THE EVOLUTION OF COMPUTER ARCHITECTURES

Luigi Dadda
Dept. of Electronics - Politecnico di Milano
Piazza Leonardo da Vinci n° 32 - 20133 Milano, Italy

Abstract

The evolution of computer architectures is reviewed, starting from the introduction of the von Neumann sequential machine and going through its various innovations up to the microprocessor. Its limitations are examined and two main lines of further development are considered: the "parallel" von Neumann architectures and the "new" architectures based on new models of computation (data flow, reduction and manage an information system is beyond the capabilities of At the beginning of the computer era (late forties) the main efforts devices were demanding tasks. The need for a suitable hardware archi- based on few elementary components can be said to be the art of an tations are examined and two main lines of further development are through its various innovations up to the microprocessor. Its limi- features, in many respects, from their material composition (many millions of elementary components, the transistors) to their software (basic and application programs with many millions of code lines) to the amount of the information stored and to the processing speed (measured in gigabytes and gigalops, respectively). To fully un- stand and manage an information system is beyond the capabilities of a single human being. How, then, have we succeeded in conceiving, building, managing them? Through a proper structure of the systems, i.e. through a proper partition into parts and a suitable definition of their relations, i.e. through a proper "architecture".

It is well known that this term has been used in the past to denote the art of managing the conception and the construction of very large buildings (e.g. a cathedral) and cities (with billions of bricks), i.e. the most complex single objects and systems that man has learned how to build since few millennia. Knowing how to produce complex and useful (and beautiful) artifacts based on few elementary components can be said to be the art of an architect, no matter what kind of objects he is dealing with. At the beginning of the computer era (late forties) the main efforts have been mostly devoted to the computer as an hardware machine: performing logic and arithmetical operations with available electronic components (vacuum tubes), inventing new memory and input-output devices were demanding tasks. The need for a suitable hardware archi- tecture was also felt and, after few attempts to translate electronically the architecture of existing mechanical or electro-mechanical comput- ing devices (e.g. with ENIAC), the concept of a machine controlled by a sequential program stored in the electronic memory along with the data was formulated by John von Neumann (and implemented in EDVAC). This has been the basic architecture for the computer de- veloped in the following decades, through a series of innovations: the following chapter will describe the most significant ones. Very soon, however, software became an essential issue in computer development, with the introduction of programming languages, of oper- ating systems and of data bases. Correspondingly, new "architectural" concepts were conceived.

A computing system can then be considered as composed by a com- puter (comprising all hardware components), an operating system, an application program. In order to fully describe a computing system we need to describe both its hardware and software parts, with their specific architectures and the relations between them; in other words, an architecture of the whole system. In the following, we will concentrate on the development of the hard- ware architecture of computer. We must, nevertheless, note that this architecture is necessarily linked with the various software and system architectures. We will therefore underline the impact on computer ar- chitecture of developments in software (although the influences in the reverse direction have also been strong).

An unifying architectural concept can be considered to be the algo- rithm implemented through software. Moreover, algorithms them- selves can be described in term of their specific architectures, and we will consider also the problem of the relation between computer architectures and algorithms architectures.

2 Past evolution of computer architectures

Computers in the past decades have undergone a profound evolution from the point of view of physical technologies (from vacuum tubes to magnetic logic devices to transistors to integrated circuits, from mer- cury or magnetostriiction delay lines main memories to cathode ray tube memories then to magnetic cores then to semiconductor memo- ries, from magnetic drum to magnetic or optical disc mass memories, etc.). Innovations have been even more profound in software, with the invention of programming languages and operating systems. As a re- sult, computing power has gone from few hundreds to several billions instructions per second, programs size from few thousands to several millions of code lines.

It is surprising to observe that the basic von Neumann sequential compu- tation model has remained essentially the same through more than four decades (and it is still valid for most of the machines produced today). The architecture of machines embodying such model has cer- tainly gone through a number of innovations and in this chapter we will analyze some of the most significant ones.

The main reason for the success of the von Neumann model of se- quential computation is the fact that it has operated as an efficient bridge between software and hardware, permitting the hardware to be developed almost independently from the software, and vice versa. It is worth noting here that sequential programmed machines have been shown to be universal computing machines by Alan Turing [9] and that the von Neumann model can be considered as a pragmatic embodiment of the Turing machine. Turing has shown that a computer may be seen as a special purpose machine executing a particular program and also a universal machine capable of simulating any special purpose machine. It has also been possible to show that this simulation can be implemented efficiently, so that special purpose machines do not offer major advantages over gen- eral purpose machines, cheaper to produce. Moreover, general purpose machines make possible high-level languages and consequently transportable software. All this explains the success of electronic computers in the past decades.

The relatively few architectural innovations, within the framework of the von Neumann model, characterizing today's sequential computers can be summarized as follows [1-8].

The instead modification of addresses and the memory hierarchy ideas were conceived in 1949 by a group at Manchester University. The index registers permitted the execution of loops without modifying the instruction addresses (one of the most powerful characteristic of von Neumann architecture) and the automatic realisation of programs in memory. The memory hierarchy idea led later to the cache and virtual machines concepts.

In 1951 Wilkes proposed the microprogrammed control, as a new and systematic way of controlling the operation of computers. The concept had a profound effect on all subsequent machines.

In 1958 Barton proposed the stack architecture as a tool for compiling and executing expressions, in order to have the machine architecture reflecting the organization of the programming language. The same concept has been subsequently recognized particularly advantageous for operating systems in managing subroutine invocations and in gen- eral program context.

Multiprocessors developed in the fifties, particularly with separate I/O processors (channels) and also [Bull Gammon 50] with multiprocessors, greatly improving computer performance. More recently co- processors have been widely used as floating point processors and as attached array processors.
Vector arithmetic units or processors are examples of innovations providing efficient machine operations on data-structures (vectors, in the above cases). They are at the base of the success of Cray-I [1972] and of the corresponding vector-supercomputers. The pipeline architecture (used also in the above processors) obtains faster operations by decomposing them in steps to be executed by cascaded sub-units. All sub-units operate simultaneously on different operands. It is therefore a general architectural idea, widely used in modern computers.

An evolution and generalization of the pipeline architecture can be considered the systolic array architecture, characterized by identical processing elements (even programmed processors) connected in a linear or multi-dimensional array, each processing element being connected to its adjacent elements only. Pipeline and systolic architectures have found widespread use in dedicated VLSI components and in special purpose (e.g. for signal and image processing) high performance computers.

The increased achievable component density in integrated circuits and the corresponding lower cost, has permitted [1972] the implementation of a complete, simple processor on a single chip, the microprocessor. The evolution to higher component density has subsequently led to microprocessors with increasingly complex instruction set and, in general, to the hardware implementation of several functions previously obtained through software. A negative result of this has been the overall slow-down of microprocessors speed. Moreover, most of those complex instructions were seldom effectively used in programs. This suggested the idea to go back to architectures with a small number of instructions (those most frequently needed) implementable with simpler circuitry and therefore working at a considerably higher instruction rate. RISC - Reduced Instruction Set Computers - were then opposed to the dominating CISC - Complex Instruction Set Computers - with good success.

An innovation concerning systems rather than single computers is given by their interconnection through communication lines, to form Computer Networks. This has required the definition and implementation of communication protocols. Their standardization has been undertaken by ISO (International Standard Organization) by first defining an "open" architecture (i.e. a standardization of homogeneous computers and characterized by a layered structured (ISO-OSI, ISO Open System Interconnection architecture).

The above architectural innovations, along with innovations in components and in software technologies have permitted the spectacular computer improvements seen in the last decades. This on the basis of the traditional von Neumann architecture, in which the performance of the processor was enhanced by innovations based on an increase of parallelism of its internal structure, as in vector processing units.

The fastest computers, the supercomputers, were obtained using the fastest technologies available for semiconductor components. Monoproessor machines cannot, therefore, obtain a computing speed much higher than what has been already achieved.

In the late sixties, under the pressure of an ever increasing demand of computing power, new architectural concepts were investigated and experimented, in two general directions. The first was directed to totally new architectures, based on models of computation drastically different from the classical sequential model at the basis of the von Neumann machines.

The second, more pragmatic, consisted in an extension of the von Neumann architecture, based on several sequential machines operating in parallel on cooperating processes. This led to architectures that may be called "parallel von Neumann architectures". One of the first machines of this kind was based on a master unit connected to several other units operating under its control (slaves). It was then relatively easy to solve the synchronization problem.

A far more difficult problem had to be solved in the case of a distributed control, each machine interacting with the others, all at the same level. The synchronization of processes has been obtained with special conceived circuits (e.g. for handling interrupts) and with new synchronization primitives added to programming languages (e.g. the "semaphores").

Several parallel machines have already been produced and used, with up to several tens of thousand of processors, and parallel computation has become a widely studied research theme. It is a widespread opinion that in this decade most of the research and development efforts in computer architecture, algorithms design, systems programming and computer applications will be concentrated on parallel computation.

We will consider now parallel von Neumann architectures.

3 Parallel von Neumann architectures

The main motivation for parallel computers is their potential ability to scale-up their performance by adding more processors. Performance can be expressed in term of processing speed, but also in term of throughput (the maximum rate of input and output data) and in term of response time (in those applications requiring fast responses). More parallelism in solving a problem is needed for two purposes: in order to solve bigger problems in the same time (scale-up), or to solve the same problem in less time (speed-up); or both.

In several applications scale-up can be obtained by executing more copies of the same program in correspondingly larger number of identical processors. This is easier to obtain than speed-up, requiring the partitioning of a program into many parallel parts to be executed by separate processors.

For these reasons multiprocessors with many processors are necessary. "Massively" parallel computers will also permit to adopt technologies cheaper than those used in today's vector supercomputers, compensating a smaller component speed with more parallelism.

Parallel von Neumann architectures can be classified according to various criteria. A well known computer taxonomy is the one proposed by Flynn [10] that classifies architectures by the number of instruction and data streams that they can process in parallel. Four categories are therefore defined:

- Single Instruction stream, Single Data stream (SISD)
- Single Instruction stream, Multiple Data streams (SIMD)
- Multiple Instruction streams, Single Data stream (MISD)
- Multiple Instruction streams, Multiple Data streams (MIMD)

SISD machines are typically represented by von Neumann machines, operating by means of sequential instructions on sequential data streams, stored in same memory or on separate memories. One form of parallelism is pipelining in the instruction execution, with partial overlapping of instruction execution stages. Another form is given by the parallel execution of multiple instructions (taken in parallel in consecutive blocks from the single instruction stream) on the same data stream, by means of multiple arithmetic units (Wide Instruction Word Computers). In SIMD machines (fig. 1) multiple processing units operate on distinct operand streams and receive the same instruction, broadcast by a single control unit. In those machines, a local variable (set by the central unit) can obtain to skip the execution of one or more operations, thus providing greater flexibility. SIMD machines can be very effective in large classes of numerical computation, e.g. in image processing and in the solution of partial differential equations.

In MIMD machines (fig. 2) represent the most general class, since in them multiple data streams may be operated with different instruction streams. Many of the existing multicomputers belong to this class.
Flynn taxonomy certainly accounts for the most important parameters of a multiprocessor system, i.e. the number of processors and some aspects of their connection but, due also to the large number of different models already implemented, mostly falling in SIMD or MIMD classes, it appears inadequate for more significant comparisons.

An extension of Flynn taxonomy will be illustrated later. A most important characteristic of multiprocessors is the method used for coupling processes composing a complex computation. Two main methods have been devised for such purpose, based respectively on a Common or Shared Memory (fig. 1a and 2a) and on a Message Passing Network (fig. 1b and 2b).

In the former, all data reside in a Data Memory whose locations can be accessed, with the same latency, by any processor. Each processor is often provided with a local Data and Instruction Memory in order to obtain a faster operation. Synchronization among processes is obtained by means of shared variables. The coupling between processors is therefore the strongest possible, hence these systems can be said “tightly coupled”.

In a second way of synchronizing and communicating, data are distributed among private memories, directly accessed by each processor. Communication between processes is obtained with messages sent by each processor to one or more other processors, through an Interconnection Network. The coupling of processes residing in different processors is therefore weaker than in the preceding case, thus the scheme is designated as “loosely coupled”.

Fig.2

In a Shared Memory MIMD multiprocessor (fig. 2a) a number of processors access a number of memory modules via an Interconnection Switch. If the number of processors is small, the switch can be based on a simple shared bus, while a complex network is required for a large number of processors.

It is beyond the scope of this writing to extensively describe Interconnection Switches and Networks for shared memory as for message passing network schemes: see [15, 16, 20] for more information. We will here briefly introduce to the most relevant concepts.

Interconnection Switches and Networks have to be recognized as the most important sub-systems necessary for the construction of multiprocessors. Several types of such interconnected networks exist, and a major distinction is to be made between the parallel Von Neumann computers, along with the traditional sub-systems such as processors and memory modules. In parallel computers the Interconnection Switches and Networks strongly affect the overall performance and accounts for a large percentage of the cost.

In a shared memory system all references to data residing in it require an address to be issued by each processor, to be processed in the interconnection switch in order to access the addressed module and the memory location in it, to write the new information (following the address) in it or to read its content. In the latter case, the content has to be transmitted through the Interconnection Switch to the processor(s) that has issued the request, along the same route in the network “opened” during the first phase.

Three main problems arise in connection to Interconnection Switches. In a ideal switch no delays exist in the implementation of the various phases, while in a real switch such delays exist, mainly because the packets comprising the address and the attached data have to be in serial form for practical reasons. Data requested by each processor will then be available after a certain latency.

Since several (even all) processors might request to write or to read to or from the same memory location, certain write or read disciplines should be decided. This is a pure logical requirement, characterizing different models of Parallel Random Access Machines or PRAM’s.

Three classes of PRAM’s are defined [27] according to restrictions on common memory access. In an Exclusive-Read Exclusive-Write (EREW) PRAM simultaneous access to any memory location is forbidden for both reading and writing; in a Concurrent-Read Exclusive-Write (CREW) PRAM simultaneous reads are permitted but not simultaneous write; in a Concurrent-Read Concurrent-Write (CRCW) both reads and writes are permitted. Write conflicts can be resolved with various methods:

a) in the COMMON method all processors write the same value in the same location;  
b) in the ARBITRARY method the algorithm should work correctly no matter which processor succeeds in writing;  
c) in the PRIORITY method processors are linearly ordered and the minimum numbered processor writes its value, in a concurrent write.

To implement Concurrent Read, the switch detects the read request for the same location and let only one to proceed while all the other are stopped, but not forgotten; then when the location content has been read from memory it is copied back to all requesting processors.

A switch for a large number of processors and modules is usually implemented as a multistage interconnection network, such as the butterfly switch, the banyan, the omega network, etc. [15, 16, 20, 27], obtained by combining smaller switches. This lead to contention problems, when too many requests concentrate on a single memory module or on a single internal switch module.

Contention is particularly harmful for the overall computer efficiency, due also to the regular data reference patterns common to many algorithms, such as access to the rows and columns of an array. In order to counteract the occurrence of traffic “hot spots”, a simple mean is adopted consisting in address “hashing” in order to scramble the destination memory modules [25, 27].

The effect of contention can be treated theoretically on the basis of specific assumptions about the operation of the processors, the interconnection switch and of the memory modules [15, 20, 30].

A Message Passing or Distributed Memory MIMD multiprocessor (fig. 2b) a number of processors with their own local memories exchange messages via a communication network, which can be fixed or reconfigurable. In order to have a feasible communication network, each processor must have a limited number of output-input ports, so that only the same number of processors can be directly connected, the communication to the other processors being obtained through intermediate processors. The technique most widely used is the Store and Forward Message Passing, involving each message to be received by a processor before being relayed to the next one, along a route connecting the transmitter to the receiver. A delay proportional to the message length and to the number of intermediate processors is then suffered. Such a delay can be drastically reduced if each bit (or byte) of the message can be re-transmitted as soon as it has been received by an intermediate processor. Such word-level technique obviously requires the whole route to be set in advance (through a suitable procedure and therefore requiring a certain setting time) and is suggested by the property of locality of memory references (i.e. the high probability of the destination address to be used several times in the future).

The address hashing previously described for Shared Memory systems can be applied also to the Message Passing scheme, in which the routing problem has to be considered, i.e. the determination of messages paths.

A distributed algorithm is desirable for such a purpose, capable also of determining paths with a certain degree of randomization, in order to avoid traffic hotspots (see [25] for an overview).

A parallel computer consists of p processors located each in a node of a p-node graph, whose edges represent transmission lines. A simple interconnection scheme is given by regular arrays or grids (linear, planar, spatial, n-dimensional). Algorithms for several important problem classes map naturally on such schemes (e.g. Partial Differential Equations in finite differences).

For a general purpose machine however some general network parameters should be optimized, such as: the degree d, i.e. the maximum number of edges from any node (it should be small due to physical packaging constraints); the diameter, being the minimal length path between the pair of nodes for which this length is maximal, should be small; graphs looking isomorphic when viewed from any part of the graph are desirable, since easier to program; recursive decomposability permits the construction from a small number of components and an easier partitioning of the graph.

11
A large number of papers have been published on interconnection networks: few of them are cited here, covering different aspects [16, 18, 19, 20, 28].

One class of networks has been extensively studied and found to exhibit good properties. It is the class related to the butterfly n-dimensional cube, containing the butterfly and the shuffle-exchange networks. Several important algorithms map very well into them.

An application program can be expressed as a set of communicating processes (12). Processes can be numerous and small (fine-grain partitioning) or (relatively) few and big (coarse-grain partitioning). In several applications, processes are connected regularly, e.g. as a linear (a pipeline) or planar, etc. network, where processes communicate only with their nearest neighbors. Such networks of processes can be mapped easily to a network of processors. The simplest mapping can be done by assigning a processor to each process, but there are important reasons suggesting to assign to a processor a group of processes. One of the reasons is to obtain a balance between computation and communication. As an example, assume a two-dimensional array of processes, where each process communicates with the four adjacent processes. In a fine-grain processing we can assign a single process to each processor, and all communications between processes are performed via messages through processors ports. In a coarse-grain processing we assign a number of adjacent processes to each processor. We assume that "internal" processes are executed in each processor operating in multiprocesing mode so that communications between internal processes do not involve communication ports. Consequently, the ratio: number of processes executions/number of messages increases with the number of internal processes. This can be used in order to obtain a good balance between communications and computations, by choosing a number of internal processes (i.e. the processing grain size) so that the processes execution time is (approximately) equal to the communication time.

It must be noted, as a consequence, that in decreasing the grain size in order to increase the speed (speed-up), a limiting factor is given by the communication speed.

A partial solution to this problem has been found (e.g. in SUPRENUM machine) by grouping processors in clusters. Communication within a cluster is via a fast internal (first level) network, while clusters are connected via a second level network.

In the above example the structure of the algorithm matches the structure of the machine. In several important algorithms this is not the case, since no mapping of processes to processors exists satisfying the condition that all interacting processes reside in adjacent processors, so that some of the interactions can be obtained only via a number of intermediate processors, at the cost of larger delays.

In order to be able to support a wide range of algorithms, some machines have been provided with specific means for supporting high speed non-local communication. One of such means is the "worm hole" technique, already described. Communication delays, though minimized with the above mentioned techniques, strongly affect the speed performance of parallel computers. An additional reduction of its effects is obtained through a technique called delay hiding [31]. This requires a number of processes assigned to each processor, capable of multiprocesing operation. Whenever a process is waiting for a message, the de-scheduled and another process (among those ready for execution) is activated. The de-scheduled processes can again be activated at the time of arrival of the necessary data. It is easily seen that a processor will never idle as a result of communication delays, provided that there are sufficient processes in each processor. Note that what has been said in relation to delays in receiving messages in a distributed memory MIMD computer can be said for the delays in reading data in a shared memory MIMD computer.

In order to be able to determine how many processes have to be assigned to a processor, the communication delay, or an upper bound of it, must be known, along with the transmission capability.

Note also that since hiding communication delays requires a number of processes in each processor, i.e. a coarse grain, more parallelism is required in algorithms in order to obtain a high processing speed.

In the next chapter, computer architectures based on entirely new computing models will be considered, as a way to achieve high performances through the exploitation of parallelism inherent in algorithms and based on computation models capable of exploiting such parallelism.

4 New architectures

The von Neumann architecture has dominated the electronic computer field since its beginning, and it is still very stable in it, despite the many criticisms issued during more than forty years.

The sequential organization and execution of programs is considered the main obstacle in achieving the higher computing speeds necessary to solve important, yet unsolved problems.

In fact, the machine executes the program instructions exactly in the ordering chosen by the programmer among the (usually) many equivalent possible orderings. In the von Neumann model of computation it is impossible to avoid such choice: in this model therefore the execution ordering is over-constrained. Moreover, languages inspired by the von Neumann model do not contain any concept of parallelism, i.e. of computing more than one thing at a time. When the need arose to program concurrent processes, parallelism was first simulated through strictly sequential programs by first adding communication and synchronization primitives to existing languages, later on by means of separate synchronized processes.

Moreover, the strong difference between the machine language and the various high level languages (on the other hand, very different one from the other) i.e. the "semantic gap", is considered the main cause of software high cost and development time, of its unreliability, etc.

Why, then, the von Neumann architecture has been so successful and stable? One reason could be traced to its conceptual simplicity and, at the same time, to its proven power. Note that the latter can be considered a consequence of the former.

Its simplicity is certainly the result of a bright intuition, but it is also the effect of the high cost (and unreliability) of the original electronic components (the vacuum tubes), leading to a minimal hardware conception.

The transistor first, and the integrated circuits of increasing complexity then, have drastically changed the situation, so that new architectures became feasible.

A large number of new architectures have been proposed, several of them have also been tested on prototypes and a smaller number even produced and commercialized. Possibilities and constraints of semiconductor technology are both represented in the design philosophy proposed by Mead and Conway. A "good" VLSI architecture must satisfy the following rules:

- Only few different types of simple cells have to be used.
- Cells are connected through local and regular data and control paths; long distance or irregular paths must be minimized.
- Pipeline and multiprocesing are extensively adopted, so that a large number of cells are active at any time.

A more detailed examination of this technology shows that the interconnections in modules and boards occupy most of the volume of a machine. Switching devices, i.e. transistors, occupy a small fraction of the silicon area. An improvement will therefore come from manufacturing processes implementing more interconnect levels in chips, in modules and in boards. However, the limiting overall factor will continue to be the wires density.

Scaling down the size of devices on chip will continue for many years. In present technology transistors gate length of about 1μ has been achieved. More devices on a chip will be put also as result of the increase of chip size.

A reduction in transistor size causes also a proportional decrease of delay. This and the quadratic increase in devices number are good reason for more parallelism.

The effect on signal propagation delays must also be considered. Since the signal delay in a metal track on silicon is determined by its resistance and capacitance, reducing the width of the track and of the separation between tracks causes a signal propagation delay that is increasing relative to switching delay.

Note that, in a good transmission line connecting chips, a speed not too smaller than the speed of light (3×10^8 cm/sec) can be achieved, while on chip in today's VLSI technology a speed of about only 5×10^6 cm/sec is obtained. Note also that a transistor can switch in about the same time as a signal propagating by 1mm on a metal track.

The search for new, feasible architectures should be based on the above technological constraints but also on new model of computer.

Computation models are reflected on one side on the logical organization of the machine, i.e. the abstract machine, on the other side on
computing languages. In effect, programming languages usually correspond to the computation model of the machine, in order to obtain a good match and consequently a good performance.

It can be observed that in the past many very different languages has been implemented on von Neumann machines (the only ones available), but it must be observed that procedural languages (e.g. Algol-like languages) exhibited a better match (offering a better performance) than non-procedural languages.

Non-procedural languages, comprising functional (e.g. LISP) and logic or relational (e.g. PROLOG) languages, have then suggested new interesting architectures.

In functional languages a program consists of a (non-ordered) set of equations specifying a calculation; the corresponding statements are not therefore executed sequentially, the order of the calculations being implied by data dependency, i.e. the order in which, starting from the input data, the computed data become available; every time new data become available, all equations whose arguments become known are computed. This leads naturally to a parallel computation.

Computers designed according to this computation model are known as data flow (or data driven) machines.

Their computation model is a graph in which operations are represented by vertices and data by tokens flowing along the directed edges. Input data are represented in vertices with an outgoing edge only. An operator "fires" according to rules specified by the chosen form of data flow. A firing absorbs a datum from each input and produces results that are sent out on the outgoing edges. In a model, each edge may contain one token at most, while in another model each edge may contain more than one token, with a specified queuing discipline. Fig. 2a and fig. 2b may represent the architectural schemes of data flow machines, provided that in fig. 2a Instruction Memories are removed and in fig. 2b the memory modules are considered as Data Memories; moreover in both figures processors do not include Instruction Processors, since no instructions have to be executed. This because only data dependencies, detected by the Data Processors, determine the operations to be performed. As an example, fig. 3a scheme may represent the Manchester Dataflow Machine [13] and the Arvind’s machine [21]. Fig. 3b represents another scheme for a dataflow machine. In the preceding schemes Data Processors may have a selector for selecting operators to be executed and Data Memories may be assumed as active role of pushing executable nodes toward processors for executing the corresponding operations.

Another computation model, the execution of the operations to be performed is triggered not by the availability of its arguments but by the request of their results: we have then the family of demand driven or graph reduction machines.

This model can be represented as a graph of function in which the root of each subtree is an Apply, the left branch is a function and the right branches are descriptions of arguments.

Assume that a function (e.g. \( f = (b - 1) + (c - 1) \)), represented as a tree, must be computed. The invocation of \( f \) calls for arguments \( f' = b - 1 \), and \( f'' = c - 1 \), i.e. functions that call in turn for arguments \( b \) and \( c \). If these are available, the following phase (reduction) is executed with a mechanism similar to the one of the preceding data flow models.

At any stage, a number of subtrees may be ready for execution. An executable sub-tree is called "redex". Multiple processors can be used to operate the redexes.

Several variations of this model have been considered and implemented [6, 14, 23].

Like data flow machines, also reduction machines do not need Instruction Memories or Instruction Processors. The structure of the graph can be considered as the program and the content the data.

Their general schemes can be the same of data flow machines, as in fig. 3a.

Considerable work on reduction machines has been done in Europe. Fig. 1, 2 and 3 have been drawn according (with slight variations) to the taxonomy proposed by Skillcorn in 1988 [24] as an extension of Flynn's taxonomy and offering a more detailed representation of architectural characteristics.

In a first level, Skillcorn taxonomy defines a set of basic functional units:

- IP or Instruction Processors, i.e. the interpreter of machine instructions fetched from an
- DM or the Instruction Memory.

Such units exist only when the computation model is based on some kind of instructions. Such is not the cases in fig. 3.

- DP or a Data Processor acting as a transformer of data (e.g. an arithmetic unit) taken from:
- DM or Data Memory.

Instruction and Data Memories are intended as hierarchical structures (cache, main memory, mass memories). The above functional units are connected in various ways by a "switch" SW. Four type of switches are considered in Skillcorn's taxonomy:

1-1: a single line connecting a single functional unit of one type with a single functional unit of another type.

n-n: a set of \( n \) lines, the \( k \)-th line connecting the \( k \)-th unit of a set of functional units with the \( k \)-th unit of another set.

1-n: connecting one functional unit to other \( n \) units.

n-n: each functional unit of one set can communicate with any unit of a second set, and vice versa. This is the switch necessary in a shared memory system, as in fig. 1a, 2a and 3a.

It seems useful to define a further switch to represent a message passing network, as in fig. 1b, 2b and 3b, calling it an n-switch or Interconnection Network, whose operation appear quite different from the preceding one (in which Skillcorn includes 1).

Skillcorn taxonomy defines an architecture at first level giving the types and numbers of functional units and the types of switches connecting them. This is much more than the Flynn taxonomy can describe. Moreover, in Skillcorn taxonomy a second level, giving details such as state diagrams of processors, pipelining, etc., and a third level describing implementation details, are considered.

The architecture of a given system can then be characterised quite accurately, permitting among other thing a significant comparison among machines. The framework provided by a taxonomy is effective for the identification of trends in computer development and, on the basis of both theoretical investigation and testing results on existing machines, also the identification of promising new developments. Skillcorn's taxonomy permits among other things, as shown in Table A, the listing of all possible architectures; it can be seen that all proposed architectures are represented and, moreover, new, unexplored architectures are indicated.

5 Towards general purpose parallel computers

The numerous parallel computers built as experimental machines and those that have been produced commercially have permitted to test on real problems different parallel schemes and to produce results of great help not only for the improvement of computer architecture but also of software and algorithms suitable for them.

Important results for determining the evolution of parallel computer architecture are also coming from theoretical researches in software and in parallel algorithms.
The main scope of the research activities in parallel computation is directed to obtaining a truly general purpose parallel computer, comparable to a general purpose sequential computer in the sense that a program can run efficiently on different machines. Today's parallel computer can be said to be general purpose only in the sense that each of them has been efficiently used for solving quite a large variety of problems. Programs are, however, written in languages such as FORTRAN, C, and LISP only apparently general purpose, being in general manufacturer's proprietary languages, or proprietary extensions of existing languages. We need therefore standard programming languages accepted by parallel computers of different manufacturers, with different number of processors.

The lack of common software is a great obstacle in the use of parallel machines. A necessary condition for the solution of this problem is the common architecture for general purpose parallel computers. Further conditions will need to be fulfilled by the algorithms and by the software for such machines.

In order to gain insight into this fundamental step in parallel computer evolution, it is useful to recall some results obtained by theoretical investigation [31].

A formalization of the classical von Neumann machine directed to investigate algorithms and programs efficiency is the Random Access Machine or RAM proposed by Cook and Reckhow [11]. Its extension to parallel machines has been done by several authors by defining various models of Parallel Random Access Machines or PRAM's (see [26] for surveys) consisting of several independent sequential processors, each with its own private memory, communicating with one another through a common memory. In one unit of time, each processor can read one common or local memory location, execute a single operation, and write into one common or local memory location.

The different PRAM models, described in chapter 3, permit to evaluate the computational power for a variety of algorithms, in a situation free from interprocessor communication. The theory deals also with how the various models can simulate each other. An interesting general result is that the various PRAM models do not have very different computational power.

In several studies the trade-off between the execution time of a parallel computation and the number of processors required has been considered, leading to the definition of so called "efficient" algorithms, whose execution time is bounded by a fixed power of the logarithm (polylog) of the processors number, and in which the processors x time product exceeds the time (number of steps) in an optimal sequential algorithm by at most a polylog factor. Several efficient parallel algorithms have been designed: it is worth noting that these algorithms are completely different from the best sequential algorithms for the same problems.

Though some implementations of the PRAM model has been obtained (see e.g. [30]), the Message Passing model has been more widely chosen, and a large number of studies has been devoted to it. The results so far obtained suggest that the goal of a general purpose parallel computer should not be too far away.

Valiant [25, 26] has proposed for the purpose a new model, the BSP (Bulk-Synchronous Parallel). The main scope to be reached for a truly general purpose machine is to efficiently hide from the programmer both storage allocation and interprocessor communication. This require the program to be partitioned in v processes, with v rather larger than the number p of processors (i.e. it must have excess parallelism, or "parallel slackness"). (Note that a similar conclusion has already been drawn in a preceding paragraph in relation to methods intended to counteract the interprocessor communication delay). This in order to give the programmer a parameter, v, for the explicit control of the degree of parallelism. v = p, for instance, would not permit an efficient storage allocation and communication (even worse for parallelizing sequential code, e.g. v = 1).

The operation of a BSP computer is based on supersteps, i.e. a multiple L of the basic machine step. Assume all processors are assigned tasks at the beginning of a superstep. During this each processor operates, exchanging messages with other processors. No synchronization occurs within the superstep, but only at its end: if all processors have completed their tasks new tasks will be allocated; if not (even for a single processor) a new superstep will be allowed, until all processors have completed their tasks. An abstract model of this machine is the XPRAM [25, fig. 4], in which each processor is a universal sequential machine with its local instruction set which can access words only in the local memory in unit time, and a set of global instructions for accessing other processors memories in g > 1 time.

The basic "universality" proof consider a specialized p-processors machine, executing an algorithm in time T, with a p'T "operation count". It then shows that the proposed general purpose machine can simulate, with an operation count O(pT), any special purpose machine M' composed by p' processes residing in p' processors, under the condition that the algorithm to be executed has enough excess parallelism, i.e. p' > p, e.g. p' = p log p.

Another condition to be satisfied is that a "good" hash function can be implemented in time proportional to time g, for mapping M' on M. The purpose is to spread out the memory accesses as uniformly as possible among the memory modules of M even when the memory accesses of M' are arbitrary non uniform.

The BSP machine, using a Message Passing Interconnection Network, can be considered as a pragmatic implementation of an XPRAM, in a much the same sense that a von Neumann machine is a pragmatic embodiment of the Turing Machine. More details can be found in [25, 26].

The following table (Table A) contains examples of architectures that fit the proposed model, to which we will refer when describing our results. Some of these models are "exotic", but their consideration is useful in understanding the limits of the proposed general purpose machine. For instance, the HEP model is intended to counteract the interprocessor communication delay. It then shows that the proposed general purpose machine can simulate, with an operation count O(pT), any special purpose machine M' composed by p' processes residing in p' processors, under the condition that the algorithm to be executed has enough excess parallelism, i.e. p' > p, e.g. p' = p log p.

Another condition to be satisfied is that a "good" hash function can be implemented in time proportional to time g, for mapping M' on M. The purpose is to spread out the memory accesses as uniformly as possible among the memory modules of M even when the memory accesses of M' are arbitrary non uniform.

The BSP machine, using a Message Passing Interconnection Network, can be considered as a pragmatic implementation of an XPRAM, in a much the same sense that a von Neumann machine is a pragmatic embodiment of the Turing Machine. More details can be found in [25, 26].
6 Alternative and future architectures

The above described computer architectures represent the main directions of evolution, originating from the early von Neumann architecture an based on classic logic and memory components implemented in the well established semiconductor technology. For sake of completeness we mention here alternative approaches to computing, and new technologies promising to initiate new computer families.

A totally new approach to computing is based on neural networks, evolved from researches in artificial intelligence, that appear particularly suitable for instance in computations related to signal and image processing. Neural networks are based on relatively simple and uniform basic processing components, the artificial neurons, and derive their power from a the very large number of connections among neurons (while in tradition computers the role of logic components prevails). A very large number of research projects has been launched and some interesting results have been obtained; it is nevertheless doubtful that this new “computing paradigm” will replace the traditional one, except in those (artificial intelligence) applications were these have failed to offer an efficient solution. An impact on computer architecture is also likely to come from new basic technologies, now in their early stage of development: optronics and molecular electronics.

It is well known that optronics has already generated a basic new communication technology (fiber optics), while an obstacle in adopting it in logic comes from the lack of an integration technology. An interesting possibility offered by optronics is the implementation of a very complex and flexible switching unit, based on the capability of electrical driven fast switching of light beams, that would permit a new basic functional unit for massively parallel computers.

Molecular electronics is based on the idea of using single molecules as the basic processing unit, thus promising an unprecedented components density. Molecular electronics is in its very early stage of development and it seems premature to speculate now about related processing architectures.

The evolution of computer architectures can be viewed in a more general frame. Considering the case of the tools used for the study of nature, we might observe that first the direct observation of phenomena, then the laboratory experiment have formed the core of the scientific method, along with a mathematical model. Analog computers, first, modeling the system under study in analogue terms, then digital computers used to simulate the system under investigation through the mathematical model, have been adopted in order to draw consequences to be tested with new experiments, thus giving new strength to the scientific method.

An efficient general purpose parallel architecture will permit to extend the application of computer simulation to new scientific and technological problems.

For certain problems, in particular for those exhibiting some sort of “natural” parallelism, special purpose machines might permit a cheaper and faster solution through a direct modeling of the systems, thus leading to a sort of analogue (as far as the structure is concerned) and digital (for the computational part) architecture. An old example (early '50s) has been the Digital Differential Analyzer. Another example can be considered the family of modern massively parallel computers with a grid interconnection, on which Partial Differential Equations represented in finite differences or in finite elements methods can be directly mapped.

7 Conclusion

The evolution of computer architectures have been shown to have developed along few very general lines, from the classical multiprocessor von Neumann sequential machines to the massively parallel machines. It has also been shown that in the latest developments a very significant role has been played by theoretical computer science. A progress in the conception of new parallel algorithms and languages is as important as a progress in basic components. In the latter field, the interconnection networks represent the new basic functional unit: its improvement is a necessity for the development of efficient parallel machines.

A fundamental driving force is represented by challenges still unanswered in science and technology [29], as shown synthetically in fig. 5. Computers seems to be, after few decades of intense evolution, still in a stage where applications push the search for new more powerful architectures.

References

Books


Papers


[26] Valiant L.G., Bulk-Synchronous Parallel Computers, TR-08-89, Aiken Computation Laboratory, Harvard University, 1989


