Chapter 1:

Parallel Algorithms Fundamentals

Section 1:

1.1 Parallel Models

1.1.1 Random Access Machine model for sequential algorithms

The architectural and operational models of computers help us to comprehend the complex behavior of computer systems. Different models exist for different levels of the architecture and the operation of computers as well as for the interconnection networks in parallel systems. We will focus on the models – be they belonging to those of architectures or algorithms of operation – that are important in order to understand the behavior of parallel algorithms and also the design and development methodologies for mapping them onto given architectures.

An early model for the sequential algorithm and program development has been the **Random Access Machine (RAM)** model. It is a simplified model that helps us to examine the basic parameters for sequential algorithms independently of the computer itself. It is based on a simple execution platform (the computer) having a central processing unit and a random access memory (like the one that will be explained later in Figure 1.1.3.a.), and some input-output means. The **running time** for a sequential algorithm is very easy to estimate with the **data-flow diagram of the algorithm** and the RAM model. Supposing the so called **uniform cost criterion** is valid for this model, the *running time* of a sequential algorithm is expressed in terms of the number of **basic operations** that are needed by the algorithm as a function of the input size n (number of input items). Uniform cost criterion states, that all basic operations have of equal cost value (execution time value), the cost does not depend on the data size (word length) and the operation type. Basic operations include basic arithmetic and logic operations on scalar values (e.g. addition, subtraction, multiplication, division, comparison of two numbers, bit-wise logic operations like AND, OR, XOR) and also memory read and write operations.

The data-flow diagram of the algorithm (soon we introduce such a model, the so-called directed acyclic graph model) directly represents the basic operations, thus the *running time or* **cost function of the algorithm** may be calculated based on this model.

1.1.2 Parallel model types

Modeling parallel computation is a task incomparably more difficult than the one in the case of sequential algorithms and computers. We have to find models that well fit *for describing and analyzing parallel algorithms*.

Several (partly conflicting) requirements the model has to satisfy. Some of them:

- Simplicity: This is a very important feature that allows easy description for the behavior and operation of the algorithm and also analysis of the performance like execution time, communication work, memory utilization. It should be independent of the *hardware architecture*, as much as possible.
- Implemantability: The algorithm, which is to be developed, based on this model should be easily transferable, implementable on parallel computers, at the condition, that basic parameters estimated using this model should remain valid after the implementation.

So far, all satisfactory models that are accepted by most researchers do not exist. Below, we briefly describe some of the models based on literature. These models include the **Parallel Random Access Machine (PRAM), Directed Acyclic Graphs, Shared memory model, and Network model.** These models suffer from oversimplification, in terms that important architectural features are neglected or not included into the model description.

That is why further in this chapter we attempt to have a closer glance into those implementation issues, that basically have to be taken into account when evaluating a parallel algorithm or mapping the algorithm onto an existing parallel architecture with a given communication network topology.

1.1.3 Parallel Random Access Machine (PRAM)

This is a theoretical model; an idealized computer based on global (shared) memory. PRAM is the extension of the *sequential RAM model*, where the single processor is replaced by *p* processors, and naturally an interconnection network. Processors access the same memory using the same address space. Neither the memory storage limits nor the communication bandwidth limits of the interconnection network are considered, i.e. communication takes place at no cost. Processors have a global synchronization clock, however they may execute different instructions. A processor executes one basic instruction in a time unit (clock cycle). PRAM models can be regarded for synchronous Multiple-Instruction-stream Multiple-Data-stream (MIMD) computers.

Since the model uses a shared memory, more than one processor may access the memory simultaneously. Based on the way as simultaneous memory access is handled, four subclasses exist for the PRAM model:

- *EREW* exclusive read, exclusive write *PRAM:* No concurrent read or write operation is allowed to the same memory location.
- C*REW* concurrent read, exclusive write *PRAM:* Multiple read operations (no destructive memory operation) are allowed; however write operations have to be serialized if approaching the same memory location.
- *ERCW* exclusive read, concurrent write *PRAM:* Concurrent read is not allowed, however concurrent write operations may be executed for the same memory address in this model.
- C*RCW* concurrent read, concurrent write *PRAM:* In this class both read and write operations are allowed to be executed concurrently to a common memory location by many processors. The most powerful PRAM model.

Because read operations do not change the content of the location, there is no problem with them. However, concurrent write needs some arbitration protocol. The most popular protocols are:

- *Common,* the (concurrent) write operation is allowed only if all processors want to write the same value.
- *Arbitrary,* in this case an arbitrarily selected processor may write its data value to the memory, others fail.
- *Priority,* processors have a predefined priority structure, and the currently highest priority processor succeeds with write operation, others fail.
- *Sum:* an associative operator for the data coming from the concurrently approaching processors (e.g. summing up all data, selecting the maximum, etc.) is defined for the concurrent write operation; it is executed and the result of the operation (the 'sum') is written to the memory location.

1.1.4 Directed Acyclic Graphs

Directed acyclic graphs (DAGs) are data-flow graph models for parallel algorithms. A DAG does not consider architectural features, yet a lot of computations may be presented by it in a natural way. The nodes of the DAG represent basic operations; while operands come via directed arcs form the other nodes. Inputs are shown by nodes without incoming arcs, and outputs are nodes whose outdegree is zero. Internal nodes have an indegree value of at most two. Incoming and outgoing arcs define the **precedence constraints** in which the operations have to be executed.

DAGs may represent computations without branches. A DAG with *n* input nodes reflects the computation operations for input size *n*. An algorithm with variable input size is represented by a family of DAGs *{G(n)}*, in which *G(n)* is an algorithm with a given input size. In the case of branches the graph can be duplicated, the sequence of its operations repeated. Unit cost criterion is assumed to be in force, and the execution time of an operation represented by a node takes a unit of time.

The DAG model may be used to analyze the performance of parallel algorithms if interprocessor communication and data accesses are at no cost. In some particular cases **scheduling** can be represented in a way that the task of each node is assigned to a particular processor. An extension of this latter case is the model where each node may posses a pair of parameters, the processor identification (index) and the time, which should be devoted to the node operation. Further extension of this model is when each node (subtask) assigned to a processor represents a subgraph of the subtask.

Performance analysis via DAGs is a straightforward process. For a given schedule the execution time is the maximum of time calculated by all of the time and precedence constraints along a path. Considering several implementation versions, i.e. schedules, the **parallel complexity** of the algorithm is defined by the minimum of overall schedules that use *p* processors.

Figure 1.1.1 represents a very simple task: summing up 8 data items with two schedules: a not optimal (serial one) and a parallel one.

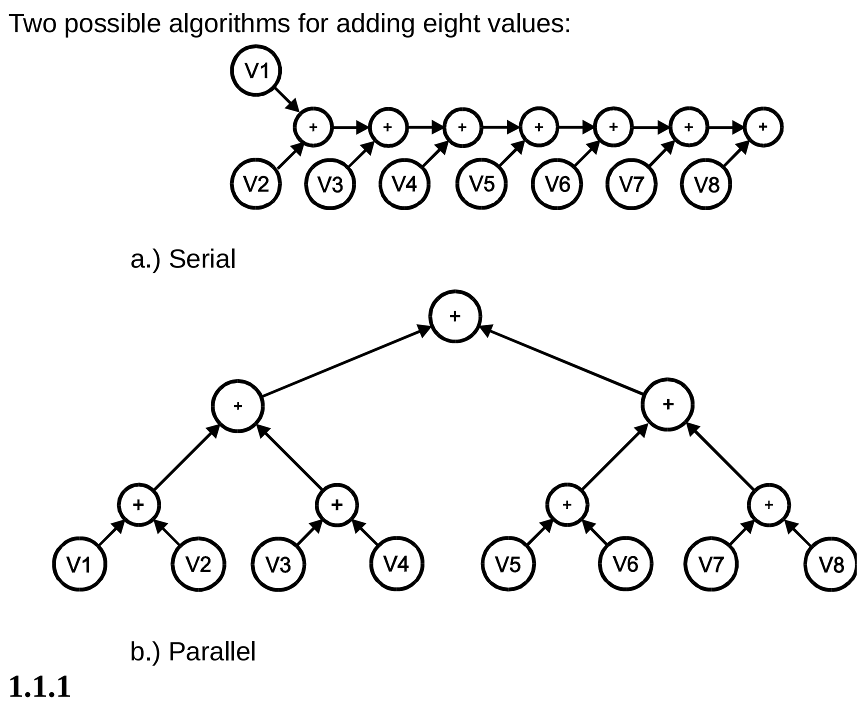


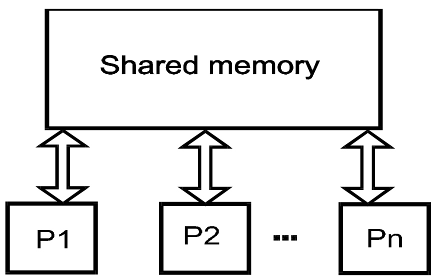
Figure 1.1.1

1.1.5 The Shared Memory Model

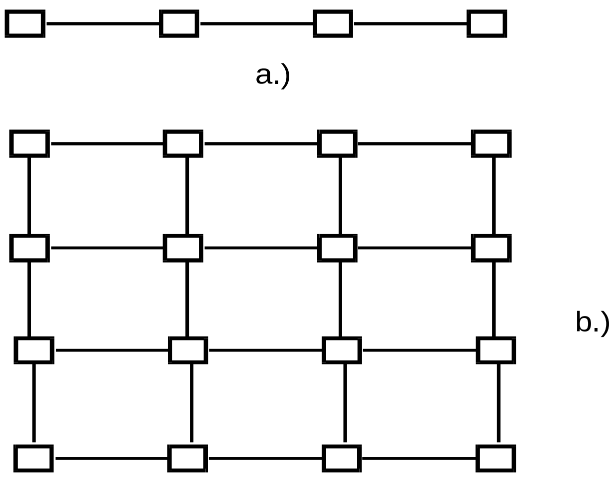
This model (in upper half of Figure 1.1.2.) is very similar to the PRAM model. It consists of *p* processors, each of them may have its own local memory, own local program stored in local memory. They are connected to a **shared** or **global memory** and communicate to each other, exchange data via this shared memory.

Two basic modes of operations may be defined for the model:

- *Synchronous,* when all the processors work under the control of one global clock. This is exactly the PRAM model.
- *Asynchronous,* each processor has its own control clock signal. In this case processors operate independently from each other between two data exchange operations. Thus it is the programmer's or program responsibility to set synchronization points when data exchange is requested. At these points the processor reaching the point has to wait (become idle) until the second one also gets ready for communication. The shared memory model reflects the **multiple-instruction-stream multiple-data-stream** type of operation. For the algorithm the amount of data which is transferred to and from local and global memories is the amount of the **communication** required.



The Shared-memory model



Networks: a) linear array, b) 4 x 4 mesh

Figure 1.1.2

1.1.6 The network model

As its name suggests this model pays attention to the topology of the parallel computer including both the number of processors *p* and the structure of the interconnection network.

The network itself may be modeled by a graph *G(N,E)*, in which each node of the graph $i \in N$ represents a processor, and the edges $(i,j) \in E$ connecting the nodes represent two-way communication links. Each processor has its attached local memory; no shared memory exists in the network.

Similarly to the shared memory model, the operation of the network model may be: *synchronous or asynchronous.* The latter one is commonly used.

Exchanging messages between each other coordinates the processors operation in asynchronous networks. This is the **message passing model.** For coordinating the exchange of messages two additional language constructs are required:

send (X,i)
receive (X,i).

Processor, which is ready to send message *X* to processor *Pi,* issues the **send** instruction. *Pi,* when receiving the message issues a **receive** instruction, suspending other activities until the data arrives, then stores it into its memory and resumes the suspended operation. Processors exchanging data do not need to be immediately adjacent to each other. If they are not directly connected, then messages travel through other processors (in more details see Chapter 1, Section 3.). The process defining the communication path for a message in the network is called **routing.**

Synchronous networks are operated by one general clock signal. In this case the total data-flow is synchronized through the network. This is the so-called **systolic operation** or **systolic paradigm.** These systems are usually very fine-granularity networks and used in special algorithms of **data-domain parallelism**.

The topology of the interconnection between the processors defines the network structure. Topology closely affects the parallel algorithm design and mapping onto the architecture. There are parameters for network evaluation. For evaluating the cost of communication one of the most important parameters is the maximum distance (in terms of connection links and nodes, the so-called **hops**). Two directly connected processors are at a distance equal to *1 hop*. The maximum distance between any of two processors within the network is the **diameter** of the network.

A network maybe characterized by the **maximum degree of any node.** For the evaluation of communication, network throughput and reliability, the **connectivity** parameter is important. It characterizes the multiplicity of paths between any two processors. The minimum number of arcs that must be removed to break the network into two totally disconnected parts defines **arc or edge connectivity**. A similar parameter is the **bisection bandwidth, which** is the number of communication links that have to be removed to cut the network into two separated parts.

Two examples of network model arrangements are shown in Figure 1.1.2.a and b.

1.1.7 Comparison of the parallel models

DAG models are very simple. However it could be very difficult to analyze the parallel algorithm, since these models in general do not represent scheduling and processor allocations. Substantial weakness of the model is that it does not have mechanisms to handle the communication issues among processors and between processors and the memories.

Because of the wide variety of network topologies it is very difficult to analyze the algorithms in general on the base of the **network model.** The model depends very much on the particular topology. Changes in the topology need totally different algorithms.

For a wide range of parallel algorithms the **shared memory model** provides a convenient way for algorithm development and evaluation. This model has been in use for a long time; well-developed techniques exist for it. Especially the **PRAM, which is the synchronous version of the shared memory model,** is popular. Its popularity is due to the following facts:

- Well-developed techniques and methods exist for a wide variety of parallel algorithm types, which help to analyze the algorithms
- Algorithm analysis is rather simple because communication and synchronization problems are removed from the model
- Operations may be examined in each time instant; processor-job allocation can be easily studied
- The PRAM paradigm is robust; many network algorithms may be derived from PRAM algorithms
- It is easy to incorporate synchronization and communication operations into the model
- PRAM algorithms may be applied to more general problems that can be thus easily analyzed.

1.1.8 Models for parallel computers

The models for parallel algorithms that have been discussed before have very little information on the real parallel computer and implementation issues that may fundamentally affect basic parameters and effectiveness of the algorithm when executed by the computer. In the following parts of this Section 1.1, we pay attention to models which contain more information on the communication, access methods between the processors and the memory modules, and also on the way how the interprocessor communication topology is organized.

The architecture of the sequential computers follows the basic idea introduced by John von Neumann. This is the well-known **single-instruction-stream single-data-stream (SISD)** computer organization. As ever since the introduction of stored program computers the speed and data storage capacity demands have been increased, a lot of improvements and modifications have been added to the hardware including the processor and memory organization and also their operations, but the fundamental ideas of John von Neumann remained in force. Figure 1.1.3 goes through the main issues of this evolution.

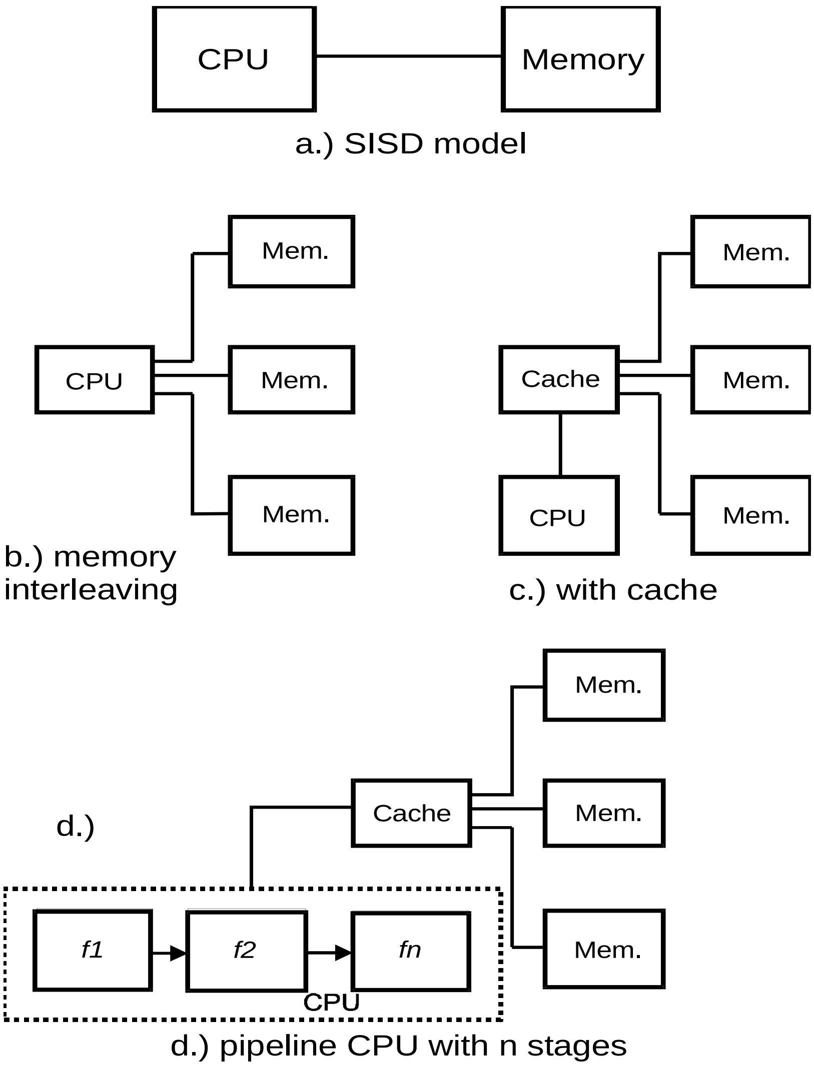


Figure 1.1.3

The original Neumann model contained two basic units: the Central Processing Unit (CPU) and the memory. (Naturally, that time CPU was shown as consisting of two modules, the Control Unit and Arithmetic Logic Unit, and also some means of input/output are requested. However, the CPU and the memory are those which define the computational power of this model). The computational model is based on a single instruction sequence fetched from the memory, and this sequence operates on a single sequence of data. The speed of operation of both the CPU and the memory are key issues to the execution time of the program (algorithm of operation). *Interleaved memory organization* and adding *cache memories* are two important steps on the way to increase the memory communication bandwidth, and *pipeline operation* has been implemented in processors to achieve higher throughput.

An alternate way for speed improvement is the use of multiple processors and memory units.

1.1.9 The taxonomy of parallel architectures

Parallel computers with multiple processors and memory units have a wide variety of architectures, they differ from each other along a lot of dimensions. Our taxonomy focuses on those classification dimensions, which fundamentally affect the behavior of the parallel algorithms. The most important ones are:

- the control mechanism of the processors' parallel operation
- the memory address-space organization, as the processors access the memory
- the interconnection network
- the granularity of the processors.

Control mechanism

P processors in a parallel computer can work under one centralized control unit: this is the **single-instruction-stream multiple-data-stream (SIMD)** model in (Figure 1.1.4.a.). In these computers a single control unit fetches the instructions and dispatches them to each processing unit. The only difference in the operation of the processors is that some of them may be selectively switched off during an instruction cycle. These computers are mostly used for **data-parallel programs** to implement **data-parallel algorithms.** One special purpose control unit controls the operation and many data processing processors execute the calculations. General-purpose microprocessors can not be applied for such computers as building blocks.

Another control possibility is the independent one for each of the processors: the **multiple-instruction-stream multiple-data-stream (MIMD)** model (Figure 1.1.4.b) works in this way. Each CPU has its own independent program. Naturally in MIMD computers the operation of the *p* processors is asynchronous, so they need some extra hardware and software means for synchronizing the data (and/or control) exchange actions. Cheap general-purpose microprocessors are suitable in MIMD computers.

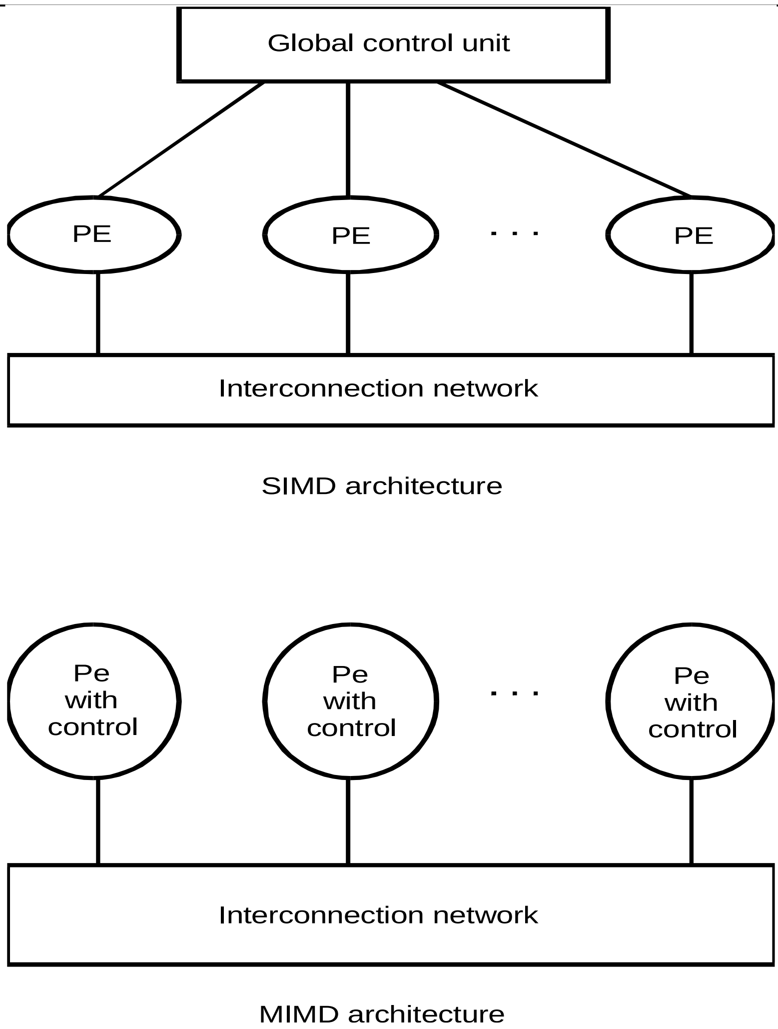


Figure 1.1.4

Address-space organization

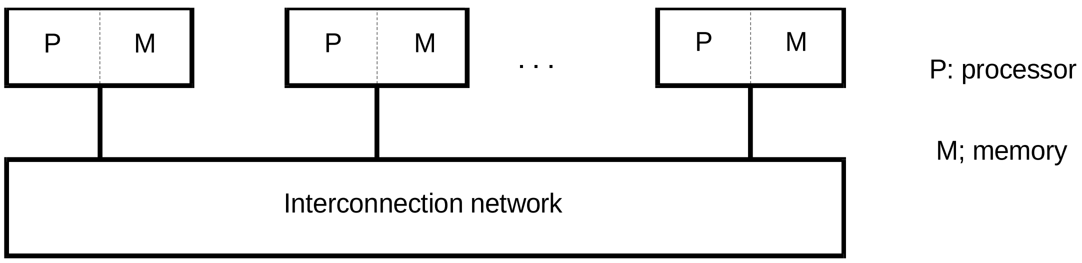
In parallel computers the processors have to communicate to each other. For data exchange either some storage locations in the memory, or input-output messages may serve. Considering how the memory and input-output address spaces are treated by the processors, we may distinguish between **shared address space** and **message passing** architectures.

There are hardware units in the **shared address space architecture** that allow read-write operations for all processors using these addresses. Processes communicate via data stored at given places in the shared address locations. These parallel computers are often called **multiprocessors.**

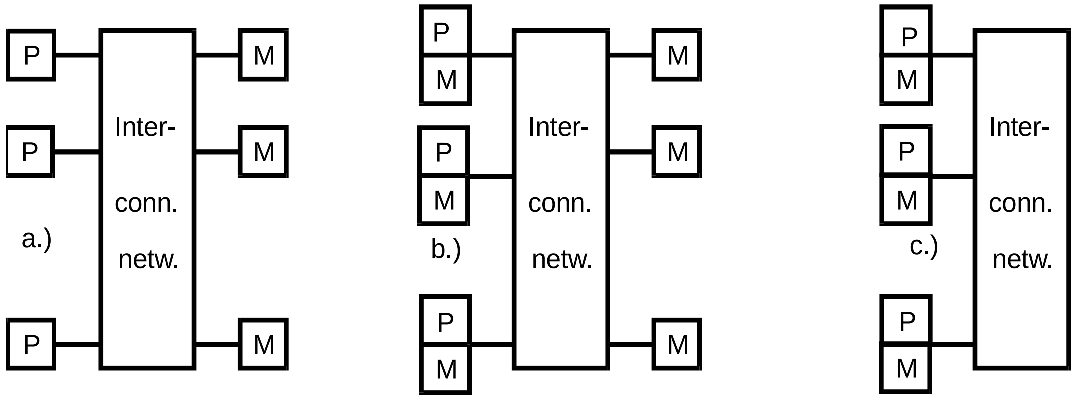
If the processors are provided with commonly accessible memory address space (with **common** or **global memory**) then we speak about **shared memory** parallel computers. The global memory communication tends to become a bottleneck in these computers. To reduce the effect of the bottleneck problem, **private** or **local memories** may be attached to the individual processors in order to store local programs and data.

In **message passing architectures** of the parallel computers an interconnection network is built among the processors. The program of the processors is stored in **private** or **local memories,** which are accessible only by the processor they are connected to. Processors exchange data via passing (sending) messages. The other name for this architecture is **distributed memory architecture.**

Figure 1.1.5 presents the model for a message-passing computer and also three versions of shared memory architectures for parallel computers.



Message passing architecture



Shared-Memory Access architectures

a.) UMA; b.) NUMA with global and local memories; c.) NUMA with local memories only

Figure 1.1.5.

Shared-address space architectures

Due to the memory transfer congestion problem 'global-only-memory' architectures do not exist in practice. Processors have their own local memories. An extension of this concept is when physically all memories are built as local memories, however hardware units are built in the system to provide access to the private memory of other processors without disturbing the operations of that. In these architectures, naturally local memory reads and writes are executed faster, and the so-called remote reads and writes take more time.

Based on the access time shared memory parallel computers may be categorized as **uniform memory access (UMA)** computers if the access time does not depend on what processor accesses which memory. On the contrary, in **non-uniform memory access (NUMA)** computers the access time is shorter when local memory operation is requested than in the case of remote operations.

Cache memories may be used in parallel computers. Usually separate caches are attached to each of the processors. Certainly the data coherence problem exists in these cases. Very careful design is requested in order to avoid data inconsistency, which appears when more than one copy exists for a data item, and changing the value of one copy (e.g. in a local cache) is not immediately reflected by the same changes in all other copies (including the copy in the global memory and caches of all other processors).

An interesting solution could be the usage of just one global memory and *p* cache memories at the separate processors. This solution is called the **cache-only-memory access (COMA)** architecture.

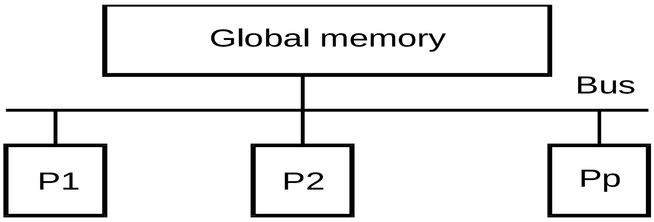
Interconnection networks

The interconnection network in both *shared-address-space computers* and *message-passing computers* may be constructed in a wide variety of ways. The two main categories for their construction are the networks with *point-to-point* communication, or using built-in *switches and communication links.* In the first case we speak on **static** or **direct networks.** While the second case allows run-time changes in the communication paths of a message, that is why they are called **dynamic networks** or **indirect networks.** Switches can be programmed to set up the communication path.

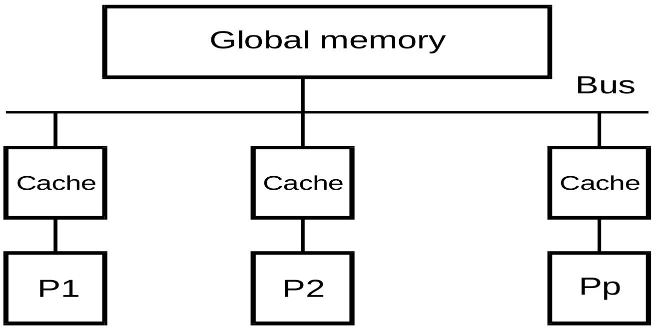
The most important types of switches and architectures with **dynamic interconnection** networks are:

- ⁎ bus-based architectures (Figure 1.1.6)
- ⁎ crossbar switches and architectures (Figure 1.1.7)
- ⁎ and architectures with multistage interconnect networks (Figures 1.1.8 and 1.1.9).

Bus-based architectures have been popular due to the simplicity and low cost of the interconnection network. However, the bus communication bandwidth may be the bottleneck of the whole system. Crossbar architectures provide real concurrent data exchange for many processors if they approach different memory modules. However, this is at the cost of sophisticated and expensive intelligent switches that are capable to serve as arbiters in the case when a memory module is to be used by more than one processor at any time. Multistage networks provide cost effective solutions if *p* processors are to be connected to *m* memory modules at the same time. A switch consists of switching elements capable to handle two input and two output buses. Two states of the switch would provide two possible connections of its inputs and outputs: direct connection or cross connection. Applying *log p* stages and *p/2* switches in each stage *p* processors can be interconnected to *p* memory modules or to each other in arbitrary pattern. Note: that the multistage switches inputs, outputs could be any module with a bus interface (memory or processor).

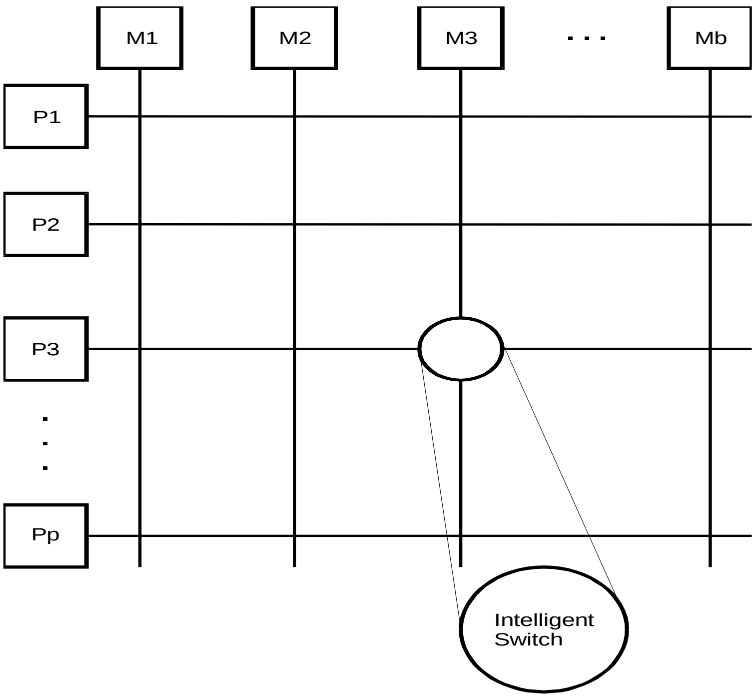


Bus-based architecture



Bus-based architecture with cache memories

Figure 1.1.6



A crossbar parallel computer with
p processors and *b memory blocks*

Figure 1.1.7

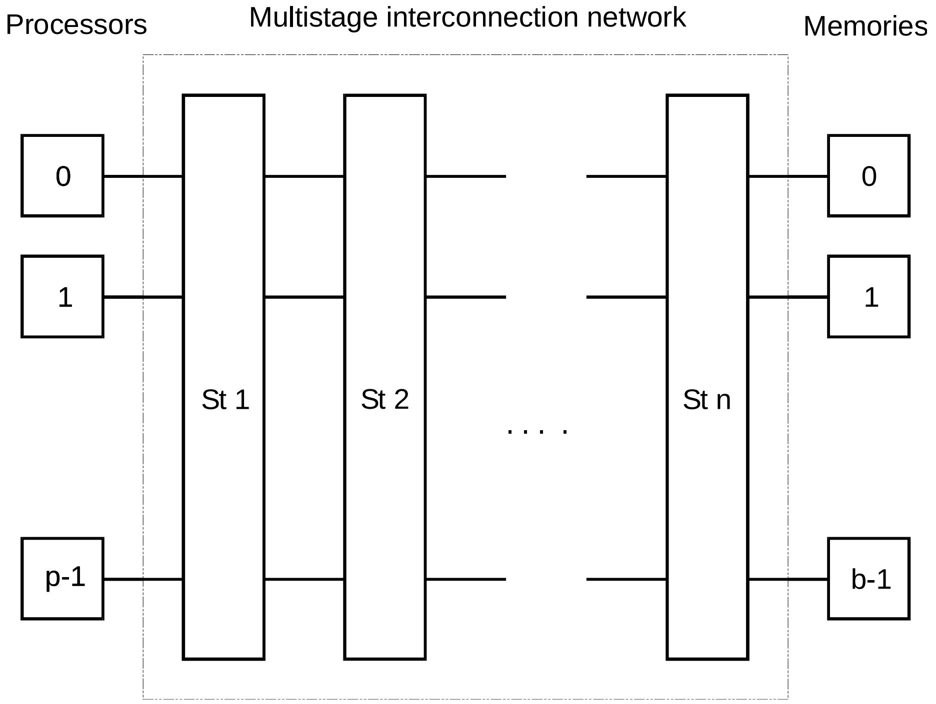


Figure 1.1.8

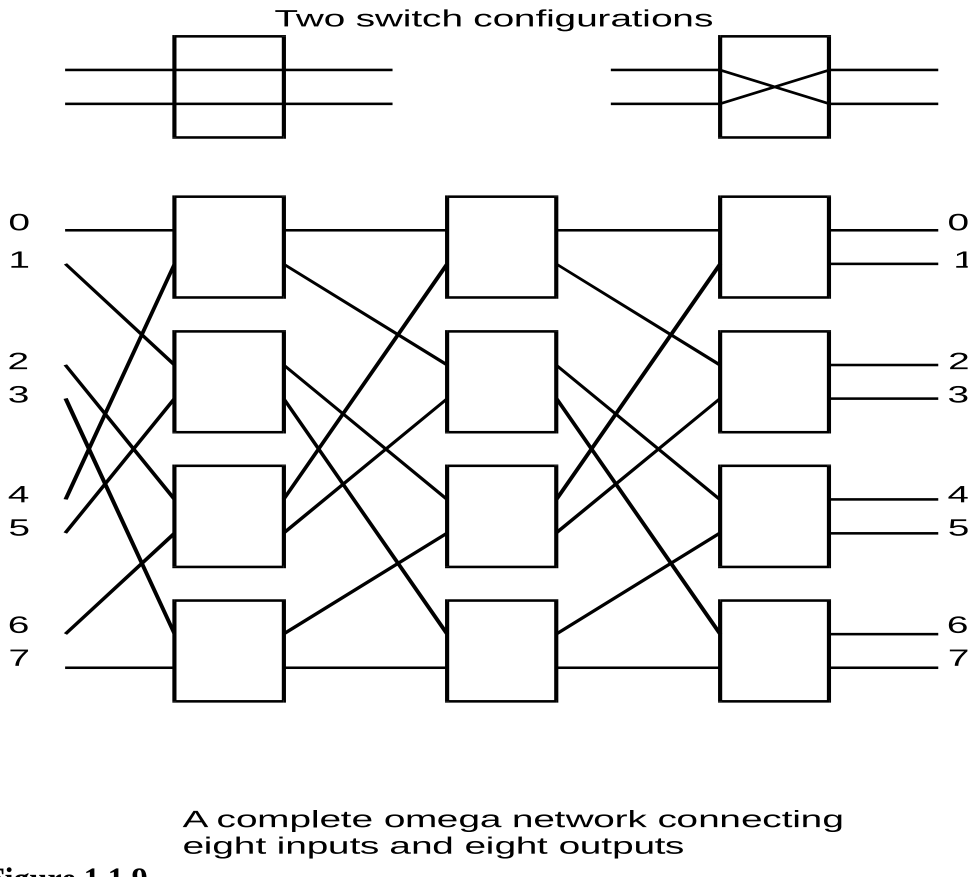
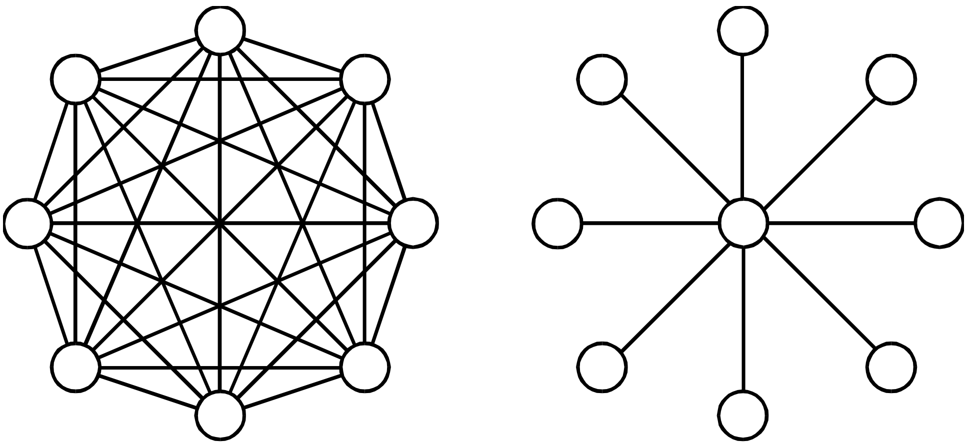


Figure 1.1.9

The most important types of **static interconnection** networks are:

- the fully connected network (Figure 1.1.10. a.)
- star with one switch in the center of star (Figure 1.1.10.b)
- one-, and two-dimensional arrays: linear array, ring, mesh and torus (Figure 1.1.11)
- tree-based architectures (Figure 1.1.12.)
- n dimensional hypercube (Figure 1.1.13 and 1.1.14)



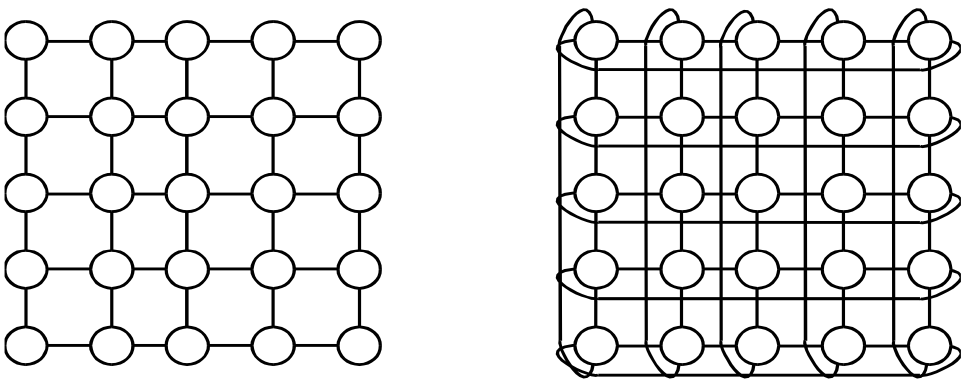
Completely connected network of eight processors

Star connected network of nine processors

Figure 1.1.10

A five processor liear array

A five-processor ring



A two dimensional (5 by 5) mesh

A two dimensional (5 by 5) torus

Figure 1.1.11

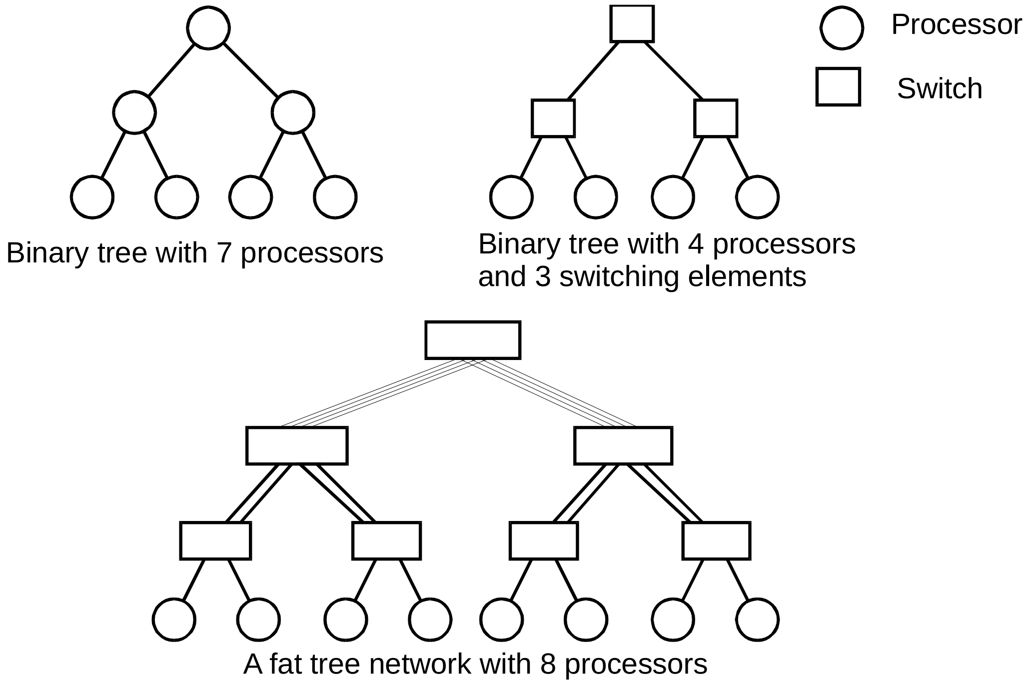


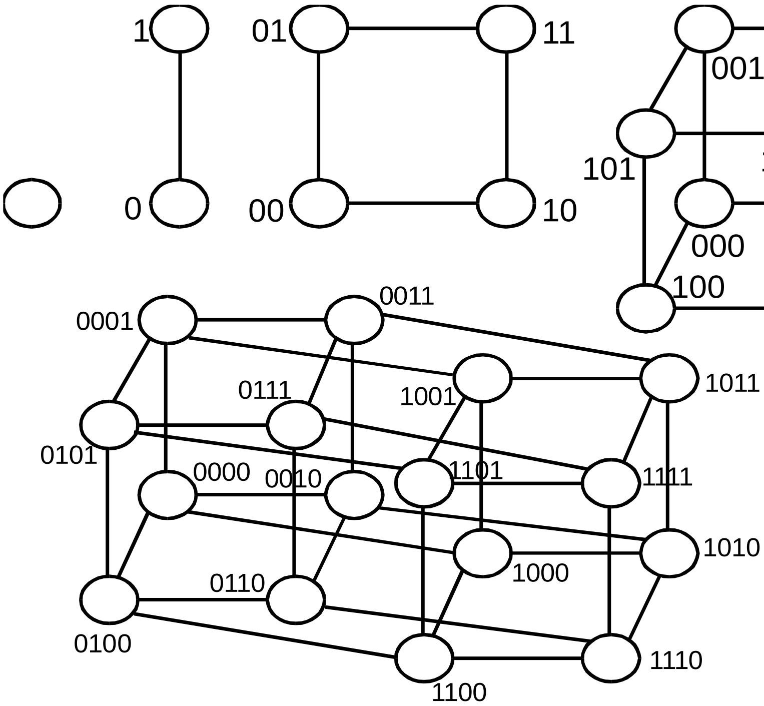
Figure 1.1.12

Meshes and hypercubes are the most popular topologies used. If an array or mesh has a wraparound link, then we get ring (1-D array) or a torus (2-D array). Hypercubes are becoming popular thanks to the fact that their communication links grow only in logarithmic dependence with the number of processors. Also the diameter of the network is just in logarithmic dependence of the number of processors. So, an 8-D hypercube will have 256 processors, but the longest communication path has only $log\ 256 - 1 = 7$ hops distance.

Granularity of processors

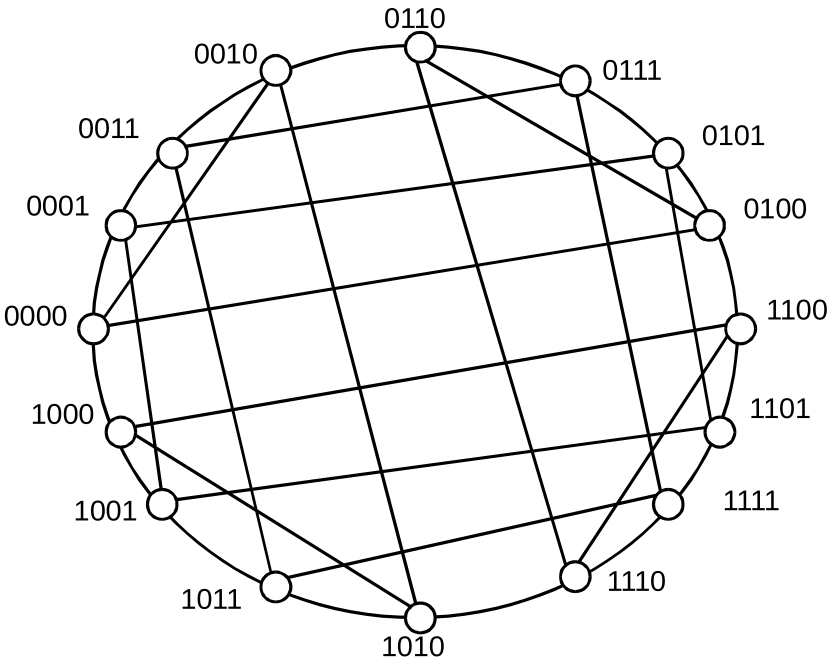
Granularity of the processors may be defined as the ratio characterizing the time that is required for communication operations to the time for a basic computation operation. Algorithms with frequent communication requirements are better executed with fine-grain parallel computers. In these algorithms the size of computation between two communication operations is small.

Parallel computers usually are classified into three granularity categories: **fine-, medium-,** and **coarse-grain** computers**.** Processors for coarse-grain parallel computers are much more complex and more expensive than processors in fine-grain computers. Medium grain size of parallel machines allows the usage of inexpensive microprocessors.



Hypercube connected architecture of zero, one, two, three and four dimensions

Figure 1.1.13



Another way to create a hypercube: arranging processors along a circle, then numbering them by Gray-codes and connecting processors with numbers of unit Hamming distance

Figure 1.1.14

1.1.10 Evaluating static interconnection networks

Cost and performance of static interconnection networks may be evaluated based on some fundamental performance criteria. These are as follows:

- **Diameter** of the network is the longest distance between any two processors in terms of hops.
- **Connectivity** is the measure of how many alternative paths exist between any of two processors. In networks with higher connectivity there is less traffic congestion probability. Usually, under the connectivity of the network we understand the minimum number of edges the removal of which causes the network to fall into two separate parts.
- **Bisection width** is similar; it is the minimum number of communication links whose removal partitions the network into two halves.
- **Channel width** is the number of bits over a link.
- **Channel rate is the** peak rate at which the channel can operate.
- **Channel bandwidth** is the product of channel width (in bits) and the channel rate.

Below, Table 1.1.1. summarizes the basic parameters of some of the networks.

Table 1.1.1.

Network	**Diameter**	**Bisection width**	**Arc connectivity**	**No. of links (cost)**
Completely-connected	1	$p^2/4$	p-1	p(p-1)/2
Star	2	1	1	p-1
Complete binary tree	2log((p+1)/2)	1	1	p-1
Linear array	p-1	1	1	p-1
Ring	p/2	2	2	P
2-D mesh	$2(\sqrt{p-1})$	$\sqrt{p}$	2	$2(p-\sqrt{p})$
2-D wraparound mesh	$2\sqrt{p}\,/\,2$	$2\sqrt{p}$	4	2p
Hypercube	log p	p/2	log p	(p log p)/2
Wraparound k-ary d cube	d k/2	$2k^{d-1}$	2d	Dp

1.1.11. Summarizing models

The development and implementation of parallel algorithms requires special methods, strategies and tools. The most important tools and related theories are as follows:

Algorithm Theory
Computer Languages
Software Implementation
Architectures
Hardware Implementation.

One of the key issues for successful parallel algorithm implementation on parallel computers is the question of modeling the behavior of both the parallel algorithm and the parallel computer that executes the algorithm.

A lot of models exist for algorithm development. Application of parallel computers requires an approach when the overall task solution with parallel computers uses a multilevel organization (not necessarily all levels must be comprehended by the same person, computer specialist). Similarly to the multilayered approach, models exist at various levels beginning from a highly theoretical level, to the very practical engineering level of implementation. Below, we summarize some of the levels where various models may be used in the research and engineering practice when one faces problems to be solved using parallel algorithms.

Models exist for all levels:

Data-flow Model
Control-Flow Model
Functional Model
Operational Model
Topological Model
Connectivity Model
Routing Model
Data Exchange Protocol Model
Signal-Flow Model