Study of multiprocessor architectures for a hardware accelerator

Luís Miguel Silveira

INESC/IST

R. Alves Redol, 9

1000 Lisboa

PORTUGAL

Abstract

This work is intended as an overview on parallel processing and parallel processing techniques and architectures. The principal constraints and design options for the adaption of a CAD tool to a general-purpose multiprocessor machine are debated. Sugestions are made as guidelines for a future implementation of the circuit simulator CINNAMON in a multiprocessor environment.

This work was partially supported by the Esprit 1058 *Project*

Contents

1 Overview on Parallel Processing

Parallel Processing has emerged in the past years as a feasible and inexpensive solution for the increasing demand for high-performance computers.

As the technologically advanced societies rely more and more on computers in fields such as administration, medicine, industry and commerce, defense and even leisure, interest is directed into building more powerful and less expensive machines. More so, since computers are not only replacing human activity in either manual or scientific work, but are also giving rise to a whole new range of products and industries.

Specific areas like weather forecasting, medical diagnosis, physics research, aerodynamics simulation, industrial automation, among many other scientific and engineering applications, are continuously demanding more powerful machines, and improvements in those areas are really depending on the computing power available.

The quest becomes one for higher speed, higher performance, more reliability and lower costs. This four requests are, over a certain extent, mutually exclusive, and a balance must be found in order to achieve the proposed goals [1].

Advanced computer architectures are nowadays centered around the concept of **parallel processing** since that seems to be the best answer to the 4 previously mentioned requests. Parallel processors, however, are such a radical departure in computing architectures that they don't fit in existing and conventional classifications of computers: new ones had to be devised, depending either on memory sharing, interconnection schemes or control schemes.

Some prefer to divide state-of-the-art parallel processing into three structural classes: pipelined computers, array processors and multiprocessor systems [2]. Others consider the price factor as very important and present a matrix-style division with system price ranges on one axis and basic architectures styles on the other [3]. Several others present different characterizations, depending on the angle by which they face the problem. The division is therefore rather conceptual and subjective and most existing parallel machines have features that sometimes fit into more than one category.

Parallel computers are those systems that emphasize **parallel processing**, either **temporal parallelism**, **spatial parallelism** or **asynchronous parallelism**. A system that performs **temporal parallelism** must overlap computations, therefore providing a kind of pipeline where instructions flow. Achieving **spatial parallelism** requires the existence of multiple processing units operating in a lock-step fashion performing the same function at the same time. **Asynchronous parallelism**, finally, is related to the existence of multiple processors of approximate capabilities sharing access to memory, I/O and some peripheral devices.

The three parallel approaches mentioned are not mutually exclusive, not even conceptually. In fact most existing machines are pipelined and some have an array or multiprocessor structure.

Pipelining is directly related to the steps required in the process of executing an instruction in a digital computer, which usually involves four major tasks: "instruction fetch","instruction decoding", thus identifying the desired operation, "operand fetch", if necessary, and "execution" of the operation. If each operation is assigned a stage in a pipeline, there is no need for an instruction to complete the four steps before the next one starts. Overlapped instructions are then possible by linear cascading the pipeline stages. At any time, each stage is dealing with one instruction and one of the steps for the execution of that instruction is being performed. At time t, while the first stage of the pipeline is fetching instruction I_n, the second stage is decoding instruction I_{n-1}, the third is fetching the operands (if any) for instruction I_{n-2} while the forth is executing I_{n-3}. Of course the pipeline can have as many steps as possible and it is easily seen that in an n-stage pipeline, we can have n instructions being overlapped thus, theoretically, increasing the speed by a factor of n.

Not all stages of the pipe have the same delay. The slowest stage dictates the duration of the pipeline cycle and the frequency of the clock that triggers the flow of data along the pipeline, thus preventing collision. The operation of all stages is therefore synchronized under a common clock control.

The main questions in designing a pipeline computer are job sequencing, the prevention of collisions in data access, the control of congestion, reconfiguration and hazard resolution. Memory conflicts, data dependency, branch and interrupts are some of the factors that can slow down or cause congestion in the pipe and thus have to be considered with extreme care.

Pipelining is widely used to speed up CPU operations or to perform repetitive arithmetic-logic tasks. In the latter case, every change of operations requires a reconfiguration of the pipe, thus slowing down the mechanism. As stated before, most of the existing pipeline machines also have an array or multiprocessor structure and therefore pipelining is usually used in supercomputers and sometimes regarded as a speedup mechanism rather than an architectural basis.

Array computers achieve spatial parallelism by having multiple ALU's (usually called Processing Elements or PE's) working synchronously in parallel. A control unit is in charge of appropriate data-routing among the PE's and control of the connecting network. The control unit also performs scalar instructions. Vector instructions are broadcasted to the PE's local memory, i.e., with operands fetched locally. Instruction fetch and decode is therefore done by the control unit exclusively. The PE's are passive devices having no action in the fetching and the decoding. They are of course less complex circuits, otherwise most of their capabilities would be wasted.

The connection network among the PE's which is also controlled by the control unit, can differ in several architectures, from a mesh-network to a crossbar network connection scheme. Array computers are SIMD machines, i.e. single instruction-multiple data, for the same instruction is broadcasted to all PE's which executes it with their local data.

Multiprocessor systems are composed of two or more processors of approximate capabilities sharing access to common sets of memory, I/O and peripherals. Each processor may have its own local memory and data transfer among processors can be made in a multitude of ways, from memory mapping the entire system to providing direct links between processors, i.e., by shared memories or through an interrupt network. A single integrated operating system controls the entire system, the processors interactions and their programs execution.

Most multiprocessor system are **MIMD** machines, i.e., **Multiple Instruction over**

Multiple Data, which implies interactions among the processors if memory is to be shared. If no data would be shared, the structure would rather be that of a MSISD, Multiple (Independent) SISD uniprocessor system. These systems are distinct from multiprocessors and are called multicomputers. The distinction is based on the extent of resource sharing and cooperation in solving a problem: in a multicomputer, there are several processors each having its own operating systems, which may or may not communicate with each other in a network fashion. On the other hand, in a multiprocessor system, one single operating system provides processor scheduling, interactions between processors and their programs at all levels: process, data set, data elements, etc.

A usual distinction made for these machines distinguishes them on bus-based or nonbus-based architectures. Bus-based systems can further be divided in two subcategories: **tightly coupled systems** and **loosely coupled systems**.

Tightly coupled multiprocessors communicate through a shared main memory. The rate at which data can be communicated is of the order of the memory bandwidth. The connectivity between processors and memory can be achieved either by an interconnecting network of one or more high-speed buses, or with a multiport memory. Memory contention when more than one processor tries to access the same memory concurrently, is a factor of performance degradation and a limiting factor to system expansion, although there are ways of reducing contention and increasing the throughput of such systems in some applications.

Loosely coupled multiprocessor systems are systems in which the interaction between processors is not high. In these systems, each processor has its own large local memory and a set of input-output devices. A special message transfer system provides for data exchange between processes executing on different processors. The degree of coupling is therefore very loose and such systems are efficient only when the interactions between tasks are minimal. Performance degrades if the number of messages passed among processors increases.

On the end-line of **loosely coupled multiprocessors** we find systems that some authors still consider as parallel processing [4]. Such systems are more on the multicomputer side and include from cluster connections to Local Area Networks (therefore excluding wide-area networks). As stated, such multicomputers are not really parallel processing machines, although they work in parallel and can cooperate in solving a task. The level of cooperation is however very limited and usually limited to data exchange over the network by the different operating systems.

Nonbus architectures include from completely interconnected systems to star architecture systems. The most widely used structure, however, is the **Ncube**, or **Hypercube**, that will later be described.

The approaches to parallel processing mentioned before are conventional control flow structures, or Von Neumann machines, because instructions are executed sequentially as controlled by a program counter. Recently however a new approach based on the flow of data and henceforth called data-flow computers, was suggested.

The main idea is to have no program counters and to execute an instruction whenever its operands become available. Instruction initiation depends on data availability, independently of the instruction's physical location in memory, thus leading to unordered execution of instructions which theoretically allows for maximal concurrency to be exploited, constrained only by the hardware resource availability.

Data-flow computers are an even greater departure from conventional architectures than parallel control flow machines, which leads to great difficulties. Not only there are no commercial machines yet, as the existing ones where done in a research environment and full documentation cannot be found. Development of such a machine is very complex, as problems will be found both on the development of the architecture itself and on the software, as data-flow computers take no advantage of existing programs and tools. Therefore state-of-the art data-flow computers are still on the beginning of its evolution although some practical special-purpose systems have been presented [5][6].

2 Hardware accelerators for CAD tools

The increasingly important role of computers in today's world brings out a new problem which is itself a challenge as important as the current quest for more computer power.

That problem is of course the existence of tools, programs that allow the user to take full advantage of the possibilities that computers make available in all fields. The quest for powerful computers implies then another quest for software tools that will enable that power to be used efficiently.

In fact, increasing the computational capabilities of a system, is of no interest if those capabilities cannot be fully exploited. On the other hand on some fields of research and study, existing state-of-the art tools are already requiring more resources. Therefore work must progress simultaneously in the development of the architecture of computers and electronics, and on the development of software to fully exploit those same resources.

A particular field of great interest, relates to the development of software tools that will enable the progress of electronics and computers itself. Such tools are aimed at improving reliability, upgrading performance and at decreasing the turnaround time as such improvements are generated and become available.

Examples of such tools, are Printed Circuit Boards Automatic Layout, Integrated Circuit Design, IC placement and routing, circuit simulation, block simulation, system simulation, etc.

Research is being conducted in all these areas and breakthroughs like switch-level simulation [7], the use of AI techniques in routing or placement in the form of Expert Systems [8], or newly proposed simulation techniques [9], make such tools become more powerful thus requiring more resources for itself.

As circuits become more complex and the fabrication techniques evolve, a necessity emerges for an accurate verification of each project before the system is built (be it a complete all-in-one system or a single chip system). Full verification of the functionality of a system must not only be reliable, but also fast enough to provide results in an acceptable time interval.

Parallel Processing thus appears again to be a good solution for simulation speedup, as it is for general problems. The trend here is to provide algorithms with inherent parallelism to efficiently exploit the resources made available.

Several examples of parallel processing architectures as **special-purpose hardware accelerators** are available from research groups around the world [10]. A significant percentage of those accelerators are implementations of logic simulation algorithms [11] [12][13], although some analogue [14] and mixed-level simulators [15] can also be found. The reason for that has to do with the simulation techniques themselves. Logic simulation in fact, leads itself more easily to be performed in a distributed environment due to the event-driven characteristics of the algorithms used, in contrast with the time integration scheme of analogue circuit simulators [16].

These accelerators do not usually make any use of pipeline techniques. That would require specific processors which are also usually not necessary, except for some array processors (for instance solving large sets of differential equations) that were built to very specific time-consuming functions. Some of these machines are made from commercially available products and the multiprocessor structure is the most widely chosen for reasons that have to do with the availability of low-cost special-purpose processors and with the circuit partitioning characteristics of the algorithms. In fact it is easier to have a partitioning algorithm that divides the circuit into several subcircuits and a scheduling algorithm that chooses a processor to receive the circuit and to simulate, having its own local copy of the simulator and some memory for the waveforms. Once the simulation ends, the computed results must be read, and the processor eventually loaded with another subcircuit.

3 Study of Multiprocessor Architectures for a Hardware Accelerator

The main motivation for **multiprocessor systems** generation is either to **increase performance**, **increase reliability**, **meeting distributed applications requirements**, **building general-purpose computers** or to **build supercomputers**.

The availability of low-cost powerful general-purpose microprocessors dued to the progress in electronics in the last decade, makes it easier for the designer to build powerful machines in simple environments.

Microprocessors have become key components in the progress of technology reducing computers to circuit components thus opening up vast application areas. Their low cost has made computers available to everyone, and their usage in interconnecting mechanisms provided a means to tie computers together.

In general, microprocessors and related circuits, offer the designer a wide range of options from single-chip microcomputers to sophisticated end-level computers approaching conventional mainframes. Their amazing progress and low-cost, which still shows no sign of slowing down, makes them attractive and desirable for the design of hardware accelerators for CAD tools.

Being general-purpose and with the possibility of building small computers starting from microprocessors, the multiprocessor structure seems the more reasonable to adopt because it is the one that takes greater advantage of the capabilities of such devices.

We are then lead to **multi-microprocessor** systems [17] because it seems more attractive to use off-the shell components than to design and build special hardware.

Furthermore, microprocessor manufacturers have recognized this trend and the newer types of microprocessors already present features that enable them to be interconnected via a common shared bus. **Multi-microprocessor systems** are however still in an experimental research phase.

3.1 Multiprocessor Systems Architecture

In multiprocessor systems, a number of processors are connected to each other and operating in parallel while exchanging data and co-operating to achieve a common objective.

Such systems are either custom built with multiprocessing in mind or just standard ones gathered in a multi-processor structure. In any case due to the low-cost of general-purpose processors and to the existence of a large number of ancillary circuits of some complexity, the performance/cost ratio of such systems is always high, even if one is not taking advantage of all the possibilities of the processors (like in a master-slave connection, to be described later).

By using standard low-cost general-purpose processors instead of custom built ones, the designer has no way of reaching the internal circuits, and all the connections have to be done through external buses, i.e., only the processors own facilities are available to be used in the desired structure. Nevertheless, as mentioned before, several manufacturers became aware of the wide field of applications to be explored and are offering special features or even circuits to help designers build multiprocessor systems.

Although all minimum workable configurations require a set of memory, which is a common property, the way by which the processors share data and communicate, is in fact what distinguishes the various types of multiprocessor structures.

In any of those interconnecting schemes, memory plays an important role. In a shared memory system, in which all processors have access to the same memory set (for which bus control and contention resolution is required), or in a multiport memory connection, memories are providing for instruction and data storage for every individual processor, storage for common instructions to several processors and also temporary storage for data transfers between them. We can, of course, gradually separate memory into banks, local to each processor. On the other end, we will find processors having its own memory and communicating either through a system bus or an input-output linked structure. Even in such structures, a memory mapping system could allow different processors to access each other's memory as part of their own.

The basic components in a multi-processor are then the **processing elements** (not to be confused with array-processors PE's, discussed before), consisting of a **processor-memory pair** (or, in hierarchical structures, an aggregate of processors and memory), the **switch**, performing intermediate communication such as data routing between processors, and the **interconnection path**, the **transmission line**, either a serial teletype-like line, a fast synchronous line or a set of parallel lines allowing byte or word transfer [18].

Many architectural options exist for system design. Those options can usually be classified as decision trees in which each level presents a new architectural problem.

The main consideration to be put, is the **transfer strategy**, i.e., the strategy for data or messages to be exchanged. The transfer strategy is either **direct or indirect**. In **direct transfer strategy** there is no intermediary between processing elements exchanging messages. The sender specifies the location of the receiver and selects the path; the message suffers no change during transmission

Several types of connections use the direct transfer strategy: one of them is the **complete interconnection**, i. e. each processor connected to all the others, a structure that is rarely used due to the huge number of interconnections lines required for a medium number of processors (grows with the square of the number of processors).

The **loop** is another dedicated path direct transfer method that reduces the number of communication lines. It is an incomplete interconnection providing in fact the minimum number of connections, one unidirectional path to the next neighbour. For one processing element to communicate with another, the message is put on the loop and passes through the necessary switching elements until the destination is reached.

The **hypercube** is still another dedicated path structure using direct transfer strategy [19]. Also called the **N-dimensional cube** or the **N-cube** it consists of 2^N nodes, i.e. computers, each one connected to N others to form a communication network that can be visualized as following the plan of an N dimensional hypercube.

Direct transfer strategy, however, can also be implemented in a **shared path** fashion. One approach is the **shared bus** strategy: before gaining control of the bus, the processor goes through a request procedure to ensure availability. Once the bus is allocated, a master-slave relation is established, the master being the processor in control of the bus, all the others having a slave status.

A way of dealing with competing requests and establishing priorities must be provided, and some processors include features to ease that job. If the shared transmission medium used is a serial line, like the contention bus (Ethernet), a switching element must interface between the processors and the line. If two elements try to use the path simultaneously, the message becomes corrupt and the processing elements must realize that fact and try to retransmit (strategies for retransmission must be adopted so that both retransmissions don't take place at the same time thereby avoiding a deadlock situation: a possible one is to generate a random wait time period and then retransmit).

Communication can also take place using the memory as an intermediary medium. This can be accomplished either by using multiport memories or, more generally, having a uniport memory connected to a system bus (shared memory structure). Procedures for memory read and write access must be provided in hardware or software.

In **indirect transfer strategy** an intermediate stage is in charge of message collecting and processor mapping. That brings up problems such as system expandability, reliability and configurability. The intermediate stage could be a unique device in the system, or a set of localized units. The first approach is a centralized routing transfer control method, while the later is a decentralized one. Both strategies can be considered as having shared or dedicated paths.

In **centralized routing**, a dedicated path structure can be achieved by a star configuration in which a central switch to whom all processors are connected via dedicated paths, performs message collecting and routing. A mailbox scheme where a separate storage area is allocated for each processor communication is implemented. Changing source-destination relationship is achieved by logical mapping of adresses to physical memory which is an interesting possibility in terms of reconfigurability. A loop with a control switch is another possible dedicated path structure.

To build a shared path centralized routing machine, a central switch controls all message transfers by controlling the common bus. Usually the central switch is itself a processor.

Decentralized routing techniques lead to structures that consist of processing elements with dedicated paths to switching elements which in turn are connected through some medium. The topology of switch connections depends on specific requirements or on geographical constraints thus leading to regular (usually seen in multiprocessors used for image processing) or irregular networks. Shared path structures are based on the bus window mechanism in which part of each processor's memory is mapped as a set of memory for other processors in a hierarchical fashion.

As previously stated, the memory structure and the interconnection scheme define the machine's structure. All possible alternatives have strong and weak points that can only be clearly seen and discussed when thinking of the goal for which the system is built.

3.2 Factors influencing Multiprocessors Design Options

The design of a hardware accelerator using a multiprocessor structure meets several sometimes conflicting factors, that should be considered in order to avoid both over- and under-design.

Major motivations for building such systems were already mentioned. The performance and reliability concerns are the most directly related with hardware accelerators for circuit simulation at any level.

Reliability in itself is an important design consideration. Detection of faults and system availability requires increasing attention as the complexity grows. Reliability should be considered both in hardware and software [20] and may sometimes become the main consideration: for example in loop connection or memory shared systems, a failure in a loop switch or the memory path could prevent normal functioning.

Deadlock is another factor to be considered since in many structures a number of resources are shared and the interdependency of processes may be great. Mechanisms for deadlock prevention and avoidance are essential to every structure and can be done either in hardware or software. Also deadlock detection and recovery is important and this usually requires time-out mechanisms and therefore estimations of the time taken in actions resolution. Such predictions are unfortunately not always an obvious task.

Hardware considerations such as **topology and physical interconnection pattern** are related to the requirements of homogeneous or non-homogeneous structures (similar processors in every node or not) and to whether we are dealing with localized or geographically distributed systems . Usually the pattern chosen is a more flexible option and a cube [19] or mesh or other forms of regular or irregular patterns present specific advantages and disadvantages to be considered. The interconnection pattern relates to the data exchange mechanism already discussed and therefore to switches and data paths.

Data paths are characterized by the degree of concurrency. The connectivity level of the nodes in the structure goes from a fully connected structure to a shared bus-like. Shared data-paths present the contention solving problem of having several processors trying to gain control of the path; this can lead to performance deterioration.

Throughput is related to path length: parallel transmission is faster but requires more wires; it is usually used only in shared-bus tightly coupled systems. Typically we have serial transmission of bytes, i.e., sequences of 8 parallel bits. The transfer discipline can be based on a circuit switched mode in which fixed paths uni- or bidirectional going through a number of switches are established between communicating process. It can also be packet-switched with source-destination information injected in the communication line that goes round the switches until destination is reached. Packet-switched is more suitable for indirect transfer strategy, while circuit-switched mode lends itself to direct communication. Data transfer can also be done synchronously as in a direct memory access or asynchronously, waiting for data to be ready. Whether or not a fixed path is allocated, sender and receiver must ascertain between them that data can be sent and that it was received correctly after transmission terminus.

Memory organization is also crucial because it is the main connecting link between processors in some structures. It can either be global and shared or each processor can have local memory coexisting or not with a global memory (which is the most common option).

Last but certainly not the least, the **software development** is one of the most important factors influencing design options. The operating system for a start, must handle several processes running on the same processor concurrently. To have a global operating system or localized copies with communication features depends crucially on the task to be assigned to the nodes and on the estimation of message exchange needs. Database management must be considered carefully, because a centralized system could run into contention problems while a decentralized system, with several copies creates a consistency problem.

In summary, the design options depend on the goal and the system requirements but also on some of the mentioned factors.

4 Circuit Simulation boosted by a Hardware Accelerator

Circuit simulation is a well established design technique for the verification of circuits. This was the starting statement of INESC's tasks proposal to cooperate in Esprit Project 1058.

Circuit simulation's importance is easily seen from the vast research conducted on the subject in industrial, governmental and university environments, and any breakthrough in the matter will have a dramatic impact in industry's near future.

Today's IC technology has evolved so much that the available circuit simulators can't cope with the complexity of most circuits. Two problems seem to be the vast memory needed for circuit and waveforms data storage and the speed of simulation itself. Special design architectures can solve, or help solving the speed problem [21] and the memory needs of conventional algorithms can be minimized using large machines. However, the real bottleneck are the time-integration numerical methods used to solve the set of non-linear differential equations that describe the circuits.

For moderately large circuits, the time required for numerical integration is not acceptable and conventional algorithms simply cannot deal with very large circuits, not to mention the intrinsic numerical problems of those algorithms.

New techniques for solving the nodal equations [9] are very interesting because they can take full advantage of the latency observed in digital circuits, in a natural way, by discretization of the voltage axis in a number of fixed values. This can be viewed as an automatic time step control, which is automatically adjusted while calculating for each node in the circuit the time when it will reach the next allowed voltage value.

Results obtained so far by simulators using this new technique are very promising and have proven the method's efficiency together with large improvements in the simulation time.

In the context of Project 1058, INESC has committed itself to study and develop a simulator, to allow the verification and exploitation of several design alternatives for an integrated circuit.

This simulator, **CINNAMON**, now being developed [22] [23] at INESC, uses an **explicit event oriented integrated algorithm** based on **voltage discretization techniques**.

Substantial improvements in simulation execution time over other circuit simulators have been reported, while keeping the accuracy reasonable. Further, the choice of an event-driven technique provides a dynamic partitioning of the network, because at each time step, only the node that had been scheduled and its neighbours are simulated. This property leads to the idea of adapting and developing the algorithm to a multiprocessor environment.

The increase in performance resulting from the implementation of the simulator in a parallel processing machine will depend on the multiprocessor structure chosen and on the algorithms to be developed in order to allow the exploitation of the inherent parallelism of an event-oriented simulator like CINNAMON.

4.1 The explicit event-driven integration scheme

The event oriented integration algorithm used in CINNAMON is similar to the widely used technique of event driven logic simulation. As stated, the greatest percentage of hardware accelerators reported, were related to logic simulators which leads to the thought that event driven algorithms present inherent parallelism. In fact, the event driven scheme leads itself easily to an implementation in a multiprocessor environment.

In CINNAMON, at every step an event-list is processed. Basically, the algorithm will find out the time at which a node voltage will change by ΔV (the voltage axis discretization) and schedule a new event for the node, inserting it into the list with a label specifying the time the event was scheduled for. The algorithm simulates all the events on the list and generates for each scheduled node, and its neighbours, new events that are in turn inserted on the list. The algorithm has two characteristics that can make it suitable to be adapted to a parallel architecture. One is the event-flow scheme: as simulation steps are "controlled" by an event-scheduler, a dynamic exploitation of latency pervasive in digital circuits is possible. The other derives from the fact that, for each event, only the node associated with it and its neighbours are simulated, which means that a dynamic partition of the circuit is being performed. For each event, only a small set of nodes of the circuit needs to be reevaluated. It is possible that, in a large circuit, several contiguous events in the event list, are related to different sets of nodes and could therefore be simulated simultaneously, if several processors where available.

There are two obvious drawbacks in this parallel simulation scheme. Consider that as simulation proceeds, we solve the equations associated to an event A in one of the processors. Suppose that there is a method to determine if the next event in the linked list, B, can be simulated simultaneously or not, and it happens that it can, and henceforth for the following events. The first problem that can be foreseen is that distinct nodes involved in the calculations related to two different events can cause events to be scheduled on each other via intermediary nodes. If an event A indirectly causes the reschedule of a node that was involved in event B already calculated, and that event is scheduled before event B, then there must exist a method for recovering the status before event B calculations were done and start over again. This problem stands from the event-oriented nature of the algorithm and also appears in logic-simulators, where it has been carefully and extensively treated.

The other problem relates more directly to the partitioning, and to the decision whether events that follow each other in the events list, can be treated simultaneously. If too many restrictions are imposed in order to consider two sets of nodes “independent” and eventually able for simulation in parallel, it may not be possible, in strongly connected circuits, to fully profit from the possibilities of exploiting latency, of circuits that the event-driven algorithm allows. On the other hand if loose measures are used, to allow simultaneous calculations, it might turn out that the first described problem appears several times.

In order to fully exploit the inherent parallelism of the event-oriented integration scheme, the partition algorithm and the multiprocessor structure must be carefully chosen.

4.2 Circuit Partitioning and Processor Scheduling

The implementation of a circuit simulator on a multiprocessor environment requires an algorithm for **circuit partitioning** and **processor scheduling**. Those algorithms are closely related to the structure of the multiprocessor used.

Circuit partitioning can be done statically or dynamically. **Static partitioning** involves the use of a preprocessor that reads the circuit description and separates it into tightly coupled groups. For each group a list of internal nodes and nodes connected to other groups, is kept. Each time a node in the group is reported changing, the group is assigned a processor and simulated. Changes in its nodes will trigger events that will cause itself or other groups to be reevaluated at a later time.

The partitioning algorithm must take into account the resources, i.e., the number of processors available, in order to do proper allocation of groups to processors. If at one extreme we had as many groups as processors, each group could be assigned to a specific processor forever. Any multiprocessor structure could be used in this case, with identical performance (except for possible interprocess communication), and the topology could be downloaded once at the beginning of the simulation. The only data exchanges would be caused by changes in nodes shared by several groups.

Static partitioning, in any practical case, will yield a number of partitions larger than the number of processors, if latency of circuits is to be exploited while keeping all processors busy, therefore not wasting resources. In this situation however, some problems can be foreseen: bus shared structures can suffer from bus contention if each simulation requires the download of the topology and waveforms; non-bus structures (like the hypercube) present the problem of simulation control and sharing of data between processors that are not neighbours.

IMEC's initial proposal for the architecture, a master-slave connection, seems a simple and attractive structure [24]. The machine is composed of several processors, connected over a common bus to a VAX-like machine that hosts the entire system. All processors are slaves in the sense that all data transfers are done by the host computer. The host must keep a record of all subnetworks and each time a group is scheduled, it chooses a processor and downloads the topology and the linked list of events to be simulated on that group. When the simulation ends, the slave sends a message to the host and waits for it to read the simulation results and perhaps to download a new topology.

The master-slave connection has three main disadvantages. First, all processors must wait for the host to read their results before proceeding; if the host is computing, which partition should be simulated next, the slaves which have have finished their simulations become idle, since they can't take control of the bus to write their results into memory. Second, this configuration is not expandable beyond a limiting point where the computing/communication ratio becomes high. Third, it is not a very usual structure and therefore is not easily found commercially, although some existing commercial structures can be configured in a master-slave connection.

The Philips experimental multi-processor [25] is also a bus-oriented machine. It has a global shared memory, a host controlling the whole system, and up to fifteen processors connected over the shared bus. Each processor has local memory. A peripheral bus exists which minimizes the problem of of bus contention. All communications between processors and between processors and the host, are done through the bus, but any processor can gain control of the bus. As the system is memory mapped, all processors can access the global memory and each processor's own local memory. These features enable them to communicate without the host direct intervention and, furthermore, message transfer does not have to be done exclusively through the shared memory set. This may become interesting because a processor can schedule new events on a group already being simulated, without the need for the host to centralize all the information. On the other hand, the lack of a floating point co-processor may turn out to be a bottleneck for the implementation of a circuit simulator.

In the bus-world, other machines, should be considered as well. The Multimax computer from Encore Corps., the Balance computer from Sequent Inc. or the Flex/32 from Flexible Comp. are three multiprocessor with fast data transfer rates, expandable to twenty (twelve for the Balance) processors, with fast cache memories. Each one presents some software tools developed in order to exploit the parallelism of the machine. They are certainly very promising computers if a bus machine is to be chosen.

From the non-bus world, hypercube interconnection schemes represent the main commercial architecture. They avoid the bandwidth problem of shared bus systems, thus providing expandability, by having dedicated paths between each node and a number of its neighbours. In such a computer structure, sets of groups resulting from a network partition and having more than average common nodes, should be assigned to processors directly connected. The hypercube connection presents however two significant problems.

On one hand simulation control would become difficult because the paths from the controlling node to a processor are not the same for every processor; if we choose to have more than one controller distributed around the nodes, then the consistency problem becomes very important and difficult to handle. On the other hand, every time a processor wants to communicate a change in one of its outputs, he must do it to its direct neighbours and that information must also be routed via intervening processors. This can slow overall processing rates if it occurs often.

Worth mentioning is that although it might seem a good partitioning technique to have all groups of similar size, that may not always be true, especially in bus-shared computers. In fact, this could increase the probability of having more than one simulation ending at the same time, and the processors would have to wait orderly until the bus is delivered to them or the host reads their results; not only the host is idle while the processors are simulating, but also each processor is idle while all the others are being served. This is obviously a waste of available resources.

Dynamic partitioning may prove itself interesting in balancing the average load in the different processors. Further, groups that are found to have high ratios of message exchange, could be simulated together. Nodes in groups that are found to be scheduled many times could lead to further partitioning of groups, therefore taking advantage of the information about processor utilization and communication. Dynamic partitioning has basically the same architectural needs than static partitioning, although a decrease in average communication needs and processor idleness can be obtained if a clever algorithm is used.

Perhaps the most important problem associated with a multiprocessor implementation of a circuit simulator is related to **data consistency**. If one wants to exploit the resources available in a multiprocessor computer, one has to simulate in different processors two partitions of the circuit that may have common nodes. One could then find out that some of these simulations were performed with out-of-date data if we receive information about changes in nodes that were supposed to be stable at the beginning of the simulation. Therefore some computations may have to be redone.

To solve this problem and to avoid checking for precedence constraints one has to simulate every subnetwork as if the values of the nodes were correct for all times, and keep a history book for each subnetwork, encoding the value of the inputs at the time the simulation started. When an output node in a partition changes, it sends a message to other partitions (possibly also or only to the host), for which that node is an input. The receiving partitions update their record book and, if necessary, redo their simulation. To avoid redoing all the simulation process because node value was found to be false, one could employ a **checkpoint strategy** and a **roll-back procedure** in order to find the time for which that transition in the input should have been considered. The set of checkpoints allows us at any moment to go back to past values and re-simulate, and thus present a complete history of the simulation.

Ideal checkpoint strategy should optimize the tradeoff between the amount of resimulation necessary (due to the roll-back mechanism) and the size of storage area. This imposes constraints in the checkpoints frequency, because the strategy must provide at least one consistent state to fall back into, and the possibility to progress if an unexpected synchronization, done probably under host control, happens. Checkpoint strategies will have to be considered with extreme care.

The partitioning algorithm is important in all the above mentioned problems. It must be able to partition a given topology into subgroups. Several different schemes for partitioning can be devised. SLOCOP, developed in the context of project 1058, uses a partitioning algorithm which looks for known structures such as gates or logic units, using AI techniques. Obvious advantages of such a partitioning method is that once a well known logic unit is recognized, its behavior is immediately known and, if available, macromodels can ease the simulation work. A drawback is that the possible subcircuit types must be defined using a special purpose language.

A partitioning algorithm, [7], that searches for all the nodes in a bidirectional path and groups them, may be more advantageous because it is systematic and fast. On the other hand, it does not provides any knowledge about the subnetwork, which means that no knowledge of the group behaviour is available at partitioning time.

Other algorithms that could stay midway between the two described hypothesis must be considered, especially for easy to identify common structures.

Processor scheduling is an important factor to be thought of, and is highly dependent on the chosen architecture. **Random scheduling** is easily done and though it follows no judgement, it may prove to be acceptable. In a bus-based structure, random scheduling seems as good choice as any other one. It is fast and as all communications are always done through the bus, all processors are indistinguishable by location, whether the processors are slaves and bus control is exclusive of the master, or if they have the ability to initiate message transfer,

In hierarchical multiprocessor structures, like bus-window systems, loop connections or hypercubes, random scheduling is not very appropriate. In these structures, connected partitions should be assigned to neighbour processors to avoid establishing large communication paths. In a hypercube where each node has dedicated paths to its N neighbours, it is convenient to keep a record of all processor assignments and of nodes common to various subnetworks, and to try to have strongly connected subnetworks under simulation in neighbour nodes of the structure.

As stated, most of the algorithms to be implemented, must be adapted to the multiprocessor structure where they will lay. Also the decision between synchronous or asynchronous algorithms depends on the architecture of the machine and on the available tools. Synchronous algorithms are easier to implement and debug. Asynchronous algorithms, although more elaborated will perhaps allow us to better exploit the resources available on a multiprocessor environment. Both possibilities will be studied along with the different partitioning techniques and processors scheduling.

References

[1] - José C. Delgado,

"The Incremental Machine"

[2] - Kai Hwang, Fayé A.Briggs,

"Computer Architecture and Parallel Processing",

McGraw-Hill, 1984

[3] - Tom Manuel,

"Parallel Processing: the Pace quickens",

Electronics, September 1986

[4] - Paul Wallich, Glenn Zorpette,

"Minis and Mainframes.",

IEEE Spectrum, January 1986

[5] - Winfried Hahn, Kristian Fischer,

"MuSic: An Event-Flow Computer for Fast Simulation of Digital Systems",

22^{nd} Proc. ACM/IEEE Design Automation Conference, 1985

[6] - Gary Catlin, Bill Paseman,

"Hardware Acceleration of Logic Simulation using a Data-Flow Architecture",

International Conference on CAD, 1985

[7] - Randal E. Bryant,

"An Algorithm for MOS Logic Simulation", LAMBDA, 4^{th} Quarter, 1980

[8] - Rostam Joobbani,

"WEAVER: An Application of Knowledge-Based Expert Systems to Detailed Routing of VLSI Circuits",

Phd Dissertation, CMU University,

Pittsburg, PA, USA

[9] - Luís Vidigal et al.,

"CINNAMON: Coupled INtegration and Nodal Analysis of MOs Networks",

23^{rd} Proc. ACM/IEEE Design Automation Conference, 1986

[10] - Jeffrey Arnold, Christopher J. Terman,

"A Multiprocessor Implementation of a Logic-Level Simulator",

International Conference on CAD, 1985

[11] - Hiroshi Uno et al.,

"A Parallel Implementation of MOS Digital Circuit Simulation",

International Conference on CAD, 1985

[12] - Fohru Sasaka et al.,

"HAL: A block level hardware logic simulation",

20^{th} Proc. ACM/IEEE Design Automation Conference, 1983

[13] - Monty E. Denneau,

"The Yorktown Simulation Engine",

19^{th} Proc. ACM/IEEE Design Automation Conference, 1982

[14] - David M. Lewis,

"A Hardware Engine for Analogue Mode Simulation of MOS Digital Circuits",

22^{nd} Proc. ACM/IEEE Design Automation Conference, 1985

[15] - Tohru Sasaki et al.,

"HAL II: A mixed-level Hardware Logic Simulation Systems",

23^{rd} Proc. ACM/IEEE Design Automation Conference, 1986

[16] - Georg K. Jacob et al.,

"Direct-Method Circuit Simulation using Multiprocessor",

International Conference on CAD, 1985

[17] - Y. Paker,

"Multi-Microprocessor Systems",

Academic Press, 1983

[18] - J. M. Hancock, S. DasGupta,

"Tutorial on Parallel-Processing for Design Automation Applications",

23^{rd} Proc. ACM/IEEE Design Automation Conference, 1986

[19] - Charles L. Seitz,

"The Cosmic Cube",

Communications of the ACM, January 1985

[20] - Harold Lorin,

"Parallelism in Hardware and Software: Real and Apparent Concurrency",

McGraw-Hill, 1972

[21] - David Tsao, Chin-Fu Chen,

"A "Fast Timing" Simulation for digital MOS circuits",

International Conference on CAD, 1985

[22] - Luís Vidigal, Horácio Neto,

"A Novel Circuit Simulation Algorithm: Implementation Issues",

2º Simposio Electrónica das Telecomuni cacões,

May 1986, Lisboa

[23] - Luís Vidigal, Horácio Neto,

"CINNAMON: New Results and Improvements",

submited to publication to the European Conference on Circuit Theory and Design,

Paris, 1987

[24] - Workpackage B, Task B.b, "Esprit Proposal: Knowledge Based Design Assistant for Modular VLSI Design",

January 1986

[25] - Project 1058 Internal Communication

"The Philips Experimental Multi-Processor System"