

COMPUTER SYSTEMS

Sotirios G. Ziavras, Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, New Jersey 07102, U.S.A.

Keywords

Computer organization, processor, memory hierarchy, peripheral devices, bus architectures, multiprocessors, multicomputers, computation models, supercomputers.

Contents

1. Introduction
2. Sequential/Conventional Computers
 - 2.1. Basic Resources
 - 2.1.1. Central Processing Unit
 - 2.1.2. Main Memory
 - 2.1.3. Associative Memory: Instruction and Data Caches
 - 2.1.4. Peripheral Devices
 - 2.1.5. Resource Connectivity
 - 2.2. Computer Performance
 - 2.3. Program Control
 - 2.4. System Software
 3. Parallel Computers
 - 3.1. Multiprocessors
 - 3.2. Multicomputers
 - 3.3. Vector Supercomputers

Glossary

Bus: A set of wires used to transmit data, addresses, or control signals between directly-connected components of a computer. They are called data, address, and control busses, respectively.

CPU: Central Processing Unit. The words CPU and processor are used interchangeably in this article.

Distributed processing: running a single program on computers of a network.

DRAM: Dynamic RAM. Its contents must be refreshed very often to avoid the loss of data.

Massively-parallel computer: a parallel computer containing hundreds or thousands of (micro)processors.

MIMD: Multiple-Instruction streams, Multiple-Data streams.

Multicomputer: a parallel computer containing many processors which are interconnected via a static point-to-point (i.e., processor-to-processor) physical network.

Multiprocessor: a parallel computer containing many processors which can exchange information through a shared memory. They access this memory via a dynamic network. The exact interconnection scheme is determined each time by the application program.

Parallel computer: a computer that contains many processors.

RAM: Random-Access Memory. They can be read and written at run time by programs.

ROM: Read-Only Memory. They cannot be written by programs. Their contents can be modified only by plugging them into specialized hardware programmers.

SIMD: Single-Instruction stream, Multiple-Data streams.

SRAM: Static RAM that is much faster than DRAM.

Supercomputer: a computer capable of delivering performance many orders of magnitude larger than that of any single-processor computer. This term is currently associated with massively-parallel computers and vector supercomputers.

System boot up code: The part of the operating system that initializes the computer.

Tri-state gate: a digital circuit that has three possible output states, namely 0, 1, and high-impedance. In the high-impedance state, the output is disabled and seems to be "floating" (that is, it does not affect and is not affected by any other signal applied to the corresponding terminal).

Vector supercomputer: a computer capable of delivering performance for array (i.e., vector) operations many orders of magnitude larger than that of any conventional computer. It contains specialized parallel units for vector operations.

Summary

The fundamentals of computer systems design and organization are presented, and the conventional procedure for the execution of computer programs is described. An overview of the major features and interactions of the hardware and software components of modern computer systems is also included. Numerous computer systems have been designed and built to aid humans in information processing and numerical calculations. As a result, several models have emerged in the field of computer systems design. These models differ in the architecture of the processors, the underlying model of computation, the architecture of the main memory, or the techniques used to interconnect the basic resources within the computer. This article presents a summary of the most fundamental computer models. The performance analysis task of computer systems is touched upon to facilitate comparisons. Advances in the technology that integrates transistors on chips improve the performance of all design models by increasing the depth of the instruction execution pipelines and the number of functional units in processors, the speed of all major electronic components, the size of on-chip cache memories, etc.

1. Introduction

Modern computers are electronic and process digital information. The physical machine consists of transistors, digital circuits implemented with transistors, wires, and mechanical components in peripheral devices used for information storage. These physical entities are collectively called hardware. System and application programs are called software. A general purpose computer system is a programmable machine that can

solve problems by accepting inputs and instructions on how to use these inputs. The instructions are included in computer programs (that is, software) that normally contain sequences of them. Graphical languages are rarely used to represent computer programs as collections of instructions with relationships between arbitrary pairs of them. Programs are often written in high-level languages (HLLs) that have to be translated (by appropriate software compilers) to produce machine-readable (that is, machine language) code that can be run directly by the given computer system. The machine language code contains sequences of primitive instructions for the given computer in binary representation. On the other hand, HLLs employ mnemonics of more powerful instructions, and appropriate structures to make programming easy and independent of the target computer.

From the software point of view, a computer is a six-level system consisting of the digital logic (collections of electronic gates), microarchitecture (a collection of functional units, such as ALUs - Arithmetic Logic Units, and their interconnectivity), instruction set architecture (the complete set of machine language instructions), operating system (code that monitors and controls the activities of the computer), assembly and machine language, and high-level language. The assembly language is very close to the machine language of a computer; it basically replaces the binary representation of machine instructions with mnemonics in a one-to-one fashion. From the hardware point of view, a computer is conveniently assumed to be a five-level hierarchy. The five levels correspond to network ports for connecting to the outside world (these ports may not be necessarily available, as a computer may be a standalone information processing and/or computing machine), peripheral or mass-storage devices for (applications and system) program and data storage, main memory, program and data caches (fast memories for retrieving data by content), and CPU (Central Processing Unit) or processor. Special emphasis is given in this article to the description of computer systems based on this five-level representation. Many other components are also included in computer systems in order to enable the aforementioned basic components to function properly. For example, control and data busses are used to transmit data between any two successive levels of the hardware hierarchy and glue logic is used to implement the appropriate interfaces. The design of a computer system most often begins with the selection of a particular CPU. The other components are selected progressively based on performance requirements. Analytical techniques, software simulations, and software or hardware prototyping of the complete or partial computer system are used to make final decisions about the design. Special attention is given nowadays to hardware-software codesign, where the selection or design of components is made in unison with the development of the corresponding system software.

There exist several types of general purpose computer systems. These types are grouped together into two major computer classes, comprising sequential or conventional computers, and parallel computers, respectively. The class of sequential or conventional computer systems comprises:

- **Laptops and palmtops.** These are small, portable computer systems. Laptops often contain very powerful processors and have capabilities very close to those of PCs (see below); their major drawbacks are smaller screens, smaller memory, and fewer

peripheral (that is, I/O - Input/Output) devices which are, however, portable. These single-user computers are implemented in the form of microcomputers. The prefix micro denotes the inclusion of a microprocessor that resides on a single chip and serves as the CPU. Other components also are included, of which the critical ones are a keyboard for data input, a screen (monitor) for information display, memory chips for temporary storage, and a hard disk for data storage.

- **PCs (personal computers) or desktops.** These computers also are of the microcomputer type. Figure 1 shows the basic components of a microcomputer. The RAM and ROM form the main memory that stores system and application programs, and data. The ROM contains only part of the operating system, and most often the part that initializes the computer. The software stored in the ROM is called firmware. Resource interface units form the required glue logic for the implementation of the required data exchange protocols. The control bus transfers the control signals produced by the microprocessor. To access an element in the memory or a peripheral device, the microprocessor first issues the address of that item on the address bus. The address bus is unidirectional, from the processor to the other units. While the latter value is still present on the address bus, the microprocessor issues the appropriate control signals to read or write from the corresponding location. The address issued by the microprocessor is decoded by external logic to choose the appropriate memory module or I/O device. The data is finally transferred on the bidirectional data bus. Details on how microcomputers execute programs are presented in Section 2.
- **Workstations.** They appeared in the 1980s as single-user computers with much better performance than PCs, primarily because they contain very advanced microprocessors. They often include proprietary co-processors to facilitate graphics functions because they basically target the scientific and engineering communities. They were uniprocessor computers in the early days, but multiprocessor workstations appeared for the first time in the market a few years ago. They are now often used as multi-user platforms.
- **Minicomputers.** High-performance cabinet-sized computers that can be used simultaneously by a few dozens of users. They are often used in engineering and scientific applications. They have been replaced recently by advanced workstations and networks of workstations.
- **Mainframes.** Very powerful computers that can serve many dozens or hundreds of users simultaneously. IBM has produced numerous computers of this type. They have been replaced recently in many occasions by networks of workstations.

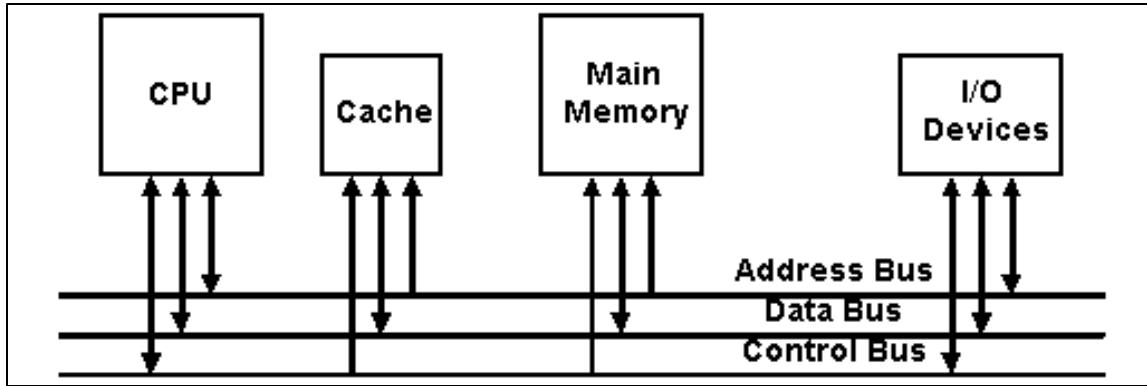


Figure 1: The architecture and basic components of a microcomputer.

Contrary to sequential computers that use a single CPU to solve a problem, parallel computer systems employ many CPUs in appropriately connected structures. This new class of computers comprises multiprocessors, multicomputers, and vector supercomputers. These types of computer systems are discussed in detail in Section 3. Also, distributed computer systems can be developed, where several complete computer systems are connected together in a networked fashion to solve a problem in parallel. For the sake of brevity, we do not discuss distributed computer systems any further. Section 2 presents in detail the class of sequential computers. Without loss of generality, for a better description emphasis is given in Section 2 to microcomputers.

2. Sequential/Conventional Computers

2.1. Basic Resources

Let us present briefly the basic components of a sequential computer and discuss their interoperability. The basic resources of complete computer systems are the CPU, instruction and data caches, main memory, peripheral devices, busses for the interconnection of these components, and some interfaces that implement the data exchange protocols. They are studied in the following subsections.

2.1.1. Central Processing Unit

The CPU is the heart of any computer system. In terms of computing power, the CPU is the most important component. The two fundamental units of a CPU are the arithmetic logic unit (ALU) and the control unit. The former unit performs arithmetic and logic operations. The latter unit fetches instructions from the memory, decodes them to determine the operations to be performed, and finally carries out these operations. All processes are initiated by the control unit that issues appropriate sequences of micro-operations in a timely fashion. To complete the execution of an instruction, the control unit may have to issue appropriate control signals to the ALU unit. Conventional control units use the program counter, a CPU register, that always contains the address of the

next instruction to be fetched from the program memory. After an instruction is fetched into the CPU, the program counter is incremented appropriately to point to the next instruction in the program. The only exceptions to incrementing the program counter are with jump instructions, procedure calls, and interrupts (also known as exceptions, which are system or program initiated subroutine calls that preempt the execution of programs) that load the program counter with an arbitrary value. To distinguish among different types of bus activities, the microprocessor uses explicit "status" control lines or the type of bus activity can be inferred from some combination of active control signals. CPU attached devices use this information to respond appropriately to CPU-initiated tasks.

On large computers the CPU components may be distributed on one or more printed circuit boards. A single chip most often contains the entire CPU on PCs and small workstations. In this case, the CPU is also known as a microprocessor. There are two basic types of conventional CPUs, namely hardwired and microcoded. This distinction is made based on how their control units are implemented. The capabilities of a hardwired CPU are determined at design time and are fixed after the chip is manufactured. On the other hand, microcoded CPUs include on-chip ROM memory that contains for each machine instruction a microprogram. Each microinstruction in the microprogram specifies the control signals to be applied to the CPU datapath and functional units, and contains the address of the next microinstruction. Maurice Wilkes first proposed the use of microprogramming in 1954. Microcoded CPUs became preeminent in the 1970s. Microprogramming allows modifications in the machine's instruction set by changing the contents of the microprogram memory; that is, no hardware changes are needed. Therefore, the great advantage of microprogramming is flexibility in designing the instruction set. However, the main disadvantage is the slower execution of programs because of the overhead related to microprogram memory accesses.

There exist two types of microcoded control units. They contain horizontal and vertical microinstructions, respectively. Each bit in a horizontal microinstruction controls a distinct control signal (or micro-operation). To increase the utilization of the microinstruction bits and to reduce the size of the microinstructions, vertical microinstructions group together mutually exclusive control signals into the same field through encoding. Vertical microinstructions require additional decoding to produce the control signals. To avoid decoding, some CPUs use the encoded instructions as addresses to access a second-level memory, called nanomemory, where the nanoinstructions that correspond directly to control signals can be accessed. However, hardwired CPUs are the most common choice because of their low overheads in instruction decoding and the issuing of appropriate control signals.

Electronic computer design is an ever changing field. Early electronic computers in the 1940s contained vacuum tubes. The introduction of transistor technology brought about significant changes in the 1950s. Discrete transistors were followed by integrated circuits (ICs) of transistors, very-large-scale integration (VLSI) circuits with thousands of transistors, and ULSI (Ultra LSI) circuits with millions of them. Continued advances in silicon integration technologies double processor performance in about every 1.5 years. As computer components continue to shrink in size, more and more functions are

possible by CPUs. There currently exist several classes of CPUs. What CPU we choose for a computer system can have a dramatic effect on its overall performance. The preeminent classes of conventional CPUs are the CISC (Complex Instruction Set Computer), RISC (Reduced Instruction Set Computer), and VLIW (Very Long Instruction Word computer) classes.

CISC processors have very large instruction sets and generally implement in hardware several different versions of a single instruction. For example, the different versions of an instruction may vary based on the word lengths of the operands or the addressing modes for accessing the operands. This approach increases the complexity of the CPU design, and may result in low performance because of longer instruction decoding and longer datapaths in the CPU chip. Studies have shown that only four types of instructions have high utilization in the majority of the applications, namely loads and stores from and to the memory, addition of operands, and branches within programs. Therefore, many instructions are rarely used if the instruction set of the processor is large. This is a major drawback because the corresponding real estate could be used to implement other, often needed tasks.

On the other hand, RISC processors implement directly in hardware a rather small set of simple instructions. Complex instructions needed by HLL programs can then be implemented by software that uses these simple instructions; the compiler makes the required conversions. The small instruction set results in reduced hardware complexity and better performance. The remaining real estate on the CPU chip is used to implement such components as MMUs (Memory Management Units), caches, etc. VLIW processors use very wide busses to simultaneously fetch many mutually exclusive, simple instructions into the CPU for execution. These simple instructions go directly to their most appropriate execution units. VLIW processors require sophisticated compilers to compact individual instructions into larger groups for simultaneous transfer to the CPU. Therefore, VLIW code is not portable because how individual instructions are grouped together depends on the type and number of functional units in the VLIW CPU.

Independently of the CPU class, the multithreading technique is widely implemented in current processors. In multithreading, different threads of programs (that is, independent sequences of instructions) may be active simultaneously by assigning to each thread its own program counter and CPU registers. Only one thread can use the execution unit(s) at any time, but a new thread is chosen when the current thread becomes inactive; a thread may become inactive while waiting to receive data from a remote resource. Multithreading increases the utilization of the CPU resources. In the case of simultaneous multithreading, operations from multiple threads are implemented in a single clock cycle. The HEP (Heterogeneous Processor) multiprocessor was an early example of a commercial machine that used multithreaded execution for processing the main instruction streams.

As a short note, an unconventional CPU design technique employs the dataflow model of computation. Dataflow has not yet been adopted in mainstream processor design. The best performance can only be derived through the maximization of parallelism. To this

effect, any ordering constraints in the execution of instructions must be minimized. Unfortunately, conventional designs assume programs composed of sequences of instructions that must be executed sequentially. Ideally, the execution of an instruction should be constrained only by instruction inter-dependence relationships and not by any other ordering constraints. The dataflow computing model has been proposed to overcome the performance limitations of the traditional sequential execution model. Only data dependencies constraint the execution of instructions. Computations are represented by dataflow graphs. Researchers have proposed dataflow languages, such as Id. However, it is not known how to implement effectively the dataflow model of computation in its pure form.

2.1.2. Main Memory

The main memory of a general-purpose computer contains sections of programs and relevant data, and is composed of RAM and ROM chips. The ROM component of the memory contains programs and data that do not change at run time. The operating system code that initializes the computer on boot-up is normally stored here. The RAM contains user and system programs and their data which are loaded from peripheral devices, and intermediate results which are produced by the processor. The main RAM memory is implemented with DRAM technology. The RAM and ROM chips are attached to the processor bus, which comprises separate address, data, and control busses. These busses are also collectively known as the memory bus. To access a location in the main memory, the processor issues the corresponding address on the address bus. It then activates (in the next clock cycle) a control signal on the control bus to indicate to the devices connected to the memory bus that a valid memory address is present on the address bus. The remaining activities depend on the type of the memory operation. For a memory write operation, the processor then sends out on the data bus the data that must be transferred to the memory. It then activates the WRITE control signal. The memory interface unit decodes the address on the address bus and determines the referenced memory location. High-order and low-order address bits conventionally select the memory chip and the appropriate word within the chip, respectively. This address decoding scheme is called high-order interleaving. The valid WRITE signal forces the memory to accept the data on the data bus as input for that location. After the data is written, the memory interface unit activates a control signal that informs the processor about operation completion. This forces the processor to release its bus. For a memory read operation, the processor uses the data bus to accept data arriving from the memory. Instead of the WRITE signal, it now activates the READ signal. An entire memory read or write operation is called a memory bus cycle.

The entire memory of a computer system is organized hierarchically. Virtual memory is a technique often used with large systems to support portability of programs, ease of programming, and multiuser environments. Programs and data are first stored in auxiliary memory (disks, etc.) and portions of them are brought into the main memory as needed by the program using the CPU. Programs are conveniently developed assuming an infinite address space. The addresses produced by programs, which are called virtual or

logical addresses, are first translated by an external mechanism to physical addresses corresponding to the main memory. This translation is often carried out by a memory management unit (MMU) which sometimes resides on the same chip with the CPU. The operating system controls the MMU to improve the utilization of its address map table; this becomes even more important if only part of the map table is present in the MMU, with the remaining parts being stored in the main memory. If a referenced item is not found in the main memory, then there exists a pointer to it in an auxiliary device. Two basic techniques exist for the implementation of virtual memory, namely paging and segmentation. There is also a hybrid technique that combines these basic ones and is called segmented paging.

With paging, the virtual/logical and physical address spaces are divided into equal-sized blocks of contiguous memory locations; these blocks are called pages. A program is assigned one or more pages, based on its size. A virtual/logical address translation becomes each time a mapping of the corresponding virtual page to a physical page in the main memory. The offsets within these two pages are identical. The advantages of paging are simple address translation and low cost. However, its major disadvantage is internal fragmentation. It occurs when the last page assigned to a program is not fully occupied because of smaller size, in which case some memory space is wasted. The basic technique of paging is shown in Figure 2.

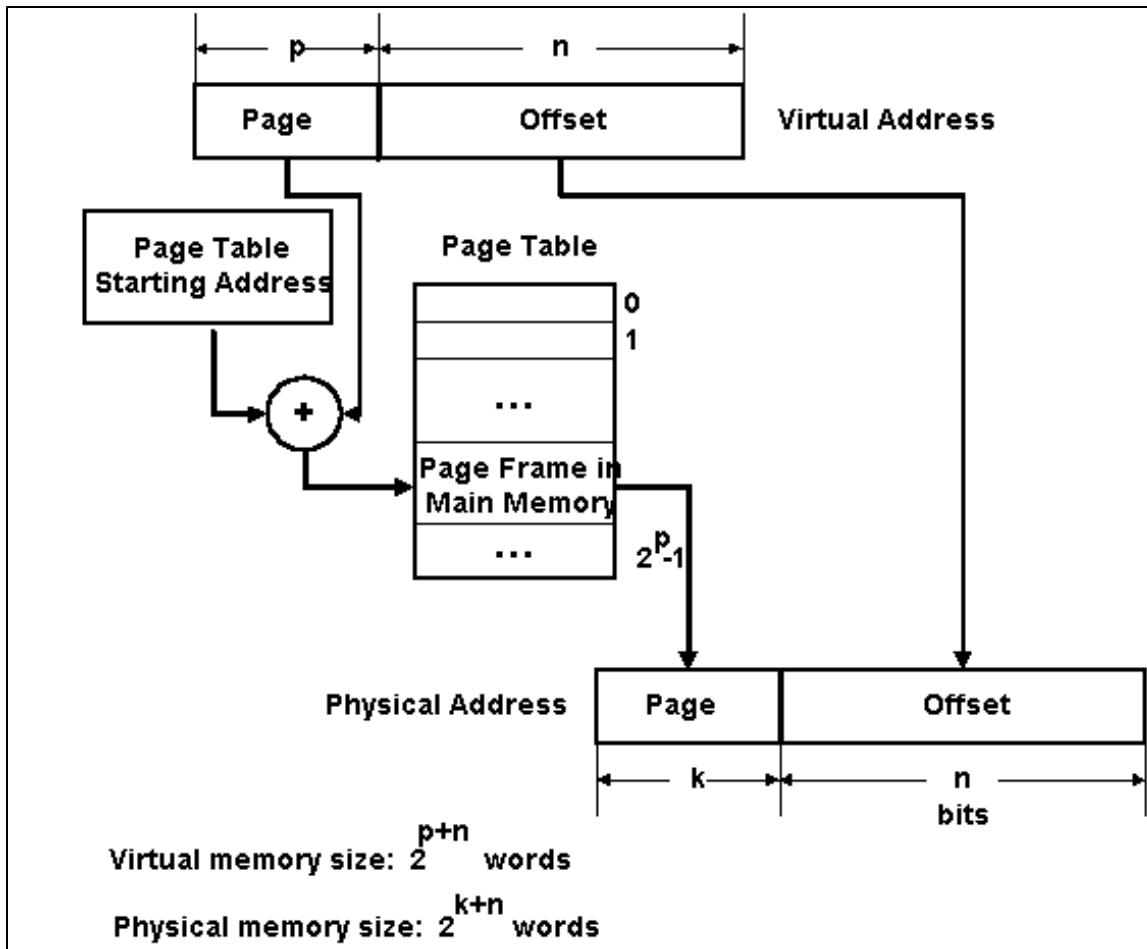


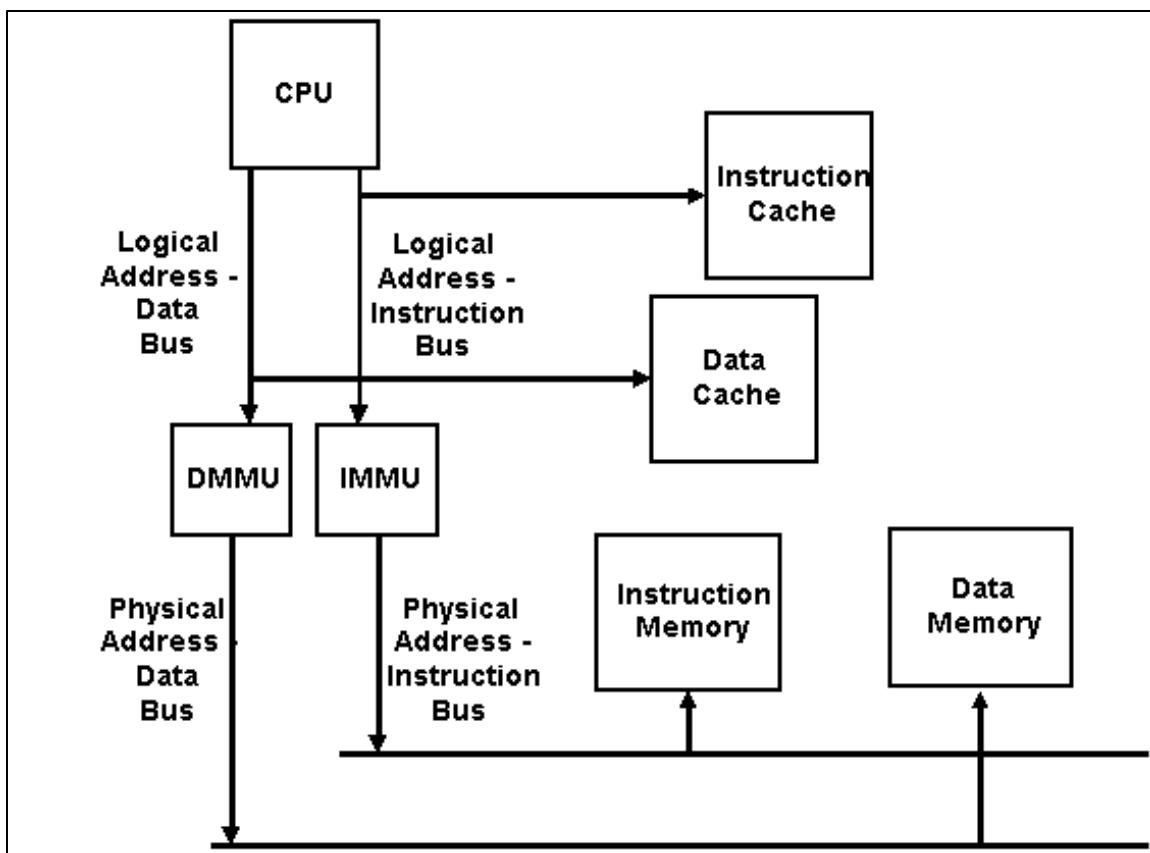
Figure 2: Paging with virtual memory.

To avoid internal fragmentation and to facilitate ease of debugging, the technique of segmentation assigns to each program exactly the space that it needs. More specifically, distinct segments of contiguous memory locations are assigned to the distinct subroutines of a program. Therefore, the starting address in the main memory of an assigned segment is arbitrary. The major advantages of this technique are ease of program debugging and no waste of memory. However, this technique suffers from external fragmentation. More specifically, despite that the collective free space may be larger than the space needed at some time by a program, there may not exist at some time during the execution of the program enough contiguous space of the size needed by the program. To solve this problem, garbage collection and memory coalescing must be carried out by identifying and moving occupied segments next to each other in an effort to create large contiguous free space. However, this technique has an adverse effect on the cost of the system. To utilize the advantages of both techniques, segmented paging is a combination of both. According to this technique, each segment is divided into pages of the same size. To speed up any address translation process, associative memories (the concept behind these devices is introduced in the next subsection) may be used to store parts of the translation table that have high access probability.

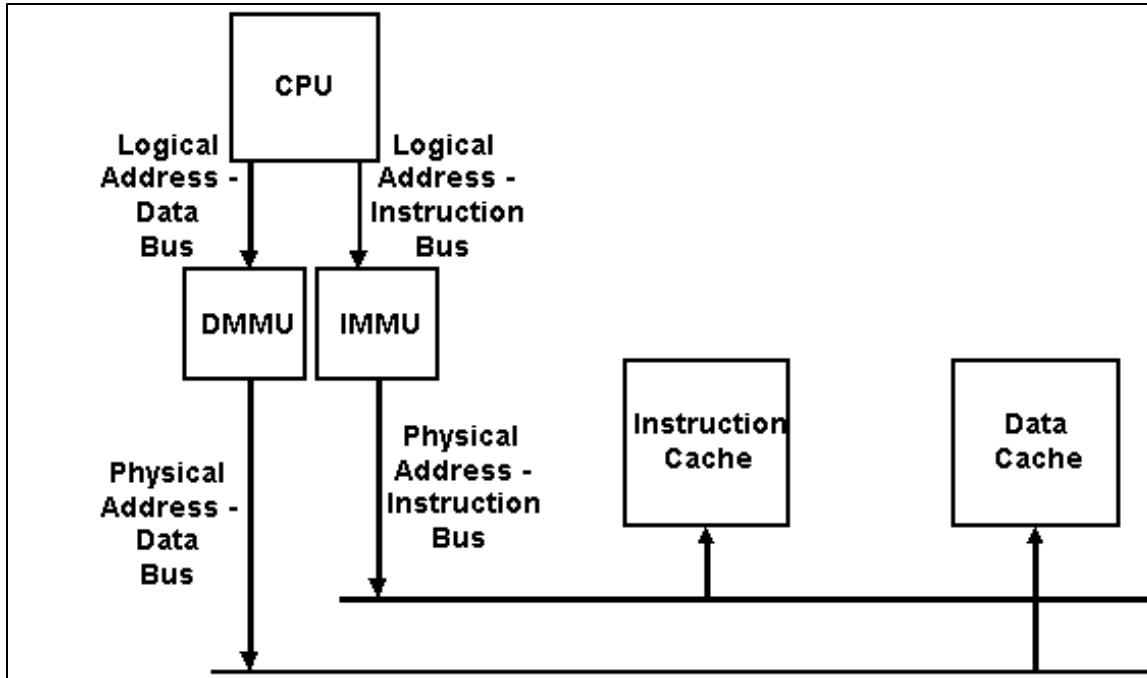
2.1.3. Associative Memory: Instruction and Data Caches

The instruction and data access times can be reduced significantly if referenced items are located directly by eliminating address decoding. This can be accomplished if the address of a referenced item is used as content to access an associative memory. The associative memory is also called content-addressable memory or cache, and is implemented with SRAM technology. In the SRAM, a value is stored on a pair of inverting gates. It is fast, but takes more space than the DRAM. In the DRAM, a value is stored as a charge on a capacitor, and therefore, it must be refreshed to be valid. The DRAM needs less space than the SRAM, but it is 5 to 10 times slower. The concept of cache was introduced by Maurice Wilkes in a 1.5-page paper of 1965. In a content-addressable memory, each item is stored along with its ID (identification number). For the sake of simplicity, we assume that this ID is the address of the item. When looking for an item, its address is sent to the associative memory and is compared in parallel to all addresses stored in it; for this to happen, a comparator circuit is associated with each location. Another circuit identifies the location of the comparison match, if the item is present, and marks it for subsequent reading. Because of the need for an exclusive comparator per location, the cache is more expensive than the DRAM. In the SRAM, a value is stored on a pair of inverting gates. It is fast, but takes more space than the DRAM. In the DRAM, a value is stored as a charge on a capacitor, and therefore, it must be refreshed to keep it valid. It needs less space than the SRAM, but it is 5 to 10 times slower.

The cache is attached to the processor bus, like the main memory. However, it is physically closer to the CPU and has shorter response time than the latter. Because of its much higher cost, only a few thousands of items are normally stored in the cache. These items are also present in the main memory, according to the data inclusion property which states that a level closer to the CPU contains a subset of the items stored at a level farther from it in the memory hierarchy of the computer. When the CPU looks for an item which is present in the cache, then the cache responds faster than the main memory; the CPU then accepts the data and terminates the memory bus cycle. Otherwise, the CPU waits until the main memory eventually responds to the request (a watchdog timer is used to terminate the bus cycle if the main memory does not respond within a prespecified number of clock cycles). Separate instruction and data caches are frequently present in computer systems. This separation is very beneficial for processors that implement the Harvard architecture which assumes two separate processor busses for instructions and data. There exist two main types of caches, namely virtual and physical caches. They accept virtual and physical addresses, respectively. Figure 3 shows these two types of caches for the Harvard architecture. Physical caches result in longer access times for items which are present because of the need to translate virtual addresses into physical addresses. On the other hand, virtual caches must be completely flushed (that is, emptied) when switching between tasks because each program uses its own virtual addresses. In contrast, all items are still valid for newly arrived tasks in the case of physical caches.



(a)



(b)

Figure 3: The Harvard architecture. Only the address busses are shown. (a) Virtual caches. (b) Physical caches. (DMMU: Data MMU. IMMU: Instruction MMU.)

Since the size of a cache is much smaller than the size of the main memory, due to its much larger cost, a cache is used to keep close to the CPU only those data and instructions that have high probability to be referenced in the near future. Techniques have been developed to decide each time what subset of the instructions and data to keep in the cache if it overflows. These techniques are based on heuristics, such as FIFO (First In, First Out), least frequently used, first-in not-used first-out, etc. Actually, entire blocks are transferred each time from the main memory based on the locality of references property which states that items in the neighborhood of the currently referenced item have very high probability to be referenced in the near future; therefore, their presence in the cache may speed up subsequent accesses. Four basic techniques exist to decide where to store a newly arrived block from the main memory. They are called direct mapping, associative mapping, set-associative mapping, and sector mapping. Associative mapping is the technique that implements the content-addressable memory concept in its pure form. The other techniques have comparable performance but reduce the cost of the cache. For the sake of brevity, we omit the description of these techniques.

Two techniques are used to deal with memory write operations when caches are involved. According to the write-through technique, a new value produced by the CPU for a program variable is copied simultaneously into both the cache (if the variable is present there) and the main memory. In contrast, the write-back technique stores the produced value only in the cache to speed up the operation; the new value is finally copied from the

cache into the main memory at task switching time or when the corresponding block in the cache has to be overwritten. Finally, evaluating the performance of a cache system is not an easy task. It may involve a combination of probability theory and simulations. Advanced caches have a hit ratio (that is, the probability that a referenced item will be found in the cache) of more than 98%; of course, this is true for well structured programs with regular instruction and data reference patterns.

2.1.4. Peripheral Devices

The I/O (Input/Output) components of computer systems, which are also known as peripheral devices, often account for most of their cost. The processor and main memory cost up to only 20% of the total cost in typical PC systems. The monitor and hard disk, which are I/O devices, cost much more. The monitor is often the most expensive component of the PC. Other peripheral devices are, among others, magnetic-tape drives, keyboards, plotters, printers, CD-ROM drives, modems, and floppy disk drives. Since all peripherals contain electromagnetic or electromechanical components, they operate at slower speeds than the CPU.

An I/O bus may connect the CPU to the peripherals. Each peripheral has its own interface unit attached to this bus. The interface unit decodes the addresses of peripherals broadcast by the CPU, recognizes commands sent to/from the peripheral, and produces the appropriate control signals for the peripheral/CPU. It also synchronizes the data transfers from/to the peripheral. Sometimes a group of peripherals share the same interface unit. The I/O bus is often identical to the memory bus. Computers having separate I/O and memory busses contain processors that have separate memory and I/O access instructions. With a single bus, processors map the peripheral devices to memory addresses, and the interface units of the latter devices recognize commands destined for the peripherals. Memory read or write operations are then used to communicate with these devices. This is called memory-mapped I/O.

DMA (Direct Memory Access) controllers are the interfaces of peripheral devices which are used to transfer large amounts of data from/to sequential memory locations. The CPU initializes a DMA transfer by sending to the interface unit the starting address and the size of the data, as well as information about the type of operation to be performed (that is, input or output). This technique allows the CPU to perform other tasks during the actual transfer. In the cycle stealing technique, the DMA controller gets hold briefly of the memory bus when the CPU releases the bus to carry out internal operations; this technique overlaps CPU operations with DMA data transfers and repeats until operation completion. Another DMA technique runs in the burst mode, where the DMA controller holds the memory bus until operation completion. Each peripheral of some computers contains a distinct I/O processor that controls all transfers from/to the peripheral. This technique is used to form "channels" in IBM mainframes. The CPU and I/O processors exchange information through the memory. The CPU initiates a transfer by storing in the appropriate memory buffer for the peripheral the operation code, the starting memory

address, and the size of the data. Similarly, the peripheral copies its status into another buffer.

The readiness of peripheral devices to accept or receive data can be determined by the CPU through polling or interrupts. Polling is time consuming because the CPU has to check the status of all devices at regular intervals. The concept of interrupts is discussed in detail in Subsection 2.3. With interrupts, the peripheral device has to activate a CPU interrupt line (an incoming control signal). If several devices share the same interrupt line, then the interrupt service routine run by the CPU uses polling to determine the appropriate device. The only exception is daisy chaining where the devices are accessed sequentially by hardware based on their interconnection in a linear array fashion. If two or more devices are active simultaneously and use the same interrupt line, the priority is determined by a hardware encoder.

2.1.5. Resource Connectivity

A processor bus consists of a set of buffers, wires, and tri-state gates. Many outputs can be connected together as long as only one of them is in an active state (that is, 0 or 1) at a time. In addition to the processor bus, the PC expansion bus is used to attach expansion boards to the processor bus. Busses can be either synchronous or asynchronous. The former busses use a distributed clock mechanism while the latter busses employ handshaking (the receiver has to send an acknowledgment signal to the sender). It is generally difficult to design good busses that avoid bottlenecks. There exist limitations in the number of interconnected devices, the length of the busses, the support for various classes of devices, and the cost. Resolving collisions also contributes to the cost and may introduce extra overhead.

Let us present briefly some of the other important busses and interfaces (ports) present in PCs. The serial port uses the RS-232 protocol of 1969. It can transfer data with as low a speed as 110 baud (bits per second) and as high as 115 Kbaud. The parallel port is used for 8-bit (byte) parallel transfers. It was introduced for speeds of up to 150 Kbytes per second. With the newer standard, it can reach speeds of up to 1 Mbyte per second. The Peripheral Component Interconnect (PCI) was introduced by Intel in 1993 for the Pentium processor and a 32-bit data bus. It can operate with a frequency of up to 33 MHz and can reach the transfer rate of 133 Mbytes per second. It is implemented with a chipset. The Advanced Graphics Port (AGP) introduced by Intel in 1997 is a specialized PCI bus for graphics accelerators. It can transfer up to 533 Mbytes per second. Numerous types of devices can be connected to the Personal Computer Memory Card International Association (PCMCIA) interface. The Universal Serial Bus (USB) can transfer up to 12 Mbits per second. It is for low performance PC peripherals (pointing devices, keyboards, modems). It has only four wire connections.

2.2. Computer Performance

Some computer systems have better performance than others for some applications. The performance of a computer generally varies with the application domain. The performance is measured according to the program turnaround time and/or the machine throughput. Either real applications or benchmarks must be used to evaluate the computer's performance. Benchmarks are synthetic programs typical of the expected workload. These benchmarks are different for different application domains. The SPEC (Standard Performance Evaluation Corporation) benchmark suite has been developed by an nonprofit organization that was formed by a group of computer companies. For example, SPECCapc is used to measure the performance for graphics operations, SPEChpc primarily for parallel and distributed computer systems, SPEC CPU2000 for CPU-intensive operations, SPECjbb2000 for server-side Java applications, and SPECweb99 for World Wide Web servers. For a given computer architecture, the performance increases with clock rate increases, CPI (cycles per instruction) reductions through processor improvements, or instruction count or CPI reductions because of compiler enhancements.

The CPI measure is defined simply as the ratio of the cumulative number of clock cycles for all instructions in the instruction set, divided by the total number of instructions. However, it is more precise to calculate the CPI based on the frequency of using the different instructions in the instruction set. Processors and complete computer systems are conveniently evaluated according to their MIPS and MFLOPS rates. These rates are derived for specific benchmarks or applications. MIPS and MFLOPS stand for million instructions per second and million floating-point operations per second, and are used to evaluate the performance for operations on integers and real numbers, respectively. The MIPS rate for a given program is derived as the total number of instructions in the program divided by the product formed by the total execution time expressed in seconds and 10^6 . The MFLOPS rate is derived similarly.

The actual performance of a computer for a given program depends on the instruction dependencies in the program (that is, the behavior of the program), the architecture of the processor (e.g., the size of its pipelines and the size of the on-chip cache), the architecture of the complete system, the silicon technology used, etc. An instruction dependence may assume the form of a data, resource, or control dependence. Two instructions are data dependent if they use the same variable as input and/or output. They are resource dependent if they use the same hardware resource and control dependent if their order of execution cannot be determined at static/compile time. A reduced number of dependencies generally implies better performance.

2.3. Program Control

The instructions of a single program routine are stored sequentially in the main memory of a conventional computer. They are fetched in sequential order from the main memory for execution. After an instruction is fetched, the program counter (PC) register in the control unit of the CPU is incremented appropriately to point to the next instruction in the routine. However, some instructions may change the value of the PC to point to a random

instruction in this routine or in another routine. That is, there exist program control instructions that change the value of the program counter, thus forcing the CPU to branch elsewhere in the program. There exist conditional and unconditional branch instructions. The former instructions look at the status of a bit in the CPU; for example, if the preceding instruction performed a subtraction, then a conditional branch instruction may force program branching if the result was zero.

Procedure calls and interrupts (also known as exceptions) also force interruption in the sequential execution of the program. However, the return address is stored in a special part of the memory called stack, in order to be able to return to the execution of the program after the exception has been serviced or the called routine has been executed. Procedure calls are implemented with program instructions and the last instruction in a called procedure is the "return from subroutine" instruction that restores the value of the PC. However, either an internal or an external event forces the processor to branch to an interrupt service routine (ISR). These events are normally unexpected. Either they are not supposed to occur under normal execution or it is not known at static time when they are going to occur. An example in the former category is division by zero during a program run while an example in the latter category is an unexpected, sudden power down of the computer. The ISR in the first case may print an error message for the user to see, while in the second case the current state of the CPU will be saved in the memory so that when the system is powered up again execution can resume from where it stopped.

The state of the CPU is determined each time by the PC content, all other CPU registers, and all status flags (one of the flags determines the mode of operation, being either the user or supervisor mode). This collection of information is often called program status word (PSW). An interrupt is always serviced only after the currently executing instruction is completed. A software trap also causes an internal interrupt. For example, respective instructions are normally used when a program wants to access a peripheral device. For security reasons, the operating system must first be invoked through a software trap. A "return from interrupt" instruction is the last instruction executed in the ISR. It forces the CPU to restore its state and resume execution of the interrupted program.

2.4. System Software

There exist six fundamental classes of system software. (a) Language processors that translate programs written in symbolic languages into machine-readable (object) code. There exist two basic types of language processors, namely compilers and assemblers. Compilers assume programs written in high-level languages while assemblers translate programs written in assembly languages. (b) Library program routines that can be invoked by application programs to perform specific tasks, thus avoiding recoding for widely used tasks and facilitating easier programming. (c) Loaders that store machine-readable code into the computer's memory for execution. (d) Linkers that combine object codes of different routines to form a single executable code for the loader. (e) Operating

systems that manage the computer's resources effectively at run time. (f) Finally, programs that monitor the computer's activities.

Operating systems play a very significant role. They provide the appropriate platform for the execution of application programs by managing the computer resources effectively. They allocate disk and memory space to user programs at static and run time, control peripheral devices by recognizing input commands or sending data/commands to output devices, attempt to maximize the utilization of devices which are closer to the CPU for better performance, assign appropriate privileges to user programs for data/routine/device access, identify faulty components, and check for erroneous results. Many of the operating system routines are memory resident and are invoked as needed. For very advanced computers, the operating system has more responsibilities. For example, in multi-user computer environments it protects user programs and data from interference and assigns resources to user programs in a fair, time-shared manner.

3. Parallel Computers

Parallelism in the computing field has been around since the early days. It has assumed several forms, such as instruction lookahead, pipelining, data parallel operations (such as, vector operations), multiscaling, multithreading, etc. However, the term parallel computer is distinctly used for computers that contain more than one processor. There exist three classes of parallel computers, namely multiprocessors or shared-memory computers, multicamputers or message-passing computers, and vector supercomputers. Hybrid versions also of these classes exist, but only the pure classes are discussed in the following subsections. The main idea behind the design of these computers is to connect many processors (and their memories) together to derive very high aggregate performance. The term supercomputer is assigned indiscriminately nowadays to any type of parallel computer that contains dozens, hundreds, or thousands of processors.

Michael Flynn has classified computer architectures based on the numbers of instruction and data streams that can be active simultaneously. He introduced four models. The SISD (single-instruction stream, single-data stream) model comprises all sequential computers, the SIMD (single-instruction stream, multiple-data streams) model is for special-purpose parallel computers that apply the same operation simultaneously on different data, the MISD (multiple-instruction streams, single-data stream) model for systolic-array computers that apply different operations on the same data, and the MIMD (multiple-instruction streams, multiple-data streams) model which is the most powerful general-purpose model for parallel computers. SIMD computers contain a single control unit that fetches one instruction at a time, decodes it, and then sends the same control signals to all processing elements (i.e., processors) in the machine; these processors do not comprise a control unit. SIMD computation appears in applications that use vector/array data structures. Many engineering and scientific applications can benefit from these computers. MIMD is the other widely used model. It assumes many independent processors (with their own control units) that run their own

programs/routines and are capable of exchanging information as implied by the application.

Three techniques exist to develop compilers for parallel computers. The first technique uses a sequential compiler and a preprocessor; the preprocessor accesses a library of object-code routines that implement parallel constructs and operations for the parallel computer. On the other hand, a precompiler performs limited program flow analysis and instruction inter-dependence checking to optimize pieces of the code for parallel execution. Finally, a parallelizing compiler detects parallelism in the source code and changes the code from sequential into parallel form. However, it is very difficult to develop fully parallelizing compilers.

Amdahls' law is used to predict the performance of parallel computers for applications with fixed workload (that is, the total number of operations is independent of the number of processors used). Assume that the number of processors in the computer is n . The law simply states that the speedup S_n with n processors, when compared to the execution of the same program on the uniprocessor, is given by

$$S_n = 1/[a + ((1-a)/n)]$$

where a is the portion of sequential code in the entire program. Therefore, the upper bound on the speedup is $1/a$. Since a has often a large value (e.g., 0.2), applications with fixed workload cannot benefit from continuous increases in the number of processors. Amdahl had concluded that the fraction of the computational load associated with data management housekeeping had been almost constant for about ten years, and accounted for about 40% of the executed instructions. This overhead appeared to be sequential in nature. He decided that in special-purpose environments this overhead could not be reduced by a factor of more than three. Therefore, he concluded that it was not important to build large parallel computers.

On the other hand, Gustafson's law is applied to scaleable applications. The workload in these applications can increase linearly with increases in the number of processors. For example, they can take advantage of the additional processors to produce better accuracy in the result. Many engineering and scientific applications are scaleable, such as weather modeling, computational fluid dynamics modeling for aircraft design, air pollution modeling, drug design, DNA folding, etc. The speedup of scaleable programs for n processors is given by $S_n = a + n(1-a)$, therefore it does not have an upper bound. For this reason, many massively parallel computers with thousands of processors have been built. This trend started in the mid 1980s and continues to our days with the involvement of major computer manufacturers, such as IBM.

3.1. Multiprocessors

Multiprocessors or shared-memory parallel computers contain main memory which is shared by all processors. Examples of commercial multiprocessors are the BBN TC-2000, Sequent Symmetry S-81, and SGI 2800. There exist three basic multiprocessor models, namely UMA (Uniform Memory Access), NUMA (Non-UMA), and COMA

(cache-only memory architecture). UMA machines support uniform sharing of all information in the memory by all processors. That is, the access time does not depend on the location of the data in the memory. Figure 4 shows the UMA model; there exist several system interconnect structures, such as the bus, multistage networks, and the crossbar. In contrast, the access time depends on the location of the data for NUMA machines (see Figure 5). The task assignment problem is trivial for UMA and difficult to solve for NUMA machines. There exist several processor interconnection networks, such as the crossbar, mesh, torus, binary hypercube, etc. Multiprocessors with caches attached to individual processors suffer from cache coherence problems. Several techniques have been developed to assure that all copies of the same variable throughout the machine have the same value during program execution.

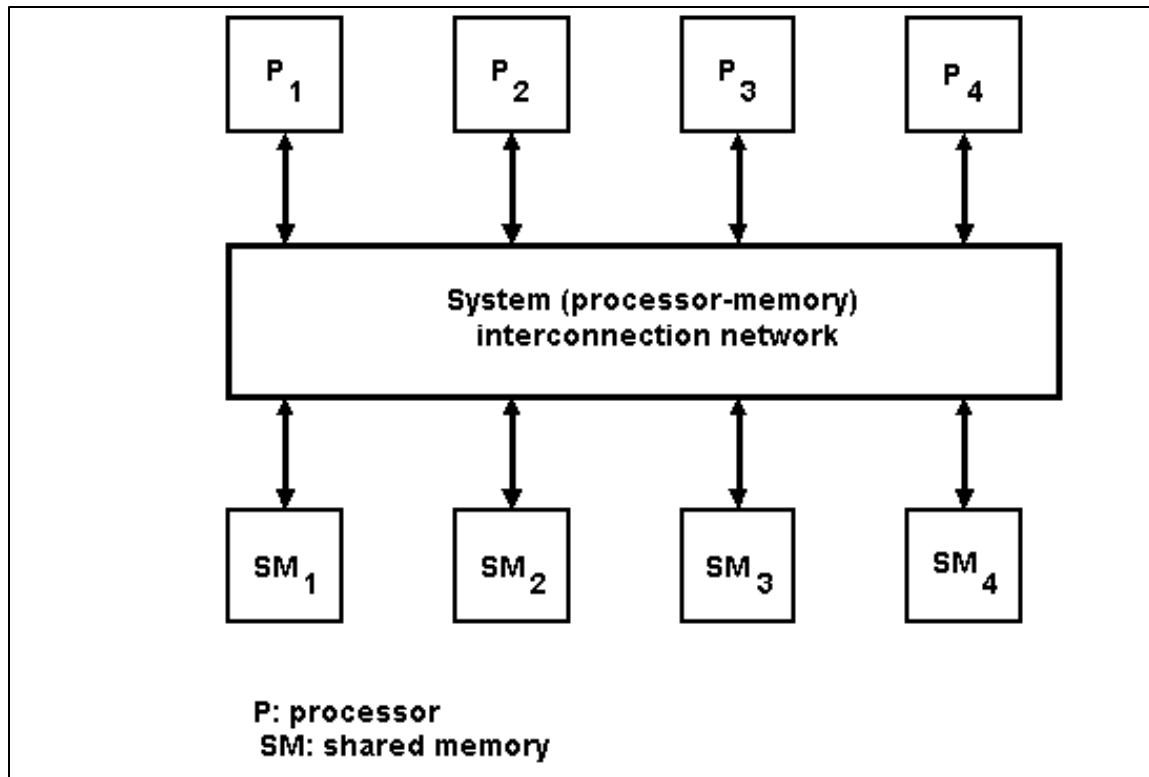


Figure 4: The UMA multiprocessor model for four processors and four memories.

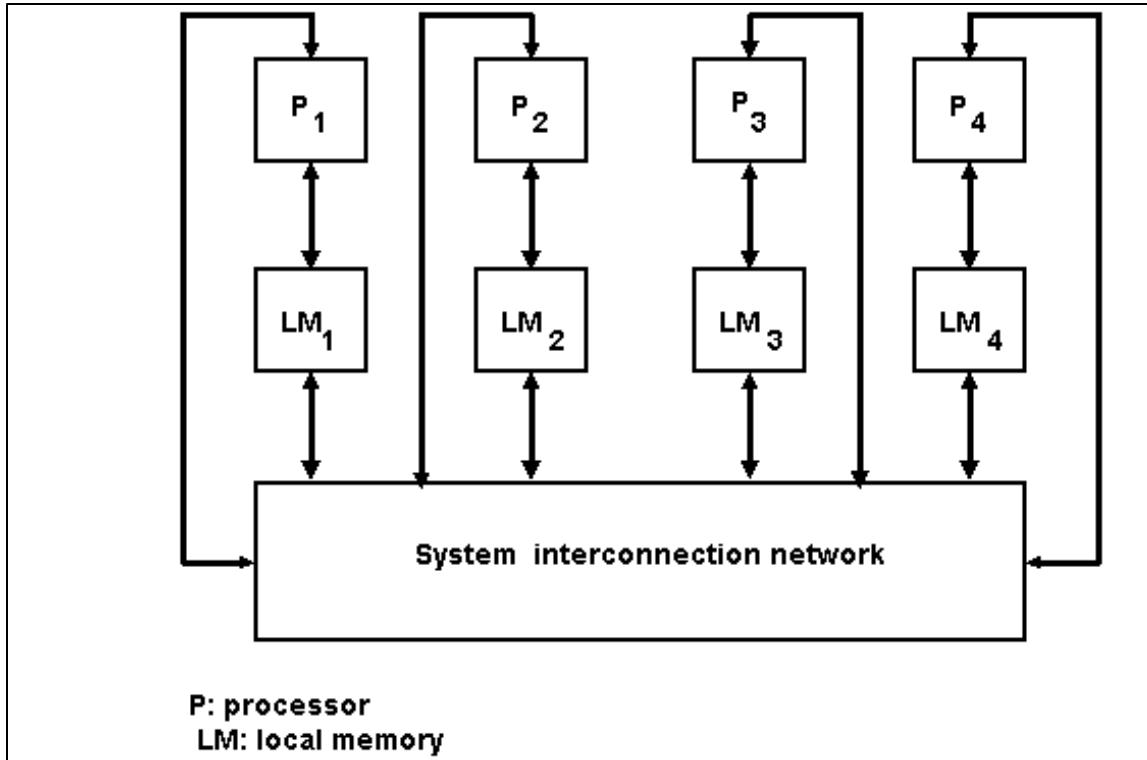


Figure 5: The NUMA multiprocessor model for four processors and four memories.

3.2. Multicomputers

In the case of multicomputers, a static point-to-point processor interconnection network is used to connect processors together. Processors exchange information by sending messages to each other. These messages may have to go through other processors before they reach their destination. There exist numerous processor interconnection networks for multicomputers. However, the two- or three-dimensional torus is the most commonly used network nowadays. The torus is a grid with wraparound connections at all its ends. Multicomputers do not normally suffer from traffic congestion for uniform communication operations because each processor has direct access only to its neighboring processors. In contrast, all processors may have to go simultaneously to the same shared memory in multiprocessors.

It is difficult to develop well-balanced programs for high performance on message-passing computers. Also, program debugging is not easy. In contrast, it is much easier to write programs and debug them on shared-memory computers. For this reason, current parallel computers that contain a message-passing interconnection network also provide support for shared memory. Specialized hardware and software on these distributed shared-memory (DSM) parallel computers allows programs to use instructions for either the message-passing or shared-memory model. Each processor on a DSM computer

contains its own local memory. However, it is part of the global memory that can be accessed by all processors.

Examples of pure multicomputers or DSM parallel computers are the Caltech Cosmic, Intel iSPC/1, IBM RS/6000 SP, nCUBE 10, Parsys SuperNode 1000, Thinking Machines CM-2 and CM-5, Cray MPP T3D and T3E, and Intel Paragon. Software was used back in the 1980s to implement the message-passing model. This slow implementation was replaced in the early 1990s by specialized fast routers that were attached to the processors. Also, the technique of wormhole message routing was introduced to implement pipelining for message transfers. Therefore, the total communication time for large messages became highly independent of the traveled distance.

3.3. Vector Supercomputers

Vector supercomputers apply the same operation simultaneously to many or all elements of vector data (i.e., arrays of elements). A vector instruction is implemented with a software loop on a conventional machine, while specialized primitive processors are used to implement in one step the corresponding vector operation on a vector supercomputer. Examples are the Cray Research supercomputers Cray-1, Cray-2, X-MP, and Y-MP, the Japanese supercomputers NEC SX/2, Fujitsu VP200, and Hitachi S820, and the mini-supercomputers Convex C-1 and C-2. Specialized vector hardware can also be found on some mainframes, minicomputers, multicomputers, advanced workstations, and multimedia processors.

There are two types of vector supercomputers corresponding to the exclusive use of memory-to-memory or register-to-register vector operations. In the former case, all input vectors are fetched by the instructions from the memory and the vector results of the instructions are also stored directly into the memory. In the latter case, the input vectors are first stored in vector registers in the processor and the result of any operation, other than a load or a store, is stored in a vector register. A vector supercomputer is attached to a scalar computer like a co-processor, that is instructions are fetched by the scalar control unit. Vector supercomputers started with uniprocessor models (Cray-1 in 1976) and they now contain more than one scalar processor. Vector supercomputers implement superpipelining, where each pipeline stage is further divided into substages for a faster system clock and a higher degree of parallelism. Also, the term supercomputer is associated nowadays with any parallel computer that can achieve extremely high performance when compared to a workstation or a minicomputer.

Bibliography

Amdahl G.M. (1967). Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities. *Proceedings of the AFIPS (American Federation of Information Processing Societies) Conference*, pp. 483-485. [This paper introduces

Amdahl's law that determines an upper bound on the speedup for the execution of programs with fixed workload on parallel computers.]

Amdahl G.M., Blaauw G.A., and Brooks F.P., Jr. (1964). Architecture of the IBM System/360. *IBM Journal of Research and Development*.

Russell R.M. (1978). The CRAY-1 Computer System. *Communications of the ACM* **21**(1), pp. 63-72.

Dennis J.B. and Misunas D.P. (1974). A Preliminary Architecture for a Basic Data-Flow Processor. *Computer Architecture News* **3**(4), pp. 126-132. [The first paper to describe the architecture of an entire processor based on the dataflow execution model. It differs from the conventional model which is control (or program-counter) driven.]

Moore G.E. (1965). Cramming More Components Onto Integrated Circuits. *Electronics*, pp. 114-117. [It introduces Moore's Law that governs advances in processor performance based on the expected degree of integration for silicon transistors.]

Goodman J.R. (1983). Using Cache Memory to Reduce Processor-Memory Traffic. *Proceedings of the 10th International Symposium on Computer Architecture*, pp. 124-131. [It introduces the concept of including cache in computer system design.]

Seitz C.L. (1985). The Cosmic Cube. *Communications of the ACM*, pp. 22-33. [Describes a processor interconnection technology that became very popular in the 1980s.]

Author Biography

Dr. Sotirios G. Ziavras received in 1984 the Diploma in Electrical Engineering from the National Technical University, Athens, Greece. He received in 1985 the M.Sc. in Electrical and Computer Engineering from Ohio University, Athens, Ohio, USA. He was awarded in 1990 the Ph.D. in Computer Science from George Washington University, Washington, D.C., USA, where he was a Distinguished Graduate Teaching Assistant and a Graduate Research Assistant. From 1988 to 1989, he was also with the Center for Automation Research at the University of Maryland, College Park, Maryland, USA. He was a Visiting Assistant Professor in the Electrical and Computer Engineering Department at George Mason University, Fairfax, Virginia, USA, in Spring 1990. He is currently an Associate Professor of Electrical and Computer Engineering, and Computer and Information Science at New Jersey Institute of Technology, Newark, New Jersey, USA. His major research interests are processor and computer systems designs, and parallel computing systems and algorithms.