, parallel computing redefines traditional

measures such as million instructions per second (MIPS) and million floating-point operations per second (MFLOPS)? New measures of performance, such as speedup, are discussed. This chapter examines several versions of speedup, as well as other performance measures and benchmarks.

Chapters 4 and 5 cover shared memory and message passing systems, respectively. The main challenges of shared memory systems are performance degradation due to contention and the cache coherence problems. Performance of shared memory system becomes an issue when the interconnection network connecting the processors to global memory becomes a bottleneck. Local caches are typically used to alleviate the bottleneck problem. But scalability remains the main drawback of shared memory system. The introduction of caches has created consistency problem among caches and between memory and caches. In Chapter 4, we cover several cache coherence protocols that can be categorized as either snoopy protocols or directory based protocols. Since shared memory systems are difficult to scale up to a large number of processors, message passing systems may be the only way to efficiently achieve scalability. In Chapter 5, we discuss the architecture and the network models of message passing systems. We shed some light on routing and network switching techniques. We conclude with a contrast between shared memory and message passing systems.

Chapter 6 covers abstract models, algorithms, and complexity analysis. We discuss a shared-memory abstract model (PRAM), which can be used to study parallel algorithms and evaluate their complexities. We also outline the basic elements of a formal model of message passing systems under the synchronous model. We design and discuss the complexity analysis of algorithms described in terms of both models.

Chapters 7–10 discuss a number of issues related to network computing, in which the nodes are stand-alone computers that may be connected via a switch, local area network, or the Internet. Chapter 7 provides the basic concepts of network computing including client/server paradigm, cluster computing, and grid computing. Chapter 8 illustrates the parallel virtual machine (PVM) programming system. It shows how to write programs on a network of heterogeneous machines. Chapter 9 covers the message-passing interface (MPI) standard in which portable distributed parallel programs can be developed. Chapter 10 addresses the problem of allocating tasks to processing units. The scheduling problem in several of its variations is covered. We survey a number of solutions to this important problem. We cover program and system models, optimal algorithms, heuristic algorithms, scheduling versus allocation techniques, and homogeneous versus heterogeneous environments.

Students in Computer Engineering, Computer Science, and Electrical Engineering should benefit from this book. The book can be used to teach graduate courses in advanced architecture and parallel processing. Selected chapters can be used to offer special topic courses with different emphasis. The book can also be used as a comprehensive reference for practitioners working as engineers, programmers, and technologists. In addition, portions of the book can be used to teach short courses to practitioners. Different chapters might be used to offer courses with

different flavors. For example, a one-semester course in Advanced Computer Architecture may cover Chapters 1–5, 7, and 8, while another one-semester course on Parallel Processing may cover Chapters 1–4, 6, 9, and 10.

This book has been class-tested by both authors. In fact, it evolves out of the class notes for the SMU's CSE8380 and CSE8383, University of Saskatchewan's (UofS) CMPT740 and KFUPM's COE520. These experiences have been incorporated into the present book. Our students corrected errors and improved the organization of the book. We would like to thank the students in these classes. We owe much to many students and colleagues, who have contributed to the production of this book. Chuck Mann, Yehia Amer, Habib Ammari, Abdul Aziz, Clay Breshears, Jahanzeb Faizan, Michael A. Langston, and A. Naseer read drafts of the book and all contributed to the improvement of the original manuscript. Ted Lewis has contributed to earlier versions of some chapters. We are indebted to the anonymous reviewers arranged by John Wiley for their suggestions and corrections. Special thanks to Albert Y. Zomaya, the series editor and to Val Moliere, Kirsten Rohstedt and Christine Punzo of John Wiley for their help in making this book a reality. Of course, responsibility for errors and inconsistencies rests with us.

Finally, and most of all, we want to thank our wives and children for tolerating all the long hours we spent on this book. Hesham would also like to thank Ted Lewis and Bruce Shriver for their friendship, mentorship and guidance over the years.

HESHAM EL-REWINI
MOSTAFA ABD-EL-BARR
May 2004

CHAPTER 1

# Introduction to Advanced Computer Architecture and Parallel Processing

Computer architects have always strived to increase the performance of their computer architectures. High performance may come from fast dense circuitry, packaging technology, and parallelism. Single-processor supercomputers have achieved unheard of speeds and have been pushing hardware technology to the physical limit of chip manufacturing. However, this trend will soon come to an end, because there are physical and architectural bounds that limit the computational power that can be achieved with a single-processor system. In this book we will study advanced computer architectures that utilize parallelism via multiple processing units.

Parallel processors are computer systems consisting of multiple processing units connected via some interconnection network plus the software needed to make the processing units work together. There are two major factors used to categorize such systems: the processing units themselves, and the interconnection network that ties them together. The processing units can communicate and interact with each other using either shared memory or message passing methods. The interconnection network for shared memory systems can be classified as bus-based versus switch-based. In message passing systems, the interconnection network is divided into static and dynamic. Static connections have a fixed topology that does not change while programs are running. Dynamic connections create links on the fly as the program executes.

The main argument for using multiprocessors is to create powerful computers by simply connecting multiple processors. A multiprocessor is expected to reach faster speed than the fastest single-processor system. In addition, a multiprocessor consisting of a number of single processors is expected to be more cost-effective than building a high-performance single processor. Another advantage of a multiprocessor is fault tolerance. If a processor fails, the remaining processors should be able to provide continued service, albeit with degraded performance.

*Advanced Computer Architecture and Parallel Processing*, by H. El-Rewini and M. Abd-El-Barr
ISBN 0-471-46740-5 

## 1.1 FOUR DECADES OF COMPUTING

Most computer scientists agree that there have been four distinct paradigms or eras of computing. These are: batch, time-sharing, desktop, and network. Table 1.1 is modified from a table proposed by Lawrence Tesler. In this table, major characteristics of the different computing paradigms are associated with each decade of computing, starting from 1960.

### 1.1.1 Batch Era

By 1965 the IBM System/360 mainframe dominated the corporate computer centers. It was the typical batch processing machine with punched card readers, tapes and disk drives, but no connection beyond the computer room. This single mainframe established large centralized computers as the standard form of computing for decades. The IBM System/360 had an operating system, multiple programming languages, and 10 megabytes of disk storage. The System/360 filled a room with metal boxes and people to run them. Its transistor circuits were reasonably fast. Power users could order magnetic core memories with up to one megabyte of 32-bit words. This machine was large enough to support many programs in memory at the same time, even though the central processing unit had to switch from one program to another.

### 1.1.2 Time-Sharing Era

The mainframes of the batch era were firmly established by the late 1960s when advances in semiconductor technology made the solid-state memory and integrated circuit feasible. These advances in hardware technology spawned the minicomputer era. They were small, fast, and inexpensive enough to be spread throughout the company at the divisional level. However, they were still too expensive and difficult

**TABLE 1.1 Four Decades of Computing**

| Feature | Batch | Time-Sharing | Desktop | Network |
|---|---|---|---|---|
| Decade | 1960s | 1970s | 1980s | 1990s |
| Location | Computer room | Terminal room | Desktop | Mobile |
| Users | Experts | Specialists | Individuals | Groups |
| Data | Alphanumeric | Text, numbers | Fonts, graphs | Multimedia |
| Objective | Calculate | Access | Present | Communicate |
| Interface | Punched card | Keyboard and CRT | See and point | Ask and tell |
| Operation | Process | Edit | Layout | Orchestrate |
| Connectivity | None | Peripheral cable | LAN | Internet |
| Owners | Corporate computer centers | Divisional IS shops | Departmental end-users | Everyone |

LAN, local area network.

to use to hand over to end-users. Minicomputers made by DEC, Prime, and Data General led the way in defining a new kind of computing: time-sharing. By the 1970s it was clear that there existed two kinds of commercial or business computing: (1) centralized data processing mainframes, and (2) time-sharing minicomputers. In parallel with small-scale machines, supercomputers were coming into play. The first such supercomputer, the CDC 6600, was introduced in 1961 by Control Data Corporation. Cray Research Corporation introduced the best cost/performance supercomputer, the Cray-1, in 1976.

### 1.1.3 Desktop Era

Personal computers (PCs), which were introduced in 1977 by Altair, Processor Technology, North Star, Tandy, Commodore, Apple, and many others, enhanced the productivity of end-users in numerous departments. Personal computers from Compaq, Apple, IBM, Dell, and many others soon became pervasive, and changed the face of computing.

Local area networks (LAN) of powerful personal computers and workstations began to replace mainframes and minis by 1990. The power of the most capable big machine could be had in a desktop model for one-tenth of the cost. However, these individual desktop computers were soon to be connected into larger complexes of computing by wide area networks (WAN).

### 1.1.4 Network Era

The fourth era, or network paradigm of computing, is in full swing because of rapid advances in network technology. Network technology outstripped processor technology throughout most of the 1990s. This explains the rise of the network paradigm listed in Table 1.1. The surge of network capacity tipped the balance from a processor-centric view of computing to a network-centric view.

The 1980s and 1990s witnessed the introduction of many commercial parallel computers with multiple processors. They can generally be classified into two main categories: (1) shared memory, and (2) distributed memory systems. The number of processors in a single machine ranged from several in a shared memory computer to hundreds of thousands in a massively parallel system. Examples of parallel computers during this era include Sequent Symmetry, Intel iPSC, nCUBE, Intel Paragon, Thinking Machines (CM-2, CM-5), MsPar (MP), Fujitsu (VPP500), and others.

### 1.1.5 Current Trends

One of the clear trends in computing is the substitution of expensive and specialized parallel machines by the more cost-effective clusters of workstations. A cluster is a collection of stand-alone computers connected using some interconnection network. Additionally, the pervasiveness of the Internet created interest in network computing and more recently in grid computing. Grids are geographically distributed platforms

of computation. They should provide dependable, consistent, pervasive, and inexpensive access to high-end computational facilities.

## 1.2 FLYNN'S TAXONOMY OF COMPUTER ARCHITECTURE

The most popular taxonomy of computer architecture was defined by Flynn in 1966. Flynn's classification scheme is based on the notion of a stream of information. Two types of information flow into a processor: instructions and data. The instruction stream is defined as the sequence of instructions performed by the processing unit. The data stream is defined as the data traffic exchanged between the memory and the processing unit. According to Flynn's classification, either of the instruction or data streams can be single or multiple. Computer architecture can be classified into the following four distinct categories:

- single-instruction single-data streams (SISD);
- single-instruction multiple-data streams (SIMD);
- multiple-instruction single-data streams (MISD); and
- multiple-instruction multiple-data streams (MIMD).

Conventional single-processor von Neumann computers are classified as SISD systems. Parallel computers are either SIMD or MIMD. When there is only one control unit and all processors execute the same instruction in a synchronized fashion, the parallel machine is classified as SIMD. In a MIMD machine, each processor has its own control unit and can execute different instructions on different data. In the MISD category, the same stream of data flows through a linear array of processors executing different instruction streams. In practice, there is no viable MISD machine; however, some authors have considered pipelined machines (and perhaps systolic-array computers) as examples for MISD. Figures 1.1, 1.2, and 1.3 depict the block diagrams of SISD, SIMD, and MIMD, respectively.

An extension of Flynn's taxonomy was introduced by D. J. Kuck in 1978. In his classification, Kuck extended the instruction stream further to single (scalar and array) and multiple (scalar and array) streams. The data stream in Kuck's classification is called the *execution stream* and is also extended to include single

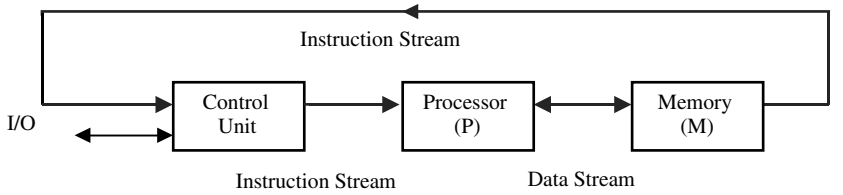



**Figure 1.1** SISD architecture.

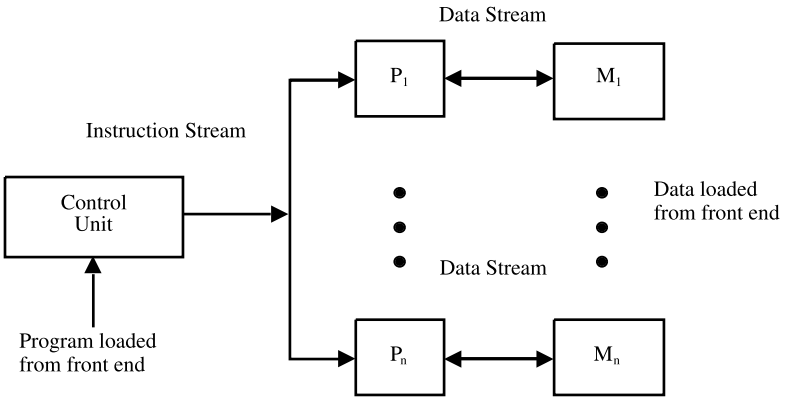



**Figure 1.2** SIMD architecture.

(scalar and array) and multiple (scalar and array) streams. The combination of these streams results in a total of 16 categories of architectures.

## 1.3 SIMD ARCHITECTURE

The SIMD model of parallel computing consists of two parts: a front-end computer of the usual von Neumann style, and a processor array as shown in Figure 1.4. The processor array is a set of identical synchronized processing elements capable of simultaneously performing the same operation on different data. Each processor in the array has a small amount of local memory where the distributed data resides while it is being processed in parallel. The processor array is connected to the memory bus of the front end so that the front end can randomly access the local

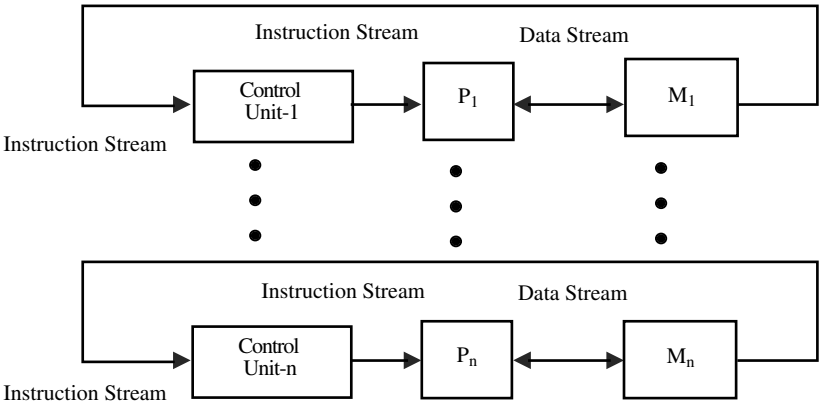



**Figure 1.3** MIMD architecture.

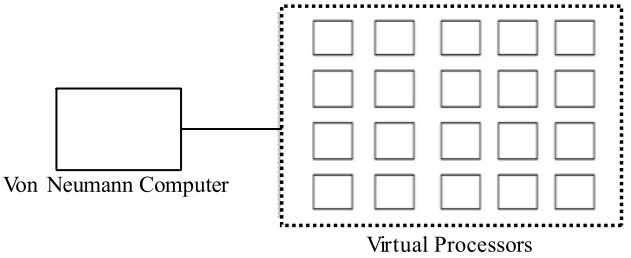



**Figure 1.4** SIMD architecture model.

processor memories as if it were another memory. Thus, the front end can issue special commands that cause parts of the memory to be operated on simultaneously or cause data to move around in the memory. A program can be developed and executed on the front end using a traditional serial programming language. The application program is executed by the front end in the usual serial way, but issues commands to the processor array to carry out SIMD operations in parallel. The similarity between serial and data parallel programming is one of the strong points of data parallelism. Synchronization is made irrelevant by the lock–step synchronization of the processors. Processors either do nothing or exactly the same operations at the same time. In SIMD architecture, parallelism is exploited by applying simultaneous operations across large sets of data. This paradigm is most useful for solving problems that have lots of data that need to be updated on a wholesale basis. It is especially powerful in many regular numerical calculations.

There are two main configurations that have been used in SIMD machines (see Fig. 1.5). In the first scheme, each processor has its own local memory. Processors can communicate with each other through the interconnection network. If the interconnection network does not provide direct connection between a given pair of processors, then this pair can exchange data via an intermediate processor. The ILLIAC IV used such an interconnection scheme. The interconnection network in the ILLIAC IV allowed each processor to communicate directly with four neighboring processors in an $8 \times 8$ matrix pattern such that the $i^{\text{th}}$ processor can communicate directly with the $(i-1)^{\text{th}}$, $(i+1)^{\text{th}}$, $(i-8)^{\text{th}}$, and $(i+8)^{\text{th}}$ processors. In the second SIMD scheme, processors and memory modules communicate with each other via the interconnection network. Two processors can transfer data between each other via intermediate memory module(s) or possibly via intermediate processor(s). The BSP (Burroughs' Scientific Processor) used the second SIMD scheme.

## 1.4 MIMD ARCHITECTURE

Multiple-instruction multiple-data streams (MIMD) parallel architectures are made of multiple processors and multiple memory modules connected together via some

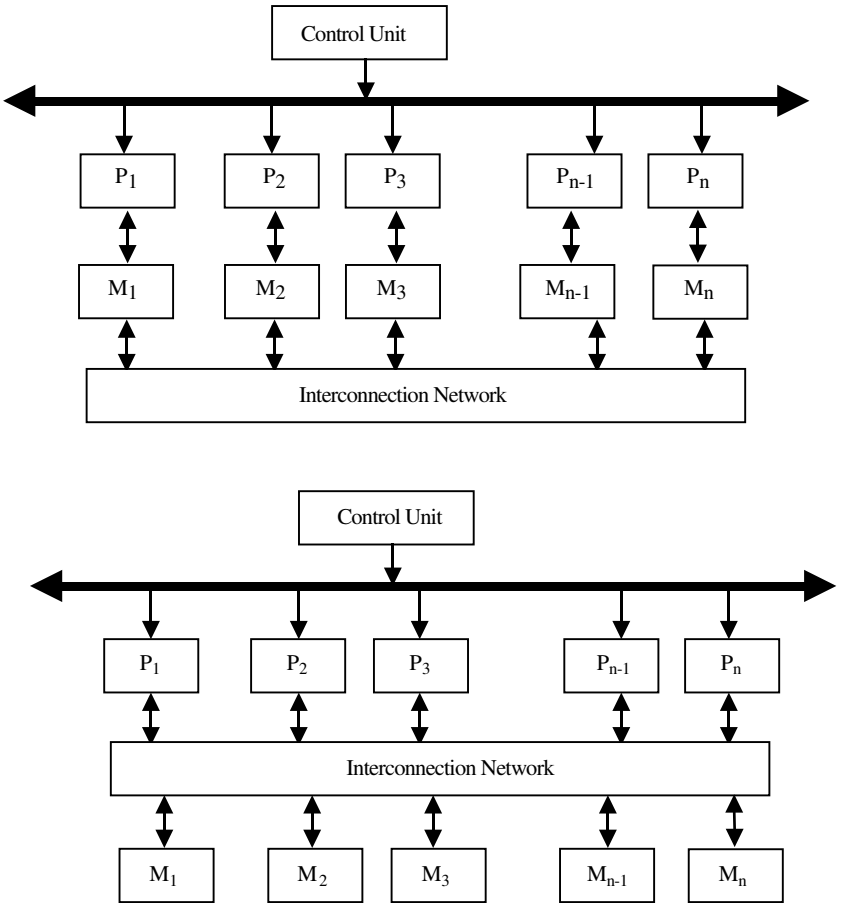
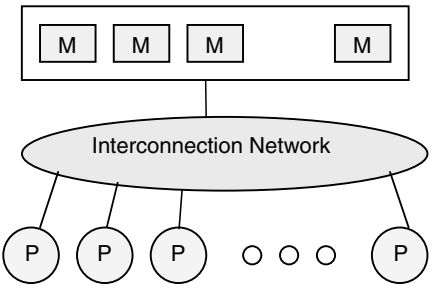


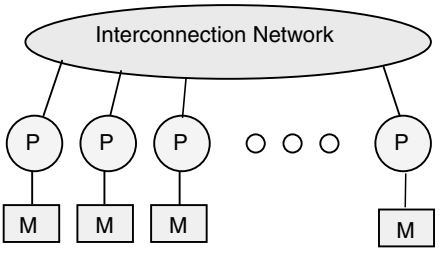

**Figure 1.5** Two SIMD schemes.

interconnection network. They fall into two broad categories: shared memory or message passing. Figure 1.6 illustrates the general architecture of these two categories. Processors exchange information through their central shared memory in shared memory systems, and exchange information through their interconnection network in message passing systems.

A *shared memory system* typically accomplishes interprocessor coordination through a global memory shared by all processors. These are typically server systems that communicate through a bus and cache memory controller. The bus/cache architecture alleviates the need for expensive multiported memories and interface circuitry as well as the need to adopt a message-passing paradigm when developing application software. Because access to shared memory is balanced, these systems are also called SMP (symmetric multiprocessor) systems. Each processor has equal opportunity to read/write to memory, including equal access speed.




Shared Memory MIMD Architecture




Message Passing MIMD Architecture

**Figure 1.6** Shared memory versus message passing architecture.

Commercial examples of SMPs are Sequent Computer's Balance and Symmetry, Sun Microsystems multiprocessor servers, and Silicon Graphics Inc. multiprocessor servers.

A *message passing system* (also referred to as distributed memory) typically combines the local memory and processor at each node of the interconnection network. There is no global memory, so it is necessary to move data from one local memory to another by means of message passing. This is typically done by a Send/Receive pair of commands, which must be written into the application software by a programmer. Thus, programmers must learn the message-passing paradigm, which involves data copying and dealing with consistency issues. Commercial examples of message passing architectures c. 1990 were the nCUBE, iPSC/2, and various Transputer-based systems. These systems eventually gave way to Internet connected systems whereby the processor/memory nodes were either Internet servers or clients on individuals' desktop.

It was also apparent that distributed memory is the only way efficiently to increase the number of processors managed by a parallel and distributed system. If scalability to larger and larger systems (as measured by the number of processors) was to continue, systems had to use distributed memory techniques. These two forces created a conflict: programming in the shared memory model was easier, and designing systems in the message passing model provided scalability. The

*distributed-shared memory* (DSM) architecture began to appear in systems like the SGI Origin2000, and others. In such systems, memory is physically distributed; for example, the hardware architecture follows the message passing school of design, but the programming model follows the shared memory school of thought. In effect, software covers up the hardware. As far as a programmer is concerned, the architecture looks and behaves like a shared memory machine, but a message passing architecture lives underneath the software. Thus, the DSM machine is a hybrid that takes advantage of both design schools.

### 1.4.1 Shared Memory Organization

A shared memory model is one in which processors communicate by reading and writing locations in a shared memory that is equally accessible by all processors. Each processor may have registers, buffers, caches, and local memory banks as additional memory resources. A number of basic issues in the design of shared memory systems have to be taken into consideration. These include access control, synchronization, protection, and security. Access control determines which process accesses are possible to which resources. Access control models make the required check for every access request issued by the processors to the shared memory, against the contents of the access control table. The latter contains flags that determine the legality of each access attempt. If there are access attempts to resources, then until the desired access is completed, all disallowed access attempts and illegal processes are blocked. Requests from sharing processes may change the contents of the access control table during execution. The flags of the access control with the synchronization rules determine the system's functionality. Synchronization constraints limit the time of accesses from sharing processes to shared resources. Appropriate synchronization ensures that the information flows properly and ensures system functionality. Protection is a system feature that prevents processes from making arbitrary access to resources belonging to other processes. Sharing and protection are incompatible; sharing allows access, whereas protection restricts it.

The simplest shared memory system consists of one memory module that can be accessed from two processors. Requests arrive at the memory module through its two ports. An arbitration unit within the memory module passes requests through to a memory controller. If the memory module is not busy and a single request arrives, then the arbitration unit passes that request to the memory controller and the request is granted. The module is placed in the busy state while a request is being serviced. If a new request arrives while the memory is busy servicing a previous request, the requesting processor may hold its request on the line until the memory becomes free or it may repeat its request sometime later.

Depending on the interconnection network, a shared memory system leads to systems can be classified as: uniform memory access (UMA), nonuniform memory access (NUMA), and cache-only memory architecture (COMA). In the UMA system, a shared memory is accessible by all processors through an interconnection network in the same way a single processor accesses its memory. Therefore,

all processors have equal access time to any memory location. The interconnection network used in the UMA can be a single bus, multiple buses, a crossbar, or a multiport memory. In the NUMA system, each processor has part of the shared memory attached. The memory has a single address space. Therefore, any processor could access any memory location directly using its real address. However, the access time to modules depends on the distance to the processor. This results in a nonuniform memory access time. A number of architectures are used to interconnect processors to memory modules in a NUMA. Similar to the NUMA, each processor has part of the shared memory in the COMA. However, in this case the shared memory consists of cache memory. A COMA system requires that data be migrated to the processor requesting it. Shared memory systems will be discussed in more detail in Chapter 4.

### 1.4.2 Message Passing Organization

Message passing systems are a class of multiprocessors in which each processor has access to its own local memory. Unlike shared memory systems, communications in message passing systems are performed via send and receive operations. A *node* in such a system consists of a processor and its local memory. Nodes are typically able to store messages in buffers (temporary memory locations where messages wait until they can be sent or received), and perform send/receive operations at the same time as processing. Simultaneous message processing and problem calculating are handled by the underlying operating system. Processors do not share a global memory and each processor has access to its own address space. The processing units of a message passing system may be connected in a variety of ways ranging from architecture-specific interconnection structures to geographically dispersed networks. The message passing approach is, in principle, scalable to large proportions. By scalable, it is meant that the number of processors can be increased without significant decrease in efficiency of operation.

Message passing multiprocessors employ a variety of static networks in local communication. Of importance are hypercube networks, which have received special attention for many years. The nearest neighbor two-dimensional and three-dimensional mesh networks have been used in message passing systems as well. Two important design factors must be considered in designing interconnection networks for message passing systems. These are the link *bandwidth* and the network *latency*. The link bandwidth is defined as the number of bits that can be transmitted per unit time (bits/s). The network latency is defined as the time to complete a message transfer. Wormhole routing in message passing was introduced in 1987 as an alternative to the traditional store-and-forward routing in order to reduce the size of the required buffers and to decrease the message latency. In *wormhole routing*, a packet is divided into smaller units that are called *flits* (flow control bits) such that *flits* move in a pipeline fashion with the header *flit* of the packet leading the way to the destination node. When the header flit is blocked due to network congestion, the remaining flits are blocked as well. More details on message passing will be introduced in Chapter 5.

## 1.5 INTERCONNECTION NETWORKS

Multiprocessors interconnection networks (INs) can be classified based on a number of criteria. These include (1) mode of operation (synchronous versus asynchronous), (2) control strategy (centralized versus decentralized), (3) switching techniques (circuit versus packet), and (4) topology (static versus dynamic).

### 1.5.1 Mode of Operation

According to the mode of operation, INs are classified as *synchronous* versus *asynchronous.* In synchronous mode of operation, a single global clock is used by all components in the system such that the whole system is operating in a lock–step manner. Asynchronous mode of operation, on the other hand, does not require a global clock. Handshaking signals are used instead in order to coordinate the operation of asynchronous systems. While synchronous systems tend to be slower compared to asynchronous systems, they are race and hazard-free.

### 1.5.2 Control Strategy

According to the control strategy, INs can be classified as *centralized* versus *decentralized.* In centralized control systems, a single central control unit is used to oversee and control the operation of the components of the system. In decentralized control, the control function is distributed among different components in the system. The function and reliability of the central control unit can become the bottleneck in a centralized control system. While the crossbar is a centralized system, the multistage interconnection networks are decentralized.

### 1.5.3 Switching Techniques

Interconnection networks can be classified according to the switching mechanism as *circuit* versus *packet switching* networks. In the circuit switching mechanism, a complete path has to be established prior to the start of communication between a source and a destination. The established path will remain in existence during the whole communication period. In a packet switching mechanism, communication between a source and destination takes place via messages that are divided into smaller entities, called packets. On their way to the destination, packets can be sent from a node to another in a store-and-forward manner until they reach their destination. While packet switching tends to use the network resources more efficiently compared to circuit switching, it suffers from variable packet delays.

### 1.5.4 Topology

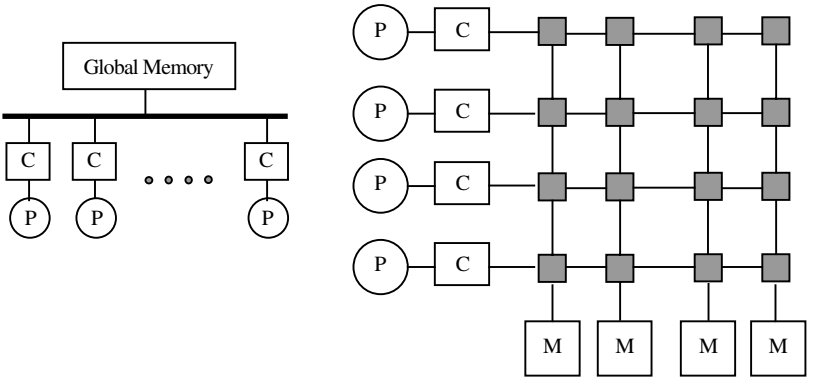
An *interconnection network topology* is a mapping function from the set of processors and memories onto the same set of processors and memories. In other words, the topology describes how to connect processors and memories to other

processors and memories. A fully connected topology, for example, is a mapping in which each processor is connected to all other processors in the computer. A ring topology is a mapping that connects processor $k$ to its neighbors, processors $(k-1)$ and $(k+1)$.

In general, interconnection networks can be classified as *static* versus *dynamic* networks. In static networks, direct fixed links are established among nodes to form a fixed network, while in dynamic networks, connections are established as needed. Switching elements are used to establish connections among inputs and outputs. Depending on the switch settings, different interconnections can be established. Nearly all multiprocessor systems can be distinguished by their interconnection network topology. Therefore, we devote Chapter 2 of this book to study a variety of topologies and how they are used in constructing a multiprocessor system. However, in this section, we give a brief introduction to interconnection networks for shared memory and message passing systems.

Shared memory systems can be designed using bus-based or switch-based INs. The simplest IN for shared memory systems is the bus. However, the bus may get saturated if multiple processors are trying to access the shared memory (via the bus) simultaneously. A typical bus-based design uses caches to solve the bus contention problem. Other shared memory designs rely on switches for interconnection. For example, a crossbar switch can be used to connect multiple processors to multiple memory modules. A crossbar switch, which will be discussed further in Chapter 2, can be visualized as a mesh of wires with switches at the points of intersection. Figure 1.7 shows (*a*) bus-based and (*b*) switch-based shared memory systems. Figure 1.8 shows bus-based systems when a single bus is used versus the case when multiple buses are used.

Message passing INs can be divided into static and dynamic. Static networks form all connections when the system is designed rather than when the connection is needed. In a static network, messages must be routed along established links.




(*a*) (*b*)

**Figure 1.7** Shared memory interconnection networks.