Chapter 1

Introduction

This chapter aims to give a short introduction to the terms used throughout the project. Because this project is dealing with the study of interconnection networks with a simulator, this introduction includes basic concepts from interconnection networks as well as from the design of simulators.

1.1 Interconnection Networks

People seem never satisfied with the speed and performance of computers. Nowadays, there are many areas in which problems are not solved quick enough, or not with the adequate precision. This is, for example, the case of weather forecasting. Although processor manufacturers are working very hard on speeding up their designs, this seems never enough. Thus, at some point in history somebody, having this in mind, must have thought *if one computer does not suit my needs, how about two?* This is where the need for interconnection networks comes.

Initially, one would think that having two computers would perform twice as good as a single one, but this is not true. The foundation for this statement is twofold. First there is the unavoidable communication overhead, and second there is the impossibility of adequately balancing the computing load among the computers. As an example we can think of two workers doing a certain job. This job should be divided into work packages and distributed among the two people. Depending on the job division and the dependencies between work packages, there will be times when one worker will be waiting for the other to complete a work package, keeping him from working on the next. This lack of efficiency can be solved, to a certain extent, by carefully studying the job and adequately dividing it. But what can not be eliminated is the time the workers spend telling each other the outcome of their work packages. The people involved with interconnection network design aim to minimize these degradations on the network's

performance.

1.1.1 Network Topology

The case of connecting two computers together is very simple. Only one link is required to transfer information between them. If a higher number of *nodes*[1] is needed, the number of links required to totally connect the nodes grows geometrically as equation 1.1 shows.

$$l(N) = \frac{(N-1)N}{2} \tag{1.1}$$

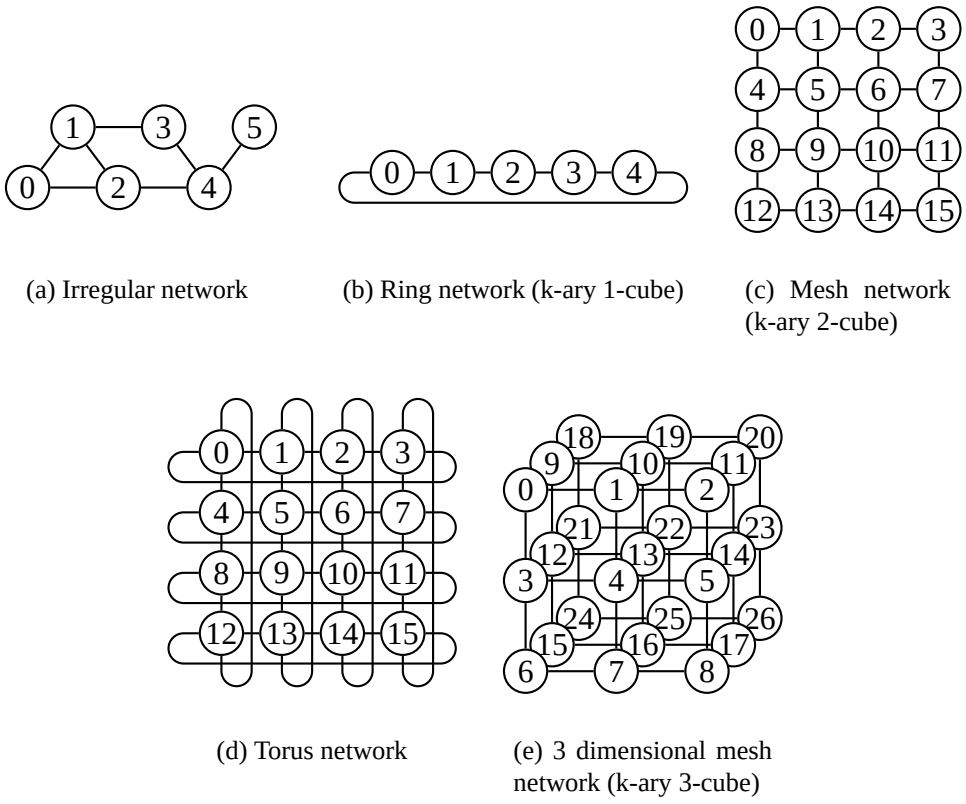
Where l is the number of links and N the number of nodes. For example, to totally connect 16 nodes the amount of links needed is 240. If we think of linking the computers in an office, this does not result affordable. Therefore, there is a need of reducing the network complexity. By adding a special element to every node, a piece of information would be able to be conveyed through the network between two nodes not necessarily directly connected. This special element is able of reading the destination of a piece of information and sending it to another node nearer to the destination if necessary. Such an element is called a *router*, in the sense that it finds routes through the network for the information to travel in.

The addition of routers to a network, enables a subset of the links in the totally connected network to be used. Then each link will not only carry information between the nodes at its ends, but also the information forwarded by these coming from nodes further away. This kind of network, in which the links are shared, are called *switched networks*.

Depending on which subset of links from the total interconnection network is chosen we get different network types or topologies. If the connection scheme does not respond to a certain rule, the network is *irregular* (Figure 1.1(a)), otherwise it is *regular*. In the case of regular networks, there is a further classification; if the view of the network from every node is the same, the network is said to be *symmetric* (Figures 1.1(b) and 1.1(d)) and if not, the network is *asymmetric* (Figures 1.1(c) and 1.1(e)). Also, depending on the number of neighbors a node can have, networks are said to be n-dimensional. Figure 1.1(b) shows a network where each router has two neighbors so it is 1-dimensional. Figures 1.1(c) and 1.1(d) show 2-dimensional networks and figure 1.1(e) shows a 3-dimensional network.

Interconnection networks can also be described by certain figures of merit. These are topological measures that allow to compare, with an objective point

[1] This is a generic way of referring the element that is to be connected. It can be a computer in a local area network, a processor inside a machine, a telephone in a telephone network...



(a) Irregular network

(b) Ring network (k-ary 1-cube)

(c) Mesh network (k-ary 2-cube)

(d) Torus network

(e) 3 dimensional mesh network (k-ary 3-cube)

Figure 1.1: Network examples

of view, different networks. The *diameter* of an interconnection network is the number of *hops*[2] necessary to communicate the two farthest apart nodes. This value can give an idea of the maximal transit time[3] a piece of information needs to cross the network.

The other value, the *average distance*, is the average number of hops necessary to communicate any pair of nodes. Therefore, this value suggests the idea of average transit time[4].

1.1.2 Parallel Computers

The networks treated in this project are intended for the use in parallel computers. A *parallel computer* is a set of computing elements that cooperate in order to

[2]the number of links a piece of information goes through.

[3]Assuming there is no congestion in the network. This is, the information flows without stopping form the source to the destination.

[4]Assuming no congestion and random selection of source and destination nodes.

solve a big problem. In order to cooperate, these elements are connected together with an interconnection network as those described above. However, there must be a higher level mechanism to structure the communication between the different processor elements. Attending to this, parallel computers can be divided into two families. Although the computers in each family have specific hardware particularities, at a software level they are able of emulating the behavior of the other family.

Message Passing Computers

These machines are also known as multicomputers. The main feature is that the programmer has to write in the code when and where should the communication occur. The communication is structured in send and receive primitives. This is done by using special libraries like MPI[5]. The processor elements in message passing computers are normally have all the typical elements of a normal computer (processor, memory, I/O) as well as a connection to the interconnection network.

Shared Memory Computers

These machines are also known as multiprocessors. Opposed to the multicomputers, the programmer needs not to care about how the communication occurs. From the software point of view, multiprocessor machines have one huge memory shared by all the processors of the machine. However, at a hardware level, the memory is divided among the processors and the network provides each with the memory of the others. Typically the processors of a multiprocessor machine share the I/O resources.

1.1.3 Routing

Focusing again on the network infrastructure of parallel computers we have a set of nodes connected to each other with a certain network topology. On each node there is a router unit that enables it to forward pieces of information to other nodes, therefore allowing networks to reduce the number of links. But now there must be a algorithm to allow information to reach its destination quickly. This is called *routing*. The routing algorithm is distributed among all the routers, this is, a router does not decide the whole path of piece of information, it just decides to which of its neighbors it will send the information to.

The routing algorithm must, at least, have knowledge of the network topology and the source and destination nodes of the information. In more complex algo-

rithms, congestion status may also be taken into account. This knowledge must enable the algorithm to find routes through the network.

Depending on which route is assigned to a pair of source and destination nodes, the routing algorithm can either be *deterministic*, in that the information traveling between two given nodes gets assigned always the same path, and *adaptive*, in which information finds its way through the network depending on, for example, traffic conditions.

In addition to getting the information to its destination, it is desirable that this is done fast. Therefore routing a piece of information can be done through a *minimum path*. This is obviously the path that connects two nodes with the minimum number of hops. Nevertheless, there are some networks, such as the irregular, or traffic conditions in which *misrouting*, i.e. routing through non-minimum paths, can give better results.

In this project we will see a static, minimum path, deterministic routing called *dimensional order routing*(DOR). In a k-ary n-cube network, such as a mesh, a piece of information has to move a certain amount of hops in both dimensions, dimension ordered routing forces the completion of the movement in one dimension before proceeding in the next. This eliminates any resource cyclic dependencies between different dimensions.

1.1.4 Flow Control

Up to now we have discussed about 'pieces of information' crossing the network. But this information should be organized somehow. The structure used in this project is as follows, we consider arbitrary sized *messages* divided into a certain number of equally sized[5] *packets*, each of these packets is composed of a sequence of *flits*. Flits are the minimum amount of information the flow control can distinguish. This is, flits travel as atomic units through the network and can not be divided whatsoever. A flit is composed, as well, by a fixed number of *phits*, which are the amount of information the physical infrastructure handles at a time, i.e. the width of the link, and are composed by bits. While the packets in a message can arrive in any order, the flits of a packet must be received properly ordered and not mixed with those form other packets.

Having the idea of the information structuring we can now describe the different flow control functions. On one side we have *store-and-forward*[6], in which a router waits until it receives the end of the packet before sending it to the next router. In the ideal case, the time a packet needs to get to its destination is proportional to its length and the number of hops.

[5]In order to have equally sized packets, the message must be padded so its length is divisible by the packet length.

On the other side we have *worm-hole* [2] flow control. With this function, a router does not wait for the end of the packet to send it over to the next router. The benefit of this approach is that the time a packet needs to get to its destination is only proportional to the number of hops and the buffering space needed is much smaller. However there is a drawback, when the head of the packet is blocked, the packet body stops advancing and blocking some channels from other routers.

There is also a mixture of both functions explained above. It is called *cut-through*[6]. This function behaves like the worm-hole, in that it sends flits to the next router as soon as possible, but once the head of the packet is sent, there must be enough space in the receiver router for the rest of it. This avoids having packets blocking several routers at a time. It can be thought of a worm-hole function when traffic is low and a store-and-forward function when the traffic is high.

None of the flow control functions presented in this project consider packet discarding or retransmission. If the network gets congested no more packets are injected.

1.1.5 Anomalies

When the design of the routing algorithm and flow control functions is not carefully done, there are various anomalies that might occur. The avoidance of some of these is critical as they can draw the network to an unusable state.

Deadlocks

It is said that, when there is a cyclic dependence on resources with no chance of being solved, the network reaches a *deadlock* state. Most of the network topologies used in this project are composed by a set of interconnected rings, therefore they are specially deadlock-prone.

There is a large amount of work on preventing and recovering from deadlocks. Deadlock recovery normally involves packet discarding so we will be concentrating in deadlock avoidance. This can be done by ordering the acquisition of resources in such a way that static dependencies among packets cannot form a cycle. One technique to avoid this involves the use of *virtual channels*. A link with various virtual channels works as various logically independent links. However, at a physical level the virtual channels share the data-path and have independent control signals. To avoid deadlock situations the traffic is split among the virtual channels so that a cycle does can not occur. Nevertheless, this kind of solution suffers from high hardware complexity and nonuniform use of link resources. Therefore other techniques have appeared based on different concepts. The *bubble flow control*, for example, is a low cost deadlock avoidance mechanism. This

consists in restricting the injection of packets to a ring, so that the packets in the ring can move[1].

Starvation

When a resource is to be shared among various consumers, these need to have a fair chance of getting the resource. If not, *starvation* may occur. In this situation, there might be a router that favors some of its inputs and neglects others, causing the latter to wait for and undetermined length of time.

Livelock

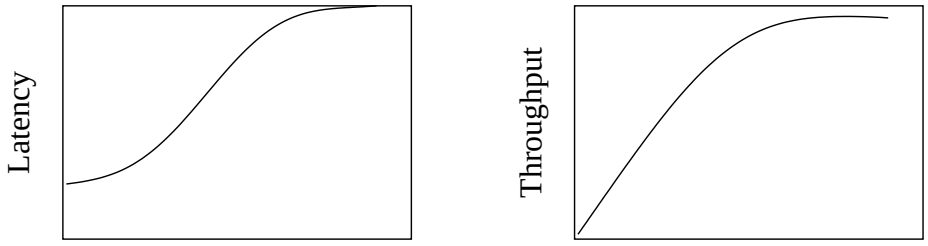
When routing packets through the network, care must be taken in order to ensure that the packets will finally reach their destination and do not wander within the network for ever. This situation is called *livelock* and can occur in when *non-minimum path* routing is used. Opposed to the concept of minimum path routing, non-minimum path routing allows packets to advance in any direction, including those that do not approach the destination.

1.1.6 Network Performance

Interconnection networks are a means to allow information to be sent among a number of nodes. A way of measuring the amount of information that traverses the network is to add up the traffic of every input or output. These will give the units of information (flits, phits, bytes, bits...) per unit time (cycles, seconds...) that enters or exits the network. This is commonly known as input or output load. However, the load is strongly dependent on the number of nodes of the network and it is sometimes interesting to be able to compare networks of different sizes, thus there is also a normalized load that can vary from 0 to 1 independently of the size of the network[4].

Without going into too much detail, there are two parameters that represent the performance of the network. These are the packet *latency* and the *throughput*. Packet latency is the time elapsed since a particular packet enters the network until its last flit reaches its destination. Of course this value is different for every packet, so to give a measure of the overall performance, the mean latency is calculated. The throughput is the amount of information per unit time, that the network has transmitted, i.e. the output load.

The values of mean latency and throughput can be calculated while varying the input load. If this data is represented in two graphs, as in figure 1.2, the relation between both values can be seen. In these graphs, two zones can be observed. A linear zone, when traffic is low. Here the latency is almost flat and



(a) Latency (b) Throughput

Figure 1.2: Typical network behavior

throughput grows linearly. In the linear zone, all packets get to their destination in a reasonable time. The other zone is the saturation zone it happens when traffic is high, here latency and throughput stabilize at their maximum values. This means that the network is not able of managing that amount of traffic, therefore, the message latency and output traffic are constant.

When the input load is at minimum, the latency is also at minimum, this latency value receives the name of *base latency*. It is important because, as there is very little traffic and there is no congestion, it tells the minimum time a packet needs to traverse the average distance network[6].

1.1.7 Network Traffic

When studying the behavior of networks under real parallel applications, it can be observed that the stress put on all nodes is not the same. Depending on which application is running, there are different *traffic patterns* appearing on the network.

Because the cost of simulating real applications is very high, synthetic traffic patterns have been used in order to analyze a network at a lower computational cost. Traffic patterns also provide a 'standard' set of stimuli that can be easily generated. Among others, there are the following traffic patterns.

Random Messages are set between pairs of random nodes.

Local Messages are sent form each node to a range of neighboring nodes.

Broadcast Messages are sent from a node to every other.

Transpose Messages are sent from node (i,j) to node (j,i).

[6] Assuming random selection of source and destination nodes.

If more realistic studies are needed there is also a possibility of using traces. These are files containing the network activity of a real parallel machine. This information is then fed to the simulator to see how the network behaves. This method offers very good results while not being very costly.

In order to simulate real network traffic with higher accuracy, a processor simulator is connected to each node of the simulated network, then a parallel application is run on the processor so that the network traffic can be studied. As this last approach simulates the processors and the network it is more costly but the results have a higher accuracy.

1.2 Simulation

Simulation is a technique that allows a cheap and fast way of studying, tuning and testing real systems without actually building them. This technique has its origins a long time ago, when scientists built scaled models of a system in order to check how it worked before building the real system. Nowadays, with the dawn of computers, these scaled models are represented by mathematical equations and rules inside a computer. Then scientists are allowed to apply some kind of stimuli to the model, in order to understand the system. The computer programs that do this are generically called simulators.

In order to test, tune and compare the performance of the different interconnection networks and routers presented in this project, we need a simulator program. To justify the structure of the simulator that will be used, a brief introduction to some simulation concepts will be given here.

1.2.1 Time Axis

Most simulator programs progress along the time axis performing calculations at certain points until the end of the simulation is reached. The nature of the system to be simulated puts some restrictions to how the time axis is treated. For example, the accuracy when integrating differential equations is highly influenced by the time step used. In the case of the simulation of synchronous digital systems, which is the case treated here, the time axis is discrete, so the simulator must be able of resolving the clock period of the system.

The simulators advance along the time axis can be done in fixed or in variable steps. The variable step approach enables the simulator to *skip* time periods of inactivity reducing the simulation time. In discrete systems this is usually done with events. Events are generated with a certain timestamp in the future and stored in a queue. The events in the queue are processed in timestamp order by the main loop of the simulator. The current simulation time corresponds to the timestamp

of the event being processed in each moment. The processing of an event causes the state of the system to change and more events to be generated.

Event-driven simulators have a drawback, when the activity of the system is high the overhead of creating and sorting events is too big and it is sometimes preferable to use the fixed step approach.

1.2.2 Accuracy vs. Simulation Time

A different aspect of simulator design, is the tradeoff existing between simulation time and the accuracy achieved. This is, the higher accuracy needed, the longer the simulation takes.

When modeling a system, there must be a selection of the aspects that will conform the model. This is, the model should only have the features that determine the functionality of the system, and not others that can unnecessarily complicate it. Therefore, the difficulty of building a model lies on the appropriate choice of the system's features depending on the purpose of the analysis.

Simulation is a numeric technique that performs experiments on a given model in order to extract data that describes the functioning of the modeled system. Normally, a simulation involves a huge amount of operations and variables. Thus, simulation is nearly always done with the aid of computers.

When using a simulator, there are various things the must be kept in mind:

- The execution of a simulation can be very long. Therefore the model must be kept as simple as possible while resembling the system's features in maximum detail.

- As the results depend on the manipulation of random variables. There should be an averaging of many experiments in order to get a proper result.

- Seldom there is a model that mimics the system completely. There is always some detail that is not taken into account. Hence, the results of the simulation are never to be interpreted as the performance of the real system. This error must be taken into account when reaching conclusions out of the results of a simulation.

In general, the error in the results of a simulation grows as the model is simplified. Therefore, at the beginning of the study of a system, the model may be fairly simple, giving high error but short simulation times. And as the study advances, a more refined model can be designed to achieve higher accuracy.

1.3 Motivation and Aims

The simulation of interconnection networks is a key issue if there is a need to study these systems. To this respect, a simulator called SICOSYS was developed and has been very successful. However, the simulator is nowadays obsolete. Although its not structurally limited, the simulation times it offers are too big to make it a useful tool.

Historically the use of super-computers has been something reserved for scientific purposes. Therefore, the study and design of interconnection networks has been done under the scientific application environment. The analysis of this kind of applications has been possible with the aid of execution driven simulators. These are able of executing an application without system code[12] and analyze the network traffic it generates.

Looking at the Top500[13] list, there are starting to appear more and more systems devoted to commercial, finance, database or WWW applications. Thus, there is an urge to study the impact of the interconnection network in these applications. When trying to analyze these applications in the same fashion as with the scientific ones, it has been noticed that those make an extensive use of operating system functions and this can not be overlooked. Therefore, the application can not be run on its own in an isolated environment.

The release of a *complete system simulator* called SimOS[11] has initiated the idea of joining SICOSYS to it. SimOS is a very complex program that simulates a parallel machine completely, it simulates the file system, network interfaces, etc. The virtual machine is capable of running almost any application in a slightly modified operating system. By replacing SimOS built-in interconnection network code with SICOSYS, there would be the possibility of studying real application traffic of almost any application wanted.

One of the aims of this project is to reduce the simulation time of SICOSYS in order to make it a feasible candidate to be joined to SimOS.

The second aim is to allow the study of very big networks. In order to solve very complex problems, the tendency is to use bigger networks with thousands of nodes[7]. The enormous size of these networks cause that small reductions in their topological dimensions make big improvements in their performance. Therefore, making SICOSYS able to test new topologies for large number of nodes is very interesting.

These two aims require an optimization of SICOSYS. This will be done in two ways. On one hand, there will be a thorough analysis of the structure of SICOSYS trying to find ways of making it run faster. These optimizations will not affect the precision of the simulator at all. On the other hand, a way of writing simplified models will be added. This will speed up the simulator greatly at the cost of a loss of accuracy.

1.4 Overview

The contents of this project can be summarized as follows:

Chapter 2 presents the simulator used in this project. This powerful tool has been very successful in the past, but nowadays the network size and complexity has grown to overwhelm the capacity of this simulator. This chapter will describe the general structure of the simulator, point out some problems and propose solutions for them. In addition, a new family of router models will be presented.

Chapter 3 quantifies the improvements in the performance of the simulator caused by the optimizations proposed in chapter 2.

Chapter 4 summarizes the achievements of the whole project and suggests future developments for the simulator.

Appendices are a collection of documents that conform the manual of the simulator. These documents were developed to give an introduction to its usage as well as a exhaustive user manual and an introduction to the programming concepts needed to extend the simulator.