eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

Department of Computer Architecture and Technology

Performance Evaluation of Interconnection Networks using Simulation: Tools and Case Studies

PhD Dissertation

Javier Navaridas Palma

Advisor: Prof. José Miguel-Alonso

September, 2009

To the memory of
my beloved Mother

To my girlfriend
for her patience
and support

To family
and friends

Acknowledgements

There are so many people that helped and supported the author that an acknowledgements list citing them one by one would be almost infinite. Consequently, I will try to make this list as brief as possible:

- To all the members of my research group, particularly to my advisor Prof. José Miguel-Alonso for his guidance.
- To all the colleagues from several different institutions that helped developing collaborative works: The University of Cantabria, The University of Burgos, The Technical University of Catalonia, The University of Adelaide, The University of Manchester and Zurich IBM Research Lab.
- Author is especially grateful to the people from the University of Manchester and to the people from Zurich IBM Research Lab for giving me the invaluable experience of working in international environments, as well as for the received human support.

Most of the research work carried out for this dissertation was supported by the Spanish Ministry of Education and Science, grants TIN2004-07440-C02-02 and TIN2007-68023-C02-02, and by the Basque Government grant IT-242-07. The author was supported by a doctoral grant of the University of the Basque Country UPV/EHU.

Abstract

This dissertation focuses on the performance evaluation of interconnection networks. It briefly introduces supercomputing and shows three different classifications of computing systems, which are used to locate the performed researching work. Moreover, the most common methodologies for the performance evaluation of such systems are discussed showing their advantages and limitations.

This dissertation thoroughly describes the simulation environment developed within the author's research group and all the related tools. Remarkable contributions to this environment are the trace-driven engine and the application-kernels that allow the evaluation of interconnection networks using realistic loads.

This environment is used to perform several researches in the area of interconnection networks that are shown in the form of four case studies. The first one is the evaluation of the twisted torus topology, a variation of the standard torus which offers better topological properties. In this study we show a pitfall of the derivation of the theoretical throughput from the bisection bandwidth. Furthermore, the two topologies are confronted in order to assess if the better topological characteristics of the twisted torus lead to better-performing execution of applications. The second case study evaluates the thin-tree topology, an alternative to the over-dimensioned *k*-ary *n*-tree topology which is shown to be more cost-effective. The third case study evaluates the interconnection network of SpiNNaker, a large-scale system-on-chip-based architecture with severe restrictions in terms of power consumption and chip area; this evaluation is specifically devised to evaluate a bespoke router mechanism focusing on the stability and the fault-tolerance of the system. The last case study measures the influence that job and task allocation policies have on the execution time of parallel applications. It is showed to be significant and dependent on the size of the system and the number of concurrent applications. The conclusion of this study is that allocating applications into isolated subnetworks leads to faster execution of applications. These evaluations have been carried out using mainly simulation, although some results have also been mathematically derived.

Table of Contents

List of Figures

List of Tables

Chapter 1. Introduction

Supercomputing is a very valuable resource which is continuously growing in importance in business and science. Most current scientific studies rely on analysis and modeling of different natural phenomena and/or technological processes, which often require a huge amount of computing power impossible to attain using regular *off-the-shelf* computers. For example, physicists, chemists or pharmaceutical researchers simulate, for several different purposes, interactions between large numbers of molecules. Likewise, in the business context, corporations demand large amounts of computing power in order to use data mining software over their huge data bases, with the objective of extracting knowledge from raw data, and to use that knowledge to their advantage. Obtaining patterns of consumer habits, boosting sales, optimizing costs and profits, estimating stocks or detecting fraudulent behavior are just a few interesting application domains.

At any rate, in both contexts, the required computing power is only limited by the available resources. In general, when resources are increased then the number of runs, the grain-size (whichever this means in the particular application context), the size of the datasets or any other parameter that affects execution time is increased accordingly to fully utilize the compute power. In other words, the magnitude of the experiments is scaled up in order to take advantage of the available resources. This means that there is a permanent demand of supercomputers able to cope with these challenging workloads.

A supercomputer is not only a piece of hardware. It is actually a multipart system that integrates a large collection of hardware and software elements. Therefore, the design of a supercomputer is a complex task that comprises the selection and design of multiple components, such as compute elements, storage, interconnection network, access elements, operating system, high performance libraries, parallel applications, etc. The performance of all these components has to be properly evaluated in order to select the most effective (again, the exact meaning of *effective* depends on the context) taking into account the purpose of the system and the workloads that are planned to be executed on them. Furthermore, the complete system has to be evaluated as a whole, because unexpected interactions among components can make the system suffer severe performance losses.

Moreover the interconnection network, a specific-purpose network that allows compute nodes to interchange messages with high throughput and low latency, is a key element of any supercomputer because its performance has a definite impact on the overall execution time of parallel applications, especially for those that are fine-grained and communication intensive. Any delay suffered by the messages traveling through the network will harmfully affect the execution time of the applications. This is the reason why we should not decide lightly about the network that interconnects computing elements in a high performance computing site. The evaluation of an interconnection network is a complex task that requires deep knowledge about how parallel applications make use of the network in order to properly mimic their behavior.

Our interest revolves exactly around this topic: the performance evaluation of interconnection networks. Special focus is put on simulation-based evaluations, on the characterization of application traffic and on the analysis of the effects of network-level decisions on the behavior of applications when being executed over different networks. Along this dissertation we will introduce several methodologies to evaluate the performance of supercomputing systems. This dissertation will also describe our in-house developed simulation environment. Furthermore it will carry out several performance evaluation studies of a variety of systems using the proposed tools and methodologies.

This first Chapter briefly introduces supercomputing, showing the classification of computer systems from several different viewpoints which will be used to bound our research. Finally, the Chapter is closed with an outline of the rest of this dissertation.

1.1 Classification of Computer Systems

This Section is devoted to provide classifications of computing system from three different points of view: the system architecture, the objectives of the system and the network organization.

1.1.1 System Architecture

A very common approach to classify computing systems is the well-known Flynn's Taxonomy [Flyn66] which is based on the amount of concurrent flows of data and instructions they manage, as shown in **Table 1**.

Therefore, four different categories can be found.

- Single Instruction, Single Data (***SISD***): A sequential computer without parallelism in terms of instructions or data. Ancient mainframes are examples of supercomputers built following this approach. Nowadays, with the broad expansion of multi-core architectures, SISD computers are relegated to *low-end* personal computing, as well as to high-mobility gadgets and several other embedded systems.
- Single Instruction, Multiple Data (***SIMD***): A computer able to perform a single instruction flow over multiple data elements taking advantage of natural parallelism. Traditional vector (or array) processors, such as the Illiac-IV [BDM72], the Burroughs Scientific Processor [KS82], the Loral's Massively Parallel Computer [Batc80] or the Thinking Machine's Connection Machine-1 [Hill85], are included in this category. The importance of pure SIMD systems in the HPC field has been progressively reduced, in favor of more scalable approaches following MIMD architectures. However, most current processor architectures include vector extensions, such as Intel's MMX [PW96] and SSE [KPK00], PowerPC's AltiVec [DDH00] or AMD's 3DNow! [OFW99], to cite just a few. Furthermore, SIMD processors are recovering part of their past importance and have become a *hot topic* in *hybrid* systems thanks to the Cell Broadband Engine Architecture [IBM] (composed by one general-purpose processor plus several vector co-processors) and to the prominent use of GPUs (that can operate over up to hundreds of data per clock cycle) for high performance computing, a technique broadly known as GPGPU [Gpgp].
- Multiple Instructions, Single Data (***MISD***): Multiple instructions operate over a single data element. This is a very uncommon architecture which is generally used for very specific purposes as, for example, fault-tolerance. Systolic processors [FW87] [KLS88], such as Carnegie Mellon's Warp [AAG87] and Saxpy's Matrix-1 [FS87], also belong to this category.
- Multiple Instructions, Multiple Data (***MIMD***): Multiple processors concurrently executing different instructions over different data elements. MIMD encompasses all kind of distributed systems, from current *off-the-shelf* multi-core systems to the largest multicomputer systems. We can be further split this subset of computer architectures among shared- and distributed-memory systems. Nowadays most (if not all) high performance computing systems belong to this category or, as explained before, to hybridizations of MIMD and SIMD.

	Single Data	**Multiple Data**
Single Instruction	SISD	SIMD
Multiple Instruction	MISD	MIMD

Table 1. Flynn's Taxonomy.

1.1.2 Objectives of the Computing System

Computing systems can be also classified taking into account the objectives of the system and the nature of the workloads that are executed on it. In this classification, we differentiate three highly related paradigms that focus on the execution of applications, and two somewhat different categories that emphasize on availability and response speed.

High-performance computing (**HPC**) include those systems devoted to host large-scale parallel applications whose aim is to be executed as fast as possible. This kind of systems has its main emphasis on the computing power achievable by the machine, usually measuring it in multiples of FLOPS, the amount of floating point operations per second performed by the system. The utilization of the HPC term is usually restricted to supercomputer and super-cluster systems.

High-throughput computing (**HTC**) is a term coined by Condor [CT]. It is highly related to HPC but with a slightly different aiming. Instead of reducing the execution times of individual applications, HTC systems aim to

maximize the amount of independent task that are executed in a given period of time. Applications executed by those systems may be parallel, but not necessarily. HTC refers mainly to clusters and computational grids, but supercomputers and super-clusters can be used following this paradigm. Actually, most supercomputing sites operate in a combination of HPC and HTC modes.

Many-task computing (**MTC**) is a very recent term that is strongly related to the two previous paradigms [Raic09]. In MTC systems, the workloads are composed of numerous tasks with some dependency relationships among them. The purpose of the system is to run this collection of tasks as fast as possible. Note that the tasks may be of very different natures, as well as the systems included in this category: clusters, supercomputers, super-clusters and computational grids.

High reliability computing (**HRC**) systems sit at a rather different place of the spectrum of computing systems. HRC applications (typically services) use to be light-weighted but its execution must be available at any given moment. For this reason such systems are also called High availability (**HA**) systems. To properly achieve the required availability, fault-tolerance and replication become critical issues for such systems. The figure of merit of such systems is the fraction of the time that the service is available, being the higher the better.

Real-time Computing (**RTC**) systems are subject to a real-time constraint [Butt04], in other words, once a given event happens the system must response/react to that event within a specified period of time that is usually short (seconds or submultiples) independently of the load of the system. Note that depending on the real-time model used in the application, being unable to react to an event in the corresponding time interval may results into a critical failure of the complete system. In other cases, this situation may generate a no-response event.

1.1.3 Interconnection Network

Finally, as our interest is on the interconnection network, we show a classification of interconnection networks, and discuses them. Taking a look at the most current Top500 list [DMS], we can see two clear trends. On the one hand, the choices of topology for custom-made, massively parallel computers are direct networks as the *k*-ary *n*-cube using *ad-hoc* technologies. On the other, commodity-based systems (super-clusters) are built with indirect networks using *off-the-shelf* networking technologies, such as QsNet [PFH02], Myrinet [BCF95] or InfiniBand [Shan02]. Often, the topologies of choice for these super-clusters are variations of the *k*-ary *n*-tree [PV97], such as the spine in the Marenostrum supercomputer [BSC], the first successful attempt to build a supercomputer using only *off-the-shelf* elements.

As their own names suggest, the difference between direct and indirect networks is in the way processing elements are connected to switching elements. In the case of direct networks, every switching element (usually called router) is directly attached to a processing element. Alternatively, indirect networks are formed by a collection of switching elements, out of which only a subset is connected to the processing elements.

Notice that there is a third alternative, a network composed of a single crossbar to interconnect all the nodes. A noticeable system following this approach was the first version of the earth simulator supercomputer [HYK03], whose compute nodes are communicated by means of a 640-crossbar. However given the large scale of current supercomputing systems, this topology is not used in practice.

It is remarkable that there is not a clear winner in the contest between direct and indirect networks in the field of supercomputers. Historically, the networks of the Top500 systems have been dominated alternatively by each of the types of interconnection network, as can be seen wandering in the Top 500 list archives [DMS].

Indirect interconnection networks have evolved noticeably from the first multi-stage networks as those proposed by Clos in [Clos53]. Those networks were built with low-radix switches (typically 4 or 8) and aimed to interconnect at most a few hundred nodes. Current spines, such as that of the MareNostrum supercomputer [BSC], have switches with hundreds of ports and are able to interconnect thousands of nodes. Former trees were low-radix: the CM-5 [LAD96] had a radix-8 data network. Current ones use switches with higher radices, as those radix-24 of the Cray XD1 [Cray]. There are also recent tree-like proposals as the Black Widow Clos network [SAK06] that takes advantage of the high availability of ports (radix-64 switches) to add side-links to the common tree-like arrangement. However, the most noticeable change in these networks is that former indirect networks were built *ad hoc* for the target systems, whereas current high-performance networking technologies like QsNet [PFH02], Myrinet [BCF95] or InfiniBand [Shan02] have favored building super-clusters with *off-the-shelf* components.

The evolution of direct networks is not that straightforward. Former networks were arranged in meshes, tori and hypercubes, such as the 38x32x2 Mesh in the ASCI Red supercomputer [SNLa] or the hypercubes within the iPSC and Ncube families of computers [Duni91]. In the last years the hypercube topology was abandoned due to its inherent difficulty to scale. An exception is Pleiades, a supercomputer currently placed in position #4 of the Top500 list that implements an InfiniBand-based hypercube of 10 dimensions [NCR]. Regarding current tori such as those at

the Bluegene/L [BCC03], the Jaguar [ORN] or the Red storm [SNLb] systems, they are similar to their predecessors, except in the number of nodes per dimension.

The latest Top500 lists show that the previous trend to interconnect the highest performance computers by means of cube-arranged networks (as in the IBM's Bluegene family [BCC03] [ABB08] and the CRAY's XT4 [ABF07] and former) is receding in favor of the use of tree-like networks based on InfiniBand. This has been the choice for the currently most powerful system, the Roadrunner [BDH08]. Anyway, it is remarkable that the number of indirect networks is growing in the middle positions of those lists.

Network bandwidth and latency have experienced notable improvements during the last 10 years, from the 800Mbps of the ASCI Red (1997) [SNLa] to the 20Gbps currently available in InfiniBand [Shan02] when using 4X, dual-data-rate connections, or the 10Gbps by Myri-10G and 10Gb Ethernet. Soon we will see offers of 100-120 Gbps (100G Ethernet, InfiniBand 12X-QDR). This takes us to a network bandwidth improvement of two orders of magnitude in less than 15 years. The latency of the full protocol and the network in the ASCI Red (taking into account message passing library) is 12µs. Both Myri-10G and InfiniBand latencies are around 2µs. Thus, latency has been improved (roughly one level of magnitude), but not as noticeably as bandwidth has.

Another interesting feature of an interconnection network is the switching method [DT04] used by the routers. Historical interconnection networks used circuit switching, a technique inherited from telephony. This switching technology reserves the path to be used between two nodes and, once it is available, the message is transmitted using those reserved resources. Circuit switching gave way to packet switching, in which messages are split into network packets that are transmitted point-to-point until they reach their destination. The first implementation of packet switching was store and forward, in which the complete packet has to be received by an intermediate router before being sent to the next one. This technology was further improved to virtual cut-though [KK79] and wormhole [NM95], which split packets into network phits, in order to pipeline the transmission of the packets, which lead to better performance in terms of network latency, making it almost insensitive to distance, provided that there is no contention in the network.

1.2 Position of this Research Work

From this tri-dimensional classification, the work discussed in this dissertation (the performance evaluation of interconnection networks) is about distributed-memory parallel computer systems (one of the branches of MIMD). The interconnection networks of interest are both direct and indirect networks. For most of the research work carried out, the evaluation was focused on high performance computing, as the network-related figures of merit to improve were those that allow a better utilization of the computing power of the system: maximize the throughput of the network, and minimize the latency suffered by the packets traveling through the network to, consequently, reduce the execution time of the selected workloads. The only exception to this rule was the evaluation of the SpiNNaker system, carried out in Chapter 6, as it is designed as a real-time system. In that evaluation, we put special attention on the stability of the system, as it is a key figure in real-time systems.

1.3 Structure of this Dissertation

The remaining of this dissertation is organized as follows. Chapter 2 presents a review of several methodologies commonly used to evaluate computer systems. These methodologies can be divided into three clear lines: analytical evaluations derive different figures of merit using solely mathematical models; simulation-based evaluations employ computational models of the system to evaluate, feeding them with different workloads; finally, empirical evaluation of an existing system can be performed executing several benchmarks over it.

In Chapter 3, we will discuss INSEE, our Interconnection Networks Simulation and Evaluation Environment. It is composed of two main modules: FSIN, a Functional Simulator of Interconnection Networks, and TrGen, a Traffic Generator module. We will discuss the different architectures that can be evaluated with INSEE, as well as all the procedures to provide workloads for the simulation.

INSEE has been used to carry out many different evaluations of interconnection networks, of very different nature: direct topologies, indirect topologies, specific purpose networks and performance of applications. The first case study will be presented in Chapter 4. It will evaluate the twisted torus, an alternative to the regular mixed-radix torus that offers better topological properties: increased bisection bandwidth, reduced distance-related figures and balanced usage of network channels under uniform traffic. One interesting finding of this evaluation is one limitation of a typical mathematical methodology to evaluate interconnection networks.

Still in the HPC environment, but focusing on indirect topologies, a second case study will be discussed in Chapter 5. It will show that the *k*-ary *n*-tree tree topology is over-dimensioned and will be, consequently evaluated

and compared against an alternative tree-based topological arrangement that offers better performance/cost efficiency figures by progressively reducing the number of switching elements in each level of the tree.

The third case study will be discussed in Chapter 6. It will evaluate the network of SpiNNaker, a large-scale, system-on-chip-based system that aims to simulate in real-time neural networks with billions of neurons. It has a rather different design and aiming, compared to the two previous networks. Two main concerns of this system are power consumption and chip area restrictions. The study of this network is focused on the fault tolerance of the system, as well as on the real-time constraints.

The last case study, presented in Chapter 7, will be oriented to the performance evaluation of parallel applications (instead of systems), and will show to what extent job and task placement policies may affect the execution time of MPI applications. This evaluation will include applications being executed over both torus and tree topologies.

Finally Chapter 8 will close this dissertation with several conclusions that summarize the contributions of our work, and with an outlook of future research lines.

Chapter 2. Evaluation Methodologies

This Chapter discusses some commonly used alternatives to evaluate the performance of parallel computing systems. When possible, the focus of the discussion will be on the interconnection infrastructure that supports these systems, but for the sake of completeness some evaluations of other parts of the system are also described and discussed. The review includes several relevant research works that cover different topics in the area of the performance evaluation of parallel computing systems, from bus-connected multi-processors to massively parallel processing systems, from raw performance figures to fault-tolerance analysis. Furthermore, the discussion spans over a long period of time, which allows understanding how the field of parallel computing has evolved during the last decades. We want to remark that this review of evaluation techniques does not intend to be exhaustive. In fact, the performed reviews should be seen as a starting point for any reader interested in the techniques discussed, and not as a thorough appraisal.

Obviously, if we want to evaluate an already existing system, a wide variety of benchmarks can be run over it in order to measure its performance. However, during the design phases of a novel architecture, the system is not available and, probably, there are several architectural alternatives to evaluate in order to select the most adequate using a predefined set of metrics, such as performance, reliability, power efficiency, cost efficiency, etc. In those cases where the actual system is not available, two alternative—but not excluding—evaluation methodologies arise. On the one hand, the system can be evaluated analytically, relying on mathematical models. On the other hand, the evaluation may rely on simulation, in which a computational model of the system is developed and executed in order to predict performance figures under some scenarios of interest. As can be seen in the literature, it is also very common to use mixed methodologies in which results of an analytical evaluation are assessed by means of simulation.

2.1 Analysis

Generally, the first step to estimate the performance of a parallel system is by means of an analytical evaluation using mathematical models. These models can provide performance measures quickly, which allows exploring a wide variety of system design options and/or parameters. Results obtained with analytical models can be seen as *raw* performance figures given that they typically require system simplifications and only support spatial and temporal distributions that are far less complex than actual applications. It is also remarkable that most analytical evaluation methodologies are restricted to evaluate the *steady-state* behavior of the system as they lack the capability to properly model and analyze *transient states*.

2.1.1 Topological Characteristics

The simplest analysis of an interconnection network is to mathematically derive some characteristics of the topology relying on Graph Theory. Note that switching elements in a network can be seen as the *vertices* of a graph, while the links joining them can be seen as the *edges*. The most important topological characteristics of an interconnection network in high performance computing scenarios are the following:

- **Bisection Bandwidth**: is the bandwidth across the smallest cut that splits the network into two equal halves. It provides a good indicator of the raw performance of a system, as *network throughput* can be easily derived from it. In Chapter 4 the bisection bandwidth analysis of two network topologies will be performed.

- **Average Distance**: is the arithmetic mean distance, understood as the number of hops took by packets to reach their destination, among any pair of nodes of the topology, taking into account only the shortest path between the nodes.
- **Diameter**: is the maximum distance among any two nodes of the topology, taking into account only the shortest path among them. Note that the average and maximum distance between nodes have a definite impact on the latency of the packets that travel along the network. For this reason, network designers try to minimize these two characteristics. In Chapter 3, when describing the topologies that are implemented in our simulation environment, we will show their distance related characteristics.
- **Vertex-Symmetry**: A graph is vertex-symmetric, or vertex-transitive, if there exists any automorphism *f* such that for all pair of vertex in the graph, v_1 and v_2, $f(v_1)=v_2$. Informally, this means that all the nodes (vertices) have the same image of the topology, which consequently allows simple routing functions.
- **Edge-Symmetry**: A graph is edge-symmetric, or edge-transitive, if there exists any automorphism *f* such that for all pair of edges e_1 and e_2 in the graph, $f(e_1)=e_2$. Informally, a graph is edge-symmetric if every link (edge) has the same local environment, so that no link can be distinguished from any other, based on the vertices and edges surrounding it. This property is desirable as it simplifies load balancing of the usage of network resources.

As an example, let us analyze the topology shown in **Figure 1**. The bisection of this topology is 5 links, as can be seen in **Figure 1**a, where one of the possible minimum cuts is shown as a dotted line (cutting edges e_1, e_4, e_6, e_7 and e_8). The topology is vertex-symmetric because, if we shift the position of each vertex of the graph one position clockwise, as shown in **Figure 1**b, we can see that the obtained graph is isomorphic to the original one. The topology is also edge-symmetric; for example, the graph shown in **Figure 1**c is an automorphism of the original graph. Finally, looking at **Figure 1**d, we can see the distance distributions from vertex V_0 (white), which is the same for all the vertices because the topology is vertex symmetric. Vertices located at distance 1 have been colored grey, and those at distance 2 have been colored black. From this figure we can derive that the network diameter is 2 (the number of hops to reach any of the black vertices). Similarly the average distance is $^7/_5$, because there are three vertices at distance 1 (grey) and two vertices at distance 2 (black).

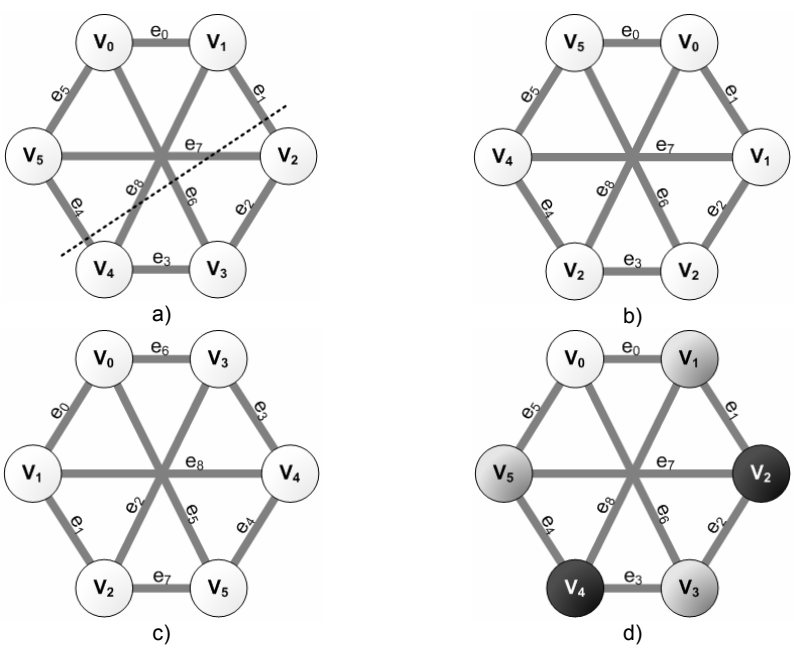

Figure 1. Example of topological study. a) Bisection bandwidth. b) Vertex symmetry. c) Edge Symmetry. d) Distance-related characteristics.

A set of models to analytically derive throughput and average latency figures in *k*-ary *n*-cubes using wormhole switching and being fed by uniform and hot-spot traffic were proposed in [Dall90]. These models took into account wire length in order to properly derive latency figures as transmission time depends on wire length. The evaluation showed that topologies having the highest number of dimensions required longer wires and so obtained worst performance figures.

Midimew networks—standing for MInimum DIstance MEsh with Wrap-around links—were presented and also evaluated using this approach in [BHB91]. This research work proved that this network topology offers optimal

diameter and average distance figures, and also that midimew networks can be implemented for any desired number of nodes. In addition, the shortest path problem for this topology was solved, which adds support for optimal routing.

In [HS95] authors analytically derived an upper bound for latency figures, valid for any vertex-symmetric topology. Their model was the first one which included congestion to compute the latency from the distance. In fact, the bound for latency is derived from a bound for the congestion encountered in the network.

The recursive diagonal torus was proposed and evaluated in [YFJ01]. Authors analytically derived the average distance and the diameter of the network, as well as the bisection bandwidth. The topological characteristics of the proposed topology were compared with some other direct networks such as 2D, 3D and 4D torus and hypercubes. However, the shortest path problem was not solved for this topology which led to propose and use two non-optimal routing algorithms.

Authors of [BMI03] proposed a class of dense circulant graphs to be used as interconnection networks. In that work the authors analytically derived some topological properties of this family of topologies. The research work also included the decomposition of these dense circulant graphs into a set of rings with the same number of nodes, which was used to derive an optimal routing function.

In [MBS08], authors presented a wide family of topologies based on Gaussian integers that includes the well-known two-dimensional square torus. In that work, the distance-related characteristics for any topology belonging to this family were analytically derived. Furthermore, they proved that all these topologies are vertex-symmetric. Finally, they proposed optimal routing functions and resource placement algorithms for systems implementing networks with such topologies.

The reader should note that the derived characteristics can be used as a bound of the achievable performance but, since those characteristics depend solely on the topology, the actual performance figures of the system may differ depending on the architecture of the switching elements, including routing function, flow-control mechanism, switching strategy, etc. In other words, the accuracy of this kind of evaluation is not very high. For example, if the actual routing does not use minimal paths all the distance figures are not valid; or, if the use of network resources is not balanced, the throughput of the system will be unable to reach the expected derived one, because of the existence of bottlenecks. Fortunately, there are other mathematical tools to evaluate interconnection networks that may include knowledge about other aspects of the design, not only the topology. We will discuss these other methodologies now.

2.1.2 Markov Chains

A Markov chain, firstly introduced by Andrey Markov in 1907 [Mark07], is a *stochastic* process obeying the Markov property. This property establishes that future states depend only on the current state and not in the previous ones—in other words, the description of the present state fully captures all the information that could influence the future evolution of the process.

At each step, the system may change its state from the current state to another state, or remain in the same state, following a certain probability distribution. The changes of state are called transitions, and the probability associated with each possible transition is called transition probability. Markov chains use to be represented as a direct graph in which vertices represent states and edges represent transitions. Each edge has attached a weight that is the probability of using that transition.

A simple Markov chain with two states (E_0 and E_1) is depicted in **Figure 2**. The transition probability to change from a state *i* to a state *j* is denoted as p_{ij}. Evidently, if *i* and *j* are equal, p_{ij} is the probability to remain in the same state.

This kind of process presents a simple, but still useful, dependency model among the random variables in a stochastic process. In the field of interconnection networks Markov chains have been used to analyze different properties. Reader should note that, for example, the wandering of a packet in a network perfectly fits the Markovian property: the next switching elements in its path depend only on the current switching element and not in those previously visited.

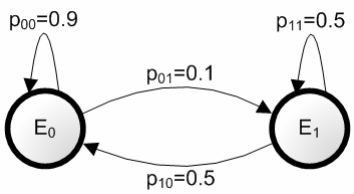

Figure 2. Example of a Markov Chain with two states.

In [SK75] memory conflicts on multiprocessor systems were modeled using Markov chains. This model gave the possibility of varying three parameters: the number of processors, the number of memory modules and the interleaving of the memory addresses. With it, an evaluation of the impact of the arrangement of instructions and data in the memory of multiprocessors was carried out, taking into account several configurations of state-of-the-art systems (at that time). The conclusion was that performance improvements could be achieved separating address spaces for instructions and data.

In [BT89] multistage interconnection networks were modeled using two-level hierarchical Markov chains for fault-tolerance and reliability contexts. This model was used to analyze the reliability of three different multistage networks with up to 1024 nodes each. Results, in terms of upper and lower bound of computing node availability, were shown for a specific case and a given failure model of not replacement and perfect coverage.

A general Markov chain-based model for wormhole switching routers was proposed in [GTB93], and applied to both hypercubes and two-dimensional meshes. From this model authors derived the average latency for several network configurations of those topologies with up to 256 communicating nodes.

An evaluation of the performance under non-uniform traffic of single buffered omega networks was performed in [AA95] using Markov chains. The proposed model was compared against another two previous models from different authors for the specific cases of uniform and hot-spot traffic. The presented results showed that the proposed model was equally accurate for uniform traffic but significantly more accurate in the case of hot-spot traffic.

Authors of [BS96] used Markov chains in the context of operating system thread scheduling to predict the number of cache misses of the threads, in order to schedule the thread that is less likely to loss performance due to memory accesses. A scheduling policy based on this Markov chain was confronted against three common policies in the context of affinity scheduling; obtained results supported the use of locality-aware policies in the scheduler of the operating system.

A model, based on Markov chains, for topology-independent wormhole routers that rely on virtual channel flow-control was proposed in [KON06]. This model allowed to measure average message latency, number of messages per queue and the probability of messages being timed-out when running under real-time constraints.

2.1.3 Queueing Theory

Queueing Theory is the mathematical study of waiting queues and was firstly introduced by Erlang in 1909 [Erla09] and normalized by Kendall in 1953 [Kend53]. Queueing theory allows the mathematically analysis of some related processes: a packet arriving at the tail of a queue, a packet waiting in a queue and a packet leaving a queue. Several performance measurements can be derived such as the average waiting time in the system queues, the probability of encountering the system in certain states—as, for example, being empty or full—, the probability of a packet being a in a certain state or queue, or even the probability of keeping the system's state for a given amount of time.

Queueing Networks are systems which contain an arbitrary, but finite, number *m* of queues. Packets move throughout the network and are delivered to the nodes. The state of a network can be described by a vector (q_1, q_2, …, q_m), where q_i is the number of packets at queue *i*. *Open* networks are those in which the packets can enter and leave the system. In contrast, *closed* networks are those in which the packets are kept within the system, in other words packets are not allowed to enter or leave the system.

A Poisson process models random independent events, meaning that time intervals among events are completely independent of each other. Queueing models based on the Poisson process can model and analyze real-life systems accurately and allows to easily evaluating the behavior of the network in *worst-case* scenarios.

Although queueing theory is generally applicable in a wide diversity of situations that may be encountered in business, commerce, industry, healthcare, public service and engineering, we focus on its application to evaluate interconnection networks. Reader should note that the process modeled by queueing theory is directly applicable to this scenario as switching elements are commonly composed by queues. This way, open networks of queues following Poisson distributions are a commonly accepted mechanism to model interconnection networks. Note also that complex flow control techniques such as Virtual Output Queues [TF88] or priority-based flow control [Dall92] will make the system more intricate but still approachable with queueing theory.

Figure 3 depicts two examples of queueing models with three queues each: a closed queueing network and an open queueing network. For all *i* in [0, 2], Q_i represents each one of the queues and λ_i is the distribution of time intervals to process a packet at the head of each queue. A packet leaves the head of Q_0 and arrives to the queue Q_2 with probability p_0, therefore, the probability to move from Q_0 to Q_1 is, $1\text{-}p_0$. In the case of the open queueing

network, ε_0 is the packet generation ratio and γ_2 is the packet departure ratio. The state of the network in this depiction is (2, 1, 2).

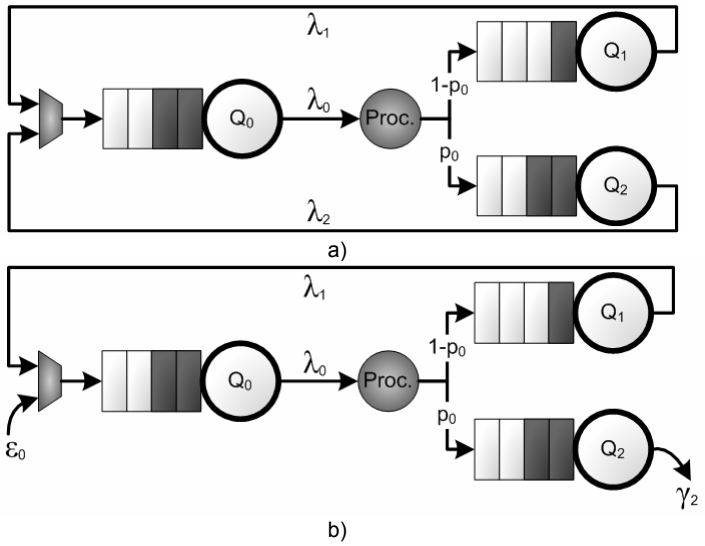

Figure 3. Examples of queueing systems with three queues. a) Closed network. b) Open network.

Authors of [DG94] relied on queueing theory principles to present a model for wormhole-based interconnection networks to be used in multicomputer systems, and stated that this model was general for any network topology and virtual channel management. This model was used to measure the average delay, average distance, and network throughput in *k*-ary *n*-cubes, including hypercubes, with and without uniform channel bandwidth. This work also included a discussion on how to extend their model to evaluate virtual cut through-based networks.

In [GKP95], a two-level interconnection network composed of regular *local* wormhole meshes connected by means of a *distributed* optical star network was modeled and evaluated using queueing theory. The evaluation focused on latency under uniform and non-uniform traffic, being the length of the optical links a key parameter of their model.

A queueing theory-based model for evaluating wormhole switching in fat-tress was presented in [GG97]. This model relied on multi-server queues to accurately model the transit of packets through the network. It was used to analyze average latency and maximum throughput of networks with up to 1024 communicating nodes.

Authors of [SOM01] proposed a model to predict average message latencies in unidirectional, wormhole switching *k*-ary *n*-cubes with fully adaptive routing using Dally's algorithm [Dall92]. This model was general and relied on queueing theory, and can be used to measure average latency for a wide spectrum of dimensions, number of nodes per dimension and number of virtual channels.

A performance evaluation of three different switching techniques was carried out in [MO02] for 2D tori. The three techniques (Wormhole, pipeline circuit switching and circuit switching) are modeled using queueing theory principles. From these models latency figures were derived for different network configurations: number of virtual channels (4 and 8), flit length (32, 64 and 128) and number of nodes (64 and 256). Authors concluded that wormhole was the best performer, provided that the flit length was short. In those cases with large flits the pipelined circuit switching outperformed the other techniques. In all cases the worst-performing technique was circuit switching.

A queueing theory model of wormhole-based networks-on-chip was presented in [GBC06]. The presented model included the capability to deal with non-uniform channel bandwidth and was used to carry out an analysis of the efficiency of quality-of-service strategies under timing constraints. The analytical evaluation focused on packet latency. This study also included and evaluated an algorithm to allocate link bandwidth in order to make an efficient use of communication resources.

2.1.4 Petri Nets

Petri Nets, which take their name from its creator Carl Adam Petri were firstly introduced in [Petr62]. It is yet another mathematical modeling language to describe parallel and distributed systems. A Petri net is composed by a directed bipartite graph, in which the vertex represent bags or transitions, and edges represents pre-conditions (from a bag to a transition) or post-conditions (from a transition to a bag). Note that being a bipartite graph means that edges are only allowed between a transition and a bag and never between two bags or two transitions.

Those bags heading to a given transition are identified as input bags of that transition and the bags accessible from that transition are called the output bags of that transition. Bags can contain any non-negative number of tokens; the presence of a token in a bag is usually interpreted as the availability of a resource. The token distribution over the vertices of a net is called a marking. When there are tokens in all the input states of a transition, it may be executed, which means that one token in each input state is consumed and, in exchange, one token is generated in each output state. The execution of a transition is atomic, a single non-interruptible step. Petri nets operate in a nondeterministic way: if a transition is enabled, it may be executed, but it is not forced to do so. Furthermore, when multiple transitions are enabled at the same time, any one of them may be executed, but as execution is atomic, only one of the transitions can be executed at once.

Petri nets offer a graphical notation for stepwise processes that include choice, iteration, and concurrent execution. Petri nets have an exact mathematical definition of their execution semantics, with a well-developed mathematical theory for process analysis. For this reason, Petri nets are an attractive mathematical model as they provide a balance between modeling power and capability of analysis. Therefore, there is plenty of literature on Petri nets and several new classes have been developed based on the original model. Some of these new classes (such as colored Petri nets) are reducible to the original Petri net, while some others (such as timed Petri nets) add properties to the model that cannot be modeled by regular Petri nets but, in return, they can not make use of the full range of mathematical tools available to analyze regular Petri nets.

A simple Petri net is depicted in **Figure 4**. P_0, P_1 and P_2 represent the bags and the represented state has a marking {3, 1, 0}. The transitions are denoted by t_0 and t_1. Transition t_1 is enabled as there is a token in the input state P_0, thus it can be executed, consuming a token from P_0 and generating a token in states P_1 and P_2. Transition t_0 is unavailable because there is no token available at state P_2 and, therefore it, can not be executed.

A Petri Net-based model to evaluate multiprocessor systems composed of different number of processors, buses and memories was proposed in [MCB84]. This model was used to bound (upper and lower limits) the performance of such systems, understood as the average number of processors being able to execute their thread, as they get access to the data in the shared memory. However, the main contribution of the paper was the introduction of a new class of Petri nets, denoted as Generalized Stochastic Petri Nets (GSPN) that has been widely used in subsequent research work. For example, this new class of Petri nets was used to model and evaluate a hexagonal mesh with wrap around-links in [KCR94]. The GSPN-based model only included the topological arrangement of the network and did not include any knowledge about switching mechanism, routing algorithms or flow-control technique. It was used to measure latency figures for different scales of the proposed topology, ranging from 19 to 169 nodes.

Authors of [TH97] modeled multistage banyan networks using timed Petri nets. The aim of the model was to evaluate the performance of this kind of networks when using multicast messages. Only networks composed of 2×2-port switches were considered and evaluated under uniform traffic with some different patterns of multicast. The figure of merit in this evaluation was the network throughput for networks connecting up to 64 nodes.

In [GSZ97] timed colored Petri nets were used to model a multi-thread multiprocessor architecture based on a 4×4 torus network-on-chip. Two different models were proposed, one of them complex and the other one simplified. These two models were used to measure the effective processor utilization of those architectures depending on the number of threads per processors, as well as the probability of applications accessing the memory. As both the complex and the simple models provided similar results the authors encouraged to use the simple one in order to obtain results faster.

Authors of [GBV02] used Petri nets to model cut-through routers for direct networks fed by random traffic and using oblivious routing. The routers were modeled with queues in the input ports or, alternatively, in the output ports. These models were used to measure packet latency both for mesh and torus topologies.

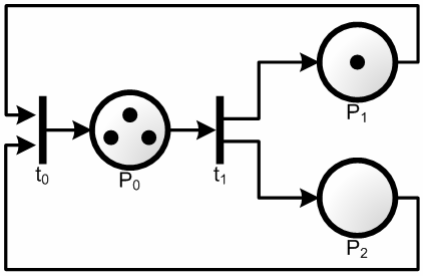

Figure 4. Example of a Petri net with three bags and two transitions. Small black balls represent tokens.

Database queries in high-end clusters were modeled and evaluated in [KTG06] by means of timed Petri Nets. Four different schemes to parallel processing and/or offloading these queries were presented and evaluated through that model. The performance was measured as the achievable speed-up.

2.1.5 Other Tools for Analytical Modeling

If we focus on the performance evaluation of parallel applications, we can find the classical analytical model proposed by Amdahl and widely known as Amdahl's law [Amda67], which states that the maximum speed-up reachable by the parallelization of any application is harmed by the part of the application that is inherently sequential. Amdahl's intention was to show the unfeasibility of parallel computing as a way to improve performance. However, in [Gust88] Gustafson relied on the same model to show that parallelization was not only viable but also advisable as the parallelizable part of most applications can be scaled up in order to make the sequential part almost negligible.

This simple model was extended in [WL08] in order to derive the performance of many-core architectures focusing on their power-efficiency. This model supported that hybrid architectures composed of a single fast computing core plus a set of slower but power-efficient computing cores obtain better performance/efficiency ratio figures than homogeneous architectures, composed by several computing cores of the same type independently of these cores being fast-computing or power-efficient ones.

2.2 Simulation

Even when analytical models are fast, they typically include simplifications that may affect the accuracy of the obtained results. For this reason, simulation is widely used in evaluation contexts, as a support for analytical evaluation or, simply, as an evaluation toolkit by itself. The main advantage of simulation is that the computing model can be as accurate as needed. It is to be said that the higher the accuracy, the more computing resources are required to carry out the simulation.

2.2.1 Simulation Engine

With no doubt, a fundamental part of a simulator is its simulation engine, as its performance will determine the performance of the whole simulation process. Focusing on how simulation events are managed, we can found two categories of simulation engines: time-driven and event-driven.

On the one hand, *time-driven* simulators are usually simpler to implement as they do not need any data structure for the simulation engine. They require using a discrete time space for the simulation, usually measured in terms of *abstract* cycles. At each cycle, the simulation engine checks all the structures of the simulated elements and executes events as needed. For this reason, to obtain good performance figures this behavior requires a high density of events per unit of time. It is also remarkable that, as all the simulated elements have to be checked, this simulation model does not scale very well as spatial locality of memory accesses is not preserved. The basic structure of a time-driven simulator is shown in **Algorithm I**.

```
while not finished
  for all elements
    check element
    if something to execute
      execute events in element
  increase clock
```

Algorithm I. Basic structure of a time-driven engine.

On the other hand, *event-driven* simulators work in an *on-demand* fashion. They keep a data structure that stores all the events to be executed, and execute them in chronological order. The execution of an event may trigger the scheduling of new events. Simulation is finished when all the events have been executed or when dictated by user needs. Event-driven simulation allows modeling a continuous time space as the simulation clock can be a floating point number. The basic structure of a time-driven simulator is shown in **Algorithm II**.

```
while not finished
  take first event to execute
  modify clock
  execute event
  if new events generated
    schedule new event(s)
```

Algorithm II.Basic structure of an event-driven engine.

Another aspect to be taken into account is that the data structures needed for event-driven simulation and their implementation have great impact on its performance [Jone86], something that we will explore in Chapter 3. For example if events are stored in a single sorted queue, retrieving an element from the head of the queue can be done in constant time, but adding a new event keeping the chronological order is $O(n)$ in the number of events—which in the case of a large number of events may result on slow simulation.

Most data structures proposed in the literature to be used in event-driven simulators have operations for inserting and taking the head element which are constant or close to constant. However, they usually need some other control operations of the structure that may require longer times, but that are seldom invoke. A general approach to obtain constant order insertion and access operations is the calendar queue [Brow88]. A calendar queue is composed of a set of *n* queues each of them of a given length *l* in terms of execution clock. This way, when an event timestamped *t* has to be scheduled it will be stored in one of these queues determined by the modulo-*l* applied to *t*. Provided that the number of queues is large enough and that their use is well-balanced, the number of events stored in each queue will be very small and, therefore, the access time to this structure will be close to linear. However, if the use of the queues is not balanced, it may happen that most of the events go to the same queue and, therefore, the access time will be of linear order. For this reason the calendar queue has an operation to re-balance the use of the queues and also to increase or decrease the number of queues in order to obtain better performance figures. Note that, if there is a very high density of events, the use of the queues is difficult to balance and, consequently, the calendar queue will not perform well. For this reason, we can find in the literature several proposals of modifications of this data structure, such as the dynamic calendar queue proposed in [OA99] or the several optimizations presented in [ELL00].

Another common data structure to perform event-driven simulation is the use of Lazy queues [RRA91]. This structure is actually composed of three structures that store events that are scheduled in the near-future, middle-future and far-future, correspondingly. Only the near-future list stores the events sorted in chronological order. The other lists do not store events in-order and keep most of the events. The middle-future structure is composed of several list that are accessed by means of a modulo operation but that do not store events in-order. The far-future list stores events without any order. These two lists are sorted by demand, when their events become near-future. This way, the scheduling of most events is done in constant order as they are stored in non-sorted lists. The data structure also implements methods to sort the events in middle- and far-future lists, generally Quicksort [Hoar62], which sort the elements in an array in $O(n \cdot \log n)$. There are also implementations of hybrids between calendar queues and lazy queues; for example in [HT02] authors propose a calendar queue which includes a far-future list in order to keep the amount of events stored in the calendar queue as reduced as possible.

2.2.2 Simulation Frameworks

A simulation framework is a piece of software that offers support to build simulators without dealing with the complexity of the simulation engine itself. This software is generally offered in the form of a set of libraries that perform the simulation, plus a set of related applications that simplify the simulation process. They tend to have human-friendly approaches to define the elements taking part into the simulation, as well as comprehensible tools to analyze the simulated system and its behavior. Some of the most important simulation frameworks are reviewed in the following paragraphs.

SimPack [UF] is a simulation framework, composed of C and C++ libraries and applications. The framework [Fish92] supports several simulation models, including discrete event simulation. SimPack includes some fairly extensive simulation facilities, and is in use by various instructors, researchers and industrial analysts for their modeling and simulation experiments.

OMNeT++ [OC] and its commercial version OMNEST [SI] are public-source, flexible and generic simulation frameworks. It can be used to model and simulate any process that can be mapped/translated to active components that communicate by passing messages. For example, it can be used for simulating queueing networks, multiprocessor systems, hardware architectures (routers, optical switches, file servers, etc.). Several open source simulation models have been published, in the field of communication networks simulations (IP, IPv6, MPLS, etc), mobility and *ad-hoc* simulations and in other areas. They offer an interactive execution environment, which allows examining simulation progression. There is also extensive library support for packet tracing etc. A review of Omnet++ was carried out in [Varg01].

OPNET [OT] stands for Optimized Network Engineering Tools, a network modeling and simulation software. OPNET is a modular, hierarchical, object-oriented simulation environment that allows for general-purpose network specification, simulation and performance analysis. It has a comprehensive analysis tool with graphical capabilities that is specially designed for interpreting and synthesizing output data. OPNET's design is discussed in [Chan99].

2.2.3 Levels of Detail

Another important aspect of the simulation that must be taken into consideration is its level of detail. The detail of a simulation can have two different, but related aspects: the model of the components and the model of the workloads.

2.2.3.1 Modeling Workloads

The workload to feed the network is a very important aspect when it comes to evaluate the interconnection network of parallel and distributed systems. For example, in the first phases of the design of a system, it may be desirable to obtain results as fast as possible even when they are not fully accurate. Alternatively, a complex model providing high fidelity may be required in the last phases just before deployment, to ensure that all the pieces of the system perform as expected when they are put together and also that the applications make efficient usage of the underlying hardware.

Synthetic traffic patterns from independent sources [DT04] provide a good first approach to evaluate a network, because they allow us to asses rapidly what the raw performance of a network is, and because it can be supported by any of the previously discussed analytical studies. Very often, randomly generated traffic is used to evaluate systems: uniform, hot region and hot spot traffic patterns have been used in a large collection of studies. Other commonly used patterns are those that send packets from each source node to a destination one as indicated by a certain permutation, usually defined as a function that takes as input the address of the source. Some examples of permutations are: bit reversal, matrix transpose, bit complement, butterfly, perfect shuffle, etc. The permutations in our environment will be defined in Chapter 3.

Nevertheless it is not common to find actual applications that are internally implemented using patterns like these synthetic ones, in which traffic-generating nodes produce messages without coordinating among them. We can state that synthetic traffic patterns do not accurately mimic the behavior of any actual application. For this reason, trace-driven simulation is often preferred to perform a more realistic evaluation of a system. Feeding a simulator with a trace is not an easy task. To evaluate only the network of a parallel system we could implement a *dummy* model of the processing node, allowing it to inject messages into the network as fast as it can, ignoring the causality of messages and the computation intervals. This approach is a stress test of the network, because of the contention caused by all nodes injecting at the maximum pace.

Alternatively, it would be more realistic to maintain the causal relationship between all the messages in the trace; in other words, if the trace states that there is a reception before a send, the node has to wait for that reception to be completed before starting with the send. This mechanism provides more fidelity than the inject-at-will model. To further improve the accuracy of the simulation, compute intervals (periods in which nodes do not inject load into the network) should be taken into account, maybe applying a *CPU-scaling-factor* in order to simulate a system with faster (or slower) CPUs than those used to capture the trace.

Still, there are some problems with the trace-driven approach that we should not ignore. Firstly, the information captured within the trace could be inexact due to the intrusion effects of the trace logging mechanism. Secondly, traces may reflect some of the characteristics of the system in which they have been obtained. Finally, traces from actual applications running in a large set of processors are difficult to obtain, store and manage, and these are precisely the ones of interest in our performance evaluation studies.

A hybrid between the utilization of synthetic traffic patterns and traces is the estimation of probability distributions for destinations, inter-generation times and message lengths, using data extracted from actual traces to feed some distribution-fitting program as for example BestFit [PS]. For example the spatial distribution of several application prototypes were shown in [ACG06]. Once we have the distributions that resemble the traced application, we can generate random traffic resembling the traced application. However, as stated before, in actual applications causal relationships among messages are common, and this technique does not capture them. And, again, the inexactitude of the information within the trace (due to the characteristics of the system in which the trace was captured, and the intrusion of the logging process) may generate estimated distributions, or parameters of those, that are not valid.

In order to introduce causality in the simulation and fill the gap between trace-driven simulation and independent sources traffic, a bursty traffic model can be used. This model uses the previously discussed synthetic traffic patterns, but emulates application causality using a coarse-grained approach. The message generation process passes through a certain number of *bursts* or steps. During a burst each node is able to inject into the network only a given number of packets (b); after doing so, it must stall until the burst finishes, that is, until all the packets of the burst (generated at all the nodes) have been injected *and* received, being the network completely empty. Simulations with short bursts emulate tightly-coupled applications and, correspondingly, long bursts emulate loosely-coupled applications.

Synchronization among nodes is included in this model, but in a very primitive way (roughly a barrier every *b* packets); fine-grained synchronization among messages/tasks are not considered.

A further refinement of the bursty model is by means of application kernels. These application kernels are implemented using point-to-point synchronization and communication primitives, and can include different levels of causality such as long chains of dependencies. These application kernels emulate the behavior of small parts of actual applications, but when compared with regular traces they are more flexible in the sense that they are fully configurable in terms of number of communicating tasks, message size, task coupling, etc. This gives an advantage when compared to traces, as the latter are difficult to capture and manage for large-scale systems. Furthermore as they are only small parts of applications their execution is orders of magnitude faster than trace-based configurations, while still providing a reasonable level of accuracy. These application kernels are inspired in communication patterns observed in actual applications. In some cases they reproduce virtual topologies, or implementations of collective communication primitives, while in others they reproduce programming models such as master-slave. The set of application kernels included in our simulation environment will be discussed in depth in Chapter 3.

The most accurate methodology to evaluate a parallel computer would be running a detailed full-system simulation that includes the interconnection network, the compute nodes, the operating system, some support libraries and the applications running on them. This is a very complex, error-prone task, as well as a high resource-consuming methodology that could need a system similar in dimension to the one we want to evaluate. These are the main reasons to justify the limited utilization of execution-driven simulation to evaluate medium-to-large size distributed memory parallel computers.

Table 2 closes this section summarizing the methodologies to generate traffic to fed simulations. The columns show the spatial pattern of the workload, the causality among messages, the complexity of generating, managing and performing simulation, and the kind of evaluation supported by the traffic model.

traffic model		spatial pattern	causality	complexity	evaluates
independent traffic sources	**random**	random	no	very low	raw performance
	permutation	worst case	no	very low	raw performance
	estimation of distributions	application-like (origin-dependent)	no	low / medium	selected application (origin-dependent)
bursty traffic sources	**random**	random	coarse-grained	very low	raw performance
	permutation	worst case	coarse-grained	very low	raw performance
	estimation of distributions	application-like (origin-dependent)	coarse-grained	low / medium	selected application (origin-dependent)
application-based	**application kernels**	application-like	application-like	low / medium	usual communication patterns
	trace-driven (inject-at-will)	application-like (origin-dependent)	no	medium	raw performance when congested
	trace-driven (causal)	application-like (origin-dependent)	application-like (origin-dependent)	medium / high	selected application (origin-dependent)
	execution-driven	actual application	actual application	very high	selected application

Table 2. Description of different traffic models used for simulation-based evaluation of parallel systems.

2.2.3.2 Modeling System Components

A parallel computer is a very complex system comprising a multitude of components and, depending on the specific area of research, only a subset of those components may be of interest. For example, if our research is focused on the interconnection network, probably it is not needed to use a detailed model of the computing nodes, a simple model of their behavior being enough. Alternatively, if our interest is on the study of cache memory infrastructure, the interconnection network becomes a secondary issue, and it is probably not included or, at most, included in a very simplistic way. This is because including a very accurate but complex model of elements that are hardly related with our main concern can, and probably, will unnecessarily increase simulation time. In other words, a *trade-off* has to be found among *what* takes part of the simulation (and *how*), and the computing resources needed to perform the complete simulation.

There are simulators specifically designed to evaluate network performance which incorporate very complex models of the router architecture and the network topology, but very simple, or none at all, model of the computing nodes. Chapter 3 will be devoted to thoroughly discuss our own simulation environment which is especially focused on the interconnection network. In the literature we can find references to a multitude of network simulators. Let us review a small selection of them.

SICOSYS [UC] [PGB02] has very detailed models of several router architectures, which allows obtaining very accurate performance measurements, close to those obtained with a hardware-level simulator, but at a fraction of the required computing resources. It is implemented in C++, and has been used for evaluations focusing on performance and on fault-tolerance, both in context of interconnection networks and networks-on-chip. In addition, it allows feeding the simulator with several traffic models with different levels of detail: synthetic traffic and traces of MPI or ccNUMA applications. It can also interface with RSIM [PRA97] and SIMOS [RHW95] to perform full-system simulation of interconnection networks (see below).

The Chaos router simulator [UW] has a detailed model of *k*-ary *n*-cube networks (meshes, tori and hypercube). Switching can be both packet-switching and wormhole. Routers can implement dimension-order oblivious routing or, alternatively, Chaotic adaptive routing [KL94]. It also incorporates a wide variety of traffic patterns to feed the simulation. The chaos router simulator may also run in animated mode, in which the temporal evolution of the simulation is graphically represented. One of the main drawbacks of this simulator is that all the changes in the design of the network to evaluate must be done at compilation-time, in other words, every change in the model requires the application re-compilation.

FlexSim [USC] is a C-based simulator for *k*-ary *n*-cube networks with any number of dimensions greater or equal than 2, any (power of two) number of nodes per dimension, and any arbitrary number of virtual channels. It has support for several router models, traffic patterns and failure models. This simulator has been used to conduct research in fault-tolerance and deadlock-free routing.

The NS-2 simulator [ISI] has a slightly different design as it focuses on research on wired and wireless TCP-based communication networks. Although high performance computing systems use to rely on high performance interconnects such as InfiniBand [Shan02] or Myrinet [BCF95] for parallel computing, most of them support Ethernet-based networks for storage and control purposes. Furthermore, new versions of Ethernet as 1Gb, 10Gb and the early-to-come 100Gb Ethernet can be used as low-cost interconnection networks.

There are also simulators based on the frameworks discussed in the previous subsection. An interesting example is a simulator [OCS] of InfiniBand networks [Shan02] that has been implemented using OMNeT++.

In the other extreme there are those simulators, denoted as full-system simulators [MHW02], which model with high fidelity all the hardware and software of the computing nodes: devices, drivers, operating system and even applications. They are designed to perform different architectural evaluations of the hardware within the node, such as CPUs, caches, networks-on-chip, etc. as well as to predict the performance of applications running on the simulated systems. Often, they do not have a very detailed model of the network. The two most important, in terms of volume of publications, are RSIM [PRA97] and Simics [MCE02]. Several tools have been designed to extend their functionality, allowing even more accurate models of the system under evaluation.

RSIM is designed to simulate scientific shared-memory multiprocessor workloads, focusing on user-level applications: applications are interpreted and no operating system is simulated. RSIM has a very complex and accurate model of *state-of-the-art* multiprocessors. RSIM includes a simple 2D mesh network model able to simulate wormhole and packet-switching flow-control. ML-RSIM [SP06] enhances this model by including a more complex input/output infrastructure, and a fully-simulated operating system able to manage virtual memory.

Simics has a high level of detail in the model of the components that form the computing nodes, and performs the simulation of all the hardware and software. However it uses a very simplistic memory timing mechanism as all instructions and memory accesses last the same amount of time. The GEMS simulator [MSB05] enhances Simics by adding a more accurate timing model and more complex memory and CPU hierarchies: from broadcast-based SMP to a hierarchical directory-based Multiple-CMP system with different models of Network-on-Chip and coherence protocols. However it does not allow scaling to multi computer systems as does not include a complex model of the interconnection network. Another relevant extension to Simics is SIMFLEX [WWF06]. It uses statistical sampling to accelerate simulations, using established statistical sampling methods from a library of application checkpoints. Chip multiprocessors and distributed-shared-memory multiprocessor systems are modeled but still the interconnection model is very simplistic.

Mambo [AWI] is another simulator to have into consideration as it models the Cell Broadband Engine processor [IBM], whose current generation (IBM PowerXCell 8i) is part of the Roadrunner [BDH08], the current most powerful system of the Top500 list [DMS].

Some other simulators were designed with the evaluation of applications behavior in mind. A notable example is Dimemas [BLG03] which can reconstruct the execution of a parallel application in any supported architecture using a trace of that application. Dimemas models the system elements in a simple way: the network is modeled as a collection of buses and the computing nodes are modeled as a set of execution pools. On the other hand the workloads within the traces are modeled in detail, with lots of significant states available for each application thread. However, to obtain traces with such a high level of detail, an instrumented kernel is required. This environment can be used to search for bottlenecks and/or unbalancing in parallel applications, which may harm their performance.

2.2.4 Examples in the Literature

As stated in the previous sections, simulation is a widely accepted method to validate the results obtained by means of analytical studies. Most of the analytical studies discussed above [Dall90] [GTB93] [DG94] [KCR94] [AA95] [BS96] [GG97] [SOM01] [YFJ01] [GBV02] [GBC06] [KON06], corroborated their models by means of simulation.

However, there are also plenty of published works on performance evaluation of interconnection networks that rely solely on simulation. To cite just a few, for example, [ABI93] evaluated the convenience of using adaptive routing in two-dimensional mesh and torus networks implemented with cut-trough switching routers. An unrealistic router model with infinite queues was used to measure the top bound of achievable performance. Then, a more realistic model with finite queues was used. The latter showed that the performance gains achieved with the infinite-queues model can never be achieved with the finite-queue model because of the restrictions imposed by the deadlock avoidance mechanism. Authors concluded that new deadlock avoidance schemes must be proposed to take full advantage from adaptive routing.

An algorithm to enhance source routing with adaptivity for multistage interconnection networks is proposed in [ASA96]. This algorithm is evaluated by means of simulation using different synthetic traffic patterns in networks with up to 128 nodes. The study was focused on packet latency figures.

In [PV97], authors proposed the *k*-ary *n*-tree topology, which has become a *de facto* standard topology in super-clusters. This research work relied only on simulation to evaluate such topology. The evaluation included throughput and latency figures under a wide variety of synthetic traffic patterns. The evaluated system was a 4-ary 4-tree composed of 256 nodes and using one, two and four virtual channels.

Three different adaptive routing algorithms were confronted using simulation in [KJA02]. The performance evaluation of these schemes focused on throughput and load balance of both irregular and torus networks. The main conclusion found by the authors was that the performance of each scheme depended on the topology.

A pre-silicon version of the interconnection network of the IBM Bluegene/L Supercomputer was evaluated by means of simulation in [BCC03]. The evaluation was performed using random and hot-region spatial traffic patterns from independent sources. It is remarkable that this performance evaluation study allowed finding a bug in the VHDL model of the router.

A performance evaluation of three different fault-tolerance methods for Networks-on-Chip was performed in [PLB04]. This evaluation relied only on simulation and was focused on three different aspects. The first one was the fault-resilience of the methods in different scenarios of transient and permanent failures, measured as the number of packets successfully delivered. The second aspect was the latency suffered by the delivered messages. Finally, the area and energy consumption of each method was measured because the hardware implementation of each method might be affected by them.

A congestion control mechanism for multistage topologies, codenamed RECN, is proposed and evaluated by means of simulation in [DJF05]. The study compares temporal evolution of the network using the proposed mechanism against some other router configurations that do not use congestion control and that can be seen as a worst case (a single queue without any use restriction) and the best case (using virtual output queues to completely avoid head-of-line blocking).

Immucube, a fault-tolerance routing scheme for *k*-ary *n*-cubes was proposed and evaluated in [PG07]. The performance evaluation of the scheme relied only on simulation, and the figures of merit were the length of the transient state after failures, the system throughput with and without network failures, and the degradation suffered both by distance and latency due to these failures. This study only modeled a worst-case scenario in which there was no spatial locality using random uniform traffic.

Simulation was used in [VRV07] to measure the performance of delta networks when fed with uniform traffic with two priority levels. This work proposed a novel priority scheme and compared it by means of simulation against three previously proposed schemes, while discussing their limitations. Furthermore, a thorough evaluation of

performance figures, both in terms of throughput and latency, for networks ranging from 6 to 10 stages are shown for the whole network and for each one of the priority levels (high and low priority).

A comprehensive evaluation of two simple local congestion control mechanisms was performed using simulation in [MIG08]. The techniques were designed to be implemented within any router intended to be used in direct topologies. The evaluation used several synthetic traffic patterns, from independent and bursty traffic sources, to fed two dimensional torus networks of 8×8, 16×16 and 32×32 nodes.

2.3 Empirical Evaluation

Providing that the system to evaluate is available, it is possible to perform an *in-situ* evaluation of its performance. There are several benchmarks especially designed to evaluate the performance of high performance computing systems. Note that some of them test the system as a whole, while others aim to evaluate a specific piece of the system. It is also very common to make use of actual applications—those that are going to be executed by actual users—to evaluate the performance of a given computing system, to this way, optimize the execution of such applications.

2.3.1 Benchmarks

The well-known NAS Parallel Benchmark (NPB) Suite [NAS] is devised to measure the performance of distributed memory parallel computers. It is composed of several applications in the area of fluid dynamics which have a wide variety of needs in terms of CPU *versus* network usage, usage of point-to-point and collectives, and complexity of benchmark control structures. In addition, it also provides the capability to test input/output infrastructure. With such a wide diversity, this suite allows to test different parts of the evaluated systems, from the hardware itself, to the compiler, the protocol stack and the implementation of the communication library. Furthermore, the included applications have several versions to test a wide variety of environments: MPI, OpenMP, Java, high performance Fortran (HPF), hybrid platforms and even grid computing. It allows evaluating emerging hybrid architectures, such as systems based on the Cell family of processors [IBM], on the near future Intel's Larrabee processor [SCS08] or on graphical processing units [Gpgp]. Discussion about the applications forming the suite can be found in [BBB91], [TS99] and [FY02]. This suite is widely used in performance evaluation studies, both for real systems, as for simulated ones. Some experiments performed in Chapter 4 and Chapter 7 will use traces of the MPI version of some benchmarks of this suite.

Linpack [PWD] [Dong87] [DCP03] is a software library for performing numerical linear algebra that is composed of several coarse-grain applications that barely make use of the network to communicate, and that exhibit low degree of synchronization among threads (or no synchronization at all). Linpack has become the *de facto* benchmark to measure the floating point computing power of a system. It is the sorting key for the Top500 list of supercomputers [DMS] and, consequently is widely used to test high performance computing systems. In fact, most supercomputer sites make a huge effort to fine-tune this benchmark and customize the code or even the programming model in order to take the best of the system.

Linpack is also part of the High Performance Computing Challenge Benchmarks [HPC] [LDK05] widely known as HPCC. This suite includes several benchmarks to test computing performance of parallel system. HPCC also includes several applications to test the system under some *challenging* memory access patterns, which aim to stress the memory hierarchy infrastructure in order to measure not only the peak computing power but also how much it can be throttled due to memory bottlenecks.

The Sequoia benchmark suite [LLNa] is a set of benchmarks, composed by kernels and applications, that can be used to evaluate most of the components of a parallel computing system. It has several benchmarks to test single processor performance, as well as the scaling efficiency of the system. It also includes other benchmarks to test the memory infrastructure using stressing access patterns. Additional benchmarks are used to test the performance of OpenMP and MPI libraries, separately or working together. Tests to measure network bandwidth and latency, as well as collective operations latency, are also included. Finally it includes several benchmarks to test the input/output infrastructure, by means of stressing access patterns, and also by means of putting all the nodes to access dynamic libraries.

The SPEC Benchmark suite [SPE] is a benchmarks suite that allows evaluating several performance properties of high performance computing systems. It includes a wide variety of benchmarks that can test some characteristic features of these systems such as the computing power, the power consumption, the parallel file systems, the MPI and OpenMP libraries, as well as the performance of several actual scientific applications.

The Sweep3D benchmark [LAN] generates a wavefront communication pattern in a (virtual) 3D mesh topology. In each iteration of the benchmark, all the nodes make computation over the data matrix they posses and then send

the values in the boundaries of that matrix to the neighboring nodes. This benchmark is highly scalable provided that the data matrix is large enough, and therefore allows the evaluation of the raw processing power of a parallel machine.

Intel MPI Benchmarks (IMB) is a piece of software designed to evaluate MPI library performance [ISN]. It is composed of several low-level benchmarks which focus on measuring the performance of three different parts of the MPI library related to communication operations. First of all, some of the benchmarks allow testing the performance of point-to-point and collective operations. Secondly, some other benchmarks can test the performance of I/O operations. Finally, a set of benchmarks to measure the performance of one-sided communications is included. The output results are always in terms of raw performance: throughput and latency.

Splash-2 [USa] is another classical benchmark suite that was designed for the evaluation of shared-memory systems, with specific emphasis on high performance computing and computer graphics. The applications included in this suite are discussed in [SWG92] and [WOT95]. However, the included programming models are those in use about twenty years ago, so state-of-the-art programming models for shared memory architectures are not evaluated. For this reason other shared-memory benchmarks have appeared. Parsec [UP] is one of the most important, as it extends the evaluation range to other fields outside of high performance computing and also incorporates new methodologies and programming models. An overview of the included applications and their behavior can be found in [BKS08]. A thorough comparison among Parsec and Splash benchmarks is performed in [BKL08].

STREAM [UVa] [McCa95] and STREAM2 [UVb] are benchmarks to measure the memory bandwidth under very simple, but memory intensive, scenarios such as initializing all variables in an array, making a copy of an array and other memory-stressing kernels. It can be used to evaluate the memory infrastructure of single- and multi-processors.

The Interleaved or Random (IOR) Benchmark [IHB] is a benchmark specifically designed to evaluate the performance of parallel file systems, focusing on parallel/sequential read/write operations that are typical of scientific applications [SS07].

STAMP [USb] is a benchmark suite designed for research in the newborn area of transactional memory which can be used to evaluate hardware and software implementations. It consists of several applications with a variety of transactional and runtime behaviors. Furthermore, it allows the comparison of transactional and regular memory models as sequential and transactional version of the applications are provided. More information about the included applications can be found in [MCK08].

Table 3 summarizes the discussed benchmarks, what pieces of the system they aim to evaluate, the type of benchmarks, the area where benchmarks come from and some comments about them.

Benchmark	Test	Type	Area	Comments
NPB	Computing Power, OpenMP and MPI Libraries, HPL, Grids, Multi Zone, Communication Infrastructure, I/O	Applications and Kernels	Fluid Dynamics	
LINPACK	Computing Power	Application	Mathematics	Sorting key of Top500 list
HPCC	Computing Power, MPI Library, Memory Hierarchy, Communication Infrastructure	Applications and Kernels	Mathematics	Includes LINPACK and STREAM
STREAM	Memory Hierarchy	Kernels	None	Included in HPCC
SPEC	Computing Power, Power Efficiency, OpenMP and MPI Libraries, I/O, Communication Infrastructure	Applications and Kernels	Multiple Areas	Divided in specialized suites
SEQUOIA	Computing Power, OpenMP and MPI Libraries, I/O, Communication Infrastructure, Memory Hierarchy	Applications and Kernels	Multiple Areas	Includes IOR
SWEEP3D	Computing Power	Application	Nuclear Physics	
IMB	MPI Library, Communication Infrastructure	Kernels	None	Formerly Pallas
SPLASH	Computing Power, Open MP Library, I/O	Applications and Kernels	Multiple Areas	Old-fashioned
PARSEC	Computing Power, Open MP Library, I/O	Applications	Multiple Areas	Some non-scientific applications
STAMP	Hardware and Software Implementations of Transactional Memory	Applications	Multiple Areas	
IOR	I/O	Kernels	None	Included in SEQUOIA

Table 3. Brief description of the discussed benchmarks.

2.3.2 Examples in the Literature

Several years ago, most of the benchmarks discussed in the previous subsection did not exist and empirical evaluation of parallel systems was done by means of synthetic applications designed *ad-hoc* to measure the raw performance of the system. For example in [Duni91] some synthetic benchmarks were used to compare five different computing systems arranged as hypercube topologies. These benchmarks aimed to measure the computing power, the memory access, the communication infrastructure and the I/O architecture.

Synthetic tests were also used in [MAB96]. In this research work, a performance evaluation of the IBM SP2 was performed comparing two consecutive generations of the system in which the communication infrastructure and MPI library were improved. The evaluation used several synthetic tests to evaluate the point-to-point and collective performance, in terms of throughput and latency. However in this case, some of the NAS parallel benchmarks and a parallel simulator of interconnection networks were used to test the systems running complete applications.

More recently, the design of the BlueGene/L supercomputer was discussed in [ACG04]. This discussion included the evaluation a prototype of the system composed by 512 nodes using the Linpack and the NAS Parallel Benchmarks, as well as five actual scientific applications from different fields: gas dynamics, nuclear physics, molecular dynamics, astrophysical simulation and material science and engineering. A huge effort was required to port some of these applications to the Bluegene/L architecture, due to some constraints of the light-weight kernel installed on the computing nodes.

In [KWO05], several aspects of the interconnection network—routing, buffer size, multicast and topology—were evaluated over an actual cluster composed of 64 nodes. The performance of the barrier collective and the execution times of the NAS parallel benchmarks were used as figures of merit for the analysis.

The Columbia cluster, composed by 20 multi-processor nodes with 512 cores, was evaluated with the NAS parallel benchmark in [SCH06], focusing on scalability. Its performance figures were compared with those obtained in an Opteron-based cluster with 64 bi-processor nodes. This study showed that for most of the benchmarks scalability was good. The only exceptions were CG and IS benchmarks because the high density of processors of the Columbia nodes generated congestion in memory access because of their hierarchical organization.

Authors of [ABK06] evaluated the architecture of the AMD Opteron Dual-Core by means of a wide set of benchmarks and two actual scientific parallel applications. The used benchmarks were Stream, the High Performance Computing Challenge, the Intel MPI Benchmarks and the NAS Parallel Benchmarks. The applications were two simulators of molecular dynamics and oceans respectively. However the study only included small-scale evaluations of up to 16 cores (8 dual-core processors).

A thorough evaluation of Cray's XT4 was performed in [ABF07], comparing the system with its ancestor the XT3. The High Performance Computing Challenge benchmark was used to evaluate memory and network performance. Network performance was also evaluated using ping-pong-like micro kernels. Furthermore the parallel file system (Lustre) was evaluated using two Input/Output benchmarks: the IOR benchmark and a customized benchmark codenamed as CustomIO1. Finally a set of five scientific applications were used to assess that previously obtained performance figures correlated with the performance of actual applications. These applications simulated different physical phenomena: modeling atmosphere, ocean, bio-molecular interactions, turbulent combustion and nuclear fusion.

Several synthetic kernels as well as four actual parallel scientific applications simulating diverse physical phenomena were used in [OCC08] to evaluate the performance of six computing platforms. The evaluated platforms were heterogeneous as they were implemented using different processor architectures, network technologies and topologies. The scientific applications were from four different areas: magnetic fusion, plasma physics, astrophysics, and material science.

The architecture of the Marenostrum Supercomputer was discussed and evaluated in [RBL08]. STREAM2 and Intel MPI Benchmarks were used to evaluate the in-node memory performance as well as the performance of MPI-based network communications. From those tests, memory and network bandwidths were obtained and several mathematical models were derived. Finally, the performance of some NAS parallel benchmarks was measured for up to 64 processors.

In [SCG08] the High Performance Computing Challenge and the Intel MPI Benchmarks were used to compare five different computing platforms (SGI Altix, Cray X1, Cray Opteron Cluster, Dell Xeon and NEC SX-8), comprising a wide variety of processor architectures (both scalar and vectorial) and network infrastructures (a SGI Numalink4, a Cray torus network, a Myrinet interconnect, an InfiniBand network, and a NEC IXS crossbar).

The currently most powerful supercomputer, the RoadRunner built by IBM, was evaluated in [BDH08]. Firstly, this study used some low-level benchmarks to measure the performance of memory and interconnection architecture,

comparing the performance of two models of the cell chip: the preceding Cell Broadband Engine processor and the most recent IBM PowerXCell 8i processor. Finally, the system performance and scalability were tested by means of the Sweep3D benchmark.

2.4 Conclusions

We have performed a review of a wide range of performance evaluation methodologies. The discussed methodologies can be separated in three main groups: analytical evaluation, simulation-based evaluation and empirical evaluation. From the wide range of analytical methods discussed in the literature, we have selected a few because of their adequacy to the field of interconnection network evaluation, and its ability to modeling these networks in a simple way. The considered methodologies have been the following: derivation of topological characteristics, Markov chains, Queueing theory and Petri nets. The overview of simulation-based evaluations has included the details to bear in mind to perform the simulation: the performance of several different simulation models, the different levels of accuracy, the modeling of the elements of the system and the modeling of the workloads. Finally, the review of empirical evaluations has included several benchmarks to test the different components of computing systems (raw computing power, memory architecture, I/O infrastructure, interconnection network, etc). We have shown the capabilities and the limits of each performance evaluation methodology, which can be summarized as follows:

- Mathematical Analysis: fast and simple, but not very accurate. Evaluation is restricted to the steady-state of the system under a given set of temporal and spatial distributions.
- Simulation: wide range of complexity and fidelity to the simulated system depending on the used models. The more detailed are the models, the more resources are needed to perform the simulation.
- Empirical: requires the availability of the system under evaluation. Benchmarks based on specific kernels may not fully characterize applications behavior. Alternatively, benchmarks based on one single application may not be representative of the full range of applications.

Chapter 3. Simulation Environment

This Chapter is devoted to discuss our workbench to simulate and evaluate interconnection networks: INSEE (standing for Interconnection Networks Simulation and Evaluation Environment) [RM05]. INSEE is composed of two main modules: a Functional Simulator of Interconnection Networks (FSIN) and a TRaffic GENeration module (TrGen). Our environment includes also some other additional modules that interface with TrGen to provide application-driven workloads: a modified MPICH library to obtain traces, and some modules implemented in Simics to perform Full-System simulation. A schematic depiction of the structure of INSEE is shown in **Figure 5**.

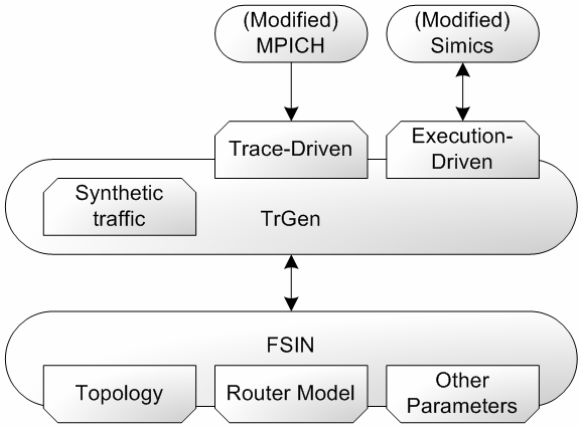

Figure 5. Overall design of INSEE.

INSEE tools have been successfully used in our research group to carry out a wide variety of studies in the field of performance evaluation of interconnection networks, including: the effect of head-of-line blocking (HoLB) at injection [IMG06], the impact in the performance of several injection interfaces as congestion control mechanisms [IMG05], the performance of local congestion control mechanisms [MIG08], the evaluation of several topological proposals [NMR] [CMV07] [NLM09], and the effect of task and node allocation on the performance of parallel applications [NPM09] [PNM09], among others. A remarkable feature of INSEE is that it allows simulating large-scale networks of up to 64K nodes in a regular *off-the-shelf* desktop computer with 2GB of RAM.

3.1 FSIN

The heart of our environment is FSIN, a flexible, lightweight functional simulator (meaning that the router functionality is modeled in detail, but the hardware is not) that allows us to *rapidly* assess the performance of large-scale systems. Time is measured in terms of an abstract cycle, defined as the time required by a router to route and forward a *phit* (physical transfer unit). Furthermore, FSIN is able to simulate a wide variety of router models and topologies.

3.1.1 Simulation Engines

FSIN was firstly designed for the evaluation of interconnection networks under scenarios of heavy utilization (congestion). For this reason a time-driven simulation engine was implemented. Nowadays the interests of our

research work also include the performance evaluation using applications and, consequently we are developing an event-driven engine that will allow us to evaluate the performance of complete applications in reasonable time.

3.1.1.1 Time-Driven Simulation Engine

The time-driven FSIN works as follows. At each cycle, all the elements of the system are checked. Packets are generated, moved and consumed as needed. However, to accelerate the execution of application workloads, routers are not checked if the network is known to be empty. The time-driven engine is a mature piece of software that is known to be accurate and has been used in most of our research studies. However, it is not very efficient when simulating computation-biased applications because, as explained before, the low density of events makes time-driven engines to perform poorly.

3.1.1.2 Event-Driven Simulation Engine

We are currently adapting FSIN to incorporate an event-driven engine. This adaptation requires a significant effort in terms of implementation time. By the time of writing this dissertation, the event-driven engine itself is fully implemented and tested. Furthermore it has been used to model and simulate several processes that do not require using the full functionalities of FSIN, such as the booting process of the SpiNNaker system [KNR09].

As stated in the previous Chapter, it is very important to select a well-performing data structure to store the events, as the execution time of the simulator is greatly influenced by this choose. For this reason we have implemented a modified version of the calendar queue introduced by Brown [Brow88]. A depiction of the data structure needed by a regular calendar queue is shown in **Figure 6**. In a calendar queue, a collection of lists contains all the events and accessing to any list is $O(1)$ when the events are well-balanced in the structure.

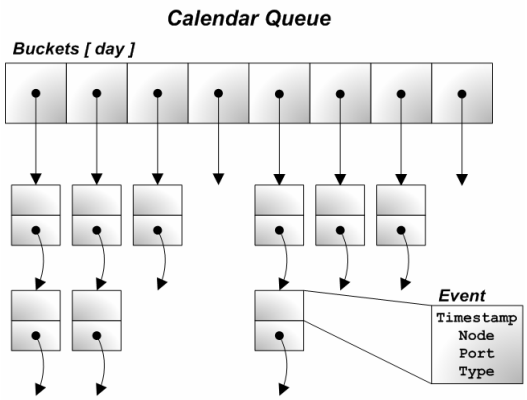

Figure 6. Data structure in a standard calendar queue.

However, bearing in mind that FSIN measures time using discrete cycles, it will occur that there are lots of events that have to be executed just at the same instant of the simulated time (meaning that they have exactly the same timestamp). This behavior would lead to an unbalanced utilization of the calendar queue. To avoid this unbalance we have modified the original conception of calendar queues in such a way that we use *meta-events* that contain all the events that have to be executed simultaneously. Each meta-event is further divided into several event sets taking into account the even types. This is because in the original design of FSIN all the events with the same timestamp are executed in order of their type, first injection, then routing, and finally movement of packets. Note that the order of the events included in an event set is irrelevant, and for this reason the performance loss in comparison with the regular calendar queue is negligible. This modification reduces the number of elements (meta-events *versus* events) within the calendar queue, which simplifies its operation. The final structure of our modified calendar queue is shown in **Figure 7**.

To show the impact of the event-list implementation on simulation time, we have measured the time needed to execute several different configurations of the SpiNNaker boot process [KNR09] using event-driven simulation with lists, the regular calendar queue and our modified calendar queue. The exact parameters of each configuration are not given as SpiNNaker boot-up process is outside of the scope of this dissertation, and the experiments are only to illustrate the effects of an improper data structure. Simulations were executed in a commodity PC with an AMD Athlon 64 X2 Dual Core 5600+ processor and 4GB of RAM. The measured execution times are shown in **Table 4**.

We can see how using a simple list led to awfully long execution times, up to two orders of magnitude longer than those achieved with the two calendar queues. Moreover, we can see how, obviously, the larger is the simulation time, the longer is the difference between the two implementations of the calendar queue and the list; given that the

simple list is O(n), it does not scale very well. It is also remarkable that the modified calendar queue performs better than the regular calendar queue; this is because in these simulations it is very common that several events happen in the same timestamp. For this reason, our modified implementation could not be profitable in other circumstances.

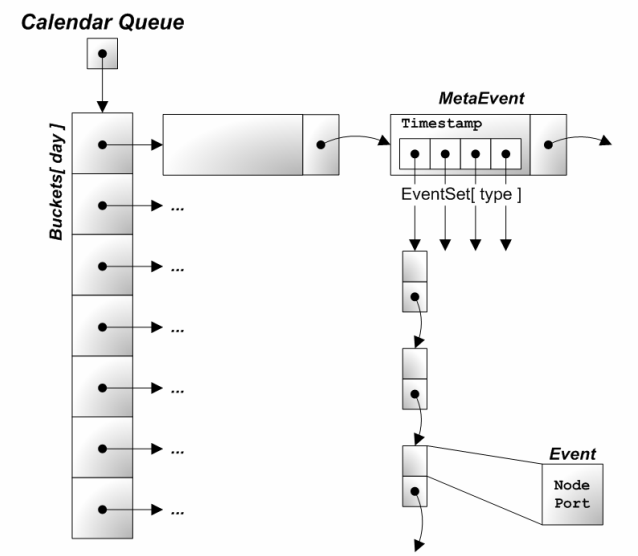

Figure 7. The data structure in our modified calendar queue.

Note that, as the event-driven engine is not completely integrated with INSEE, all the discussion about INSEE in the remaining of this dissertation will always refer to the time-driven simulation engine.

	Simple List	Calendar Queue	Modified Calendar Queue
Configuration 1	26.269 s	1.133 s	0.259 s
Configuration 2	34.691 s	0.707 s	0.317 s
Configuration 3	115.416 s	1.156 s	0.573 s
Configuration 4	383.078 s	6.141 s	0.983 s

Table 4. Time needed to simulate the SpiNNaker boot process.

3.1.2 Modeled Routers

Models of a wide variety of router architectures with different purposes have been implemented in FSIN. The Dally router and the Bubble router are designed to be used in direct networks, arranged in the typical 2D or 3D topologies commonly used in massively parallel processors. The multistage switch allows the simulation of tree-based, multistage, indirect topologies, such as those used in many large-scale clusters. Finally the SpiNNaker router is a bespoke router architecture designed for fault-tolerance and low consumption.

3.1.2.1 Dally router

A router architecture designed to be used in toroidal *k*-ary *n*-cube topologies was introduced by Dally in [DS87]. This architecture uses a virtual channel scheme that allows deadlock-free routing using two virtual channels as Escape channels following oblivious routing (DOR). The cycles embedded in each dimension ring are eliminated by means of restrictions in the use of the Escape channels, which cut the physical cycle into non-cyclic combinations of virtual channels. In **Figure 8**, a depiction of the behavior of this deadlock avoidance scheme is shown. Adaptivity is possible increasing the number of virtual channels: two virtual channels (usually 0 and 1) are used as Escape channels (using deadlock-free oblivious routing) while adaptive routing can be used in the remaining ones. The overall arrangement (according to Duato's theorem [Duat95] is an adaptive, but deadlock-free routing algorithm.

Several strategies to route packets are implemented in FSIN to be used together with the Dally router:

- In **trc** all packets are injected in the same Escape channel. Packets remain in that channel, until reaching the wrap-around link of a toroidal topology. Then, it is switched to the other one. In other words, the cycle is transformed into a spiral (acyclic) as can be seen in **Figure 8**b.

- In **basic** those packets that have to cross the wrap-around links of a dimension—let us call them P_W—circulate through one Escape channel in that dimension. Those that do not have to use the wrap-around links—P_D, standing for direct packets—use the other Escape channel. This way, the physical ring is split into two separate virtual chains, which do not include cycles—see **Figure 8**c.
- **improved** is an optimization of the basic policy to obtain better balancing in the utilization of both VCs. P_W packets are forced to transit by one of the Escape channels. P_D packets choose randomly the Escape channel to use. This way, P_W packets cannot block P_D packets, and thus P_D packets eventually will use the other channel and reach their destination. Note that this policy may lead to starvation of P_W packets, as PD packets may make intensive use of the two Escape channels.
- **adaptive** is the same as the improved policy, adding routing adaptation capabilities. Two virtual channels work as Escape channels following the improved policy and the remaining virtual channels are used as adaptive channels (always using minimal paths). The packets randomly select a viable adaptive output port and, only in the case that no adaptive channel is available, the packets try to use an Escape channel as dictated by the improved strategy (P_W packets are restricted to one of the Escape channels, while P_D packets are free to use any of the two Escape channels).

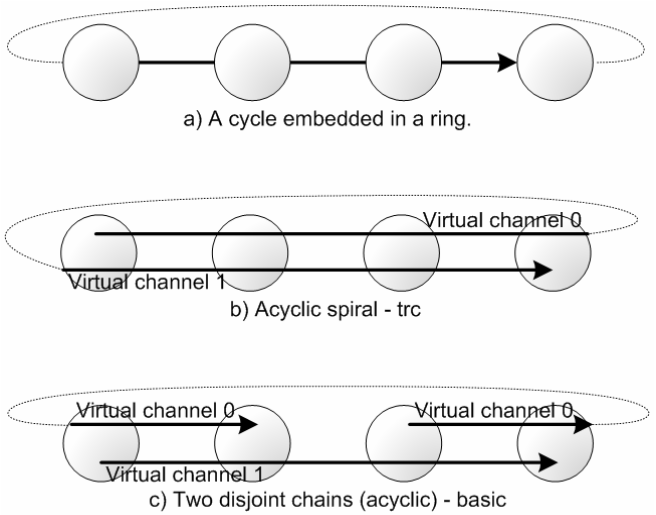

Figure 8. Dally scheme to avoid deadlock in a unidirectional ring. a) A cycle in a ring. b) The cycle is cut by changing the virtual channel at crossing the wrap-around link (trc policy). c) The cycle is removed by splitting the physical ring into two separate virtual chains (basic policy).

The modeled router has queues in the input ports and a crossbar interconnection among input and output ports as depicted in **Figure 9**. Note that the crossbar has as many ports as virtual channels (plus additional ports for injection and consumption). Therefore, the utilization of VCs augments crossbar complexity.

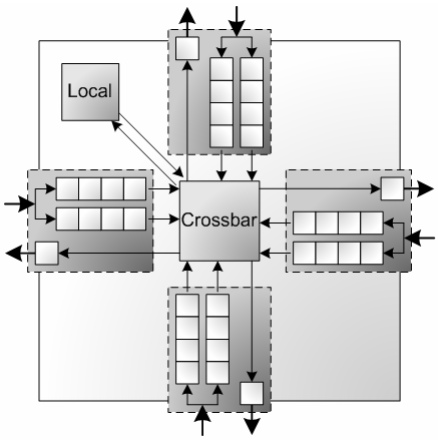

Figure 9. Architecture of Dally and Bubble routers for a 2D topology and two virtual channels per physical link.

3.1.2.2 Bubble router

The bubble router [PIB01] is also designed to be used in *k*-ary *n*-cube topologies, having the same architecture of the Dally router (see again **Figure 9**) but follows a different approach to avoid deadlocks; instead of avoiding cycles, this scheme avoids that all the buffers in a ring (cycle) become full. To do so, packets can only enter an Escape channel, from injection or from other channel, when there is room to store at least two packets, one for the entering packet and another one to ensure that there is, at least, one free slot (*bubble*) to be used by the packets inside the Escape channel. This behavior, depicted in **Figure 10**, requires only one Escape channel to avoid deadlock. In the depiction, only node 0 and node 3 can inject into the ring, as they have room enough in the queues of the ring to comply with the bubble restriction. Alternatively, the other two nodes (node 1 and node 2) are not allowed to inject as they do not comply with that restriction.

Adaptivity can be easily incorporated by adding additional virtual channels with adaptive routing, but always checking the bubble restriction when moving packets from an adaptive channel to the Escape channel.

The bubble router can route packets following several strategies (*request modes*) to decide, when a packet is awaiting at the head of the input queue (or at the injection queue), to which output VC a request will be made to forward the packet.

- When using the **random** request mode, the router tries to route packets waiting at an input port through the output port of the same VC. If this is not available, any profitable adaptive VC can be requested, choosing it randomly. Finally, if no profitable adaptive VC is available then packets try the Escape channel.
- The **shortest** request mode works as the previous one but, when trying a profitable adaptive VC, selection is done considering the space available in the channel's queue, choosing the one with more room.
- If the **smart** request is selected, a packet is injected initially into a random channel and tries to continue in the same dimension, way and virtual channel. If this is not possible, then it is moved to a profitable adaptive virtual channel in another dimension. If no adaptive, profitable channel is found, the packet tries to move to the Escape channel. This strategy tries to avoid congested links by changing the traveling dimension every time it reaches a port that is in use.
- With **oblivious** request mode all VCs follow the bubble restrictions (all of them behave as Escape channels), using DOR routing. Once a packet enters into a VC, it never abandons it—in short, there is no adaptivity.

Several other experimental strategies have been implemented for the bubble router but are not discussed for the sake of simplicity.

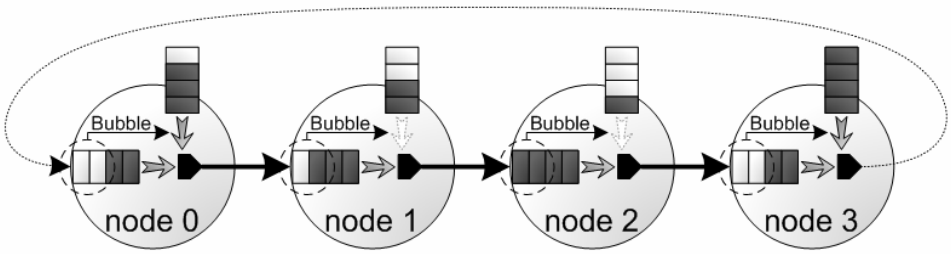

Figure 10. Bubble scheme to avoid deadlock in a unidirectional ring.

3.1.2.3 Multistage router

Routers for multistage IN are simpler than those for *k*-ary *n*-cubes, as routing in multistage topologies is deadlock-free when shortest path algorithms (such as up*/down*[SBB90]) are used. The router has a centralized crossbar and can have any arbitrary number of ports and virtual channels. Queues are at the input ports. The model of the router for multistage topologies is depicted in **Figure 11**. Routing can be static or adaptive, but in both cases shortest path routing is used to guarantee deadlock freedom:

- In **Static** routing the upward path is defined statically and depends on the source of the packet (*source id* mod *k*). The downward path is also static and depends on the destination of the packet (*destination id* mod *k*). Furthermore, if physical links are split into several virtual channels, one virtual channel is randomly selected at injection and the packet never leaves it.
- When using **Adaptive** routing the downward path is static and depends on the destination, but the packets can adapt when traveling upwards. A credit-based adaptive routing is used; it works as follows: a packet in

the head of an input queue tries to go through the profitable output channel with most available free room in the queue of the neighbor input port (*credit*). If several output channels have the same credit, one of them is selected randomly. Note that credit transmission is performed *out-of-band* and, therefore they do not interfere with regular traffic.

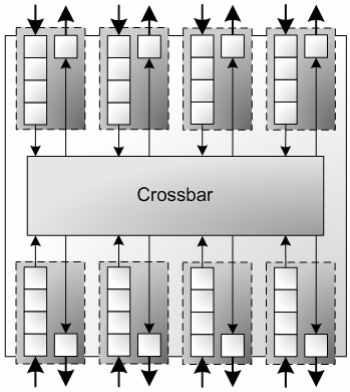

Figure 11. Architecture of a multistage router with eight ports and one virtual channel per physical link.

3.1.2.4 SpiNNAker router

The SpiNNaker router [PFT07] is also implemented within INSEE. This is a specific-purpose router designed for a large-scale machine with strong hardware constraints. The main focus of the SpiNNaker system is on low-consumption and fault-tolerance, and the design of the router also follows these concerns. As crossbar-based switching engines require a high amount of hardware resources, they can not be part of the SpiNNaker router; therefore, it was designed using a more frugal approach. All the ports of the router are hierarchically merged in such a way that the routing engine is accessed by a single packet at once. This way, a packet that can not be forwarded will force all the packets in the router to wait until it is forwarded or discarded, an undesired effect known as Head-of-Line Blocking. Fortunately, as the routing engine is relatively faster than the transmission ports, this situation is unlikely to happen. A depiction of the architecture of the router is shown in **Figure 12**.

Routing is performed by means of the routing tables inside each router. However instead of being destination-based, the routing tables choose the output port(s) for a given packet taking into account the source of the packet. This behavior makes the source nodes and the packets being unaware of the destination node(s).

A more detailed description of the architecture of the SpiNNaker system, including the router and the network, will be provided in Chapter 6, where a performance evaluation of the interconnection network of SpiNNaker is carried out.

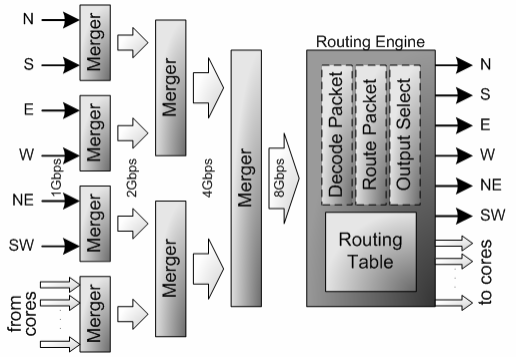

Figure 12. Architecture of the SpiNNaker router.

3.1.2.5 Other Router Parameters

The **phit length** (physical transfer unit, measured in Bytes) and the **packet length** (measured in phits) can be set to any arbitrary value. Note that the packets have a fixed length and when a smaller packet is required, it will carry some empty phits. Furthermore, the **transit queue length** and the **injection queue length** (measured in terms of room to store packets) can be set to any arbitrary value larger than one packet.

The routers can be implemented with any arbitrary **number of injectors**. Several **Injection** policies are implemented to be used with direct topologies:

- **Shortest** policy inserts a newly generated packet into the injection queue with more room. This is the policy used in the multistage topologies, as *pre-routing* does not have sense in this context.
- When using **dor** policy there is a dedicated injection queue per network dimension/direction, which significantly reduces Head-of-Line blocking effects [IMG06]. A dimension order pre-routing phase is carried out to decide in which queue will be injected the new packet. If the selected queue is not available, the injection will be stopped.
- The **Dor+shortest** policy tries to inject using the **dor** policy but, if the selected injection queue is full, then it uses the shortest policy.
- In the **shortest profitable** policy, the packet will be injected in the valid queue (once pre-routed) that has most empty space.
- **Longest path** policy injects packets in the queue associated to the direction through which the packet has to travel a highest number of hops.

The **arbitration** of output ports (selection among all the requesting input queues) can be performed in several ways, some of them are unaware of the traffic, and some others take into account some attributes of the packets to give priority to certain flows. The arbitration policy can be defined for all the router models but the SpiNNaker router, as it treats the packets once at a time. The defined policies are the following:

- **Round Robin** arbitration.
- **Random** arbitration selects randomly among all the requesting input ports.
- **FIFO** arbitration selects the first input port that requested the output.
- **Longest** arbitration selects the input port having the highest queue occupation.
- **Age** arbitration selects the input port containing the oldest packet, measured since the packet was injected to the network.

The direct routers allow two alternative **Consumption** modes:

- **Single** consumption policy treats the consumption port as a regular output port following the selected port arbitration strategy.
- **Multiple** consumption policy assumes a consumption port wide enough to accept simultaneously a phit from each input port, meaning that several packets can be consumed simultaneously.

Furthermore, we have implemented and evaluated several **Congestion Control Mechanisms** around the Dally and the bubble routers. Two of them are local mechanisms that detect congestion taking into account only the state of the queues of the router itself. The other one is a global mechanism that throttles the injection taking into account the state of the whole system. These mechanisms are described as follows:

- In-transit Priority Restriction (**IPR**). For a given fraction P of cycles, priority is given to in-transit traffic, meaning that, in those cycles, injection of a new packet is only allowed if it does not compete with packets already in the network. P may vary from 0 (no restriction) to 1 (absolute priority to in-transit traffic). This mechanism can be used along with the two router models and is the method applied in the IBM's BlueGene/L torus network [ABC05].
- Local Buffer Restriction (**LBR**). This mechanism has been designed specifically for adaptive routers that rely on Bubble Flow Control to avoid deadlock in the escape sub-network. A previous study showed that the bubble restriction also provides congestion control for the escape sub-network [IMG06]. LBR extends this mechanism to all new packets entering into the network. That is, a packet can only be injected into an adaptive virtual channel if such action leaves room for at least B packets in the transit buffer associated to that virtual channel. The parameter B indicates the number of buffers reserved for in-transit traffic. In other words, congestion is estimated by the local buffer occupancy.
- Global Congestion Control (**GCC**). This mechanism estimates network congestion by examining the status of the whole network and stopping injection when it reaches a given threshold G, which is usually given as a fraction of the whole network capacity. In our implementation, the network utilization is measured every a given number of cycles T, and transmitted out-of-band to all nodes, in such a way it does not interfere with application packets.

3.1.3 Topologies

Several topologies, both direct and indirect, have been implemented in FSIN. Some of them are widely used in actual systems. Some other less frequent topologies are part of our research work. We will show some characteristics of the topologies: number of nodes N, number of dimensions d, number of ports per router or radix R, diameter D, average distance $\overline{d}$ and theoretical throughput for uniform traffic θ. At any rate, implementing new topologies in FSIN is quite simple, as the only requirement is to provide a neighborhood function and a routing function.

3.1.3.1 Perfect Crossbar Network

This is a special network that is able to interconnect nodes in an unblocking any-to-any fashion, that is, each node is able to send a message to any other node with just two hops, one from the source network interface card (NIC) to the crossbar and another one from the crossbar to the destination NIC. We make the assumption of a perfect crossbar, which allows to manage up to the number of ports messages at once, provided that all those messages come from different sources and do not compete for the destination ports. In other words, when bottlenecks appear, they are caused by contention at injection or consumption ports. In our studies the perfect crossbar is used to establish an ideal bound for the execution time of any proposed workload and, consequently, it can be used as the yardstick in simulation-based performance evaluation studies; an example of such behavior can be found looking at [NMR08]. A depiction of a 6-node perfect crossbar is shown in **Figure 13**.

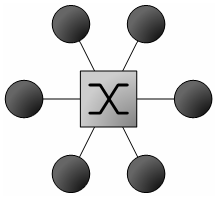

Figure 13. A sample of a 6-node Perfect Crossbar.

It is remarkable that this network is implemented using a single switching element as the previously discussed multistage router. A summary of the characteristics of a perfect crossbar network is shown in **Table 5**.

Nodes	N
Radix	$R = N$
Diameter	$D = 2$
Average Distance	$\overline{d} = 2$
Throughput	$\theta = 1$

Table 5. Characteristics of a Perfect Crossbar network for a given number of nodes N.

3.1.3.2 Direct topologies

Direct topologies are those in which every switching element or router is connected to a compute node. FSIN has topological models of the standard meshes and tori for 1, 2 and 3 dimensions. Furthermore models for several topological studies have been added: 2- and 3-dimensional twisted tori, the Midimew topology and the SpiNNaker topology. All these topologies will be discussed in the following paragraphs and are depicted in **Figure 14**. In these graphs nodes (and their corresponding routers) are represented by dark circles and links are represented by lines between them.

In the following sections that summarize the topological characteristics of indirect networks, N denotes the total number of nodes, and N_X, N_Y and N_Z the number of nodes in each dimension. The number of dimensions is denoted by d and the radix of the routers is R. The coordinates of a node are represented by x, y and z, and those for the corresponding neighbor are x', y' and z'. Note that when showing the neighborhood relationship among nodes, if the coordinate in any dimension of the neighbor node is not explicitly stated, it implicitly takes the same value as this coordinate in the origin node (that is, $x'=x$, $y'=y$ and $z'=z$). In the case of the midimew, n is a node and n' is its corresponding neighbor. Note also that $\perp$ represents a null link, that is, a link that is not connected. The operations **mod** and **max** correspondingly represent the positive remainder of the integer division and the maximum value among a set of values. Finally, *i.o.c.* stands for *in other case*. The following neighborhood descriptions are done for the highest number of dimensions available; it is straightforward to reduce these descriptions to a lower number of dimensions. Note that, in FSIN all direct topologies but meshes can be built using unidirectional or bidirectional links.

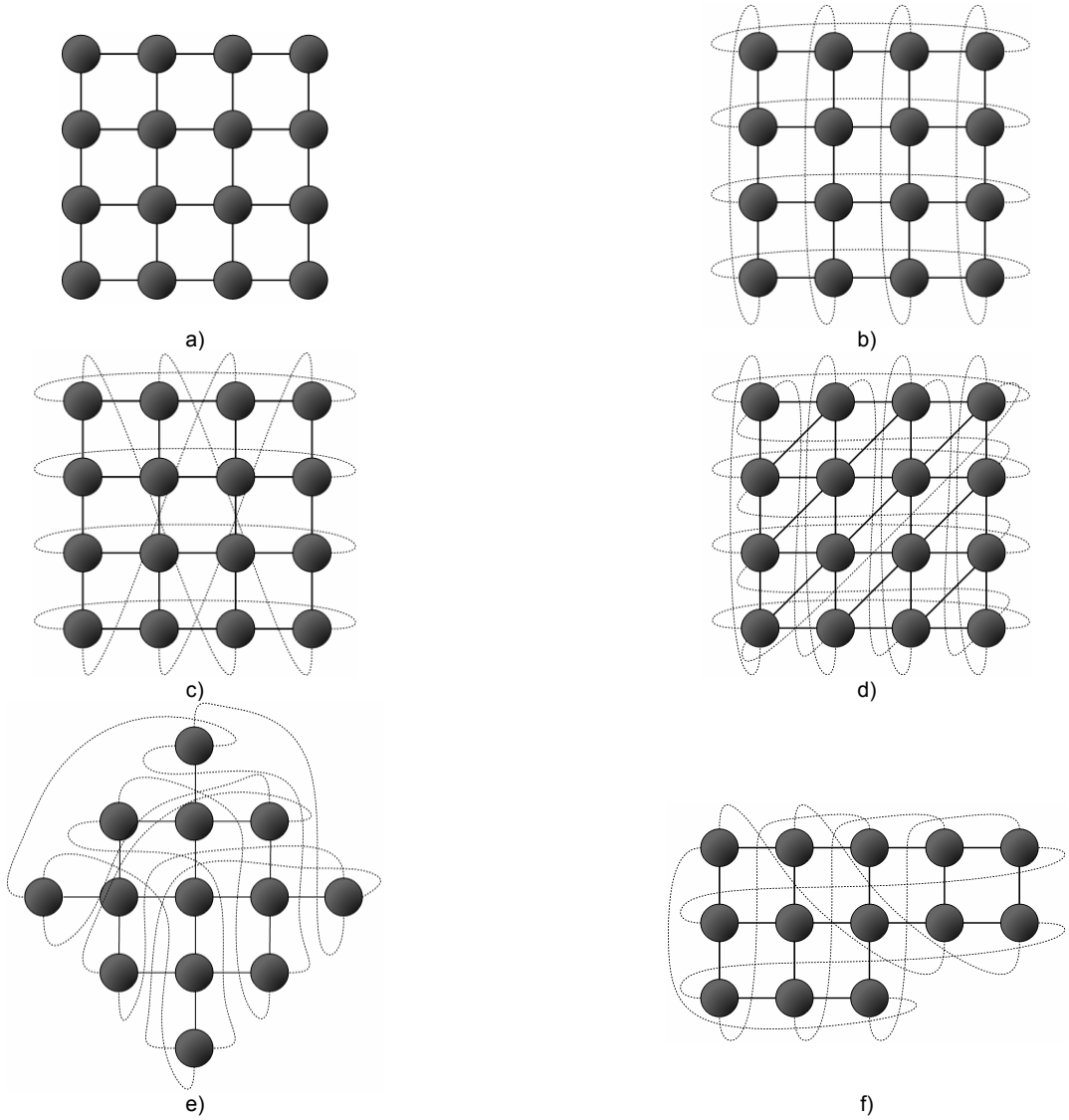

Figure 14. Examples of the direct topologies implemented in FSIN. a) 4×4 mesh. b) 4×4 torus. c) 4×4 twisted torus with skew 2. d) 4×4 SpiNNaker topology. e) and f) 2 representations of a 13-node midimew.

3.1.3.2.1 Mesh

A mesh [DT04] is the simplest direct topology; the nodes are arranged in a *d*-dimensional array and identified by their Cartesian coordinates. Some links at the peripheral nodes are disconnected—there are not wrap-around links. Meshes of one dimension are widely denoted as chains. A 4×4 mesh topology is shown in **Figure 14**a. This topology was historically used in HPC systems because of its ease to scale up as new nodes can be attached to the unconnected links. Nevertheless, currently this topology has been abandoned in this context in favor of networks with better topological characteristics, such as the torus. It is remarkable that the mesh topology is gaining popularity in the context of Networks-on-Chip because of the simplicity to deploy them in silicon [MKH07]. The neighborhood description of a mesh is as follows:

$$X+:\quad x'=\begin{cases}\perp & x=N_x-1\\ x+1 & i.o.c.\end{cases}\qquad X-:\quad x'=\begin{cases}\perp & x=0\\ x-1 & i.o.c.\end{cases}$$

$$Y+:\quad y'=\begin{cases}\perp & y=N_y-1\\ y+1 & i.o.c.\end{cases}\qquad Y-:\quad y'=\begin{cases}\perp & y=0\\ y-1 & i.o.c.\end{cases}$$

$$Z+:\quad z'=\begin{cases}\perp & z=N_z-1\\ z+1 & i.o.c.\end{cases}\qquad Z-:\quad z'=\begin{cases}\perp & z=0\\ z-1 & i.o.c.\end{cases}$$

Some interesting topological properties of the mesh are summarized in **Table 6** .

Nodes	$N = N_x \cdot N_y \cdot N_z$
Dimensions	$d \in \{1, 2, 3\}$
Radix	$R = 2 \cdot d$
Diameter	$D = N_x + N_y + N_z - 3$
Average Distance	$\overline{d} \to \frac{N_x + N_y + N_z}{3}$
Throughput	$\theta = \frac{4}{\max(N_x, N_y, N_z)}$

Table 6. Topological properties of the mesh.

3.1.3.2.2 Torus

The torus [DT04] is another well-known topology that has been historically used to interconnect massively parallel processors. Nodes in a torus are arranged in a *d*-dimensional array and identified by their Cartesian coordinates. In opposition to the mesh, the nodes in the boundaries of the topology are connected among them by means of wrap-around links. Single-dimensional tori are also known as rings. A 4×4 torus is shown in **Figure 14**b. The neighborhood description of tori is as follows:

$$X+: \quad x' = (x+1) \bmod N_x \qquad X-: \quad x' = (x-1) \bmod N_x$$

$$Y+: \quad y' = (y+1) \bmod N_y \qquad Y-: \quad y' = (y-1) \bmod N_y$$

$$Z+: \quad z' = (z+1) \bmod N_z \qquad Z-: \quad z' = (z-1) \bmod N_z$$

The topological properties of tori are summarized in **Table 7**.

Nodes	$N = N_x \cdot N_y \cdot N_z$
Dimensions	$d \in \{1, 2, 3\}$
Radix	$R = 2 \cdot d$
Diameter	$D = \left\lfloor \frac{N_x}{2} \right\rfloor + \left\lfloor \frac{N_y}{2} \right\rfloor + \left\lfloor \frac{N_z}{2} \right\rfloor$
Average Distance	$\overline{d} \to \frac{N_x + N_y + N_z}{4}$
Throughput	$\theta = \frac{8}{\max(N_x, N_y, N_z)}$

Table 7. Topological properties of tori.

Currently, the torus topology is used in several supercomputer systems around the world such as the IBM's Bluegene family of super computers [BCC03] and the Cray XT family [ABF07].

3.1.3.2.3 Midimew

The midimew [BHB91]—standing for MInimal DIstance MEsh with Wrap-around links—is a two-dimensional symmetric topology based on circulant graphs. It provides the best distance-related characteristics for any given number of nodes using routers of radix 4. Two different representations for a 13-node midimew network are shown in **Figure 14**e and **Figure 14**f. The neighborhood description of a midimew is the following:

$$X+: \quad n' = \left(n + \left(\left\lceil \sqrt{\frac{N}{2}} \right\rceil - 1 \right) \right) \bmod N \qquad X-: \quad n' = \left(n - \left(\left\lceil \sqrt{\frac{N}{2}} \right\rceil - 1 \right) \right) \bmod N$$

$$Y+: \quad n' = \left(n + \left\lceil \sqrt{\frac{N}{2}} \right\rceil \right) \bmod N \qquad Y-: \quad n' = \left(n - \left\lceil \sqrt{\frac{N}{2}} \right\rceil \right) \bmod N$$

The topological properties of the midimew topology were derived in [BHB91] and are shown in **Table 8**.

Nodes	$N \geq 2$
Dimensions	$d = 2$
Radix	$R = 4$
Diameter	$D = \begin{cases} \left\lceil \sqrt{\frac{N}{2}} \right\rceil - 1 & N \leq 2 \cdot \left\lceil \sqrt{\frac{N}{2}} \right\rceil^2 - 2 \cdot \left\lceil \sqrt{\frac{N}{2}} \right\rceil + 1 \\ \left\lceil \sqrt{\frac{N}{2}} \right\rceil & i.o.c. \end{cases}$
Average Distance	$\overline{d} = D \cdot \left[1 - \frac{2 \cdot (D^2 - 1)}{3 \cdot (N-1)} \right]$
Throughput	$\theta = \frac{8 \cdot (2 \left\lceil \sqrt{\frac{N}{2}} \right\rceil - 1)}{N}$

Table 8. Topological properties of midimew.

3.1.3.2.4 Twisted Torus

The twisted torus [CMV07] is an optimization of the regular torus as it provides, for the same number of nodes, better topological characteristics: bisection bandwidth, average and maximum distance. An interesting property of this topology is that, for networks with the double of nodes in one dimension than in the others ($2a \times a$ and $2a \times a \times a$) symmetry can be maintained, which help balancing the use of the channels of the different dimensions: the network does not present bottlenecks. A representation of a 4×4 twisted torus with a twist of 2 from dimension Y over dimension X is shown in **Figure 14**c. The neighborhood description of twisted tori is below—note that s_{uv} stands for the skew (or twist) of the dimension u over the dimension v.

$$X+: \quad \begin{aligned} x' &= (x+1) \bmod N_x \\ y' &= \begin{cases} (y + s_{xy}) \bmod N_y & x = N_x - 1 \\ y & i.o.c. \end{cases} \\ z' &= \begin{cases} (z + s_{xz}) \bmod N_z & x = N_x - 1 \\ z & i.o.c. \end{cases} \end{aligned} \qquad X-: \quad \begin{aligned} x' &= (x-1) \bmod N_x \\ y' &= \begin{cases} (y - s_{xy}) \bmod N_y & x = 0 \\ y & i.o.c. \end{cases} \\ z' &= \begin{cases} (z - s_{xz}) \bmod N_z & x = 0 \\ z & i.o.c. \end{cases} \end{aligned}$$

$$Y+: \quad \begin{aligned} x' &= \begin{cases} (x + s_{yx}) \bmod N_x & y = N_y - 1 \\ x & i.o.c. \end{cases} \\ y' &= (y+1) \bmod N_y \\ z' &= \begin{cases} (z + s_{yz}) \bmod N_z & y = N_y - 1 \\ z & i.o.c. \end{cases} \end{aligned} \qquad Y-: \quad \begin{aligned} x' &= \begin{cases} (x - s_{yx}) \bmod N_x & y = 0 \\ x & i.o.c. \end{cases} \\ y' &= (y-1) \bmod N_y \\ z' &= \begin{cases} (z - s_{yz}) \bmod N_z & y = 0 \\ z & i.o.c. \end{cases} \end{aligned}$$

$$Z+: \quad \begin{aligned} x' &= \begin{cases} (x + s_{zx}) \bmod N_x & z = N_z - 1 \\ x & i.o.c. \end{cases} \\ y' &= \begin{cases} (y + s_{zy}) \bmod N_y & z = N_z - 1 \\ y & i.o.c. \end{cases} \\ z' &= (z+1) \bmod N_z \end{aligned} \qquad Z-: \quad \begin{aligned} x' &= \begin{cases} (x - s_{zx}) \bmod N_x & z = 0 \\ x & i.o.c. \end{cases} \\ y' &= \begin{cases} (y - s_{zy}) \bmod N_y & z = 0 \\ y & i.o.c. \end{cases} \\ z' &= (z-1) \bmod N_z \end{aligned}$$

The topological properties of particular instances of twisted tori were derived in [CMV07] and are summarized in **Table 9**. RTT stands for rectangular twisted torus and refers to $2a \times a$ topologies with a skew $s_{yx} = a$. PTT stands for prismatic twisted torus and refers to $2a \times a \times a$ topologies with only one skew, $s_{yx} = a$. Finally, PDTT stands for prismatic doubly twisted torus and refers to $2a \times a \times a$ topologies with two skews, $s_{yx} = a$ and $s_{zx} = a$.

	RTT	PTT	PDTT
Nodes	$N = N_x \cdot N_y = 2a^2$	$N = N_x \cdot N_y \cdot N_z = 2a^3$	$N = N_x \cdot N_y \cdot N_z = 2a^3$
Dimensions	$d = 2$	$d = 3$	$d = 3$
Radix	$R = 4$	$R = 6$	$R = 6$
Diameter	$D = a$	$D = \frac{3}{2}a$	$D = \frac{3}{2}a$
Average Distance	$\overline{d} \to \frac{2}{3}a$	$\overline{d} \to \frac{11}{12}a$	$\overline{d} \to \frac{7}{8}a$
Throughput	$\theta = \frac{6}{a}$	$\theta = \frac{6}{a}$	$\theta = \frac{48}{7a}$

Table 9. Topological properties of twisted tori.

3.1.3.2.5 SpiNNaker topology

The SpiNNaker topology [PFT07] is a triangular toroidal network, in other words, a torus network with additional diagonal links to add redundancy to the design. This redundancy is deliberately devised to be exploited for fault-tolerance as will be discussed in Chapter 6. A 4×4 instance of the SpiNNaker topology is represented in **Figure 14**d. The neighborhood description of the SpiNNaker topology is as follows (note that the diagonal axis is denoted as T):

$$X+: \quad x' = (x+1) \bmod N_x \qquad X-: \quad x' = (x-1) \bmod N_x$$

$$Y+: \quad y' = (y+1) \bmod N_y \qquad Y-: \quad y' = (y-1) \bmod N_y$$

$$T+: \begin{array}{l} x' = (x+1) \bmod N_x \\ y' = (y+1) \bmod N_y \end{array} \qquad T-: \begin{array}{l} x' = (x-1) \bmod N_x \\ y' = (y-1) \bmod N_y \end{array}$$

The topological properties of a square SpiNNaker topology with $N_x = N_y = n$ being n even, were derived in [PFT07] and are shown in **Table 10**.

Nodes	$N = n^2$
Dimensions	$d = 2$
Radix	$R = 6$
Diameter	$D = \left\lfloor \frac{2 \cdot n}{3} \right\rfloor$
Average Distance	$\overline{d} = \frac{\sum_{i=1}^{\frac{n}{2}} \left(6 \cdot i^2\right) - 3 \cdot \frac{n}{2} + \sum_{i=1}^{\left\lfloor \frac{n}{6} \right\rfloor} 2 \cdot \left(\frac{n}{2} + i\right) \cdot \max\left(1, 9 \cdot \left(\left\lfloor \frac{n}{6} \right\rfloor - i\right) + 3 \cdot \left(\frac{n}{2} \bmod 3\right)\right)}{n \cdot (n-1)}$
Throughput	$\theta = \frac{16}{n}$

Table 10. Topological properties of the SpiNNaker topology.

3.1.3.3 Indirect topologies

Indirect topologies comprise all topologies in which there are switching elements that are not directly attached to computing nodes. This includes multi-stage and multi-level topologies. In multi-stage interconnection networks

(MIN) all the traffic within the network flows in the same direction (usually represented from left to right) and the distance between every pair of nodes is the same: the number of stages of the network. In contrast, in multi-level topologies, the traffic must go up from the source to one minimal common ancestor of source and destination, and then down to the destination. This way, the distance between two nodes depends on the number of levels needed to reach a common ancestor. Two indirect, tree-based multilevel topologies are implemented in FSIN, the *k*-ary *n*-tree and a reduced version of this well-known topology that we have called *k*:*k*'-ary *n*-thin-tree.

In the descriptions that follow we assume that all switches used to build a given network have the same radix. Consequently, the upward ports of the topmost level of switches are unplugged. This assumption has advantages in terms of simplicity in the descriptions, and also provides scalability. In practical implementations, all ports of the highest switch level may be used as downward ports, eventually resulting in a network connecting more nodes. Alternatively, we may consider a single switch as an aggregation of lower radix *virtual* switches, which results in a smaller number of switches in the topmost stage of the system. In the graphical representations of the topologies (see **Figure 15**), boxes represent switches and lines represent links between them. Note that we neither show the compute nodes connected to the first-level switches and their links, nor the last-level of upward links (which, as we stated before, are unplugged). These elements are hidden for the sake of clarity.

We will use *n* to denote the number of levels in a network, and *N* to denote the number of compute nodes (leaves) attached to it. We will denote the total number of switches in a topology as *S*, and the number of switches at level *i* as S_i. The total number of links will be denoted as *L*. The switch radix will be denoted as *R*. We call the relation between the number of downward ports of a switch and the number of upward ports the *slimming factor*. For example, taking a look at the switches in the topology shown in **Figure 15**b, four ports are downward ports, linked to switches in the next lower level. The remaining two ports of each switch are upward ports that connect to switches in the next higher level; therefore the slimming factor is the relation between 4 and 2, that is, 2:1 (or simply 4:2).

In the topological descriptions that follow, we denote each switch port within the system as the level where the switch is, the position of the switch in that level, and the number of the port in that particular switch. We call the lowest level of switches (those attached to compute nodes) level 0; obviously, level *n*-1 is the one on the top of the tree. We number the switches in each level from left to right, starting from 0. Ports in a switch are denoted as upward (↑) or downward (↓), and numbered from left (0) to right. Thus, a port can be addressed as a 4-tuple <*level*, *switch*, *port*, *direction*>.

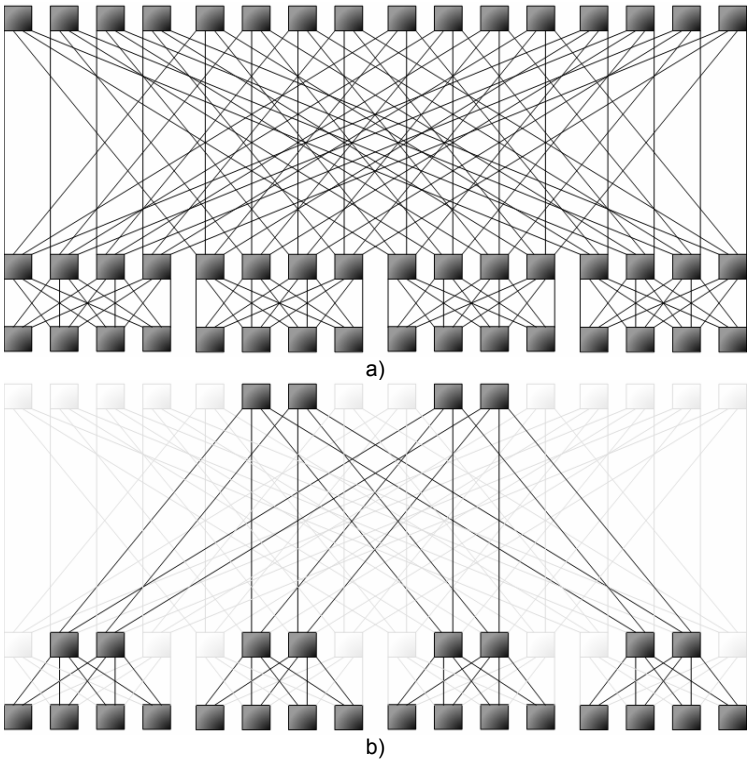

Figure 15. 64-node example networks of the multistage topologies. - a) 4-ary 3-tree (4,3-tree). b) 4:2-ary 3-thin-tree (4:2,3-tree).

Given two ports *P* and *P'*, they are linked (*P*↔*P'*) when there is a connection (link) between them. As links are full-duplex, in the expressions concerning linkage we avoid the redundancy of showing downward connections. We will call *l*, *s* and *p* the address components of a given port, and *l'*, *s'* and *p'* the address components of the port to which it is connected (its upper neighbor). Therefore,

$$< l, s, p, \uparrow > \leftrightarrow < l', s', p', \downarrow >$$

3.1.3.3.1 *k*-ary *n*-tree

In *k*-ary *n*-trees [PV97], *k* is half the radix of the switches—actually, the number of links going upward (or downward) from the switch—and *n* the number of levels. They can be seen as particular cases of the thin-trees (to be formally defined later), with a slimming factor 1:1. For simplicity, we will refer to them as *k*,*n*-trees.

A *k*,*n*-tree is typically built in a butterfly fashion between each two contiguous levels, **Figure 15**a shows a depiction of a 4,3-tree. The topological neighborhood description is as follows:

$$\forall l \in [0, n-1), \quad \forall s \in [0, k^{n-l}), \quad \forall p \in [0, k)$$

$$l' = l + 1$$

$$s' = \left(\left(p \cdot k^{l} \right) + \left(s \mod k^{l} \right) + \left(k^{l-1} \cdot \left\lfloor \frac{s}{k^{l}+1} \right\rfloor \right) \right) \mod k^{n-1}$$

$$p' = \left(\left\lfloor \frac{s}{k^{l}} \right\rfloor - \left(k^{l+1} \cdot \left\lfloor \frac{s}{k^{l}} \right\rfloor \right) \right) \mod k$$

$$\langle l, s, p, \uparrow \rangle \leftrightarrow \langle l', s', p', \downarrow \rangle$$

The main advantages of this topology are the high bisection bandwidth and the large number of routing alternatives for each pair of source and destination nodes—a path diversity that can be exploited via adaptive routing. Nevertheless, it might be expensive and complex to deploy, because of the large number of switches and links required.

3.1.3.3.2 *k*:*k'*-ary *n*-thin-tree

We define a thin-tree [NMR] as a *cut-down* version of a *k*-ary *n*-tree in which we apply a given slimming factor. It will be denoted as *k*:*k'*,*n*-tree, where *k* is the number of downward ports, *k'* is the number of upward ports and *n* is the number of levels. The slimming factor is, obviously, the ratio between *k* and *k'*. It is remarkable that *k* does not need to be a multiple of *k'* so that we can produce a thin-tree with arbitrary values of *k* and *k'*. As stated before, a *k*,*n*-tree is actually a *k*:*k*,*n*-tree.

A 4:2,3-tree is depicted in **Figure 15**b; some other examples of the thin-tree topology will be shown in Chapter 5. Note the shadowed switches and links, that represent those elements that would be removed from a complete 4,3-tree to form a 4:2,3-tree. In this topology the bisection bandwidth has been reduced, as well as the number of switches and links. For this reason, thin-trees are easier to deploy than regular trees and, if *k* and *n* values are kept, the radix of switches is smaller. In Chapter 5, we will compare the *k*-ary *n*-tree and the *k*:*k'*-ary *n*-thin-tree looking for the slimming factor that provides better cost-effectiveness.

The topological neighborhood relationships between ports in a thin-tree are described as follows:

$$\forall l \in [0, n-1), \quad \forall s \in \left[0, k \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{n-l} \right), \quad \forall p \in [0, k')$$

$$l' = l + 1$$

$$s' = \left(\left(p \cdot k'^{l} \right) + \left(s \mod k'^{l} \right) + \left\lfloor \frac{s}{k \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{l}} \right\rfloor \cdot k' \cdot \left\lfloor \frac{k}{k'} \right\rfloor^{l} \right)$$

$$p' = \left\lfloor \frac{s}{\left\lfloor \frac{k}{k'} \right\rfloor^{l}} \right\rfloor \mod k$$

$$\langle l, s, p, \uparrow \rangle \leftrightarrow \langle l', s', p', \downarrow \rangle$$

The relations between network parameters (*n*, *k* and *k'*), number of elements (*N*, *S*, S_i, *L*, *R*) and topological properties (Θ, *d*, *D*) for the two tree-like indirect topologies were derived in [NMR] and are summarized in **Table 11**.

	k,n-tree	k:k',n-thin-tree
Nodes	$N = k^n$	$N = k^n$
Switches	$S = n \cdot k^{n-1}$	$S = \sum_{i=0}^{n-1} k'^{(n-i)-1} \cdot k^i$
Switches per level $\forall i \in [0, n-1]$	$S_i = k^{n-1}$	$S_i = k^{(n-i)-1} \cdot k'^i$
Links	$L = S \cdot k$	$L = S \cdot k$
Switch radix	$R = 2 \cdot k$	$R = k + k'$
Theoretical throughput	$\Theta = 1$	$\Theta = \left(\frac{k'}{k}\right)^{n-1}$
Avg. Distance	$d = \frac{\sum_{i=0}^{n-1} 2 \cdot (i+1) \cdot (k-1) \cdot k^i}{N}$	$d = \frac{\sum_{i=0}^{n-1} 2 \cdot (i+1) \cdot (k-1) \cdot k^i}{N}$
Diameter	$D = 2 \cdot n$	$D = 2 \cdot n$

Table 11. Topological properties of the tree-based multistage topologies.

3.1.4 Output Data Generated by FSIN

Depending on the kind of research work that is being carried out with INSEE, different types of results are needed: throughput/delay analysis, channels utilization, time to deliver a workload, dropped packets, system evolution, etc. For this reason FSIN is able to capture and report a wide variety of statistics from the performed simulations. These statistics can be captured for the whole simulation or, in contrast, can be averaged from different time intervals. For debugging purposes, cycle-by-cycle information can be generated as well. As the execution of FSIN is deterministic (given a seed used to generate random numbers, if needed), a summary report of the input simulation parameters is printed in order to allow to repeat a given execution.

3.1.4.1 Run-Time Statistics

Run-time summaries of statistics can be obtained at pre-configured time intervals. This allows us to follow the evolution of the system. Note that applications usually pass through different stages of communication patterns and, therefore, of network usage. The statistics captured along the execution are the following (always related to the measurement time interval):

Injected and Accepted load: The average injection and consumption rates of network nodes, measured in phits/cycle/node.

Packets: The number of packets that have been injected, consumed and dropped due to different reasons (blocked in the head or tail of injection queues and/or dropped in transit).

Network capacity: The average number of packets in the queues of the system.

Distance: The (averaged) distance traversed by consumed packets.

Latency: Several latency-related measurements are captured including: average, standard deviation and maximum latencies. Note that latencies are measured since the moment at which packets are generated, as well as since the moment they are injected into the network.

3.1.4.2 Traffic Evolution

The wandering of packets and phits can be showed in order to have a better understanding of how the network evolves. Another interesting use of this mode is for debugging purposes. For **packet evolution**, the simulator will show in its output the next events:

- Packet generation.
- Packet injection.
- Header of the packet arriving to a node.
- Packet leaving a node.
- Header of the packet dropped.

- Tail of the packet dropped.
- Packet consumption.

For **Phit evolution**, the output of simulator will show the following events:

- Phit generated.
- Phit moved.
- Phit dropped.
- Phit consumed.

3.1.4.3 Final Report

Once the simulation is concluded, FSIN shows a larger report with a summary of the whole simulation. All the previously discussed figures are averaged for the whole simulation. In order to allow a better understanding of the results, the standard deviation and the maximum of all these figures are also shown, as well as the total simulation time, measured in cycles. The actual time required to perform the simulation is also provided.

If requested, FSIN can capture statistics to allow a more detailed understanding of the simulation. A histogram of the queue occupancy along the simulation can be shown, which is useful to understand which (and how) network queues were occupied. This can be used to detect unbalances in network utilization.

A map of the load managed by each (virtual) port of the system can also be generated. It is useful to find network bottlenecks, and also to understand the virtual channel management strategy used by the system.

If we focus on the distances traversed by packets, FSIN can show a histogram of distances, as well as a map showing pair to pair communication. Both histogram and map are generated for the distance distributions at the injection *and* at the consumption. If the captured data for injection and consumption are notably different, this can suggest that there are nodes that are suffering starvation. Note that the pair-to-pair map gives more information, but may require excessive memory if the system is large, as the memory requirements scales quadratically with the number of system nodes.

3.2 TrGen

TrGen—standing from TRaffic GENerator—is the module in charge of feeding the simulation with the desired kind of traffic. It interfaces with FSIN and passes the traffic in the form of network packets. TrGen is capable of generating purely synthetic traffic patterns, as well as of extracting communication patterns from traces or from the actual execution of parallel programs in a simulated computing environment (interfacing with Simics).

3.2.1 Synthetic Workloads

Our simulation environment provides a wide variety of synthetic workloads that can be used to measure the performance of the interconnection infrastructure. These workloads have different levels of fidelity to actual application workloads, both in terms of spatial and temporal/causal patterns.

3.2.1.1 Independent Traffic Sources

A widely accepted and used mechanism to feed simulations is the use of synthetic traffic patterns from independent sources [DT04]. This kind of workload allows tuning the injection rate of nodes, which try to inject at this rate, following a Bernoulli distribution. There is no causality among receptions and injections. In this category, the spatial traffic patterns supported by TrGen are the following:

- **Random**: When a packet is generated at a node (the source), the destination is randomly selected following a given probability distribution. The built-in modes are *uniform* (**UN**), in which all the nodes have the same probability of being selected as destination, and the non-uniform *hot spot* (**HS**) and *hot region* (**HR**), in which a given node or group of nodes, respectively, have higher probability of being selected as destination (therefore, increasing the risk of congestion in some regions of the network). Furthermore, TrGen can read and follow user-defined distributions which can be introduced as a *histogram* or as a *population*.
- **Local**: a specific case of random traffic in which the probability of selecting destination nodes decreases with the distance (so that most packets are sent to nearby nodes). This pattern is implemented for *k*-ary *n*-cubes only.

- **Distribution**: Distribution patterns send packets sequentially to each one of the remaining nodes. The initial destination node can be the next one in order of identifier (**dist**), or can be selected randomly (**rdist**).
- **Permutation**: Given a source node, the destination node is always the same, and is computed as a permutation of the source node identifier (generally bit permutations). The permutations accepted by TrGen are the following: Bit Complement (**BC**), Bit Reversal (**BR**), Bit Transpose (**BT**), Butterfly (**BU**), Perfect Shuffle (**PS**) and Tornado (**TO**). Their mathematical descriptions are shown in **Table 12**. Identifiers of source and destination are denoted as s and d, respectively. For bit permutations, l is the number of bits, s_i and d_i are the i-th bit of the source and the destination respectively. For the tornado permutation n_x is the number of nodes in the X and Y dimension of a 2D cube topology. $<s_x, s_y>$ and $<d_x, d_y>$ are the Cartesian coordinates in the 2D cube of the source and destination nodes, respectively.

Pattern	Destination	Example
Bit Complement	$\forall i: 0 \le i \le l-1, d_i = \sim s_i$	BC(11011000) = 00100111
Bit Reversal	$\forall i: 0 \le i \le l-1, d_i = s_{l-i-1}$	BR(11011000) = 00011011
Bit Transpose	$\forall i: 0 \le i \le l-1, d_i = s_{(i+l/2) \bmod l}$	BT(11011000) = 10001101
Butterfly	$d_{l-1} = s_0, \quad d_0 = s_{l-1}$	BU(11011000) = 01011001
Perfect Shuffle	$\forall i: 0 \le i \le l-1, d_i = s_{(i-1) \bmod l}$	PS(11011000) = 10110001
Tornado	$d_x = (\frac{n_x}{2} + s_x) \bmod n_x, \quad d_y = s_y$	$TO_{8\times8}$(<3,2>) = <7,2>

Table 12. Mathematical description of the permutation patterns implemented in TrGen.

3.2.1.2 Bursts

All the previously discussed spatial patterns can be used with *bursty* traffic sources [IMG05] instead of independent traffic sources. Bursty traffic emulates a barrier every a given number of packets b. To do so, nodes are allowed to inject b packets as fast as they can, and then stall until all packets of the burst have been injected and consumed by all the nodes. In this mode, the figure of merit to measure network performance is normally the time taken to deliver all the packets in a burst, being the faster the better. This burst-synchronized behavior avoids starvation of nodes, as all of them are allowed to inject the same amount of traffic into the network. Furthermore, it is remarkable that small values of b resemble tightly-coupled applications while large values of b resembles loosely-coupled applications.

3.2.2 Pseudo-Synthetic Workloads

One of the contributions of this dissertation is a set of synthetically generated traffic patterns that resemble the way scientific applications communicate. These workloads can be considered application micro-kernels as they mimic different communication patterns widely used on parallel applications. This mimicry is done both in terms of spatial pattern and causality. We distinguish three different approaches: implementations of collective operations, exploiting neighbor communication in virtual topologies and random generation of synchronized interchanges of messages. Further details can be found in [NMR08] and [NM09]. In the descriptions that follow, N denotes the number of communicating tasks and S the message length; both are provided as parameters. The algorithmic descriptions are defined from the point of view of each node, except when otherwise stated. *Send_to(d, M)* and *Wait_from(s, M)* functions do what their names suggest: put the node to send a message of length M to node d, or wait for a message of length M from a node s. When the workloads are split in several stages, these stages are denoted by t. Before entering a new stage each node needs to send and receive all the messages in the previous stage.

As stated in Chapter 2, application kernels provide a reasonable level of simulation accuracy at a reduced cost in terms of computing power. Moreover, they have the capability of generate workloads of thousands of communication nodes. For these reasons, application kernels will be used to feed some of the simulations carried out in Chapter 5 and Chapter 7.

3.2.2.1 Collectives

Most of the scientific parallel applications we have studied and used in our experimental work use MPI collectives to implement parts of their functionality: from scattering information to collecting results, or just to synchronize a group of processes. For example, the Fourier Transform, used in many applications, could be implemented by means of MPI_Alltoall() and MPI_Reduce() collectives.

In the following sections, we will provide algorithmic as well as graphical representations of these workloads. In the graphical descriptions of collectives, grey horizontal arrows represent tasks arranged from top downwards, with the time evolving from left to right. The small black arrows represent messages between tasks. Note that the arrowhead of such arrows represents a waiting state in the destination node.

3.2.2.1.1 Binary Tree

The Binary Tree pattern (**BI**) provides an efficient implementation of N-to-1 (MPI_Reduce, MPI_Gather, etc.) collective operations, used by some MPI implementations that make no assumption about the underlying network or the node placement strategy [Labo01]. In this pattern message, interchanges are performed in *O(N)* in number of messages and *O(S·log N)* in time. It starts with a message at odd-numbered nodes, and ends when a *root* node (in our model, node 0) has received the last message from all nodes whose identifier is a power of two (included 2^0=1). The spatial and causal pattern is defined algorithmically and graphically in **Figure 16**.

$\forall node\ in\ [0, N):$
$BinaryTree\ (node, S):$
  $for\ t\ in\ [\ 0, \log_2 N\ )$
    $if\ (node \bmod 2^{t+1}) == 0\ then$
      $Wait\_from(node + 2^t, S)$
    $elsif\ (node \bmod 2^t) == 0$
      $Send\_to(node - 2^t, S)$
    $endif$
  $endfor$

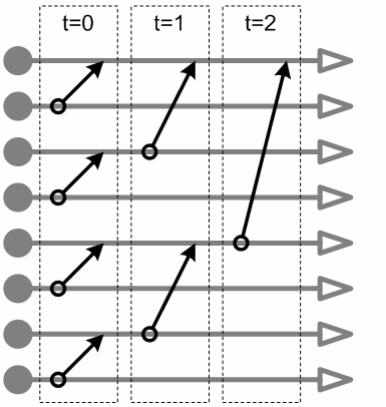

Figure 16. Definitions of the Binary Tree traffic pattern algorithmically (left) and graphically (right).

3.2.2.1.2 All-to-One

Contrarily to the binary tree, the All-to-One pattern (**A2O**) is a non-optimized implementation of 1-to-N collectives. In this pattern message interchanges are performed in *O(N)* in number of messages and *O(S·N)* in time. This pattern forces all the tasks in the group to send a message to a single root task, a situation that generates large contention around the root node. Note that this contention may spread through the network, leading to highly congested scenarios. The spatial and causal patterns of this workload are defined algorithmically and graphically in **Figure 17**.

$\forall node\ in\ [0, N):$
$All-to-One(node, S):$
  $if\ node == 0\ then$
    $for\ i\ in\ [1, N)$
      $Wait\_from(i, S)$
    $endfor$
  $else$
    $Send\_to(0, S)$
  $endif$

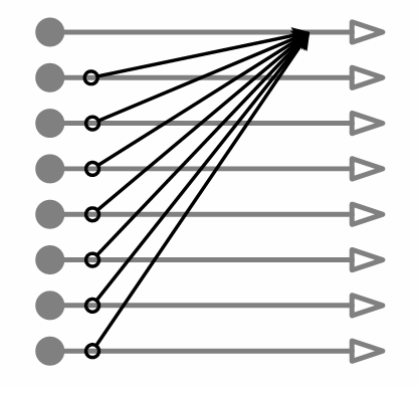

Figure 17. Definitions of the All-to-One pattern algorithmically (left) and graphically (right).

3.2.2.1.3 Inverse Binary Tree

The Inverse Binary Tree (**IB**) is a common implementation of 1-to-N (MPI_Bcast, MPI_Scatter, etc.) collectives [Labo01], with the same complexity of the BI pattern. IB starts with a single message in the *root* node and finishes when all the odd nodes receive a message. Readers may note that spatial and causal patterns are just the opposite of those of BI. IB is defined algorithmically and graphically in **Figure 18**.

$\forall node\ in\ [0, N):$
$InverseBinaryTree\ (node, S):$
  $for\ t\ in\ [\ 0, \log_2 N\ )$
    $if\ (node \bmod 2^{\log_2 N - t}) == 0\ then$
      $Send\_to(node - 2^{\log_2 N-(t+1)}, S)$
    $elsif\ (node \bmod 2^{\log_2 N-(t+1)}) == 0$
      $Wait\_from(node + 2^{\log_2 N-(t+1)}, S)$
    $endif$
  $endfor$

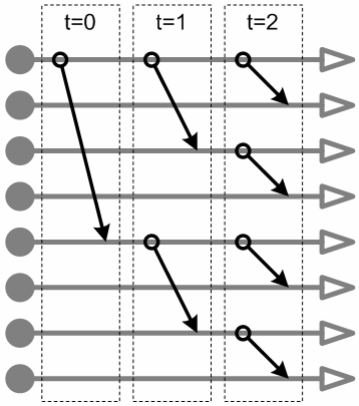

Figure 18. Algorithmic (left) and graphic (right) definitions of the Inverse Binary Tree traffic pattern.

3.2.2.1.4 One-to-All

The One-to-All pattern (**O2A**) is a non-optimized implementation of 1-to-N collectives. It is composed of a burst of messages sent from a root task to the rest of the tasks in the group. This generates contention at injection-level. O2A has the same complexity as the A2O pattern. The spatial and causal pattern is defined algorithmically and graphically in **Figure 19**.

$\forall node\ in\ [0, N):$
$One - to - All(node, S):$
  $if\ node == 0\ then$
    $for\ i\ in\ [1, N)$
      $Send\_to(i, S)$
    $endfor$
  $else$
    $Wait\_from(0, S)$
  $endif$

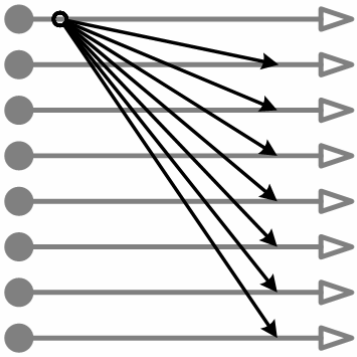

Figure 19. Definitions of the One-to-All pattern algorithmically (left) and graphically (right).

The reader should note that a usual mechanism to implement MPI_Barrier() is by concatenating a reduction and a broadcast, both with message length 0. This way, when the reduction finishes, the *root* knows that all tasks are synchronized, and starts a broadcast to let the other tasks know that all of them have reached the barrier, that is, they are synchronized. This can be implemented as a Binary Tree concatenated with an Inverse Binary Tree or, alternatively, as an all-to-one concatenated with a one-to-all.

3.2.2.1.5 Butterfly

The Butterfly pattern (**BU**), also known as *recursive doubling*, provides an efficient implementation of MPI N-to-N collectives (MPI_Alltoall, MPI_Allreduce, etc.) [TG03]. BU is *O(N log N)* in number of messages and *O(S·log N)* in time. BU pattern starts with a message at each node, and ends when all the messages are received. Algorithmic and graphical definitions of this pattern are shown in **Figure 20**.

$\forall node\ in\ [0, N)$:
Butterfly $(node, S)$:
  for t in $[\ 0, \log_2 N\ )$
    if $\left(\left\lfloor node/2^t \right\rfloor \bmod 2\right) == 0$ *then*
      $Send\_to(node + 2^t, S)$
      $Wait\_from(node + 2^t, S)$
    else
      $Send\_to(node - 2^t, S)$
      $Wait\_from(node - 2^t, S)$
    endif
  endfor

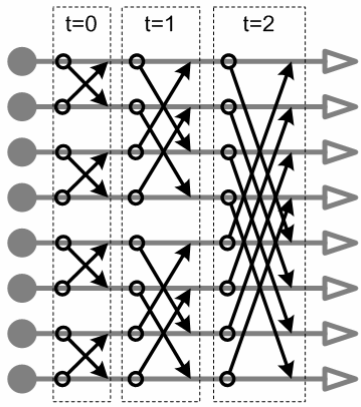

Figure 20. Algorithmic (left) and graphic (right) definitions of the Butterfly traffic pattern.

3.2.2.1.6 All-to-All

Finally the All-to-All pattern (**A2A**) is a non-optimized implementation of N-to-N collectives. In A2A, all the tasks have to communicate with the rest of the tasks in the group. In order to reduce contention at consumption, each task *n* sends messages in order starting from its next task, that is, to *n*+1, then to *n*+2, and so on. This way, if the network is crossbar-like, the messages can be delivered without any contention. In contrast, if the network is not crossbar-like, this pattern may generate a high level of resource contention. The data interchange performed in A2A scales in $O(N^2)$ in number of messages and $O(S \cdot N)$ in time. Algorithmic and graphical definitions of this pattern are shown in **Figure 21**.

$\forall node\ in\ [0, N)$:
$All - to - All(node, S)$:
  for i in $[1, N)$
    $Send\_to(node + i \bmod N, S)$
  endfor
  for i in $[1, N)$
    $Wait\_from(node + i \bmod N, S)$
  endfor

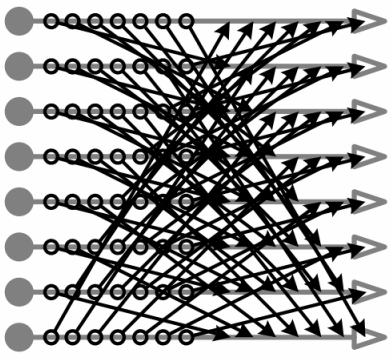

Figure 21. Definitions of the All-to-All pattern algorithmically (left) and graphically (right).

3.2.2.2 Virtual Topologies

This branch of micro-kernels reproduce the data interchanges performed in applications that rely on virtual topologies, such as the 2D meshes commonly used in matrix calculus. We have generalized these communication models in such a way that they are available for several dimensions (*D*), which along with the number of nodes, *N*, and the message length, *S*, are the parameters accepted by these workloads. Note that the virtual topology is completely independent of the actual network topology, because it is just a way to arrange MPI processes. It is remarkable that these patterns can be easily extended to other cube-like topologies as the torus or the hypercube. In the graphical descriptions, grey boxes represent nodes in a 2D mesh arranged by their Cartesian coordinates, while arrows represent the messages among them.

3.2.2.2.1 Wave-Front

The 2D and 3D wave-front patterns (**2W** and **3W**) perform a diagonal sweep from the first node to the last one in MPI virtual square (or cubic) meshes. The simulation of these patterns starts with two (three for 3W) messages in node 0, and ends with the completion of the sweep in the last node of the network. These patterns do not impose a very heavy load on the network (note that there are only a few nodes injecting at once), but create some contention in the destination nodes because they have to receive data from several neighbors. We can observe this pattern in

applications implementing the Symmetric Successive Over-Relaxation (SSOR) algorithm [BFV93], used to solve sparse, triangular linear systems. For instance, LU from the well-know NAS Parallel Benchmarks [NAS], perform several consecutive bi-dimensional sweeps of short messages. Algorithmic and graphical definitions are shown in **Figure 22**.

```
∀node in [0, N):
Wavefront (node, S, D):
  for d in [0, D)
    Wait _ from(Neighbor(node, d, '−'), S)
  end for
  for d in [0, D)
    Send _ to(Neighbor(node, d, '+'), S)
  end for
```

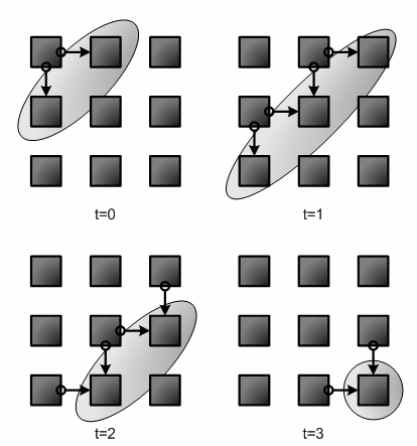

Figure 22. Algorithmic (left) and graphic (right) definitions of the Wave-front traffic pattern.

3.2.2.2.2 Mesh Distribution

The 2D and 3D mesh patterns (**2M**, **3M**) perform data distributions in MPI virtual square (or cubic) meshes from every node to all its neighbors; after that, each node waits for the reception of all messages from its neighbors. Simulation starts with all nodes injecting one message per direction (2-4 for 2M, 3-6 for 3M), and ends when all the messages have arrived to their destinations. These workloads impose a very heavy load on the network, because all nodes inject simultaneously several messages at once, before stopping to wait for the receptions. Algorithmic and graphical definitions are shown in **Figure 23**. These patterns can be observed in scientific applications using finite difference methods [AN99]. In some of these applications, the spatial pattern also includes communication in the positive diagonal: each node communicates with the nodes located at ±1 in all dimensions. Furthermore, if the virtual topology is a torus, then the nodes located in the boundaries of the virtual topology communicate between them.

```
∀node in [0, N):
Distribution (node, S, D):
  for d in [0, D)
    for w in {'+', '−'}
      Send _ to(Neighbor(node, d, w), S)
    end for
  end for
  for d in [0, D)
    for w in {'+', '−'}
      Wait _ from(Neighbor(node, d, w), S)
    end for
  end for
```

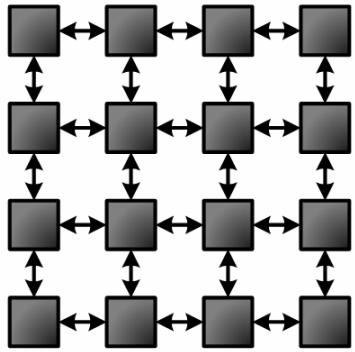

Figure 23. Algorithmic (left) and graphic (right) definitions of the Mesh Distribution traffic pattern.

3.2.2.2.3 Mesh Direction Distribution

The 2D and 3D distribution patterns (**2D**, **3D**) perform the same data distributions of 2M and 3M in virtual square (or cubic) meshes. However in these patterns data distributions are arranged in several steps, one for each direction. Simulation starts with all nodes injecting one message to their neighbors in the positive first dimension (X+). After that, each node wait for the message from its neighbor, and then sends it to the neighbor in the negative first dimension (X-), and so on for all the remaining directions. Simulation will end when all messages in the last direction (and obviously in all the other directions) have been delivered. The spatial and causal pattern is defined

algorithmically and graphically in **Figure 24**. These patterns are also common in finite difference methods [AN99]. They can be further refined by adding other synchronization models such as arranging the distribution by dimensions. Just as a curiosity, the reader should note that the formerly explained butterfly pattern can be seen as a dimension distribution in a (virtual) hypercube.

$\forall node\ in\ [0, N):$
$Direction\_Distribution\ (node, S, D):$
  $for\ d\ in\ [0, D)$
    $for\ w\ in\ \{'+', '-'\}$
      $Send\_to(Neighbor(node, d, w), S)$
      $Wait\_from(Neighbor(node, d, w), S)$
    $end\ for$
  $end\ for$

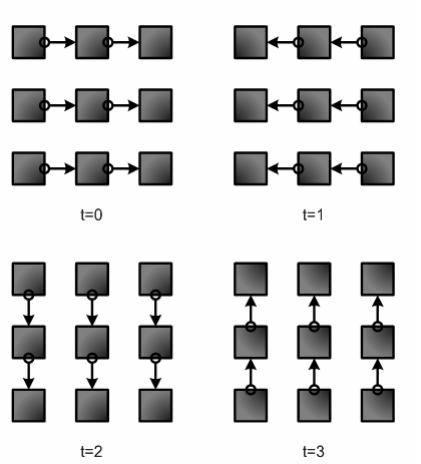

Figure 24. Algorithmic (left) and graphic (right) definitions of the Mesh Direction Distribution pattern.

3.2.2.3 Synchronized Random

The synchronized random pattern does not form part of any application but is a refinement of the bursty traffic model. This pattern supports point-to-point synchronization between nodes and also modeling different levels of application coupling. This synchronized, random workload (**SR**) is algorithmically defined in **Figure 25**, which also shows an example of its behavior. It accepts four parameters: number of tasks (*N*), message length (*S*), total number of messages (*M*) and number of messages per wave (*W*). In short, this workload generator creates waves of messages, with causal dependencies among them. A wave is defined as a set of message emissions that can be performed by tasks before waiting for the corresponding receptions. Therefore, the algorithmic definition is not done from the point of view of each node, but from the whole workload. The source and destination of the messages are selected uniformly in [0, *N*). If *S* and *W* are large enough, this workload may lead to highly congested states.

$RndComm(N, S, M, W):$
  $for\ t\ in\ [0, M)$
    $src = Rnd(N)$
    $dst = Rnd(N)$
    $Send(src, dst, S)$
    $Store(src, dst, S)$
    $if\ B\ messages\ Stored$
      $for\ m\ in\ Stored$
        $Wait(m.src, m.dst, m.S)$
      $endfor$
    $endif$
  $endfor$

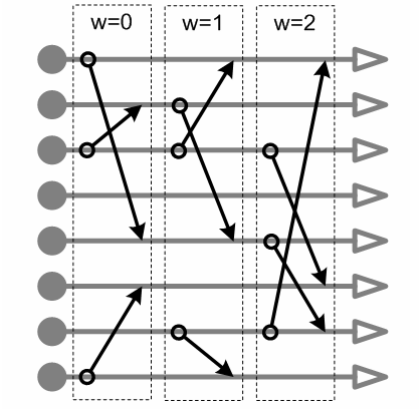

Figure 25. Algorithmic definition of the synchronized random pattern (left) and an example with 8 nodes, 9 messages in total, and waves of 3 messages (right).

If we reach the extreme scenario in which *W* and *M* are equal, all messages are sent in a single burst and, therefore, no synchronization is involved. Then, we are emulating the RandomAccess benchmark, also known as giga-update-per-second (**GUPS**) from the High Performance Computing Challenge Benchmark Suite [LDK05]. We want to remark that random uniform traffic is, on the one hand, a network stress test, because it does not take advantage of communication locality; but, on the other hand, it can be considered a best-case scenario in the sense

that, as traffic is evenly distributed through the network, it does not introduce bottlenecks. Reader should note that we could add another degree of freedom to this workload by randomly selecting the message length in each communication operation.

3.2.3 Application-Guided Workloads

Synthetic sources provide very useful insights into a network's potential. However, obtained performance metrics can be unrealistic as applications use more sophisticated communication patterns than synthetic models. For this reason TrGen can also use traces from applications to perform trace-driven simulation, and can also interact with Simics to perform full-system simulation. This subsection is devoted to discuss these execution modes.

3.2.3.1 Traces

Another important contribution is the use of traces obtained from the execution of actual applications to perform simulations that follows the traffic patterns explicitly defined in the traces, in terms of spatial distributions, packet sizes and causal relationships.

We use a modified version of MPICH [MPI], one of the most popular implementations of MPI, to obtain trace files usable with TrGen. MPICH includes an easy-to-use mechanism to obtain trace files from running applications. However, these traces are not useful for our purposes because collective operations (such as barriers, broadcasts, reductions, etc.) appear as such in the trace files, that is, the actual interchange of messages necessary to implement those operations in networks without native support for collective operations is not reflected. Internally, MPICH implements collective operations using point-to-point operations (when no better alternative is available). We modified MPICH in order to make those point-to-point operations visible, registering them in the trace files along with the corresponding collective operation. The intervals of time between MPI operations are considered as computation intervals. A cpu scale factor can be applied to these intervals in order to simulate faster or slower cpus that those of the system where traces were captured.

Trace files are slightly pre-processed before using them with TrGen. Only a few, relevant fields are selected (event type, source node, destination node, message size and tag) and organized in a simplified format more suitable for TrGen. It would be possible to use this format to feed simulations with traces obtained from sources different to MPICH, a network sniffer being a good example, just building the right pre-processor. To reproduce the causal relationships between events in the trace files, TrGen requires a special data structure to store past and future events, which is shown in **Figure 26**. Each node of the simulated applications has an event queue, which is fed from the trace file. A packet is sent to the network when an **S** (send) event is in the queue's head. If an **R** (receive) event is in the head, it is necessary to access the pending notifications queue to check if the expected event has happened already; otherwise, processing of events is blocked until the network notifies the awaited reception. The pending notifications queue at each node, thus, stores reception events that arrive before the application requests them, and it is a crucial element to keep event causality. The complete process of trace-driven simulation is as follows:

1) Enqueue in each node's event queue all the events it has to execute.
2) Initialize the pending notifications list as an empty list.
3) Nodes sequentially execute the events in their event queue.
    a) If the first event is a send, remove the event and inject the corresponding message into the network.
    b) If it is a reception, check if a corresponding message (matching origin, destination, tag and size) is in the pending notification list. If it is there, remove both entries. Otherwise, keep in this state until the required message is received by the node and is accordingly found in such list.
    c) If it is a computation event, put the node on hold for the required period of time. Note that the period of time captured into the trace may be scaled by a speed-up factor in order to simulate faster (or slower) processors.
4) When the simulator delivers a message, put it in the pending notifications list.

An example of this procedure is depicted in **Figure 26**, note that **R** represents a reception and **S** represents a send; computation events (**C**) are not considered for the sake of simplicity. In the figure, node 0 cannot advance, because it is waiting for a message from node 1, even if a message from node 15 has been already received. In contrast, node 15 can advance because the required message from node 1 has been delivered. This mechanism reproduces the actual way messages were interleaved when running the application, complying with the causal order between a reception and the subsequent sends it may trigger.

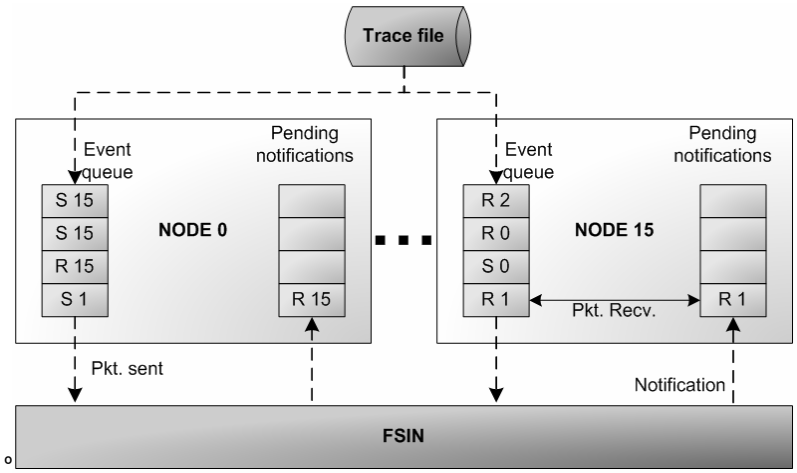

Figure 26. Data structure used to maintain causality in the trace-driven simulation.

Traces obtained from one system are often used to evaluate, via simulation, the performance potential of other system. However, this approach has some drawbacks. In the context of interconnection network design and evaluation, traces obtained with the same collection of nodes running a parallel application with two different interconnection networks A and B may be different, because properties of A and B differ and those properties have an influence on the way nodes interchange messages. For this reason, performance results obtained with traces may not be accurate [GH93]. A thorough discussion of the design, limitations and issues of the trace-driven simulation in our INSEE environment was carried out in [MNR09].

When dealing with application traffic in the remaining of this dissertation, we will be using the trace-driven engine, as it provides a good balance between realism of the simulation and required computing resources. More exactly, this traffic model will be used in the evaluations performed in Chapter 4, Chapter 5 and Chapter 7.

3.2.3.2 Interaction with Simics

TrGen can interface with Simics [MCE02] to perform a full system simulation. Simics is in charge of simulating a cluster of multiprocessors. In our experimental environment, each Simics instance is executed in a different computer and can simulate up to 8 compute nodes (limited by the available RAM of our machines). A single instance of INSEE simulates the network that interconnects all the simulated nodes. In order to perform a correct simulation of the parallel system, a set of interfacing modules were implemented in INSEE and SIMICS. These modules are in charge of the following functions:

- Transfer application traffic from Simics instances to FSIN and vice versa.
- Synchronization among all the elements taking part in the simulation (FSIN and the collection of Simics instances).

A depiction of all the elements involved in the execution driven simulation are depicted in **Figure 27**.

Every simulated computing node has a simulated Ethernet network interface card which is instrumented to put the messages into the traffic Manager of its Simics instance. The traffic Manager sends meta-information about the messages to TrGen and is in charge of sending the complete message to the corresponding Simics instance and node, once it has reached its destination in the simulated network.

It is remarkable that Simics gives support to the synchronized execution of all the nodes within a given instance. As intra-instance synchronization is maintained, we need to add a synchronization client to keep synchronization among all Simics instances as well as among them and FSIN. With this purpose INSEE is complemented with a Synchronization server which keeps the whole simulation running at the same pace. TrGen keeps track of all the messages of the simulation. The full system simulation process is as follows:

The interchange of application traffic is managed by TrGen itself which, as stated before, stores meta-information of each message taking part in the simulation. TrGen is also in charge of injecting traffic into FSIN and of monitoring it to recognize when a message has arrived to its destination. Once a message arrives to its destination in FSIN, TrGen sends a confirmation to the source Simics instance in order to send the complete message to the destination instance and node. The Traffic manager inside each Simics instance is in charge of storing the complete message and sending it to the destination node. When the Traffic manager receives a message to a simulated node it injects the message into the NIC module of the proper node.

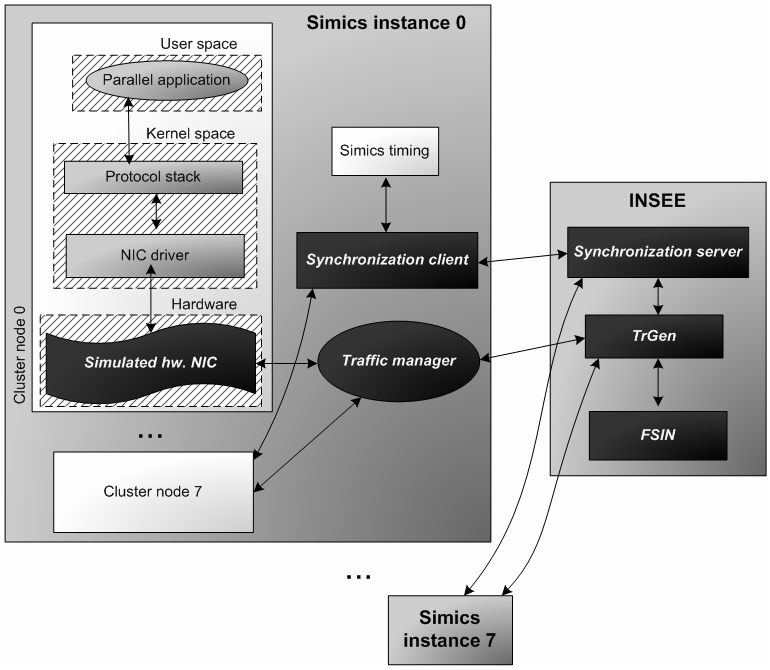

Figure 27. Elements of our full-system simulation environment that simulates an MPI application running on top of a FSIN (simulated) network. Black elements belong to INSEE. In this case 8 Simics instances simulate 8 cluster nodes each, coordinating with a single instance of FSIN.

Synchronization is fairly more complex as a two-level mechanism is implemented in order to synchronize the multiple elements taking part on the simulation. On the one hand, all the nodes simulated within a Simics instance are executed in a step-wise fashion: nodes go to execution sequentially for a given number of CPU cycles (*slice*) and, once the one in execution finishes its slice, the next node enters into execution for the same amount of time. This mechanism is part of Simics. A Synchronization Client implemented inside each instance of Simics is in charge of sending a message to TrGen to make it know that that instance has finished its slice; this is carried out when all the simulated nodes finish their corresponding slices. Once this message is sent, the synchronization client suspends the execution until a message from TrGen is received allowing the execution of a new slice. The Synchronization Server, implemented within TrGen, waits until all Simics instances have finished their slices and then makes FSIN run for a period of time equivalent to the Simics slice. After FSIN finishes its slice, TrGen sends a multicast message to the Simics instances, allowing them to resume their execution. The ratio between Simics and FSIN slices (one measured in Simics cycles and the other one in FSIN cycles) determines the bandwidth of the simulated network.

Further information about performing execution-driven simulation can be found in [RMN09]. This work shows a thorough description of the full system simulation in INSEE, and also discusses several different approaches that may be followed in order to perform an execution driven simulation. Emphasis is put on where to capture the traffic in the simulated node, and also in synchronization, showing the need to fine tune the value of the slices in order to obtain a balance between simulation accuracy and execution time. Furthermore it discusses some issues encountered by our group when setting up the simulation environment and carrying out performance-related experiments.

This traffic generation model will not be used within this dissertation because it requires a high amount of computing resources to perform simulation.

3.2.3.3 Task Placement

An interesting consideration to take into account when launching parallel applications is the allocation of computing resources to each application tasks, something that we will explore in Chapter 7. Parallel applications are usually implemented following spatial distributions that can be exploited effectively by means of an adequate allocation onto the network (actually, compute) nodes. Furthermore, supercomputing sites are often used by several users that share machine resources, with several applications being executed concurrently.

When dealing with application (or application-inspired) traffic, INSEE has the ability of arrange the tasks of a workload following different policies. Furthermore it can execute several application instances concurrently to measure the effects of the interactions among them. Several policies to arrange instances and tasks are implemented; note that some of them depend on the topology of the interconnection network:

- **Consecutive**/**Row**. Tasks and applications are placed consecutively—note that in the case of cube topologies this means filling rows in order. For example if we have two applications of n nodes each, they will be placed from nodes 0 to n-1 and from nodes n to 2n-1, correspondingly, being their tasks arranged in order. Examples of these placements are plotted in **Figure 28**a for the cubes and **Figure 29**a for the trees.
- **Shift**: Is the same policy than Consecutive, but adding a given shift, s, to the allocated node, that is, task t is located in node t+s. Examples of these placements are plotted in **Figure 28**b for the cubes and **Figure 29**b for the trees.
- **Shuffle**: This policy only makes sense in those topologies that have several computing nodes attached to each networking element. In our environment, this applies to tree-like topologies. Tasks are placed one in each switching element, in other words, if in each networking element there are p ports connected to computing nodes, nodes are placed in nodes 0, p, 2p, and so on. An example of this placement is plotted in **Figure 29**c.
- **Column**: This policy only makes sense in cube-like topologies with, at least, two dimensions. Assignment is done selecting the nodes by columns, which can be seen as partitioning the network in rectangular sub-networks, taller than wide. An example of this placement is plotted in **Figure 28**c.
- **Quadrant**: This policy only makes sense in cube-like topologies with, at least, two dimensions. When using several application instances, we can partition the network in perfect squares (or cubes). Within each square an application is placed following consecutive order. An example of this placement is plotted in **Figure 28**d.
- **Random**: All the tasks are randomly placed along the network independently of their application. To do so, we generate a random permutation of the network nodes.
- **File**: INSEE can read the placement information from a file. The format of this file is very simple: *node task application,* meaning that the task *task* from the application *application* is placed in the node *node*.

Some examples of the different placement strategies available in direct networks are depicted in **Figure 28**. In the showed depictions, four applications composed by four nodes each share the network (these applications are represented by white, grey, black and crossed, correspondingly). Similarly, some example depictions of the different strategies for tree-like topologies are show in **Figure 29**. Note that random and file placement are not depicted for the sake of simplicity. Some of these placement strategies will be used in Chapter 7 to illustrate the impact that the placement may have over the performance of parallel applications.

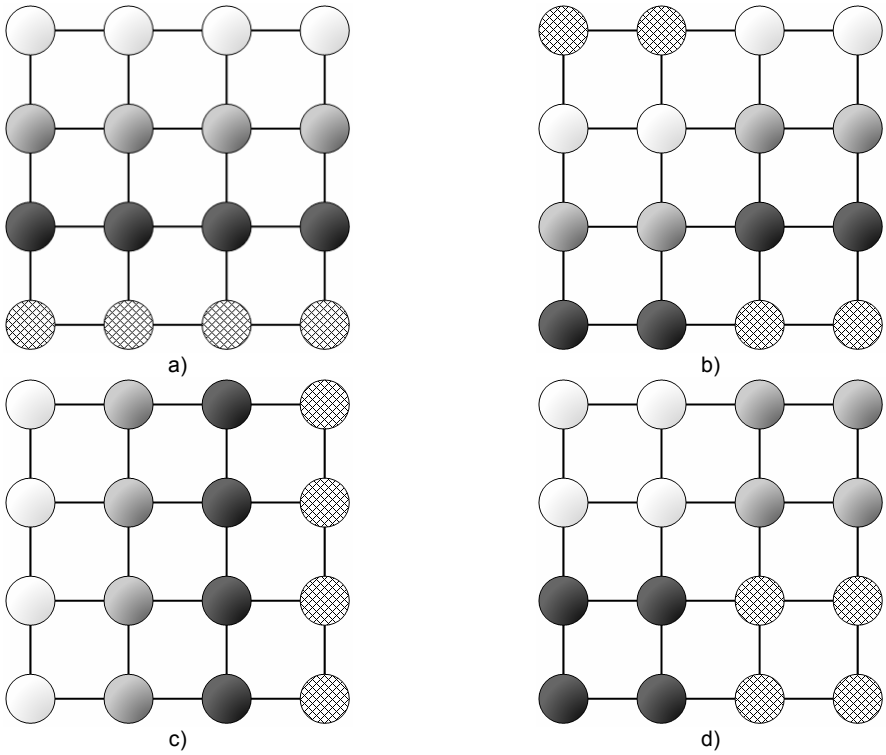

Figure 28. Placement strategies for direct topologies. a) row. b) shift 2. c) column. d) quadrant.

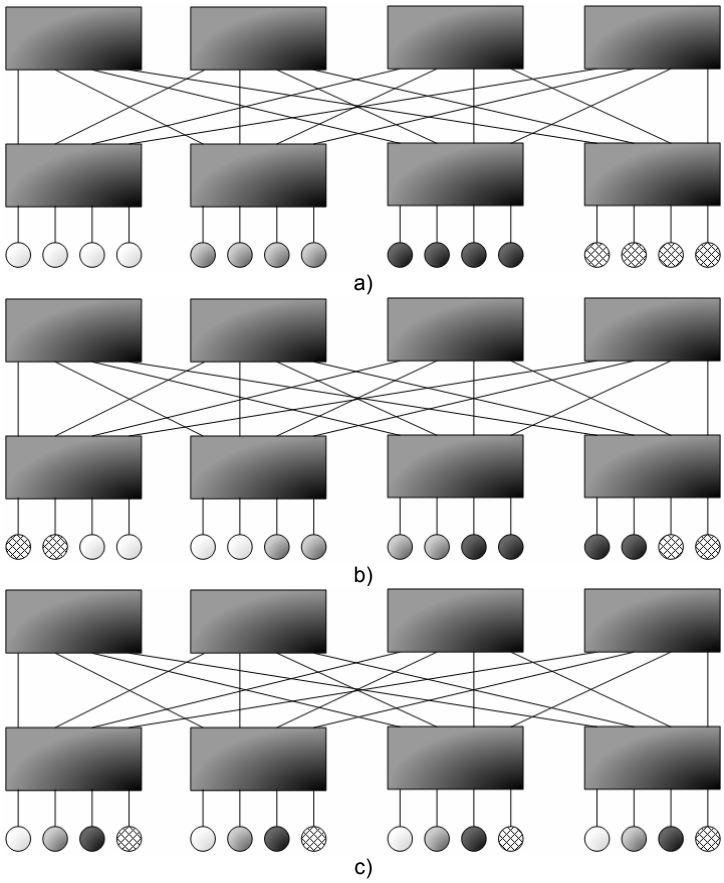

Figure 29. Placement strategies for tree-like topologies. a) consecutive. b) shift 2. c) shuffle.

3.3 Limitations of INSEE

Although INSEE has proven to be a useful tool to evaluate interconnection systems, it has some limitations that we should not forget. Some of these limitations are easily solvable, but may require considerable effort in terms of implementation time. Additionally, some modifications may increase excessively memory requirements and/or execution times, something that would go against the philosophy of INSEE. Still, in this section we discuss some modifications that could increase the usefulness of this environment.

As stated before, one of the limitations of INSEE is that, as it has a time-driven engine, it is not suitable to simulate workloads with long computation times between communication events, because the time required to perform the simulation will be excessive. We are currently developing an event-driven engine for INSEE which would accelerate the simulation of this class of workloads.

Another limitation is the simplicity of the node model, simulated as a simple traffic generator and consumer, without any internal structure. This is because we are interested in the behavior of the interconnection network, and this model is enough for our purposes. Another limitation closely related to this one is that INSEE only allows allocating a single task per node, which is not very realistic because, in current systems, nodes attached to the network are actually multiprocessors. The simulation of multiprocessor nodes can be implemented in TrGen just by allowing several application tasks (from Simics, traces or application kernels) to share a FSIN node. However it will require making some implementation decisions: how to arbitrate the injection infrastructure and how to perform intra-node communication. The implementation of these decisions may require increasing excessively the complexity of the node model.

When performing trace-driven simulations, the causality of the messages is maintained but the MPI semantics are not followed precisely by TrGen. For instance, all messages sends are treated equally, regardless of them being immediate, *rendevouz,* one-sided, etc. Implementing MPI semantics accurately requires increasing the complexity of the node which, as stated before, is not desirable. Furthermore, MPI has very complex (and precise) semantics, and implementing every operation supported by the standard would require huge efforts in terms of learning and understanding these semantics and coding them within TrGen.

A remarkable limitation of INSEE is that FSIN does not include detailed models of some *state-of-the-art* networking technologies such as InfiniBand [Shan02], Myrinet [BCF95] or QsNet [PFH02]. The reason for this is

two-folded. On the one hand, implementing these technologies would require a deep knowledge of every detail of them, as well as a non-negligible effort in terms of coding. On the other hand, given the complexity of these technologies, it would not be possible to perform large scale simulations because of the associated requirements in terms of computing resources.

A less significant limitation of INSEE is that it does not allow for parallel or distributed simulation. Although FSIN could be easily extended in this line—note that its time-driven engine is split into two separate loops and, in each loop, iterations are completely independent among them—this has not been done for two simple reasons. The first one is that the low footprint of INSEE allows for *large-scale* simulations in a single *off-the-shelf* computer, and therefore we did not *need* to parallelize it, even when we have parallel computers. This leads to the second reason: in our evaluations we usually require the execution of *many* experiments, modifying parameters such as network sizes, workloads, placements, random seeds… without any dependency between the different runs. The utilization of a cluster as a high-throughput computing resource is enough for our purpose.

3.4 Conclusions

In this Chapter we have described the simulation environment that we will use in the following Chapters to carry out the performance evaluation of several systems. The functionalities of the simulator have been explained in detail: the wide variety of network topologies, the built-in router models, the generated output data and the traffic generation modes, etc. Furthermore, we have shown some details about how simulation and traffic generation are implemented in the INSEE modules. We have also described some other additional modules required to perform application-driven simulations: a modification of MPICH to generate usable traces of applications, and some modules in Simics to perform full system simulation. Finally, we have reviewed some limitations of the INSEE environment.

Chapter 4. Case Study: Mixed-Radix Twisted Torus

The twisted torus topology was presented and studied in [CMV07] and [CMV]. The main contributions of these papers were the following. The topological characteristics of the twisted torus topology were derived and validated by means of simulation with INSEE. Optimal (shortest-path) routing functions were presented and proved for 2D and 3D topologies. Finally, authors presented folding/wiring functions for this class of networks. The discussion of the properties of this class of networks falls outside the scope of this dissertation. However, we want to point out the expression derived to compute the theoretical throughput from the bisection bandwidth. An initial analysis that was carried out following the method stated in [DT04] resulted incorrect, because of the characteristics of the twisted torus topology and the proposed optimal routing.

In this chapter we will discuss how simulation unveiled a pitfall of the preliminary analytical expression of the bisection bandwidth, and how it helped in deriving the correct one. Furthermore, in order to support the advantages of this topology, we will compare the performance of parallel applications in small instances of mixed-radix standard and twisted torus. For the sake of simplicity, we will limit our discussion to rectangular $2a \times a$ two-dimensional networks, for which a skew of a was proved to be optimal. We want to remark, though, that extending the discussion to 3D networks is straightforward.

4.1 Justification of the Twisted Torus Topology

Many parallel computers using direct interconnection networks have been designed and commercialized in the last decades, being meshes, tori and hypercubes the most popular topologies. Nowadays, hypercubes seem to decline in favor of lower degree networks such as two-dimensional and three-dimensional tori and meshes because of their inherent difficulty to scale. Different machines, such as the Alpha 21364-based HP GS1280 [Cvet03] and the Cray X1E vector computer [DVW05] used 2D tori. Others, such as the Cray T3E [ABG97], the Cray XT3 [VAD06] or the IBM BlueGene/L [AAA02] used 3D tori.

Usually, 2D topologies arrange their N nodes in a square Mesh with $\sqrt{N}$ nodes per side. In the torus, the boundaries of the topology are connected pair wise by means of $2\sqrt{N}$ wraparound links. As suggested in [Agar91], networks having above, more or less, a thousand nodes should be arranged in 3D topologies, being a cubic 3D torus of side $\sqrt[3]{N}$ the most desirable solution. Nevertheless, each dimension of a torus may have different number of nodes leading to rectangular and prismatic topologies for two and three dimensions, respectively. These topologies, denoted as mixed-radix networks in [DT04], are often built for practical reasons such as packaging, modularity and scalability. For instance, the HP GS1280 employs a 4×2 2D rectangular torus network [Cvet03], and the IBM BlueGene/L a 64×32×32 3D prismatic one [AAA02]. However, mixed-radix tori have two important drawbacks: first, they are no longer edge symmetric, and second, the distance-related network parameters (diameter and average distance) are far from the optimal values of square and cubic topologies. The edge asymmetry leads to a lack of balance in the utilization of resources, as for many traffic patterns the load on the longer dimensions is higher than the load on the shorter ones [BCC03] and, consequently, links in the longer dimensions become network bottlenecks. In addition, maximum and average packet delays are relatively long as they depend on the poor values of diameter and average distance exhibited by these networks.

4.2 Experimental Set-Up

For this study, the router model is similar to the one implemented in the IBM BlueGene/L: Virtual Cut-through *switching* [KK79] and Bubble flow control (which guarantees *deadlock avoidance* [PIB01]), with an oblivious DOR virtual channel plus another two fully adaptive virtual channels. Each virtual channel, independently of being adaptive or Escape, has a transit queue with room for 4 packets. The arbitration of the output ports is done in round robin. Routers have a single injection queue able to store up to 4 packets. We assume that the consumption interface is wide enough to allow simultaneous consumption of several packets arriving from different ports. The routers of the BlueGene family of supercomputers implement an in-transit priority congestion control mechanism [BCC03] which is also included in our router model. Finally, in our experiments packets have a fixed length of 16 phits of 4 bytes each.

First, we use independent traffic sources to assess how the topological advantages of the twisted torus lead to better performance under random uniform traffic. In this case, packets are distributed evenly along the whole network, so that no persistent bottlenecks are generated. Performance data is shown in accepted load (throughput) versus provided load. Provided load is an input parameter, and accepted load is a value, measured using simulation, that indicates how many phits/cycle/node the network is able to deliver successfully. When the network is not saturated, both values are almost identical. However, when some of the routers (or all of them) saturate, actual throughput may be much smaller than provided load.

We also evaluate the networks using traces of real applications, obtained running several of the NAS Parallel Benchmarks (NPB) on an actual multicomputer. The study is restricted to those benchmarks of the NPB suite that can be mapped onto rectangular networks and, among them, to those that make intensive use of the interconnection network: Conjugate Gradient (**CG**), Integer Sort (**IS**), Fourier Transform (**FT**), LU solver (**LU**) and MultiGrid (**MG**). Some limitations of our experimental framework do not allow us to generate traces for applications larger than 128 tasks. Hence, we restrict our study to small-sized configurations—for which we generated traces of Class A of the enumerated benchmarks using 32 and 128 nodes. Performance is measured in terms of the number of cycles required to fully inject and deliver each of the benchmarks. In some real applications, the mapping of tasks to nodes has a great impact on performance; this issue will be discussed in Chapter 7. In the experiments performed in this Chapter, we have tested two simple mappings which are in use in two actual systems implemented around rectangular torus: consecutive allocation (task *i* goes to node *i*) similar to the one implemented in the systems of the IBM's BlueGene family [ADG05], and random, which is the behavior of policy in use at the Cray's XT3 and XT4 [Ansa]. For the random mapping, graphs show the average of 20 simulation runs together with the 95% confidence intervals.

4.3 Network Throughput under Uniform Traffic

In this Section we analyze the performance of these mixed-radix networks in respect to their bisection bandwidth (B_B). The bisection bandwidth is the bandwidth between two equal partitions of the network when a minimum cut is considered. It provides a limit for the maximum random traffic injection rate that can be accepted by the network, denoted as Θ. According to the stated in [DT04], in networks with uniform channel bandwidth as the ones considered here, the bisection bandwidth is proportional to the channel count of the minimum cut of the network (B_C). We will show that, while this holds for the standard rectangular torus, for the twisted torus any optimal (shortest-path) routing makes us to consider an effective bisection bandwidth that is lower than the one calculated with the minimum cut of the network.

For a $2a \times a$ rectangular torus (**RT**) as the one depicted in **Figure 30**, a minimum cut of the network in two halves is the one that divides it in two adjacent $a \times a$ squares. In the figure, this corresponds with the cut that leaves columns 0 to 3 on one side and 4 to 7 on the other side. The cut is crossed by $2a$ horizontal links (a internal and a wraparound links). This means that $2a$ phits can traverse these links per cycle in each direction. Under random uniform traffic, the $N/2$ nodes in one half of the network generate packets with probability 0.5 to the other half; therefore a maximum of $\frac{2a}{N/4} = \frac{8a}{N}$ phits/cycle can be generated by each node. The B_B provides an upper bound on the maximum injection rate per node under uniform traffic. For a rectangular torus with $N = 2a^2$, the theoretical maximum injection rate per node is $\Theta = \frac{8a}{2a^2} = \frac{4}{a}$ phits/cycle.

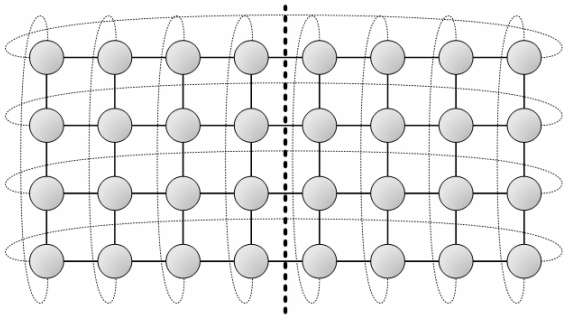

Figure 30. Depiction of an 8×4 rectangular torus showing the bisection cut.

Now, we consider a $2a \times a$ rectangular twisted torus (**RTT**) such as the one depicted in **Figure 31**. The cut separating columns 3 and 4 is again a minimal cut. However, in this case, it comprises $4a$ links, twice as much as in the RT network. If we apply the previous definition, we would find a maximum B_B of $\frac{4a}{N/4} = \frac{16a}{N}$ phits/cycle, which establishes that the injection limit of the RTT *should* be $\Theta = \frac{8}{a}$ phits/cycle per node, which is twice the value obtained for the RT.

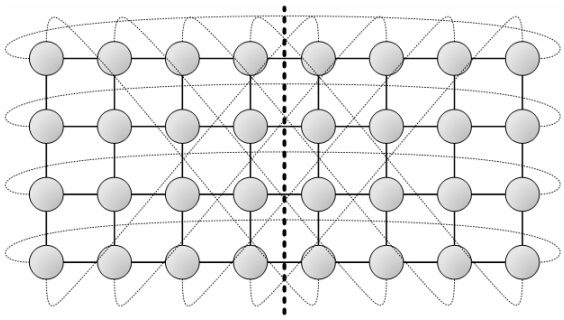

Figure 31. Depiction of an 8×4 rectangular twisted torus with skew 4 and its bisection cut.

Figure 32 plots measured throughput versus theoretical throughput following the previous computations for 32×16 systems. Note that the same figures have been measured for systems with different scales but, as their results give the same insights, for the sake of simplicity we limit the discussion to the 32×16 case. Looking at the figure, we can see how the rectangular torus reaches a throughput level that is very close to the theoretical one, and then slightly drops due to different phenomena, such as congestion and unbalanced use of network resources [MIG08]. However, if we focus on the plot for the rectangular twisted torus, we can see that measured throughput is far from the previously computed theoretical one. The first thought here would be to judge the twisted torus as a congestion-unfriendly network. Nonetheless, the goodness of being a vertex-symmetric network [CMV07] and the absence of a peak after reaching the saturation point tell us a different history: this topology seems to manage congestion properly.

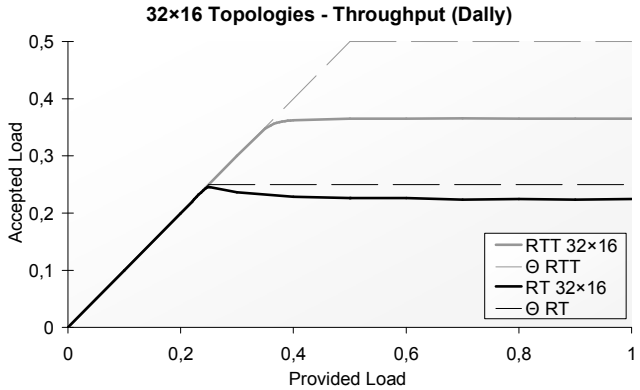

Figure 32. Measured *versus* theoretical throughput for uniform load in 32×16 rectangular torus (RT) and twisted torus (RTT).

In fact, the problems are neither in the behavior of the network nor in the simulations, but in the expression used to compute the theoretical throughput under uniform load, which is not valid for the twisted torus. The reason is that some wires of the network bisection are used by communicating nodes that are in the same half of the network, something that does not happen in the rectangular tori. **Figure 33** shows a 16×8 rectangular twisted torus (a = 8), with links omitted for simplicity. The dashed square line encloses one half of the network and represents the minimal cut. A replica tile on the top right corner is partially shown. Routing from node *A* to node *B* (both belonging to the left partition) forces the packet to follow a path going right and up, crossing the partition boundaries twice (*outwards* and then *inwards* to the original partition).

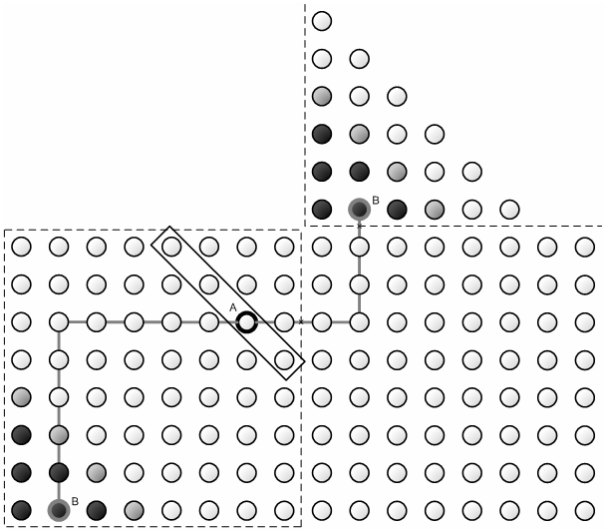

Figure 33. Rectangular twisted torus of size 16×8. Nodes in the same half of the network communicate using the bisection.

The alternative internal path inside the partition is longer and, consequently, it is discarded by the shortest path routing algorithm. For this reason part of the traffic internal to one partition crosses the bisection.

We define the *Effective Bisection Bandwidth* (E_{BB}) as the bisection bandwidth that is actually used by traffic going from one partition to the other; E_{BB} depends on the bisection bandwidth and on the amount of packets internal to one partition that are forced to cross the bisection. To quantify this value we initially consider the effect of the replica tile, shown on the top right corner of **Figure 33**. A message from node *A* to any of the black nodes in the figure will find that a destination node replica is closer crossing the partition boundaries than using an internal path. The number of black nodes is determined by T_i which is the *triangular number* for i (i =3 in **Figure 33**). The i+1 nodes in grey at distance diameter $k = a$, both in the original and the replica tile, are equally distant from node A. To balance traffic, we consider that internal and external paths are randomly chosen half of the times. Finally, the collection of destination nodes being closer in the replica tile than in the original one is the same for any of the *a-1-i* nodes in the same diagonal as *A* (4 nodes in the example **Figure 33** with i =3).

If all nodes in the partition send a packet to other nodes inside the partition (uniform traffic), the amount of packets P_1 crossing the bisection towards the replica tile on the top right corner is given by:

$$P_1 = \sum_{i=0}^{a-2}(a-1-i)\cdot T_i + \frac{1}{2}\sum_{i=0}^{a-2}(a-1-i)\cdot(i+1)$$

Considering the effect of the four replica tiles on the four corners (not shown in **Figure 33**) we have that the overall amount of packets P that, being internal to one partition, cross the bisection is:

$$P = 4P_1 = \frac{a^4 - a^2}{6} \approx \frac{a^4}{6}$$

Thus, if every node sends a packet to any other node in the network, there will be a^4 packets sent from the left partition to the right one (which obviously have to cross the bisection), and roughly $2\times\frac{a^4}{6} = \frac{a^4}{3}$ packets which are internal to each partition but have to cross the bisection twice to follow the minimal path. Hence, the traffic crossing

the bisection is increased from a^4 packets to $\frac{4a^4}{3}$, from which only a^4 are using the bisection to depart from one partition to the other. This means that, having $4a$ links in the network bisection, on average $3a$ links are used for traffic crossing from one partition to the other, while a links are used for traffic internal to one partition. Therefore, for a RTT with $N = 2a^2$, the injection rate per node under random traffic is limited by $\Theta = \frac{3a}{N/4} = \frac{12a}{2a^2} = \frac{6}{a}$ phits/cycle/node. This corresponds with an E_{BB} increase of 50% over the original RT.

In **Figure 34** we reproduce the same results of **Figure 32**, but with the correct derivation of the theoretical throughput of the twisted torus. Now we can see that the measured performance of the twisted torus is very close to the theoretical.

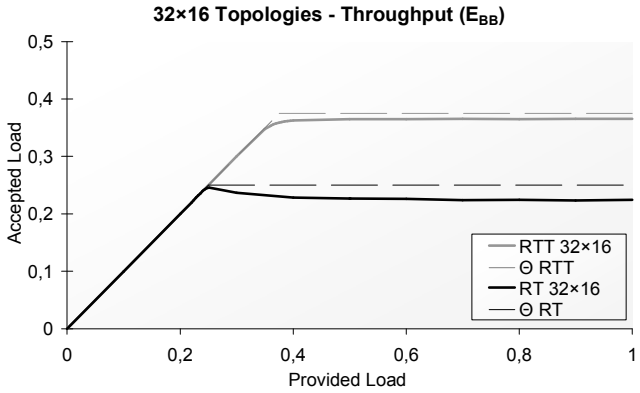

Figure 34. Measured *versus* computed throughput (Effective Bandwidth) for uniform load in 32×16 networks.

4.4 Application Traffic

In the previous section we have shown that the performance of the twisted torus is better than that of the regular torus when managing (unrealistic) uniform traffic. In order to verify that the theoretical advantages of this topology result in advantages for real application traffic, we perform additional experiments with traces from actual applications. We run the simulator, feeding it with traces generated for 32 and 128 nodes, in topologies with 8×4 and 16×8 nodes respectively. We report the *execution times* given by the simulator: the number of cycles required to inject and consume all the messages in the traces. Taking into consideration the differences in execution times, the results have been normalized to those obtained by the standard rectangular torus. Note that lower figures are better. The results for the two networks are plotted in **Figure 35**. As stated before, two simple mapping strategies are used: consecutive and random. Note that for the random mapping the 95% confidence intervals of the 20 runs are plotted.

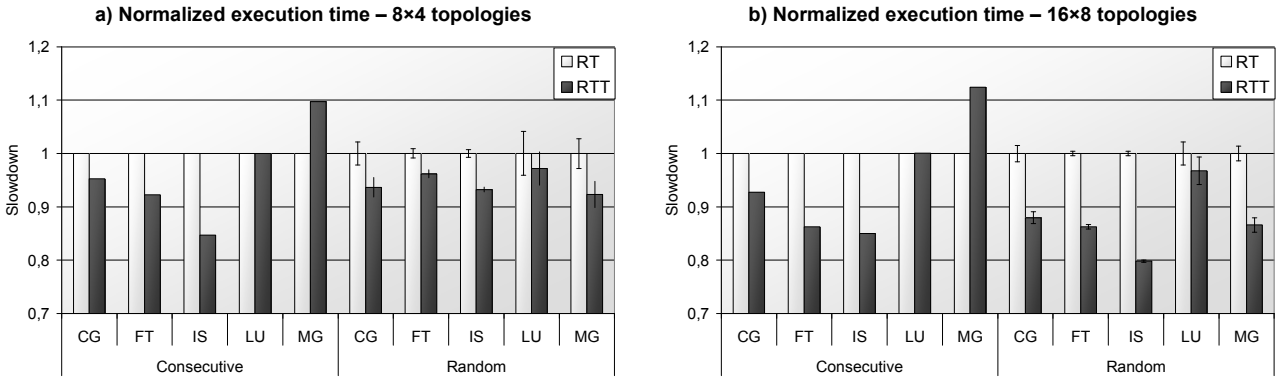

Figure 35. Normalized execution time of the NPB traces. a) 8×4 topologies. b) 16×8 topologies.

It can be seen how the rectangular twisted torus outperforms the standard rectangular torus when running the benchmarks CG (up to 10%), FT (up to 18%) and IS (up to 20%), independently of the mapping. In the case of LU with consecutive mapping, both topologies perform similarly. This is because the communication pattern of LU adapts perfectly to both topologies; in fact, this application uses a virtual mesh topology in which spatial locality is

exploited making nodes communicate with their neighbors, this way mapping a mesh over a torus or over a twisted torus leads to the same result. When LU tasks are located randomly, the rectangular twisted torus obtains lower average execution time than the standard rectangular torus, but confidence levels overlap and therefore we can not state that its performance is better. For MG the twisted alternative is worse than the non twisted one, for consecutive mapping, because of the peculiar communication pattern of MG. Still, for the random mapping, MG obtains better performance figures when being executed over the rectangular twisted torus.

It is remarkable how the advantages of the twisted topology become more evident in networks of larger sizes. For this reason, the adoption of the twisted torus as the topology to arrange large-scale systems would lead to significant improvements in terms of execution time of applications. Furthermore the confidence intervals in the rectangular twisted torus are smaller as it shows a more stable behavior.

4.5 Related Work

The idea of twisting 2D square tori in one of its two dimensions for obtaining architectural benefits is not new. A first approach appeared in the Illiac IV [BDM72]. This torus-based parallel computer employed a twist of one unit in the wraparound links of dimension X. Such twist was introduced in order to embed a Hamiltonian ring into the topology which facilitated some control and data operations. Doubly Twisted Tori were introduced by Sequin in [Sequ81] looking for optimal mappings of binary trees onto processor arrays.

Other square meshes with twisted wraparound links were proposed in [BHB87] to reduce maximum and average distance among their nodes. However, the inclusion of a twist in a square torus generates little improvement in terms of network throughput and delay: just a 4% gain in average distance can be obtained. Additionally, the twist breaks the natural full symmetry of the square network. Similar topologies were proposed in [YFJ01] as a component of hierarchical networks denoted as Recursive Diagonal Tori. More recently, in [Cvet03], it has been shown that a 4×2-node HP GS1280 computer can offer some extra performance when introducing a "shuffle" in the wraparound links of the longest dimension. Finally in [MBS08], a family of topologies based on the Gaussian integers was presented and evaluated. This evaluation includes the analytical derivation of the distance-related characteristics of such networks, and demonstrates the vertex-symmetry of all topologies belonging to this family. It is remarkable that the regular square-torus and the 2a×a twisted torus belong to this family of topologies.

Note that the inclusion of two twists (one per dimension) provides better performance, but has collateral drawbacks, such as the loss of edge-symmetry and the lack of optimal adaptive routing algorithms.

Although rectangular tori have worse relative performance than square ones, engineers have built them due to several practical reasons, such as to satisfy node count limits. Recently, mixed-radix topologies are becoming more popular. The Cray XT3 [VAD06] and XT4 [ABF07] scale from a 6×12×8 to a 40×32×24 3D torus. The IBM's Bluegene/L [AAA02] is implemented as a 64×32×32 torus and has evolved to a 72×32×32 torus in the new Bluegene/P [ABB08]. However, not too much work has been done in order to improve the performance of prismatic tori by twisting any of its three dimensions.

4.6 Conclusions

In this Chapter, we have characterized the theoretical throughput of two-dimensional twisted torus, showing the limitations of computing it from the bisection bandwidth as suggested in [DT04]. In this particular case, the utilization of a twist, forces some node pairs located in the same half of the network to communicate using the bisection. This way, the bisection is not used solely by packets that travel from one half of the network to the other, which makes the bisection bandwidth a bad estimation of the theoretical maximum throughput for uniform traffic.

Furthermore we have evaluated the performance of twisted tori by means of simulation employing a router model that incorporates all the architectural features of current packet routers, and resembles the one used in the torus network of the BlueGene/L. First of all, we have assessed the results obtained analytically performing a typical throughput analysis. Then, the networks have been tested managing application workloads from traces of the well-known NAS parallel benchmarks, in which the rectangular twisted torus has showed noticeable gains over the standard rectangular torus.

Chapter 5. Case Study: Thin-Tree Topology

The *k*-ary *n*-tree topology [PV97] is often the topology of choice to build low latency, high bandwidth and high connectivity interconnection networks for parallel computers. Its main characteristics are the low average path length and the multitude of paths from a source to a destination node, which increases exponentially with the distance (in number of hops) between nodes. The high path diversity is the main reason behind the good performance levels this topology provides for almost all kind of workloads, independently of their spatial, temporal and message length distributions. Nonetheless, its design does not take into account that parallel applications usually arrange their processes in such a way that communicating processes are as close as possible to each other, trying to obtain advantages from locality in communication. A network design that ignores locality can be a good option because it provides good performance figures even in worst-case scenarios, but the cost to pay is high. We can achieve better cost-efficiency by reducing the bandwidth among network levels, in such a way that those levels located closer to the nodes (the lower levels) are provided with higher bandwidth than those in the upper tree levels. This breaks an important property of *k*-ary *n*-trees: the constant bisection bandwidth at all tree levels.

This Chapter is devoted to compare the cost-effectiveness of *k*-ary *n*-trees with a wide variety of *thin-trees* in order to estimate how much we can reduce the topology without severe performance loss. We performed a simulation-based analysis using realistic traffic models: the traces of the well-known NAS parallel benchmarks, as well as a subset of the application kernels described in Chapter 3.

5.1 Definitions

As stated in Chapter 3, the ratio among downward ports and upward ports is denoted as the slimming factor, to avoid misinterpretations, we will use *k*:*k*' without reduction, for instance, we will use 8:4 instead of 2:1 to distinguish it from 4:2. For simplicity, the *k*-ary *n*-tree will be denoted as *k*,*n*-tree, being *k* the number of upwards (or outwards) links per switch and *n* the number of levels. The *k*:*k'*-ary *n*-thin-tree will be denoted as *k*:*k'*,*n*-tree. Similarly, *k* denotes the number of downwards ports per switch, *k'* denotes the number of upwards ports per switch and *n* the number of levels. We will use, as performance metric, the execution speed achieved by each workload. Finally, *cost-effectiveness*, *cost-efficiency* or, simply, *efficiency* will refer to the relation between the performance of a network and its cost.

5.2 Experimental Set-Up

In this evaluation we have selected two system configurations to be able to evaluate small- and large-scale systems. We have opted for simple scenarios in order to accelerate the simulations.

5.2.1 Model of the Components

Simple input-buffered multi-level switches were used in the simulations, with radices ranging from 5 to 16 depending on the topology. For the sake of simplicity, we do not use virtual channels. The switching strategy is virtual cut-through. Transit queues are able to store up to four packets. Routing is, when possible, adaptive using shortest paths. A credit-based flow-control mechanism is used, so that when several output ports are available, the one with more available credits is selected. Credits are communicated out-of-band, so they do not interfere with regular application traffic. The arbitration of each output port is performed in a random way.

Nodes are modeled as traffic generation sources with one injection queue, which is able to store four packets. Nodes are also the sink of the arrived messages. Messages are split into packets of a fixed size of 16 phits. The phit size is 32 bits. If a message does not fit exactly in an integral number of packets, the last packet contains unused phits. For simplicity reasons, we put a single task in each node and perform a direct assignment between tasks and nodes, in other words, task *i* is placed in node *i*, which results in a consecutive placement of the communicating tasks. Bearing in mind that we want to test the performance of the network, we simulated infinite-speed processors. In other words, computation intervals among communications are not taken into account.

5.2.2 Networks under Study

This study takes into account two different scales of the topology. Firstly, a small-scale system that is able to connect 64 nodes. For this configuration we fixed the number of downward ports to 4 and tested the four possible values of the slimming factor from 4:4 (the complete tree) to 4:1. A depiction of each one of these four topologies is shown in **Figure 36**. Reader should notice the reduction in the number of elements (switches and links) of the thinned topologies.

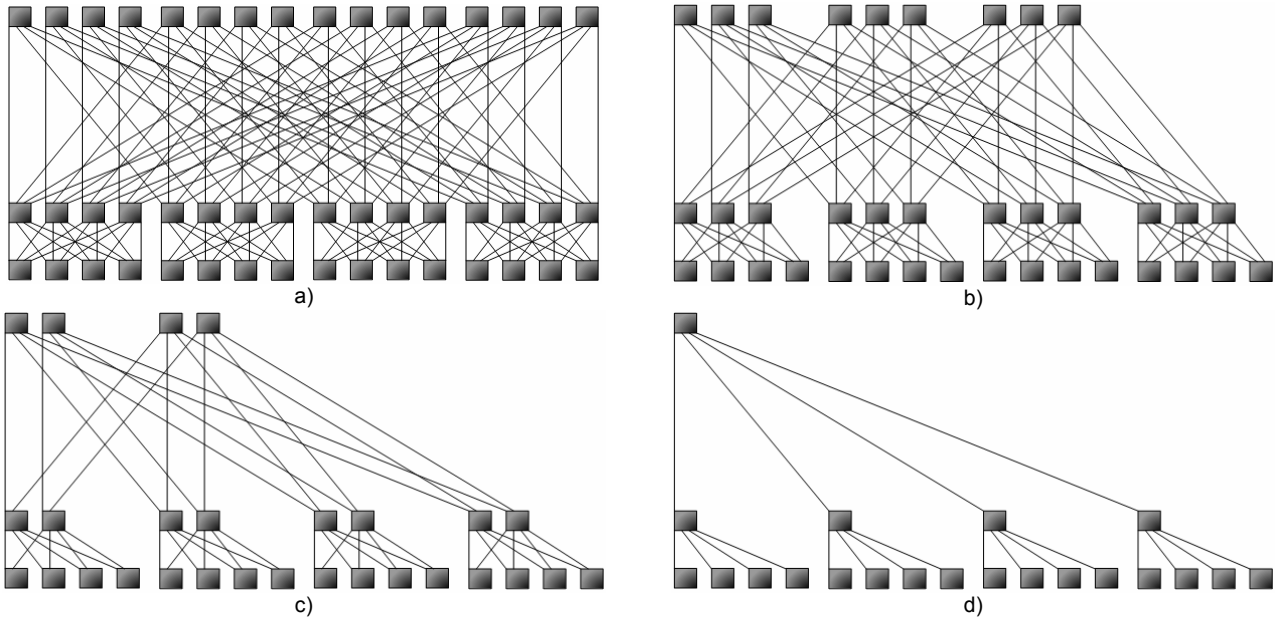

Figure 36. The topologies in the small-scale configuration. a) 4,3-tree. b)4:3,3-tree. c)4:2,3-tree. d)4:1,3-tree.

Our evaluation will also include a large scale system able to connect 4096 nodes. This configuration will be evaluated only by means of application kernels, because we can not generate traces from actual applications with such a large number of tasks. We fixed the downward ports to 8 and simulated the whole range of slimming factors, from 8:8, to 8:1. These topologies will not be represented graphically due to their complexity and size.

Table 13 and **Table 14** summarize some characteristics of the studied networks for the small- and large-scale configurations, respectively. These characteristics are the following: the number of switches and their radix, the number of links and the number of interconnected nodes. The reader should note that all the switches in each configuration have the same number of downward ports (4 and 8, correspondingly), but the actual radix of the switches is not always the same, being smaller in the more slimmed topologies. Therefore, in this evaluation, the complete trees are in advantage compared with the thinner alternatives: they use more links, and more switches that also have larger radix. Consequently, raw performance measurements are biased towards the 4,3-tree and the 8,4-tree.

	4,3-tree	4:3,3-tree	4:2,3-tree	4:1,3-tree
Nodes	64	64	64	64
Radix	8	7	6	5
Switches	48	37	28	21
Links	192	148	112	84

Table 13. Amount of elements conforming each topology – small-scale networks.

	8,4-tree	8:7,4-tree	8:6,4-tree	8:5,4-tree	8:4,4-tree	8:3,4-tree	8:2,4-tree	8:1,4-tree
Nodes	4096	4096	4096	4096	4096	4096	4096	4096
Radix	16	15	14	13	12	11	10	9
Switches	2048	1695	1400	1157	960	803	680	585
Links	16384	13560	11200	9256	7680	6424	5440	4680

Table 14. Amount of elements conforming each topology – large-scale networks.

5.2.3 Workloads

As stated before, we used a mix of workloads to test the topologies under study. First of all, for the small-scale configurations, we used traces of all the NPB [NAS] but EP, as it does not make extensive use of the communication infrastructure. Traces for 64 nodes of the class A of the selected benchmarks (BT, CG, FT, IS, LU, MG and SP) were captured in an *off-the-shelf* cluster. With these traces the simulation is fed as described in Chapter 3.

In order to expand our experimental set-up, and given that we do not have enough resources available to generate traces for the large-scale configuration, we have selected a subset of the application kernels described previously in Chapter 3. All-to-all (**AA**), binary tree (**BI**), butterfly (**BU**), nearest neighbor communication in 2D (**M2**) and 3D (**M3**) meshes, and wave-front communication in 2D (**W2**) and 3D (**W3**) meshes will complete our evaluation. All of them were generated for the given network sizes of N=64 and N=4096 communicating nodes, with messages of length 10KB. The only exception is AA, whose messages length is noticeably shorter (512B length) than for the remaining kernels, in order to be able to carry out the simulation in a reasonable amount of time—note that the number of messages in this communication pattern grows quadratically with the number of nodes.

As we are testing the capacity of the networks, none of the workloads will include computation intervals; in other words, we will simulate *infinite-speed* processors. This allows making the study independent of the actual networking and processing technologies. However, the existence of computation phases *should* (and *will*) be taken into account when looking at the results, as in actual applications these phases alternate with communicating phases.

If we focus on the ability of the workloads to stress the interconnection infrastructure, we can state that BI, W2 and W3 can be considered *light* workloads as they have highly causal communication patterns which only allow a few messages traversing the network at once; for this reason low or even no contention at all for the use of network resources is expected. In contrast, the remaining workloads can be considered *heavy* as the network is simultaneously used by most of the nodes, which can generate high congestion in the communication infrastructure. This is especially true for AA, FT and IS as their communication patterns do not exhibit spatial locality in communication.

5.3 A Justification for Thinned Topologies

To show intuitively the motivation of this study of thin trees, we simulated the delivery of the different workloads in the two complete trees in our experimental set-up, the 4,3-tree (64 nodes) and the 8,4-tree (4096 nodes). We measured the amount of packets that used each tree level, and normalize it to the total amount of packets delivered during the simulation. Note that this figure of merit is completely independent of the network architecture, the switching and the routing, provided that shortest path routing functions are used. The per-level utilization of the 4,3-tree network, when fed with NPB traces, is plotted in **Figure 37**a. Similarly, **Figure 37**b shows the per-level utilization of the 4,3-tree network when dealing with the application kernels. Finally, **Figure 37**c shows this measurement in the 8,4-tree when using application kernels.

We can see how, in all the workloads, the first level of the 4,3-tree is, obviously, used by all the packets. However, subsequent levels have lower utilization. In most cases, the second level is used by less than 75% of the packets, and the third level is used by less than 33% of the packets. The only exceptions are the FT and IS benchmarks, and the all-to-all micro-kernel. The communication patterns of these three workloads force every node to exchange the same amount of data with all the other nodes; in other words, traffic does not have locality. At any rate, even in these three benchmarks, the utilization of the topmost level is around 25% lower than the utilization of the first level.

We can see how this effect scales in the large-scale configuration. In this case, the first level of the 8,4-tree is, again, used by all the packets taking part in the simulation. The other levels have noticeably less utilization, with extreme cases in which the last level is barely used, as happens in BI (less than 0.2% utilization), M2 (around 5.5% utilization) and W2 (around 5.5% utilization). For all workloads but all-to-all, the second, third and fourth levels have utilizations below 75%, 50% and 25%, correspondingly. Again, the lack of locality of all-to-all forces a similar utilization of all levels, being the topmost level utilization below 90% compared to the first level.

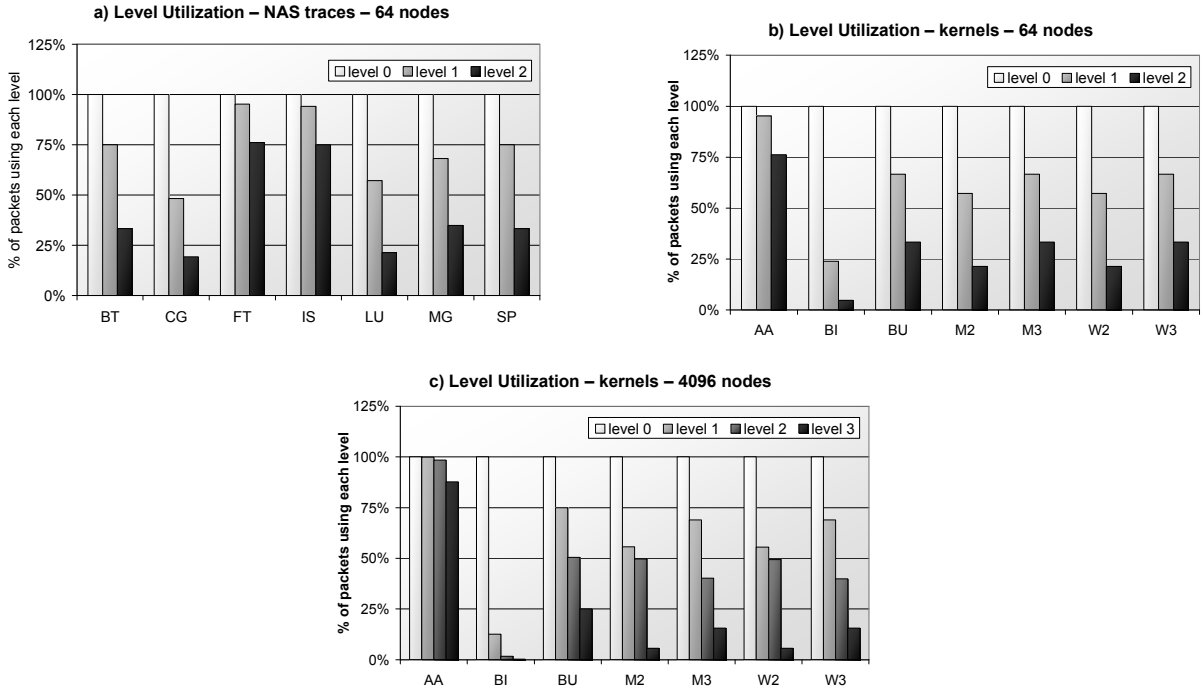

Figure 37. Utilization of the channels of each stage by the packets. a) Traces from the NAS parallel benchmarks in a 4,3-tree. b) Application kernels in a 4,3-tree. c) Application kernels in a 8,4-tree.

Summarizing, the progressively lower utilization of levels as they are closer to the top of the tree motivates the research carried out in this Chapter: the progressive reduction of the bandwidth among the levels of a tree, in a topology we have called thin-tree.

5.4 Experiments and Analysis of Results

We simulated the different networks running the selected application mix. As reported times vary noticeably depending on the workload, they are normalized to the time required by the complete trees to deliver each workload. These normalized times, that can be seen as slow-down figures, are plotted in **Figure 38**. It is remarkable that the reported simulation times correspond to the time required to deliver all the messages of the workload, and that the actual execution time of such workloads will depend on the communication/computation ratio of the applications, which in actual applications should be biased to computation, and consequently the differences among the topologies will be diluted.

If we focus on the experiments of the small-scale network, shown in **Figure 38**a and **Figure 38**b, we can see how the 4:3,3-tree provided execution times that were very close to those of the 4,3-tree. The increase was always below 20%, and in most cases below 10%. In the case of the 4:2,3-tree, the worst slow-down obtained was around 2 in the butterfly pattern. Finally, if we look at the 4:1,3-tree, we can see that an aggressive thinning of the network may lead to slowdown levels beyond 6.5, which can be considered as excessive.

The experiments performed with the large-scale configuration showed a similar picture: moderate slimming factors obtained results that are very close to those obtained with the complete tree, the performance of the 8:7,4-tree was always below a 10% increase in simulation time compared to the 8,4-tree. For the 8:6,4-tree, normalized times were below a 35% increase in all cases but M3, which was around 120% longer compared to the 8,4-tree. The thin-trees using medium slimming factors (8:5,4-tree, 8:4,4-tree and 8:3,4-tree) reached slowdowns levels around 4. Finally, in most of the cases, the most extreme sliming factors (8:2,4-tree and 8:1,4-tree) showed unacceptable slowdowns of up to 14 and 94, respectively.

Focusing on the workloads, the reader can observe that most thin-trees performed acceptably well in those traffic patterns pinpointed as *light*. With them, the complete trees were unable to take advantage of their high bandwidth and path diversity, just because the network utilization was low and so was the probability of two packets competing for an output port. In contrast, under heavy loads, the high bandwidth of the complete tree topology was able to handle the high amount of packets inside the network. In the thinned networks, there was too much contention due to the bandwidth reduction between each level and consequently, the packets were delivery significantly slower. However, in these cases, the slightly-narrowed networks (4:3,3-tree and 8:7,4-tree) required simulation times that were only a little higher than those from the complete tree.

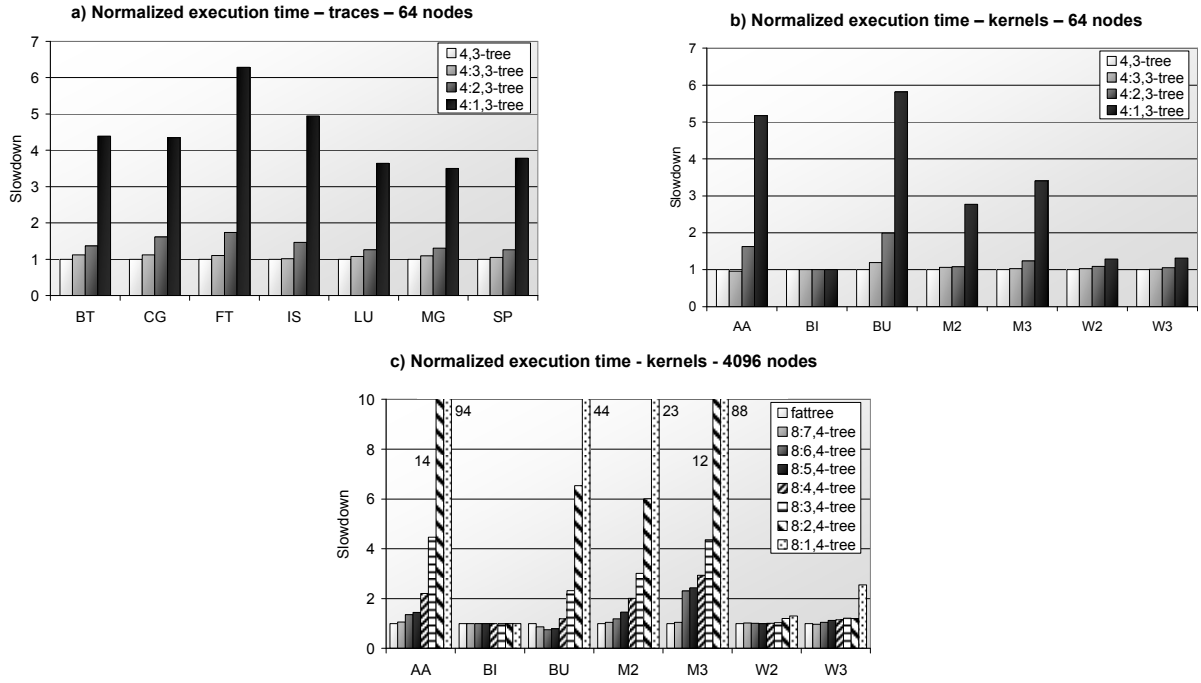

Figure 38. Normalized execution time of each workloads in the different networks. a) Traces in 64-node trees. b) Application kernels in 64-node trees. c) Application kernels in 4096-node trees.

We want to remark again that the measured times included only communication time and, therefore the actual execution time of an application will depend on its communication-computation ratio; therefore, in actual systems the advantages of implementing a full-fledged *k*-ary *n*-tree would be even less clear. Note also that the complete tree used higher radix switches and, thus has advantage over the thin-trees. If we had used switches with the same radices and the extra ports in the thin-trees had been devoted to be used as downward ports, the locality in those configurations would have been increased and, therefore, better communication times would have been obtained (if locality is correctly exploited by applications). This additional evaluation will not be carried out for the sake of simplicity, but the interested reader is encouraged to look at [NMR] for a detailed exploration of this issue.

We can conclude, from the point of view of raw performance, that the *k*-ary *n*-tree is the best-performing topology in our experimental set-up, therefore if we had unlimited (financial) resources we could just select it as the best-performing option. Nonetheless that option may not be the most cost-effective, as we will explore next.

5.5 Performance/Cost Efficiency Analysis

In this section we propose a methodology to measure the cost/performance effectiveness of an interconnection network, taking into account the workloads that are making use of it. Actual workloads vary widely from site to site, depending on the applications and data-sets of interest. In this work we are not using actual applications, but a collection of representative benchmarks and synthetic application kernels. We describe a network efficiency function in the context of these workloads that can be used with different workload types and or application mix—those applications relevant for the actual system under evaluation.

For each given workload simulation reported a (relative) time T_W. For example, we have a certain execution time T_{BU} for butterfly. Note that these values are relative to the complete trees: consequently they are always 1 for this topology. Depending on the application mix of interest in a particular computing center, we may apply a weighting factor to each workload w_W. This weight should be large for those applications that are used often. For a given network, we define its performance φ as follows:

$$\varphi = \frac{1}{\sum_{W} w_W \cdot T_W}$$

Note that, for any given application mix and set of weights, a higher value of φ represents a better-performing network. As in this work we can neither identify all representative application mixes nor even use actual applications, we decided to use a constant value of one for all the weights, with the solely purpose of illustrating the proposed methodology. With this constant weight, the denominator in our efficiency value is just the addition of the (relative)

times obtained in the experiments. This yields a value of φ=1/14 and φ=1/7 for the *k*-ary *n*-tree in the small-scale and large scale configurations, respectively. We further normalize this value to be in the range [0, 1]. **Table 15** and **Table 16** show the normalized performance values for the two configurations in our experimental set-up, the small- and large-scale systems respectively.

Performing an exhaustive cost analysis of a complete system is, clearly, a difficult task that requires the knowledge of a large number of parameters, including the choice of technologies and physical placement of the elements of the system (nodes, racks). A proper cost evaluation should take into account both deployment and maintenance costs. Deployment cost must consider the number of switches and links, that may, and probably will, have different characteristics—for instance, the use of wires of different length would be needed for most plant organizations. Maintenance costs include the power consumption and the heat dissipation.

At any rate, all this concerns are outside of the scope of this dissertation. For this reason, we will consider three simple functions to compute the cost of each network in order to be able to carry out a performance/cost comparison. In these functions, *S* represents the total number of switching elements of the network and *R* their radix. Note that in our topological model the upward ports of the topmost stage are unplugged, and therefore the last levels of the trees are formed by switches with a smaller radix. However, for the sake of simplicity, we will consider that all the switches have the same radix.

At any rate, all this concerns are outside of the scope of this dissertation. For this reason, we will consider three simple functions to compute the cost of each network in order to be able to carry out a performance/cost comparison. In these functions, S represents the total number of switching elements of the network and R their radix. Note that in our topological model the upward ports of the topmost stage are unplugged, and therefore the last levels of the trees are formed by switches with a smaller radix. However, for the sake of simplicity, we will consider that all the switches have the same radix. Furthermore, to simplify these functions, the number of links is not taken into account as it depends on the number of switches and ports.

The considered cost functions are the following:

- In the first function c_C, the cost of the switch is constant regardless of its radix, $c_C = S$. Several aspects of the manufacture of the network scale linearly with the number of switches independently of their radix, such as the cost related to the plant area, the rack space or the packaging of the switch.
- In the second function c_L, the cost of the switch depends linearly on the radix, $c_L = S \cdot R$. For instance the number of links scales linearly with the radix, as well as the cost of the hardware associated to each port.
- In the third function c_Q, the cost increases quadratically with the radix, $c_Q = S \cdot R^2$. Note that the heart of a switch (the crossbar) scales quadratically with the number of ports [EA05].

	4,3-tree	4:3,3-tree	4:2,3-tree	4:1,3-tree
φ	1.00	0.94	0.73	0.27
c_C	48	37	28	21
φ/c_C	1.00	1.22	**1.26**	0.62
c_L	384	259	168	105
φ/c_L	1.00	1.40	**1.67**	0.99
c_Q	3072	1813	1008	525
φ/c_Q	1.00	1.60	**2.23**	1.59

Table 15. Performance, cost and efficiency of the topologies – small-scale networks.

	8,4-tree	8:7,4-tree	8:6,4-tree	8:5,4-tree	8:4,4-tree	8:3,4-tree	8:2,4-tree	8:1,4-tree
φ	1.00	1.00	0.81	0.76	0.61	0.40	0.17	0.03
c_C	2048	1695	1400	1157	960	803	680	585
φ/c_C	1.00	1.21	1.19	**1.34**	1.30	1.03	0.50	0.10
c_L	32768	25425	19600	15041	11520	8833	6800	5265
φ/c_L	1.00	1.29	1.36	1.65	**1.73**	1.49	0.81	0.17
c_Q	524288	381375	274400	195533	138240	97163	68000	47385
φ/c_Q	1.00	1.37	1.55	2.03	**2.31**	2.17	1.29	0.30

Table 16. Performance, cost and efficiency of the topologies – large-scale networks.

Using the characteristics of the networks previously introduced in **Table 13** and **Table 14**, we have computed the cost of the systems using each of the three cost functions. The cost of the systems is shown in **Table 15** and **Table 16**. In these tables, we also present the cost-efficiency of the systems, calculated as the performance divided by the cost, and normalized to that of the complete tree.

For the small-scale configuration, we can see how the performance/cost efficiency of the 4:2,3-tree is the highest regardless of the cost function. This is because its performance is good, but its cost is reduced almost to one half compared with the complete tree. The 4:3,3-tree also provides better cost-efficiency than the complete tree independently of the cost function. Finally, in the case of the 4:1,3-tree, although the cost is noticeably reduced, a severe performance drop is observed and, as a consequence, it only has better cost-efficiency than the complete tree when the quadratic cost function is considered.

If we focus on the large-scale configuration, we can notice that things are not as clear as before. The 8:4,4-tree is the most cost-effective for the linear and quadratic cost functions, but its performance is reduced considerably, around 40% lower than that of the complete tree. The 8:5,4-tree may be a better option due to a noticeable boost in terms of performance, even when its cost-efficiency is not as high as the 8:4,4-tree's. Finally, it is remarkable that, even an awful-performing configuration, the 8:2,4-tree, with a measured performance of 0.17, overtakes the complete tree when using the quadratic cost function.

The reader should note that the network is only a part of the system, so that the execution time depends (greatly) on the behavior of the other components, and the interactions between all of them. In other words, the advantages or disadvantages of a given network might not be as clear as shown in our evaluations. This is an argument against the better-performing, more-expensive networks, because in real set-ups the benefit of using them will be diluted. The extent of this dilution depends on the applications and their data-sets, as well as on the architecture of the system. Furthermore, the collection of workloads used in this analysis might not be representative—and probably is not—of all actual workloads used at supercomputing centers. A thorough study should be customized for a particular site, taking into account their applications and fine-tuning their relative weights.

5.6 Related Work

Taking a look at the Top500 list [DMS], we can see the clear trend towards deploying commodity-based systems (super-clusters) that are commonly built around this class of tree-like topologies. Most of the machines in the middle positions of the list are arranged this way, which justifies our interest.

In fact, at least three super-clusters that are in positions #1 (RoadRunner), #41 (Tsubame) and #70 (Thunderbird) of the June 2009 edition of this list were built with InfiniBand networks arranged as thin trees—actually, narrowed spines. In Thunderbird the slimming factor is 2:1; RoadRunner and Tsubame goes further, to 4:1 and 5:1 respectively.

We have not found any evaluation work providing the rationale behind those decisions. However, in this work we have shown that, compared to full-fledged *k*-ary *n*-trees, thinner topologies may provide comparable performance at much lower cost. The savings in the networking elements could be invested in other ways to improve the system (faster CPUs, better networking technology, larger system, etc).

5.7 Conclusions

The *k*-ary *n*-tree is one of the topologies most widely used to build high-performance parallel computers. However, it is expensive and difficult to build. An alternative tree-like topology that requires fewer switches and links to be deployed is the thin-tree topology. Thin-trees are obviously cheaper than the complete tree in terms of cost and complexity, but at the cost of a reduction of bandwidth in the upper tree levels.

This Chapter has explored the reasons why these reduced topologies based on the *k*-ary *n*-tree are viable in terms of performance, showing that slightly thinned topologies provide competitive performance levels at significantly lower cost, which results in a higher cost-efficiency of the system. In the case of extremely thinned trees, the cost reduction is very noticeable but, in exchange, the performance of the network is degraded below tolerable limits (up to 2 orders of magnitude). Current systems are being built implementing thinned topologies, with the objective of generating cost savings in the network; savings that can be invested in other ways to improve the system, such as purchasing more computing nodes, more cores per node, faster CPUs, larger memories, better performing networking technology, etc.

Chapter 6. Case Study: SpiNNaker Interconnection Network

This Chapter discusses the evaluation of the interconnection network of SpiNNaker, a massively parallel system which aims to support the simulation of one billion (10^9) spiking neurons in real-time, housed by the more than 65 thousand nodes of the largest configuration of the system.

The Chapter starts with a description of the architecture of SpiNNaker. This system is being developed with emphasis on low-power consumption and on a specific real-time application domain. For this reason, it fits into at a rather different point of the design space, compared to the typical architecture of High-Performance Computing systems, such as those discussed in the two previous Chapters. HPC systems are commonly built with very fast processors over not-so-fast networks. In contrast, SpiNNaker is built with low-power cores working at 200MHz and an over-dimensioned network that is able to communicate at 1 Gbps. These features dictate that our evaluation will not be the traditional one for a HPC interconnection network. Throughput and latency are not the characteristics to fine tune. Instead, the amount of packets dropped and the stability of the system—understood as low variability of the performance figures—become the key characteristics to evaluate.

Taking into consideration the large scale of the system, throughout the experimental study we consider scenarios in which the interconnection network suffers different levels of hard failures. Furthermore, having the real-time restrictions in mind, we also investigate the temporal evolution of the system and observe that in failure-free scenarios the performance indicators progress stably. Furthermore the experiments show the ability of the emergency routing mechanism (to be described later) to help the system operating stably in the presence of hard failures

In order to avoid deadlock and livelock, SpiNNaker implements a packet dropping mechanism which requires some adjusting of timeout parameters. Accordingly, part of our experimental study is devoted to understanding the impact of those parameters on the interconnection network performance which will be measured in terms of the level of packet dropping under different situations and the highest injection rate managed without dropping packets. Note that the chip area constraint precludes the implementation of more sophisticated mechanisms for deadlock and livelock avoidance. Note also that these situations are very unlikely to occur, as the network utilization is expected to be low.

6.1 SpiNNaker Architecture

SpiNNaker is a system-on-chip (SoC)-based architecture designed to support the real-time simulation of large networks of spiking neurons (up to 10^9 neurons). To emulate the very high connectivity of biological systems, SpiNNaker uses a self-timed, packet-switched network which supports efficient multicast, high bandwidth, and low-delay communications. The heart of the communication infrastructure is an on-chip router and the self-timed implementation of the fabric that allows the seamless extension of the on-chip communications to include the inter-chip links. We encourage the interested reader to look at [PFT07] for a more detailed description at hardware level and at [FT07] for an engineering-oriented overview of neural networks applications and systems.

6.1.1 SpiNNaker Node

The basic block or node is the SpiNNaker chip. It contains one multi-core SoC with 20 low-power ARM968S-E processing cores [ARM] and one SDRAM chip. On-chip, each ARM core has a tightly-coupled dedicated memory that can hold 32KB of instructions and 64KB of data. Processing units are also provided with other useful modules

such as a timer, vector interrupt controller (VIC), communication controller and DMA controller. A communication-oriented model of the SpiNNaker chip is depicted in **Figure 39**; the router will be detailed later.

All the cores in a chip share a SDRAM of up to 128 MBytes in which synaptic connection information is stored. Access to this shared storage space is carried out by means of a self-timed Network on Chip (NoC) [RYK08] which is used to connect the resources in the chip. Note that the NoC provides higher communication bandwidth (8 Gbps), lower contention and lower consumption than any typical bus-based architecture [PBF08]. Detailed simulations of the chip using Verilog and SystemC proved that each core can host around 1000 individual neurons [RYK08]. The router can be accessed through the NoC, but, in practice, this is done only for configuration purposes; during normal operation the ARM cores will use the Communication Controller subsystem to send or receive packets from the network.

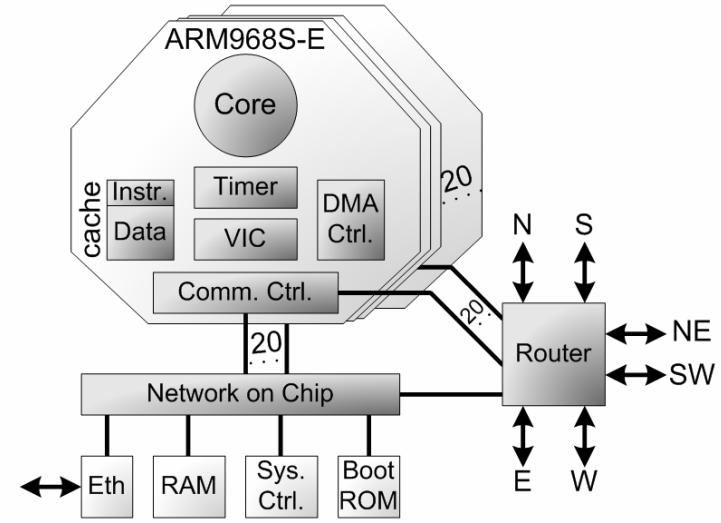

Figure 39. Schematic model of the SpiNNaker chip.

6.1.2 The On-Chip Router

Each chip incorporates a router that allows inter-chip and on-chip communications. The router is the heart of the NoC and occupies approximately 10% of the chip area. Its primary role is to direct each neural event packet to those cores where the connected neurons are located. Given the large area it occupies, popular NoCs based on two-dimensional meshes are not feasible. The chip area constraints also make unfeasible using a crossbar-based router. For this reason the organization within the router is hierarchical; ports are merged in three stages before using the actual routing engine. Note that routers are able to forward a single packet at once, but work faster than the transmission ports. Thus, most of the time routers will be idle, and router delay barely affects the pace at which packets are processed.

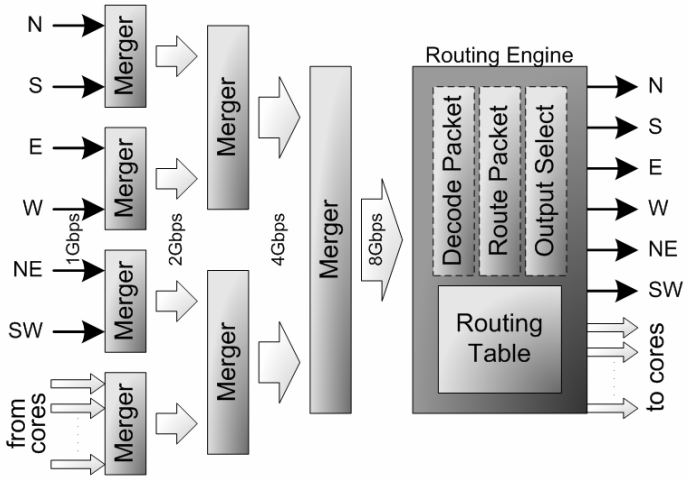

Figure 40. Architecture of the SpiNNaker router. Black arrows represent links outwards from the chip. White arrows represent hard-wired links within the chip.

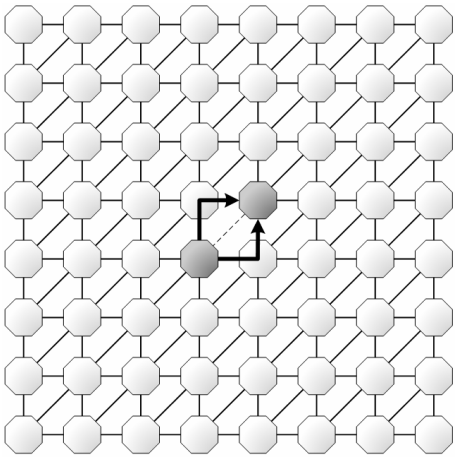

Figure 41. Example of an 8×8 SpiNNaker topology. Peripheral connections are not depicted for the sake of clarity. The regular route (thin and slashed) and the two emergency routes (arrows) between the two shaded nodes are shown.

A depiction of the router is shown in **Figure 40**. It reveals the 20 ports for internal use of the ARM cores and the six input (left) and output (right) ports to communicate with six adjacent chips. Ports are full-duplex and implement self-timed protocols.

The router is designed to support point-to-point and multicast communications, as required by the target applications. The multicast engine helps reducing pressure at the injection ports, and reduces significantly the number of packets that traverse the network, compared to a pure point-to-point alternative. Information interchange is performed using small, 40-bit packets. It is important to indicate that routers make routing decisions based on the source address (neuron identifier) of the packets. Thus, packets do not contain any information about its destination(s), only about the neuron that was fired. The network itself will deliver the packets to all chips containing neurons that have synaptic connections with the source neuron.

The information about connections is embedded in the 1024-word routing tables inside the routers, and must be preloaded using application-specific information. To minimize the space pressure on the routing tables, these offer a masked associate route look-up. Furthermore, the routers are designed to perform a default routing that sends the packet to the port opposite to the one the packet comes from. For example, a packet that comes from the North will be sent to the South. This default routing needs no entry for the source neuron in the tables. Consequently, in order to reduce the number of entries in the routing tables, the expected shape of the routes between chips is by means of two straight lines with one inflection point where the two lines are joined [KLP08].

The network topology allows two-hop routes to go from a chip to each one of its neighbors (see **Figure 41**). These two-hop paths between neighbor nodes are denoted as emergency routes and may be invoked to bypass problematic links due to transient congestion states or link failures. In practice, only one of the two possible turns is implemented in the router to minimize chip complexity (area).

SpiNNaker flow-control is very simple. When a packet arrives to an input port, one or more output ports are selected and the router tries to transmit the packet through them. If the packet cannot be forwarded, the router will keep trying, and after a given amount of time it will also test the clockwise emergency route, in other words, it will try both the regular and the emergency route. Finally, if a packet stays in the router for longer than a given threshold (*waiting time*) the packet will be dropped to avoid deadlock scenarios. Livelock situations are avoided by means of an *age* field in packet header. When two ages pass and the packet is still in the interconnection network, it is considered as *outdated* and dropped. Ages are global to the whole system and its time-span is arbitrary, a system configuration parameter. Later we will explore the bounds of the recommended values for these two network parameters (waiting time and age length).

Emulating the behavior of biological neural networks, dropped packets in SpiNNaker are not re-sent. Losing neurons (one per second in human brains) or signals does not impede the normal functioning of the biological processes; however, dropping level must be kept (*very*) low.

6.1.3 Interconnection Network Topology

SpiNNaker chips are arranged in a two-dimensional mesh topology with links to the neighbors in North, South, East, West, Southwest and Northeast. An 8×8 instance of this topology is depicted in **Figure 41** showing the emergency route between two nodes. Note that chips at the network boundaries are connected by means of

peripheral, wrap-around links that are not shown in the figure for the sake of clarity. For a 4×4 topology showing the wrap around connections look at **Figure 14**d of Chapter 3.

A knowledgeable reader may conjecture that using the 6-port router in the SpiNNaker chip, the system could be arranged as a 3D torus. In fact, the topological properties of a torus, such as bisection bandwidth and distance related characteristics are better than those of the SpiNNaker topology. However the topology of choice for SpiNNaker has some interesting properties: a two-dimensional system is easier to deploy, the diagonal links add redundancy to the design, and the previously described emergency routing can be easily implemented; note that a three-hop emergency routing could be implemented in a 3D torus but at the cost of higher chip area requirements that are not affordable. It is also remarkable that routing in a 3D torus requires more entries in the routing tables, as a regular route would be composed by three straight lines instead of two. This would increase the entries in the routing tables by roughly a 25% which may force to increase the number of entries in each table and, therefore, the chip area.

6.1.4 Archetype of Application

The main applications of SpiNNaker are to be the "mind" of robots providing real-time stimulus-response behavior [ES03], and become an experimental platform to improve our understanding of the behavior of the brain. Neural models running in the system communicate by means of spike events which occur when a neuron is stimulated beyond a given threshold and then fires. Spike events are communicated to all connected neurons, with typical fan-outs of the order of 1000. Abundant parallelism and no explicit requirement to maintain consistency characterize the biological process, as well as its resilience to failures: neurons may die and spikes may be missed. Furthermore, the biological process advances at very low pace when compared to standard computer components: milliseconds *versus* microseconds. The design of SpiNNaker takes advantage of all these characteristics to deploy a well-balanced, low-power massively parallel architecture.

Research in neuroscience provides an estimation of average neuron firing rates of 10Hz in active populations [DA01]. Thereby, for SpiNNaker, we can estimate the network usage required by each node as:

$$20\frac{cores}{chip}\cdot 1000\frac{neurons}{core}\cdot 10\frac{packets}{second\cdot neuron}\cdot 40\frac{bits}{packet}=8\,{}^{Mbps}\!/_{chip}\,.$$

This leads to an expected network utilization that is below 15% of network capacity [NLM09]. Therefore in the evaluations we should neither put emphasis on (nor extract conclusions from) the behavior of the interconnection network under saturation.

6.2 Experimental Work

6.2.1 Experimental Set-Up

A detailed model of the interconnection network of SpiNNaker was implemented within INSEE. It contains most of the features of the router, and also the topological description of the system. However, in order to be able to confront simulations of large-scale systems, some modeling simplifications were taken.

Regarding the routing tables, in the actual system they will be configured on an application basis. As our evaluation is not tied to any particular application, the table-based routing was not used; note that this reduced the computing resources required to execute the simulations. As the regular routes between chips in the actual system will attempt to use a minimal path with a single inflection point, packets were sent through minimal paths using Dimension Order Routing (DOR) which emulates the expected shape of actual communications in SpiNNaker. Note that when applying DOR, diagonal links were considered a third dimension (Z), thus the routes followed by packets are always XY, XZ or YZ—because a path XYZ is not a minimal path. It is remarkable that, while DOR is unaware of network failures, SpiNNaker will be aware of these failures, modifying the routing tables accordingly to avoid sending packets to areas pinpointed as conflictive. Hence the results about networks with failures should be taken as worst-case results. The system was evaluated under point-to-point traffic only. Consequently the multi-cast engine was not used.

The nodes (SpiNNaker chips) were modeled as independent traffic sources that inject packets following a Poisson temporal distribution, in which the injection rate (packets per cycle per node) can be tuned to any desired value. Furthermore, as all the ports from the CPUs inside a chip are merged, we modeled all of them as a single injection queue with room for four packets. If this queue is full and the cores try to inject a packet, the packet is dropped because of the lack of room to store it.

6.2.2 Timeout Parameters Optimization

We evaluated the largest configuration of SpiNNaker (256×256) under traffic with uniform distribution of packet destinations—note that the actual system is expected to use optimized mapping of the neurons keeping communicating neurons in close proximity [KLP08]. This study considered a wide range of injection rates from 0.001 to 0.068 packets/cycle per node, which roughly represent 1.6% and 109% of the system theoretical throughput. This allowed us to have a picture of the behavior of the system under different levels of communication requirements. Note that, as explained before, most of the simulated scenarios were noticeably above the expected utilization of the interconnection network. With this wide range of injection rates we can study network behavior under utilization peaks—note that although these scenarios are very unlikely to happen and, as stated before, are not relevant in our study, it is reasonable to know how the system will behave in those scenarios and will help to show the differences in behavior among the different configurations under study.

Different values of the time to wait before dropping a packet to avoid deadlock (*waiting time* from 0 to 8) were tested in order to elucidate an optimal value to be used in the actual system. Note that zero-waiting means that, if a packet cannot be transmitted, it tries the emergency route and, if it is also unavailable, the packet is dropped immediately. In the rest of the cases, the emergency route was tested in the last half of the *waiting time*.

The figures of merit were the dropped packet ratio, *i.e.* the amount of dropped packets normalized to the number of injected packets, and the injection rate at which each configuration was forced to drop packets. Note that maximum latency figures help to select a good value for the age-based packet dropping mechanism to avoid livelock in the actual system. Although not mandatory, it is preferable to keep latency low, so if several waiting times perform similarly, we will select the one with lower latency figures.

Additionally, we studied the system under low degrees of network failure, focusing on link failures. The experiments were repeated in systems with one and two random link failures. The reason to test with such reduced amount of failures is that SpiNNaker will adapt to avoid the non-working components of the system, so only a small amount of failures were expected to occur at once. If we consider a pessimistic scenario of mean time between failures average of 5 years with a sigma of 2 years, the region of interest is [30..60] failures, thus, as a worst case configuration, we tested the system with 64 random link failures. The dropped packet ratios are plotted in **Figure 42**.

Figure 42a shows the dropped ratios in properly-working systems; the lines disappearing from the bottom of the plots are those with this ratio equal to zero, in other words, without any packet dropped. The shaded area at the left of the plots represents the regular area of operation of the system. Note that the higher the *waiting time*, the higher the load the network was able to manage without dropping packets. In contrast, the lower values lead to higher dropped ratios as soon as the network became saturated. This may lead us to think that the higher the *waiting time*, the better the performance.

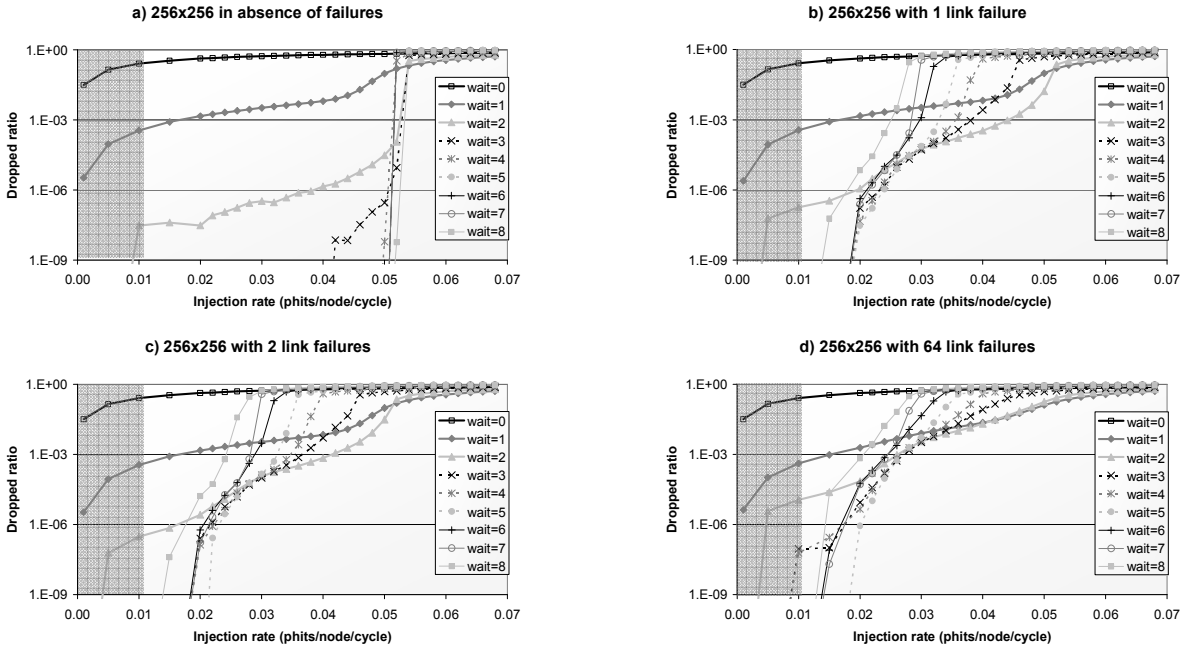

Figure 42. Packet dropped ratio for different values of the waiting time. a) Failure-free system. b) System with 1 failure. c) System with 2 failures. d) System with 64 failures.

However, when looking at the scenarios with link failures—plotted in **Figure 42**b, c and d—the picture changes drastically. Those scenarios with the highest waiting times started dropping packets before those with medium values, and reached unacceptably high dropped ratios faster. This happened because long waiting times generated congestion zones around the failure links, which spread along the whole network. The clearest example was *wait=8*, which started dropping packets noticeably before than the *waiting time* values from 4 to 7. Note also how, the lower the waiting time, the lesser the effect failures had over the dropped ratio. For the cases *wait=0* or *wait=1* we observed the same behavior independently of the number of failures. In these cases, even the slightest contention for the use of the resources lead to packet dropping. Therefore, congestion zones were neither formed nor spread. Obviously, the penalty to pay was a higher dropping ratio that surpassed the acceptable threshold.

The overall best performer of the *waiting time* values was *wait=5*, because the experiments with this value were those that started dropping packets later in all the studied scenarios. However we have to keep in mind that, in scenarios with high network pressure—which are not expected to occur—it could perform poorer than other smaller timeout values.

The experiments also helped us selecting the length of an *age* in terms of cycles, a requirement for a properly working livelock avoidance mechanism. Note that all packets being in the network for more than two ages will be dropped. An age duration that allows dropping outdated packets as soon as possible, but without dropping useful, slowly-advancing packets is desirable. Age duration could be fixed to the maximum packet latency value obtained via simulation, which depended on the *waiting time*. Note that, as ages are global to the whole network, a packet that is injected in the last cycle of an age will be tagged with that age and, therefore, is under the risk of being dropped as soon as the next age finishes, so it will only have one age length plus one cycle to be delivered. Selecting a lower value of age length may lead to unnecessary packet dropping. For example, in the case of *wait=1* an age length of 373 cycles would be a good choice. However, in this case, an outdated packet may wander around the network for up to 746 cycles. **Table 17** summarizes the measured maximum latencies, for the different *waiting time* values and number of link failures.

configuration	wait=0	wait=1	wait=2	wait=3	wait=4	wait=5	wait=6	wait=7	wait=8
0 failures	174	373	790	890	1353	1411	1809	1975	2226
1 failure	174	373	789	888	1352	1411	1811	1974	2227
2 failures	174	373	789	889	1353	1409	1812	1973	2227
64 failures	174	371	787	891	1350	1415	1807	1968	2222

Table 17. Maximum latencies measured in simulation.

6.2.3 Stability of SpiNNaker

Due to the real-time nature of SpiNNaker, our next set of experiments focused on *stability*, understood as the variability of the performance indicators as time evolves, which should be low. Another distinguishing feature of SpiNNaker is the emergency routing mechanism, and part of these experiments focus on assessing how it helps keeping the system stable.

Experiments started with a fully functioning system, fed with uniformly distributed traffic at a load of 0.02 packets/cycle/node. An increasing amount of failures is introduced every 5K network cycles, simulating a system that degrades progressively, from 0 to 1024 faulty links. Failures were introduced at once, at the beginning of every 5K-cycle block. Performance metrics were measured at intervals of 10 network cycles. In the graphs we plot accepted load (packets/cycle/node), number of dropped packets, and packet latency (average and maximum). We fixed *waiting time* to 5 as suggested by the previously reported experiments. In order to better understand the impact of emergency routing in system stability, we plot the evolution of two different systems: a regular 256×256 SpiNNaker network with this feature deactivated and the actual SpiNNaker using emergency routing. All the obtained results are plotted in **Figure 43**.

To better understand the graphs, notice that the X axis shows simulation clock (in cycles). The labels on the top (1, 2, 4, …, 1024) indicate the total number of failures at the corresponding time: during the first 5K cycles the network was fully operative, from 5K to 10K there was a single link failure, from 10K to 15K there were two failures, and so on. Each performance metric has its own unit, indicated in the Y axes: packets (for the dropped packets line), cycles (for the latency related figures) and packets/cycle/node (for the accepted load line).

Figure 43a shows how the progressive introduction of failures in a system with emergency routing deactivated resulted in a high variability of the performance metrics. If we look at results with emergency routing activated (**Figure 43**b) the system performed more stably. The only exception was the maximum delay, which showed more

variable figures for the larger number of failures, but still it showed manifestly more stable maximum delay figures than those of the system without emergency routing.

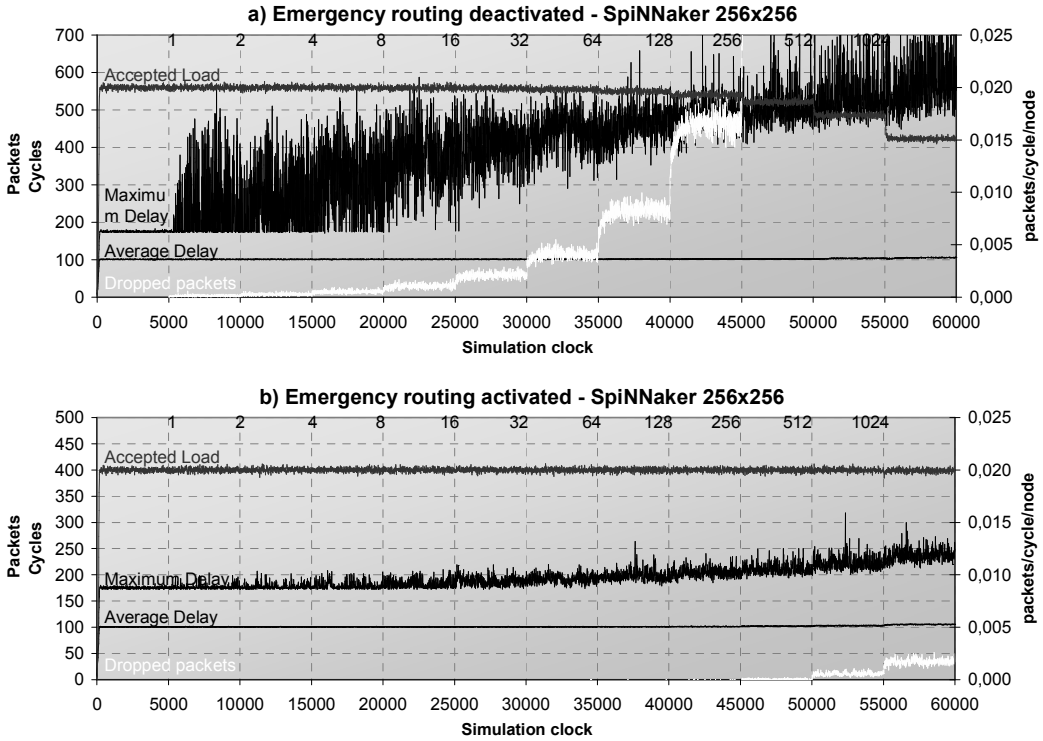

Figure 43. Evolution of the different systems under uniform point-to-point traffic at a given load of 0.02 packets/node/cycle. a) SpiNNaker without emergency routing. b) SpiNNaker with emergency routing.

If we focus on the amount of dropped packets—which is the key figure of merit, as its value *must* be kept low—we can see that the systems without emergency routing started dropping packets as soon as a single link fails. In the most extreme scenario of 1024 failures, the network *without* emergency routing dropped roughly 25% of packets while the SpiNNaker *with* emergency routing dropped only 0.2% of packets.

The conclusion of these experiments is that SpiNNaker has a highly stable network for the real-time simulation of spiking neurons, even under very pessimistic scenarios. The system did not show significant performance fluctuations, and degraded gracefully. Furthermore we showed the ability of the emergency routing mechanism to keep the system stable.

6.3 Related Work

Research in simulating biologically plausible neural networks (brain-like systems) is not new and keeps being a hot topic during the last decades. In the early nineties a team at the University of California at Berkeley worked in the Connectionist Network Supercomputer [ABF94]. This project aimed to build a supercomputer specifically tailored for neural computation as a tool for connectionist research. The system was designed to be implemented as a 2D mesh, with a target size of 128 nodes (scalable to 512). Each node would incorporate a general-purpose RISC processor plus a vector coprocessor, 16MB of RAM and a router. To our knowledge, a prototype of the node was built under the codename T0, but the system never operated as a network. Experiments using up to five nodes in a bus configuration were discussed in [FA97].

More recently, the Microelectronics Division at the Technical University of Berlin worked in a project [MDB] whose objectives were similar to those of SpiNNaker. Part of this project is an acceleration board, called SSE, implemented with a collection of FPGAs interconnected via an on-board bus. An SEE accelerator is able to perform neural computations 30 times faster than a desktop PC [HGG05]. Other projects used FPGAs for similar purposes, for example, obtaining speedups of up to 50 compared to software-only implementations. However, as these boards cannot be connected to form a network, they are not able to scale to the magnitudes of SpiNNaker.

As far as we know, the only active project comparable to SpiNNaker in terms of simulation scale is the Blue Brain project [EPF] which aims to create a biologically accurate functional model of the brain. However, the

complexity of the model used to achieve this accuracy does not allow it to work in real-time. In contrast with the biologically-inspired architecture of SpiNNaker, the Blue Brain project does not contemplate the construction of any specific computing system but uses a general-purpose supercomputer, the IBM BlueGene.

6.4 Conclusions

In this Chapter we have discussed the design and architecture of SpiNNaker. This system will be used to perform real-time simulation of spiking neural networks. Some essential goals of the design are low power consumption, reduced silicon area and high tolerance to failures. In order to be a robust system the SpiNNaker architecture relies on redundancy both in terms of computing and communicating elements.

A simulation-based study allowing the selection of optimal values for the timeout to avoid deadlock, which will be necessary at the router implementation stage, was carried out, focusing on reducing packet dropping ratios and increasing the injection rate at which the system starts dropping packets. Moreover, the measured values of maximum latencies are provided in order to help in the selection of the appropriate values for the parameters of the age-based packet dropping mechanism, implemented to avoid livelock.

Results lead to the conclusion that keeping in-transit packets waiting for too long for the allocation of output ports is counterproductive. This contention results in a backpressure that causes packet dropping at the injection queues. In most of the experiments, a waiting time of five cycles provides the best balance between the number of dropped packets at any injection rate and the injection rate at which packets start being dropped, both in the properly-working scenario and in the scenarios with link failures. Note, however, that our simulated routing model was unaware of the network failures, while the actual SpiNNaker will be aware of them, trying to route the packets through *trusted* paths. Thus, a lower degree of system degradation is expected in the actual system. Finally, we have tested the stability of the SpiNNaker interconnection under progressively faulty scenarios. It behaved in a very stable way even in highly degraded scenarios thanks to the emergency routing mechanism implemented in the routers.

It is remarkable that this study has several limitations. The most noticeable is the use of point-to-point random uniform traffic, as actual traffic in SpiNNaker will not be uniform and will use the multicast subsystem in order to reduce the amount of traffic that traverses interconnection network. In this line we plan to carry out a more realistic evaluation of the system using the routing tables and the multicast engine.

Chapter 7. Case Study: Job and Task Allocation

This Chapter is devoted to showing the impact that job and task placement has on the performance of parallel applications when the network of the computing system is based on any of the two most commonly used topologies: *k*-ary *n*-tree and *k*-ary *n*-cube. With this purpose, we carried out a simulation-based study in which we fed the networks with different workloads: application traces, and application kernels. We tested networks of various sizes, in which we explored several alternatives of task allocations for a single parallel job, as well as job and task allocations for concurrent, parallel jobs. Results support that, in most cases, random placement results in unnecessarily long execution times. We find out that some applications are insensitive to placement, but many others run very efficiently under certain topology/placement combinations. In all cases, the partition of the network into isolated sub-networks increases applications performance, because inter-application contention is prevented.

7.1 Motivation

Current high-performance computing facilities are composed of thousands of computing nodes executing jobs in parallel. The underlying interconnection network provides a mechanism for tasks to communicate. Most of these facilities are shared by many researchers and it is very uncommon to dedicate all the nodes of a site to run a single application. In most cases, nodes are time and/or space shared among users and applications. For example, in [PN99] authors describe the scheduling mechanisms in use in the supercomputers of the Numerical Aerospace Simulation supercomputer facility, located at NASA Ames Research Center.

Supercomputing sites have one or more job queues to which users send their parallel jobs; there the jobs wait until a scheduler allocates the required resources to them. It may be shocking to know that many scheduling policies disregard any knowledge about the topological characteristics of the underlying system; they see the system as an unstructured pool of computing resources. Schedulers assign free nodes—*i.e.* resources—to jobs, independently of the location of those nodes in the network, and return them to the free pool once jobs finish or are cancelled. After a certain warm-up time, which depends on the number and variety of executed jobs, physical selection of allocated resources is close to random: nodes assigned to a given job may be located anywhere in the network. In other words, resources are fragmented.

The reader should note that programmers of parallel applications usually arrange tasks in some form of virtual topology. As an example, this is a natural way of programming applications in which large datasets (matrices) are partitioned among tasks [AN99]. Programmers favor communication between neighboring tasks, under the assumption that this strategy should result in improved performance. If the assigned execution nodes are not in close proximity, programmers' efforts are totally ineffective. Furthermore, if job placement is arbitrary, the messages interchanged by a job may interfere with those interchanged by other, concurrent ones, in such a way that contention for network resources may be exacerbated. Thus, topological information should be taken into account in the scheduler's decision process to effectively exploit locality and to avoid undesired job interactions.

7.2 Experimental Set-Up

We rely on simulation to assess the impact of different simple allocation strategies on the performance of parallel applications. This is done both for *k*-ary *n*-trees and *k*-ary *n*-cubes, using systems composed by 64, 256 and 1024 nodes. The models of all the components taking part in the simulations are discussed below.

7.2.1 Workloads

Networks will be evaluated using traces taken from the NPB [NAS] and a set of application kernels. In both cases we assume *infinite-speed* processors, meaning that we only measured the time used by the network to deliver the messages, but not the time used at compute nodes to generate, receive and process them.

Regarding traces, we will use class A of the following benchmarks: Block Tridiagonal (**BT**), Conjugate Gradient (**CG**), Integer Sort (**IS**), Lower-Up diagonal (**LU**), Multi-Grid (**MG**), Scalar Pentadiagonal (**SP**) and Fourier Transform (**FT**). This study will not include Embarrassingly Parallel (**EP**) because it does not make intensive use of the network.

In this evaluation we will use the following application-kernels: all-to-all (**AA**), binary-tree (**BI**), butterfly (**BU**), distribution in 2D or 3D meshes (**M2**, **M3**) and wave-front in 2D or 3D meshes (**W2**, **W3**), all of them with a message length of 64 Kbytes.

All of the workloads used in the experiments have exactly 64 tasks. In some experiments the network has 64 nodes, so a single application uses the whole simulated system. In others, the network had $64\times N$ nodes, so N instances of an application shared the computer. Chosen values of N were 4 and 16. To simplify the experiments, we never mixed different applications. The figure of merit to measure performance is the time required to consume all the messages in the workload. When using multiple, simultaneous application instances to feed a network, reported time will be the one required to complete all the instances, in other words, the time taken by the slowest one.

7.2.2 Networks and Placement

Given the size of the experiments (up to 1024 nodes), we will consider only 2D cubes; 3D would be recommended for large-scale networks [Agar91]. In order to allow workloads to fit exactly into the network, we will use 4-ary trees. Note how these trees raise one level from configuration to configuration. In short, considering all these restrictions, the networks used in our study are the following:

- 4-ary, 3-tree and 8-ary, 2-cube (*i.e.* 8×8 torus) for experiments with a single application instance.
- 4-ary, 4-tree and 16-ary, 2-cube (*i.e.* 16×16 torus) for experiments with 4 instances of the application.
- 4-ary, 5-tree and 32-ary, 2-cube (*i.e.* 32×32 torus) for experiments with 16 instances of the application.

Note that the aim of this study is *not* to compare the torus against the tree. The evaluation of alternative network topologies goes beyond the scope of this Chapter. Our focus is on the impact that placement have on the execution time of applications of different sizes, running alone or sharing a parallel computer with other applications.

We assume that parallel jobs are composed of 64 tasks, numbered from 0 to 63. Network nodes are also numbered. In the trees, numeration of nodes is: (0,0), (0,1), (0,2), (0,3), (1,0), (1,1), etc., where (*s,p*) should be read as "switch number *s*, port number *p*". Switch numbers correspond, left to right, to the lowest level of the tree, the one to which compute nodes are attached. In the case of the tori, numeration is done using the Cartesian coordinates of the nodes.

Regarding placement, we consider *task placement* (allocation of the tasks of a single job) and *job placement* (allocation of several jobs that will run concurrently). Actually, we consider task allocation alternatives only for the experiments with a single application instance. In the other cases we evaluate combinations of task and job placement strategies. Now we describe these strategies. In all cases, we assume that assignment is done in order using firstly the job identifier and that, for a given job, nodes are assigned to task sorted by task identifier. In the case of the trees, allocation can be:

- In *Consecutive* order, switch/port assignment is done selecting, in order, node (*s,p*), increasing first *p* and then *s*.
- In *shuffle* order, switch/port assignment is done selecting, in order, node (*s,p*), increasing first *s* and then *p*.

Allocation for the torus can be:

- In *row* order, assignment is done selecting, in order, node (*x,y*), increasing first *x* and then *y*. This can be seen as partitioning the network in rectangular sub-networks, wider than tall.
- In *column* order, assignment is done selecting, in order, node (*x,y*), increasing first *y* and then *x*. This can be seen as partitioning the network in rectangular sub-networks, taller than wide.
- When using several application instances, we can partition the network in perfect squares (this is possible because our choice of network and application sizes). We use a *quadrant* scheme in which the network is partitioned this way. Allocation inside each partition is done in row order.

Moreover allocation of the tasks of N 64-task jobs to a 64×N-node machine can be done *randomly*. When running experiments with this placement, we generated five random permutations and plotted the average, maximum and minimum values of the measured execution times. Look at **Figure 28** and **Figure 29** in Chapter 3 for the graphical representation of the different placement strategies in the tori and in the trees, respectively.

7.2.3 Model of the Components

Nodes are modeled as reactive traffic sources/sinks with an injection queue able to store up to four packets. In order to model causality, the reception of a message may trigger the release of one or several extra messages as defined by the workloads. Messages are segmented into fixed-size packets of 16 phits, being phit size 4 bytes. Simple input-buffered switches were used. Transit queues had room to store up to four packets. The output port is arbitrated following the round robin policy. Switching strategy was virtual cut-through.

In the case of the trees, switches are radix-8. Routing is adaptive as discussed in Chapter 3. Tori are built using radix-5 adaptive bubble routers. Four of the ports are regular transit ports split into two virtual channels: one Escape channel plus one adaptive channel. The fifth port is an interface with the node. This interface has a single injection queue, and allows the simultaneous consumption of multiple packets.

7.3 Experiments and Analysis of Results

Results of the experiments are depicted in **Figure 44**. Execution times (actually, communication times) in cycles, as reported by the simulator, were normalized to the best performing task placement. Note that bar representing random shows the average of the 5 runs and the best and worst case as an interval. We want to remark that we are not betting for a single, *miraculous* task placement which performs best for all possible applications. In fact, we will see that some applications were not responsive to task placement. It is deliberate that plots are not arranged to allow for a direct comparison of topologies.

For the smallest networks (64-node networks and a single application instance) both in the torus and the tree, differences between the best and the worst performing placement strategy reached a 250%. This is very significant for such a small network. In general, although there were exceptions, the random placement yielded the worst results, consecutive placement was the best performer for the tree, and both row and column placements performed equally well in the torus topology.

For the medium size configurations (256-node networks and 4 concurrent application instances), the worst-to-best ratio grew up to over 300%, reaching 400% and 450% in the most adverse cases (**LU** in the tree and **BT** in the torus, respectively). Again, consecutive placement was the best performer in the tree. In the case of the torus, the best performing strategy was quadrant placement, with the single exception of **MG**, for which row and column placements worked equally well.

Finally, for the largest configuration (1024-node networks and 16 concurrent application instances), the ratio for some of the workloads when executed in the tree was around 500%, reaching a 600% in the most adverse cases. In the case of the torus, these ratios went even higher, being around 700%, and reaching 850% in the worst case. The best placement options were those described for the medium size case.

In general, the negative impact of a bad placement depended heavily on the network size. More exactly, on network distance, that depends on the height (number of levels) of the tree and on the length of the rings of the torus. Extrapolating to actual large-scale configurations with tens of thousands of nodes, we can expect that in the impact of job and task allocation may have an even more noticeable impact on the execution time required by parallel applications.

If we focus on applications, we can see how **LU**, **BT** and **SP** showed to be very sensitive to task placement, regardless of the topology. This is because their communication patterns cause a significant degree of contention for resources. The interferences between communications from different instances worsened this contention, which in turns increased even more the communication time. On the other hand, **W2** and **W3** were the workloads less responsive to task placement or topology. This is because the high degree of causality intrinsic to their traffic patterns did not cause saturation on the network; therefore, as in the absence of contention for resources, the delay depends only slightly on distance, both applications always performed well. **IS** showed to be almost insensitive to the placement, regardless of the topology.

It is interesting to observe that some workloads were very sensitive to placement when running on one topology, but not that much when the network was different. Extreme examples are **M2** and **BU**. The former adjust perfectly to a mesh topology, and was able to take full advantage of this situation when the placement allowed it. However, **M2** does not map naturally on a k-ary n-tree, so the choice of placement on this network was almost irrelevant. Regarding

BU, the perfect marriage between this pattern and the *k*-ary *n*-tree topology was exploited only with the consecutive placement, and worked on the torus equally well (or bad) with any placement. For a more detailed explanation of this effect, the interested reader can see [NMR08] in which the temporal evolution of these workloads is shown for the two topologies.

If we focus on the plots for single-instance experiments, we can see how consecutive allocation strategies were not always the best performers. Let us pay attention to results of **FT** in the torus. Row and column placements performed worse than random placement. It happens that the allocation strategies we tested were not optimal for this pattern, because its regularity lead to the occurrence of highly congested *hot paths*. Random allocation scatters these contention spots around the network, thus its performance was better. For the multi-instance experiments with **FT**, the quadrant allocation of jobs avoided harmful interferences between instances, an effect that overshadowed the bad task allocation. We would expect better results if we perform the same quadrant job allocation, but with a better task allocation, even random, inside each quadrant. Similar conclusions can be extracted from the behavior of **AA**.

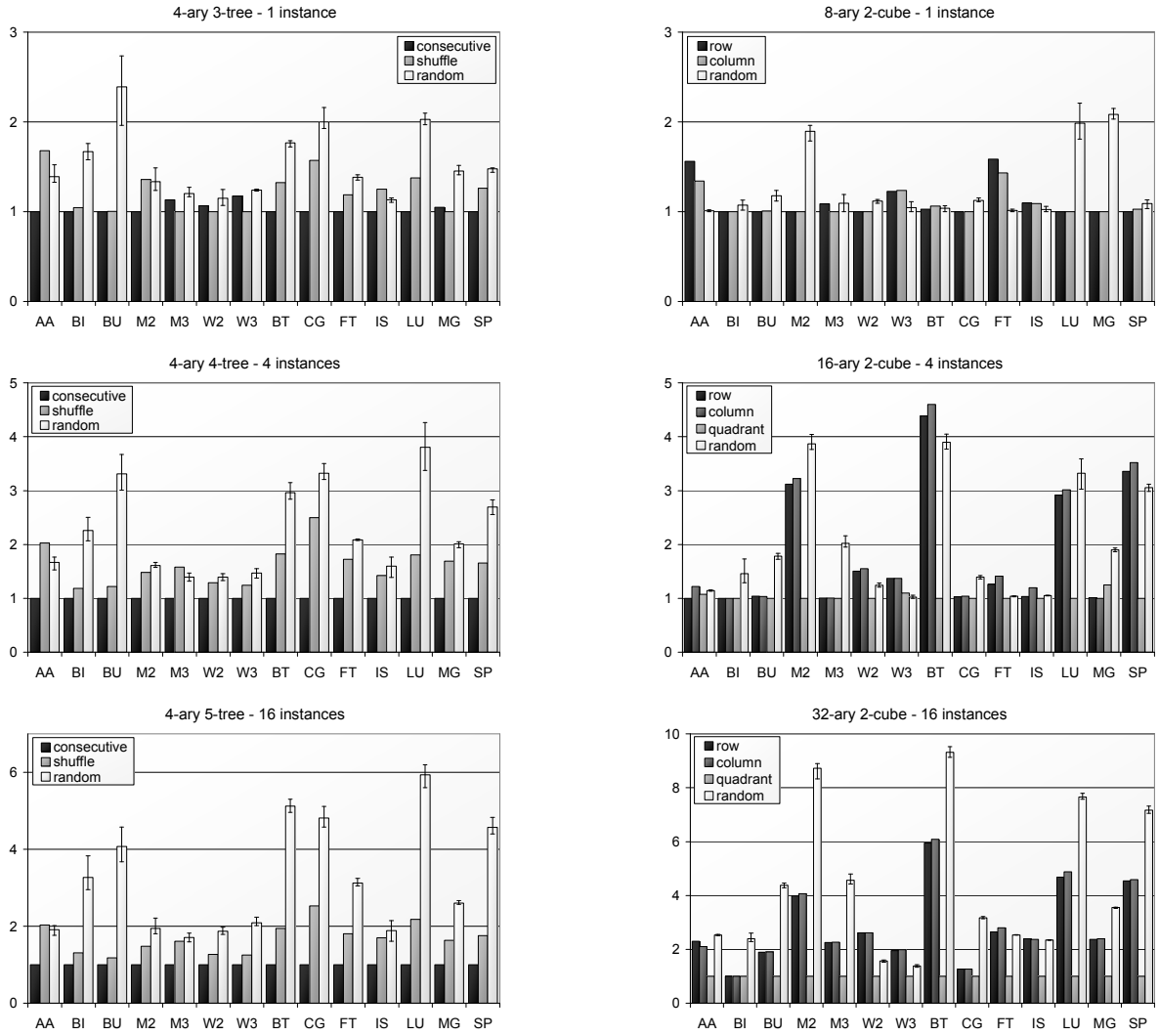

Figure 44. Normalized execution times for different workloads and placement strategies.

7.4 Related Work

We can find in the literature a variety of strategies for resource allocation and scheduling. These two problems are strongly interrelated. The use of a good allocation algorithm and a good scheduling policy decreases network fragmentation, allowing contiguous allocation of jobs in the parallel system, which is advantageous for applications.

When evaluating task allocation mechanisms, it is usual to evaluate them when the complete system is devoted to run a single application. Under this scenario, as applications are run in isolation, the impact of the interaction among traffic from different applications is not measured. For example, authors of [OSD04] proposed a communication-based mapping method which uses optimization techniques to minimize the distance among communicating tasks. In

[BK08] the positive impact that task-allocation has on the performance of three specific applications when running over the 3D torus interconnection network of a Bluegene/L is evaluated. In [DRR08], a task allocation technique for clusters is evaluated in an SGI Altix using benchmarks and applications. This technique is based on the use of multiprocessor tasks, to take advance of the faster communication within each multi-core node. More recently, in [BK09] authors perform an evaluation on how the presence of contention zones in the network of a BlueGene supercomputer affects the latency suffered by network traffic, which in turn affects the execution time of applications.

In [SKS02] we can see how the contiguous allocation of tasks resulted in improved application performance. Authors run eight sets of 16-node FFTs—benchmark FT, part of the NPB [NAS]—concurrently on a 128-node mesh, and compared contiguous vs. random node allocation. They observed a 40% improvement in runtime when using contiguous allocation. The obvious way to go is to introduce contiguous allocation strategies in schedulers for parallel machines. In some other papers addressing this issue ([LWL97] [SKS02] [ADG05] [MSD06] [LZL07]) allocation algorithms were proposed mainly for *k*-ary *n*-cube topologies. Figures of merit usually did not show how placement strategies affect the runtime of an application instance, but just the completion time of a list of jobs. Interestingly, in [LWL97] authors showed how the requirement of contiguous allocation may cause poor utilization of the system due to external or internal fragmentation. To avoid this effect, they evaluated several non-contiguous, but non-random, allocation schemes that improved overall system utilization.

A review of commercial and free schedulers shows that, by default, they are not topology aware—in other words, they do not take care of the actual placement of tasks. This is true for job queuing systems and scheduling managers such as Sun's Grid Engine [SMS], IBM's LoadLeveler [KRM01] or PBS Pro [PGW] (the latter used in Cray Supercomputers [ABF07]). Although some of them provide mechanisms for the system administrators to implement their own scheduling/allocation policies, in practice, this is not done. For example, the scheduling strategy used on Cray XT3/XT4 systems (custom-made 3D tori) simply gets the first available compute processors [Ansa]. Maui [CR] and Slurm [LLNb], in use in ASC Purple (IBM Federation network) and BSC's MareNostrum [BSC] (multistage Myrinet), have an option to take into account application placement, but they ignore the underlying topology, considering a flat network, which means that the distance between nodes is calculated as the difference between node identifiers. The most notable example of current supercomputer that tries to maintain locality when allocating resources is the BlueGene family (3D tori), whose scheduler puts tasks from the same application in one or more midplanes of 8×8×8 nodes [ADG05].

7.5 Conclusions

In this Chapter we studied the impact of job and task placement strategies on the time that parallel applications spend interchanging messages, performing a simulation-based evaluation with two kinds of workloads: traces from applications and application kernels. We focused our study on two different network topologies widely used in current supercomputers: tori and trees.

To summarize the findings of this analysis, we can state that, in some applications, the choice of task and job placements has a very relevant impact on performance, a fact that should not be taken lightly. Other applications are insensitive to allocation, and could be used to fill fragmented gaps of the network, in order to increase system utilization with a minimal impact on the overall performance. The best performing placements are those that allow for a good matching between the virtual topology (spatial distribution of application's communication) and the physical one, because this way communication locality can be exploited effectively. With very few exceptions, the random placement was the worst performer. The actual benefit of a placement strategy depends heavily on the application and the network topology, but our analysis showed that the flat-network supposition embedded in many schedulers is too simplistic and must be reconsidered in order to accelerate the execution of applications.

We want to remark again that the results presented in this work were obtained under the assumption of infinite-speed processors. Parallel applications pass through computation phases, in addition to communication phases. Our experiments showed how communication can be improved using a good placement; however, computation is not directly affected by placement. Therefore, the actual impact of placement on execution speed would depend on the communication/computation ratio of the application. In other words, the benefits we announce for good placement strategies will be diluted when running actual applications on actual machines. For example, for a 10:1 computation-communication ratio, a 450% increase in communication time will increase total execution time over 30%, which in our opinion still makes worthwhile researching on this topic, as subsequent results showed that accelerations of such magnitude are enough to make topology-aware scheduling cost-effective [PNM09].

We want to remark that, in systems implementing networks arranged as the thin-trees evaluated in Chapter 5, the exploitation of locality by applications may result in even more noticeable benefits in terms of execution time, as they are more prone to saturation and inter-application contention would exacerbate this undesirable effect.

Note that this study was focused on parallel applications running on high-performance computing systems, and more precisely, on the kind of interconnection networks used in them. However, the effects of efficiently exploiting locality could be even more noticeable when using a hierarchy of networks. Let us consider a cluster of multiprocessors. In this machine, the communication time within an on-chip network is smaller than that of the external node-to-node network, so if communicating tasks are located in the same node, the execution time should be improved. Furthermore, if the computing resource is a grid of clusters, the cluster-to-cluster communication links are orders of magnitude slower than the other networks, so the allocation of processors for the tasks of a job must avoid the utilization of these links.

Chapter 8. Conclusions

This chapter closes the dissertation with some final remarks, a summary of the main contributions of this work, and a list of current and future lines of research that complement and continue those presented here.

8.1 Final Remarks

This dissertation focused on the performance evaluation of interconnection networks. It briefly introduced supercomputing, showing the classification of computer systems from three different points of view: the system architecture, the interconnection network, and the purpose of the system. This classification is used to accurately place the aiming of our research work.

Several performance evaluation methodologies were reviewed and discussed: analytical, simulation-based and empirical. The overview of analytical methodologies revolved around the most commonly employed in performance evaluation studies: derivation of topological characteristics, Markov chains, queueing theory and Petri nets. All of them present mathematical models of the networks to evaluate, in order to calculate several figures of merit. The discussion about simulation-based studies embraced all the details to bear in mind regarding the performance of the simulation itself, as well as the different levels of accuracy, in terms of modeling both the system and the workloads. Finally, a review of benchmarks to empirically test different parts of a computing system was performed with special emphasis on those benchmarks related to high performance computing.

We thoroughly described INSEE, our Interconnection Networks Simulation and Evaluation Environment. This description included all the router architectures and network organizations currently implemented in FSIN, our Functional Simulator of Interconnection Networks, as well as all the accepted router parameters and captured statistics. Furthermore, all the traffic models and procedures to generate workloads provided by TrGen, our Traffic Generator module, are explained in detail. The workloads can be synthetic, with different degrees of fidelity to actual applications, or application-based using traces and full system simulation.

INSEE has been used to carry out many different evaluations of interconnection networks, of very different nature. The first case study evaluated the twisted torus topology which offers better topological characteristics than the torus: wider bisection bandwidth, lower distance-related figures and more balanced usage of network resources. We performed a classical analytical evaluation to derive the theoretical maximum throughput achievable by the networks under uniform traffic, showing that this methodology is not valid for the twisted torus because of its intrinsic characteristics. Furthermore, we used traces of applications to show that the improvements in topological characteristics lead to acceleration of the applications.

The second case study motivated and introduced the thin-tree topology, an alternative to the over-dimensioned k-ary n-tree topology which offers better performance/cost figures. The evaluation study carried out used traces and application kernels to measure the performance of complete trees when compared to thin-trees, and also proposed cost functions to evaluate the cost-effectiveness of the systems. The findings of this evaluation is that thin-trees can help to reduce significantly the cost of the network while showing performance figures that are only slightly worse than those of the complete tree.

A rather different evaluation was performed in the third case study in which the interconnection network of SpiNNaker was evaluated. SpiNNaker is a large-scale system-on-chip-based system with severe restrictions in terms of power consumption and chip area restrictions, especially designed for real-time simulation of spiking neuron systems. Its interconnection network is deliberately over-sized in comparison with the processing units. The analysis was focused on the fault tolerance of the system, and on its real time nature. The evaluation was two-folded: fine-

tuning several parameters of the SpiNNaker router, and proving the effectiveness of a router mechanism to maintain system operation within the adequate parameters.

The last case study evaluated the influence that job and task allocation policies may have on the execution time of parallel applications. We used traces from application as well as application kernels and executed them over torus and tree-like topologies. The study measured the execution time of a single application, as well as of several applications running concurrently, using different, simple allocation algorithms. The outcome of that study was that the impact of allocation can be very noticeable, with differences of up to 1000% in terms of communication time in our experiments, and that the larger is the system the more noticeable is the impact of allocation.

In general, several different evaluations were carried out using simulation as the main research tool. However, most of these simulation-based evaluations have mathematical backgrounds that, in some cases, are included here and, in others, can be found looking at the corresponding publications listed in Annex A. More specifically, all the topological evaluations had a throughput analysis as well as the derivation of the distance-related characteristics (showed in Chapter 3). Moreover, optimal routing functions were provided for each topology in order to fully exploit their characteristics. Finally the neighborhood relationship among networking elements had to be provided in order to be implemented in FSIN. Similarly, the application kernels required a deep examination of traces from actual applications, and the creation of mathematical models of the data interchanges.

8.2 Summary of Contributions

This section is devoted to briefly recapitulate the contributions of the research work described in this dissertation, with references to the publications in which these contributions were presented. Note that most of these contributions are described in detail in this dissertation, but some others correspond to closely related lines of work whose description have not been included in the dissertation in order to present a more coherent structure.

In the performance evaluation of different topologies for massively parallel computers:

- We have discussed how the twisted torus has better topological characteristics than a comparable regular torus, and that applications can actually benefit from them. These issues were presented and evaluated in [CMV07] and [CMV].
- We have analyzed the thin-tree topology, which offers better cost-effectiveness than the full-fledged *k*-ary *n*-tree. This work was presented in [NMR].
- The topology of the interconnection network of SpiNNaker was analytically evaluated in [NLM09].

In the evaluation of congestion control:

- The evaluation of several local congestion control mechanisms under workloads of different nature was performed in [RNM07]. The conclusion of that evaluation was that the use of congestion control mechanism positively affects the execution time of applications. This issue was not covered in this dissertation.

Fault-tolerance-related evaluations:

- The benefits in terms of fault-tolerance of the emergency routing mechanism of the SpiNNaker network was shown in [NLM09] under different levels of degradation of the interconnection network.
- Several strategies to load and distribute an application in the SpiNNaker system were evaluated in [KNR09] with special emphasis on fault tolerance and loading time. We concluded that sending packets in three directions was the *sweet point* that allowed a fast boot up, while being fault-resilient. Furthermore it was shown that using the built-in broadcast engine was enough to avoid losing packet even in scenarios of very high system degradation. This issue was not covered in this dissertation.

Topology-aware scheduling of applications:

- The impact of job and task allocation in the performance of parallel systems was measured in [NPM09], showing communication times (for the selected workloads) that differed in factors up to 10x. This fact motivated the following research work.
- In [PNM09] we investigated the influence of exploiting communication locality in the scheduling of parallel systems, and the (high) costs associated to this locality-aware scheduling. The findings were that a 10-15% reduction in the execution time of applications by means of a correct exploitation of locality is enough to make locality-aware scheduling worthwhile. This topic was not discussed in this dissertation.

In the field of simulation tools to evaluate the performance of interconnection networks using workloads from applications:

- We have developed an implemented inside INSEE a complete methodology to feed simulations with traces from actual MPI applications [MNR09].
- Similarly, we have discussed and implemented a methodology to interact with Simics in order to perform full system simulation of multicomputers. This methodology and the issues encountered during its development were discussed in [NRM07], [RMN07] and [RMN09]. This topic is covered only superficially in this dissertation.

In the characterization of workloads:

- A comparison of a bursty traffic model and trace-driven simulation was performed in the context of evaluating congestion control mechanisms [RNM07]. In that work we showed that bursty traffic results correlated with those obtained by means of trace-driven simulation. Bursty traffic is discussed in this dissertation, but the experimental comparison is left apart of the body of this dissertation.
- We have described, implemented and used in evaluation studies a collection of application-kernels. They were introduced and justified in [NMR08] and [NM09].

8.3 Future Work

Some lines of future research have been identified throughout the different sections of this dissertation. Here we provide a summary of those, and of some additional ones.

A new, event-driven simulation engine for FSIN is under development, in order to allow faster simulation of applications while properly modeling the network. This implementation will be further extended to be used in the field of job scheduling. In this line we plan to evaluate the feasibility of different topology-aware policies for the allocation/scheduling of parallel applications.

We plan to perform a more detailed evaluation of the SpiNNaker interconnection network by modeling the multicast engine, and including realistic workloads for this target system that will reproduce the way in which an actual neural network simulation will be executed.

The study on the fault-tolerance of different parts of the SpiNNaker system was the origin for our interest on the topic of fault-tolerance. We plan to integrate other fault-tolerance schemes, such as Immunet [PGV08] and ImmuCube [PG07] into INSEE in order to further explore this topic.

Another interesting line of research is the evaluation of new proposals for the architecture and organization of interconnection networks, such as, for example, the HPAR router [PGB03] or the rotary router [APP07]. The later was proposed to be used in the context of networks on chip, but its utilization could be extended to the context of interconnection networks. A proper performance evaluation contrasting it against other router models (such as those currently implemented within INSEE) is planned. A similar evaluation can be done with HPAR, a network router specifically designed to be used in CC-NUMA computers. Another interesting research in this topic is the scalability analysis of those routers, as they have been evaluated only for small-scale configurations.

We will continue with our work on characterizing realistic workloads, in order to increase our collection of application kernels. These kernels will be used in future evaluation studies in which realistic traffic models are required to provide accurate results. One interesting starting point is the review of typical high performance computing applications that can be found in [ACG06], in which 13 *dwarves* are identified. Each of these dwarves represents a prototype of application because of the communication patterns, the coupling of the task or several other details.

List of References

[AA95] M Atiquzzaman and MS Akhtar. "Performance of buffered multistage interconnection networks in a nonuniform traffic environment". Journal of Parallel and Distributed Computing Volume 30, Issue 1, October 1995, pp. 52-63. DOI: 10.1006/jpdc.1995.1125

[AAA02] NR Adiga, G Almasi, GS Almasi, et al, "An overview of the BlueGene/L Supercomputer", Proceedings of the ACM/IEEE Conference on Supercomputing, November 2002, pp. 16-22. DOI: 10.1109/SC.2002.10017

[AAG87] M Annaratone, E Arnould, T Gross, HT Kung, M Lam, O. Menzilcioglu and JA Webb. "The Warp computer: Architecture, implementation, and performance". IEEE Transactions on Computers, Volume 36, Issue 12, December 1987, pp. 1523-1538

[ABB08] S Alam, R Barrett, M Bast, MR Fahey, J Kuehn, C McCurdy, J Rogers, P Roth, R Sankaran, JS Vetter, P Worley and W Yu. "Early evaluation of IBM BlueGene/P". Proceedings of the 2008 ACM/IEEE conference on Supercomputing, Austin, Texas, USA, 15-21 November 2008. DOI: 10.1145/1413370.1413394

[ABC05] NR Adiga, MA Blumrich, D Chen, P Coteus, A Gara, ME Giampapa, P Heidelberger, S Singh, BD Steinmacher-Burow, T Takken, M Tsao and P Vranas. "Blue Gene/L torus interconnection network." IBM Journal of Research and Development, Volume 49, Number 2/3, 2005.

[ABF07] R Alam, RF Barrett, MR Fahey, JA Kuehn, JM Larkin, R Sankaran and PH Worley. "Cray XT4: An Early Evaluation for Petascale Scientific Simulation". Proceedings of the ACM/IEEE Conference on Supercomputing, Reno, NE, USA, 10-16 November, 2007. DOI: 10.1145/1362622.1362675

[ABF94] K Asanovic, J Beck, J Feldman, N Morgan and J Wawrzynek. "A supercomputer for neural computation." In Proceedings of the 1994 International Conference on Neural Networks, Orlando, FL, USA, 27 Jun-2 Jul, 1994, pp. 5-9. DOI: 10.1109/ICNN.1994.374129

[ACG06] K Asanovic, BC Catanzaro, JJ Gebis, P Husbands, K Keutzer, DA Patterson, WL Plishker, J Shalf, SW Williams and KA Yelick ."The Landscape of Parallel Computing Research: A View from Berkeley". EECS Department. University of California, Berkeley. Technical Report No. UCB/EECS-2006-183. December 18, 2006. Available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf

[ABG97] S. L. Scott and G. M. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus", Proc. HOT Interconnects IV, 1996.

[ABI93] A Arruabarrena, R Beivide, C Izu and J Miguel. "A Performance Evaluation of Adaptive Routing in Bidimensional Cut-Through Networks". Parallel Processing Letters, Volume 3, Issue 4, December 1993, pp. 469-484. DOI: 10.1142/S0129626493000496

[ABK06] SR Alam, RF Barrett, JA Kuehn, PC Roth and JS Vetter, "Characterization of Scientific Workloads on Systems with Multi-Core Processors". IEEE International Symposium on Workload Characterization, San Jose, CA. USA. 25-27 Oct, 2006. pp. 225-236. DOI: 10.1109/IISWC.2006.302747

[ACG04] G Almasi, S Chatterjee, A Gara, J Gunnels, M Gupta, A Henning, JE Moreira and B Walkup. "Unlocking the Performance of the BlueGene/L Supercomputer". Proceedings of the ACM/IEEE Conference on Supercomputing, Pittsburgh, PA, USA. 06-12 November 2004. DOI: 10.1109/SC.2004.63

[ADG05] Y Aridor, T Domany, O Goldshmidt, JE Moreira and E Shmueli "Resource allocation and utilization in the Blue Gene/L supercomputer". IBM J. Res. & Dev. Vol. 49 No. 2/3 March/May 2005. Available at: http://www.research.ibm.com/journal/rd/492/aridor.pdf

[Agar91] A Agarwal, "Limits on interconnection network performance", IEEE Transactions on Parallel and Distributed Systems, Volume 2, Issue 4, October 1991, pp. 398-412. DOI: 10.1109/71.97897

[Amda67] GM Amdahl. "Validity of single-processor approach to achieving large-scale computing capability". Proceedings of AFIPS Conference, Reston, VA. 1967. pp. 483-485.

[AN99] Y. Aoyama and J. Nakano. "RS/6000 SP: Practical MPI Programming". IBM Red Books SG24-5380-00, ISBN 0738413658. August, 1999.

[Ansa] R Ansaloni, "The Cray XT4 Programming Environment". Slides available at: http://www.csc.fi/english/csc/courses/programming_environment

[APP07] P Abad, V Puente, P Prieto and JA Gregorio. "Rotary Router: An Efficient Architecture for CMP Interconnection Networks", Proceedings of the 34th annual International Symposium on Computer Architecture, San Diego, CA, USA, June 2007, pp. 116-125. DOI: 10.1145/1250662.1250678

[ARM] Advanced RISC Machine. "ARM968E-S – ARM processor". Available at: http://www.arm.com/products/CPUs/ARM968E-S.html

[ASA96] Y Aydogan, CB Stunkel, C Aykanat and B Abali. "Adaptive source routing in multistage interconnection networks". Proceedings of the 10th International Parallel Processing Symposium, 15-19 April, 1996, pp. 258-267. DOI: 10.1109/IPPS.1996.508067

[AWI] AlphaWorks IBM. "IBM Full-System Simulator for the Cell Broadband Engine Processor". Available at: http://alphaworks.ibm.com/tech/cellsystemsim

[Batc80] KE Batcher. "Design of a Massively Parallel Processor" IEEE Transactions on Computers, Volume 29, Issue 9, September 1980, pp. 836-840. DOI: 10.1109/TC.1980.1675684

[BBB91] DH Bailey, E Barszcz, JT Barton, DS Browning, RL Carter, L Dagum, RA Fatoohi, PO Frederickson, TA Lasinski, RS Schreiber, HD Simon, V Venkatakrishnan and SK Weeratunga. "The NAS Parallel Benchmarks". International Journal of High Performance Computing Applications Volume 5, Issue 3, September 1991, pp. 63-73. DOI: 10.1177/109434209100500306

[BCC03] M Blumrich, D Chen, P Coteus, A Gara, M Giampapa, P Heidelberger, S Singh, B Steinmacher-Burrow, T Takken and P Vranas. "Design and Analysis of the BlueGene/L Torus Interconnection Network". IBM Research Report RC23025(W0312-022), December 2003.

[BCF95] NJ Boden, D Cohen, RE Felderman, AE Kulawik, CL Seitz, JN Seizovic, and WK Su. "Myrinet: A Gigabit-per-second Local Area Network," IEEE Micro, Volume 15, Issue 1, February 1995, pp. 29-36. DOI: 10.1109/40.342015

[BDH08] K Barker, K Davis, A Hoisie, D Kerbyson, M Lang, S Pakin and JC Sancho. "Entering the petaflop era: The architecture and performance of Roadrunner". Proceedings of the ACM/IEEE Conference on Supercomputing, Austin, TX, USA, 15-21 November 2008.

[BDM72] WJ Bouknight, SA Denenberg, DE McIntyre, JM Randall, AH Sameh, and DL Slotnick. "The Illiac IV System", Proceedings of IEEE, Volume 60, Issue 4, April 1972, pp. 369-388.

[BFV93] E Barszcz, R Fatoohi, V Venkatakrishnan, and S Weeratunga, "Solution of Regular, Sparse Triangular Linear Systems on Vector and Distributed-Memory Multiprocessors", Technical Report NAS RNR-93-007, NASA Ames Research Center, Moffett Field, CA, 94035-1000, April 1993.

[BHB87] R Beivide, E Herrada, JL Balcazar and J Labarta. "Optimized Mesh-Connected Networks for SIMD and MIMD Architectures", Proc. 14th Annual International Symposium on Computer Architecture, Pittsburgh, Pennsylvania, USA, 1987, pp. 163-169. DOI: 10.1145/30350.30369

[BHB91] R Beivide, E Herrada, JL Balcazar, A Arruabarrena. "Optimal distance networks of low degree for parallel computers". IEEE Transactions on Computers Volume 40, Issue 10, October 1991, pp. 1109-1124. DOI: 10.1109/12.93744

[BK08] A Bhatele and LV Kale. "Application-specific topology-aware mapping for three dimensional topologies". In Proceedings of Workshop on Large-Scale Parallel Processing, Miami, FL, USA, 14-18 April 2008. DOI: 10.1109/IPDPS.2008.4536348

[BK09] A Bhatele and ;LV Kale. "An Evaluative Study on the Effect of Contention on Message Latencies in Large Supercomputers". Workshop on Large-Scale Parallel Processing, Rome, Italy, May 29, 2009.

[BKL08] C Bienia, S Kumar and K Li. "PARSEC vs. SPLASH-2: A Quantitative Comparison of Two Multithreaded Benchmark Suites on Chip-Multiprocessors" Proceedings of the IEEE International Symposium on Workload Characterization, Seattle, WA, USA, 14-16 September, 2008, pp. 47-56. DOI: 10.1109/IISWC.2008.4636090

[BKS08] C Bienia, S Kumar, JP Singh and K Li. "The PARSEC Benchmark Suite: Characterization and Architectural Implications". Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques, Toronto, Ontario, Canada, 25-29 October, 2008, pp. 72-81. DOI: 10.1145/1454115.1454128

[BLG03] RM Badia, J Labarta, J Gimenez and F Escale. "DIMEMAS: Predicting MPI applications behavior in Grid environments". Workshop on Grid Applications and Programming Tools, June 2003.

[BMI03] R Beivide, C Martínez, C Izu, J Gutiérrez, JA Gregorio and J. Miguel-Alonso "Chordal topologies for interconnection networks". The fifth International Symposium on High Performance Computing (ISHPC-V). 20–22 Oct. 2003, Tokyo, Japan.

[Brow88] R Brown. "Calendar queues: A fast O(1) priority queue implementation for the simulation event set problem." Communications of the ACM, Volume 31, Issue 10, October 1988, pp. 1220-1227. DOI: 10.1145/63039.63045

[BS96] F Bellosa and M Steckermeier. “The performance implications of locality information usage in shared-memory multiprocessors”. Journal of Parallel and Distributed Computing Volume 37, Issue 1, August 1996, pp. 113-121. DOI: 10.1006/jpdc.1996.0112

[BSC] Barcelona Supercomputing Center. “Mare Nostrum Supercomputer”. Available (august 2007) at: http://www.bsc.es/

[BT89] JT Blake and KS Trivedi. “Reliability analysis of interconnection networks using hierarchical composition”. IEEE Transactions on Reliability Volume 38, Issue 1, April 1989, pp. 111-120. DOI: 10.1109/24.24584

[Butt04] GC Buttazzo. “Hard Real-time Computing Systems: Predictable Scheduling Algorithms and Applications”. Real-Time Systems Series, Springer-Verlag TELOS, 2004. ISBN:0387231374

[Chan99] X Chang. “Network simulations with OPNET”. Simulation Conference Proceedings, Phoenix, AZ, USA, 5-8 December, 1999, pp. 307-314. DOI: 10.1109/WSC.1999.823089

[Clos53] C Clos: “A Study of Non-Blocking Switching Networks,” Bell System Technical Journal, March 1953, pp.406-424.)

[CMV] JM Camara, M Moreto, E Vallejo, R Beivide, J Miguel-Alonso, C Martinez, J Navaridas. “Twisted Torus Topologies for Enhanced Interconnection Networks”. Accepted in IEEE Transactions on Parallel and Distributed Systems. To be published.

[CMV07] JM Camara, M Moreto, E Vallejo, R Beivide, J Miguel-Alonso, C Martinez, J Navaridas. "Mixed-radix Twisted Torus Interconnection Networks". Proceedings 21st IEEE International Parallel & Distributed Processing Symposium, Long Beach, CA, USA, March 26-30, 2007. DOI: 10.1109/IPDPS.2007.370270

[CR] Cluster Resources. “Maui Admin Manual”. Available at: http://www.clusterresources.com/products/mwm/moabdocs/MoabAdminGuide52.pdf

[Cray] Cray Inc., “Cray XD1 Overview”. Available (august 2007) at: http://www.cray.com/products/xd1/

[CT] Condor team, University of Wisconsin-Madison “High Throughput Computing”. Available at: http://www.cs.wisc.edu/condor/htc.html

[Cvet03] Z Cvetanovic,“Performance Analysis of the Alpha 21364- based HP GS1280 Multiprocessor”, Proc. of the 30th Annual International Symposium on Computer Architecture, San Diego, CA, USA, June 09-11, 2003, pp. 218-228. DOI: 10.1109/ISCA.2003.1207002

[DA01] P Dayan and L Abbott. “Theoretical Neuroscience”. Cambridge: MIT Press, 2001. ISBN 0-262-04199-5

[Dall90] WJ Dally. “Performance Analysis of k-ary n-cube Interconnection Networks”. IEEE Transactions on Computers Volume 39, Issue 6, June 1990, pp. 775-785. DOI: 10.1109/12.53599

[Dall92] WJ Dally. “Virtual-channel flow control”. IEEE Transactions on Parallel and Distributed Systems, Volume 3, Issue 2, March 1992, pp. 194-205. DOI: 10.1109/71.127260

[DCP03] JJ Dongarra, P Luszczek and A Petitet. “The LINPACK Benchmark: past, present and future”. Concurrency and Computation: Practice and Experience Volume 15, Issue 9, August 2003, pp. 803-820. DOI: 10.1002/cpe.728

[DDH00] K Diefendorff, PK Dubey, R Hochsprung and H Scale. “AltiVec extension to PowerPC accelerates media processing”. IEEE Micro, Volume 20, Issue 2, March 2000, pp. 85-95. DOI: 10.1109/40.848475

[DG94] JT Draper and J Ghosh. “A comprehensive analytical model for wormhole routing in multicomputer systems”. Journal of Parallel and Distributed Computing, Volume 23, Issue 2, November 1994, pp. 202-214. DOI: 10.1006/jpdc.1994.1132

[DJF05] J Duato, I Johnson, J Flich, F Naven, P Garcia and T Nachiondo. “A new scalable and cost-effective congestion management strategy for lossless multistage interconnection networks”. Proceedings of the 11th International Symposium on High-Performance Computer Architecture, 12-16 February, 2005, pp. 108-119. DOI: 10.1109/HPCA.2005.1

[DMS] JJ Dongarra, HW Meuer, E Strohmaier. “Top500 Supercomputer sites”. Available at: http://www.top500.org/

[Dong87] JJ Dongarra. “The LINPACK Benchmark: An explanation”. Proceedings of the 1st International Conference on Supercomputing, 08-12 June, 1987, pp. 456-474. DOI: 10.1007/3-540-18991-2_27

[DRR08] J Dummler, T Rauber and G Runger. “Mapping Algorithms for Multiprocessor Tasks on Multi-Core Clusters”. In proceedings of the 37th International Conference on Parallel Processing, 2008. Portland, OR,USA, 9-12 September 2008, pp. 141-148. DOI: 10.1109/ICPP.2008.42

[DS87] WJ Dally, CL Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," IEEE Transactions on Computers, Volume 36, Issue. 5, May 1987, pp. 547-553. DOI:10.1109/TC.1987.1676939

[DT04] WJ Dally and B Towles. “Principles and Practices of Interconnection Networks”. Morgan Kaufmann Series in Computer Architecture and Design, 2004. ISBN: 0-12-200751-4

[Duat95] J Duato. “A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks” IEEE Transactions on Parallel and Distributed Systems, Volume 6, Issue 10, October 1995, pp.1055-1067. DOI: 10.1109/71.473515

[Duni91] TH Dunigan. “Performance of the Intel iPSC/860 and Ncube 6400 hypercubes”. Journal on Parallel Computing, Volume 17, Issue 10-11, 1991, pp. 1285-1302.

[DVW05] TH Dunigan, JS Vetter, JB Whiteand PH Worley. “Performance evaluation of the Cray X1 distributed shared-memory architecture”. IEEE Micro, Volume 25, Issue 1, January 2005, pp. 30- 40. DOI: 10.1109/MM.2005.20

[EA05] H El-Rewini, and M Abd-El-Barr. “Advanced computer architecture and parallel processing” Wiley, 2005. ISBN: 978-0-471-46740-3

[ELL00] KB Erickson, RE Ladner and A Lamarca. “Optimizing static calendar queues”. ACM Transactions on Modeling and Computer Simulation, Volume 10, Issue 3, July 2000, pp. 179-214. DOI: 10.1145/361026.361028

[EPF] Ecole Polytechnique Fédérale de Lausanne. “BlueBrain project”. Available at: http://bluebrain.epfl.ch/

[Erla09] AK Erlang. “The theory of probabilities and telephone conversations”. Nyt Tidsskrift Mat. (1909). B 20, pp. 33-39.

[ES03] T Elliott and N Shadbolt. “Developmental robotics: Manifesto and application”. Philosophical Transactions of the Royal Society of London, Series A, 361. pp. 2187-2206, 2003. DOI: 10.1098/rsta.2003.1250

[FA97] P Farber and K Asanovic. “Parallel neural network training on multi-spert”. In Proceedings of the IEEE 3rd International Conference on Algorithms and Architectures for Parallel Processing, Melbourne, Vic., Australia, December 10-12, 1997, pp. 659-666. DOI: 10.1109/ICAPP.1997.651531

[Fish92] PA Fishwick. “SimPack: getting started with simulation programming in C and C++”. Proceedings of the 24th conference on Winter simulation, Arlington, Virginia, United States, 1992, pp. 154-162. DOI: 10.1145/167293.167322

[Flyn66] MJ Flynn. “Very high-speed computing systems”. Proceedings of the IEEE, Volume 54, Issue 12, December 1966, pp. 1901-1909.

[FS87] DE Foulser and R Schreiber “The Saxpy Matrix-1: a general-purpose systolic computer”,IEEE Computer, Volume20, Issue 7, July 1987, pp. 35-43. DOI: 10.1109/MC.1987.1663618

[FT07] S Furber, S Temple, “Neural Systems Engineering”. Journal of The Royal Society Interface Volume 4, Issue 13, April 2007, pp 193-206. DOI: 10.1098/rsif.2006.0177

[FW87] J Fortes and W Wah. “Systolic Arrays-From Concept to Implementation”. IEEE Computer, Volume 20, Issue 7, July 1987, pp. 12-17.

[FY02] A Faraj and X Yuan “Communication Characteristics in the NAS Parallel Benchmarks”. Proceedings of the IASTED International Conference on Parallel and Distributed Computing Systems, 4-6 November, 2002, Cambridge, USA.

[GBC06] Z Guz, E Bolotin, I Cidon, R Ginosar and A Kolodny. “Efficient Link Capacity and QoS Design for Network-on-Chip”. Proceedings of Design, Automation and Test in Europe, Munich, Germany, 6-10 March, 2006, pp. 9-14. DOI: 10.1109/DATE.2006.243951DATE '06.

[GBV02] JA Gregorio, R Beivide and F Vallejo. “Modeling of interconnection subsystems for massively parallel computers”. Performance Evaluation Volume 47, Issue 2, February 2002, pp. 105-129. DOI: 10.1016/S0166-5316(01)00058-X

[GG97] RI Greenberg and L Guan. “An improved analytical model for wormhole routed networks with application to butterfly fat-trees”. Proceedings of the 26th International Conference on Parallel Processing, Bloomington, IL, USA, 11-15 August, 1997, pp. 44-48. DOI: 10.1109/ICPP.1997.622554

[GH93] S Goldschmidt and J Hennessy. “The accuracy of trace-driven simulation of multiprocessors”. In ACM Sigmetrics Conf. on Measurement and Modeling of Computer Systems, May 1993, pp. 146-157. DOI: 10.1145/166962.167001

[GKP95] M Gerla, B Kannan and P Palnati. “Protocols for an optical star interconnect for high speed mesh networks”. Proceedings of the Fourteenth Annual Joint Conference of the IEEE Computer and Communication Societies, Boston, MA, USA, 2-6 Apr, 1995, pp. 146-153. DOI: 10.1109/INFCOM.1995.515871

[Gpgp] GPGPU.org “General-Purpose Computation Using Graphics Hardware”. Available at: http://www.gpgpu.org/

[GSZ97] R Govindarajan, F Suciu and WM Zuberek. “Timed Petri net models of multithreaded multiprocessor architectures”. Proceedings of the Seventh International Workshop on Petri Nets and Performance Models, Saint Malo, France, 3-6 June, 1997, pp. 153-162. DOI: 10.1109/PNPM.1997.595546

[GTB93] WJ Guan, WK Tsai and D Blough. “An analytical model for wormhole routing in multicomputer interconnection networks”. Proceedings of the Seventh International Parallel Processing Symposium, Newport, CA, USA, 13-16 April 1993, pp. 650-654. DOI: 10.1109/IPPS.1993.262804

[Gust88] JL Gustafson. “Reevaluating Amdahl's Law”, Communications of the ACM, Volume 31, Issue 5, May 1988. pp. 532-533. DOI: 10.1145/42411.42415

[HGG05] HH Hellmich, M Geike, P Griep, P Mahr, M Rafanelli and H Klar. “Emulation engine for spiking neurons and adaptive synaptic weights”. In Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN), Montreal, Quebec, Canada, 31 July-4 August, 2005, pp. 3261-3266. DOI: 10.1109/IJCNN.2005.1556450

[Hill85] WD Hillis. “The connection machine”. Cambridge, Mass. MIT Press, 1985. ISBN:0262081571

[Hoar62] CAR Hoare. “Quicksort”. The Computer Journal, Volume 5, Issue 1, 1962, pp. 10-16. DOI:10.1093/comjnl/5.1.10

[HPC] High Performance Computing Challenge. “High Performance Computing Challenge Benchmark Suite Website”. Available at: http://icl.cs.utk.edu/hpcc/

[HS95] FM Heide and C Scheideler. “Space-efficient routing in vertex-symmetric networks” Proceedings of the seventh annual ACM symposium on Parallel Algorithms and Architectures, Santa Barbara, California, USA, 24-26 June, 1995, pp. 137-146. DOI: 10.1145/215399.215433

[HT02] TCK Hui and ILJ Thng. “FELT: A Far Future Event List Structure Optimized for Calendar Queues”. Transactions of The Society for Modeling and Simulation International, Volume 78, Issue 6, June 2002, pp. 343-361. DOI: 10.1177/0037549702078006573

[HYK03] S Habata, M Yokokawa and S Kitawaki. “The Earth Simulator System”. NEC research & development, Volume 44, Issue 1, January 2003, pp. 21-26.

[IBM] IBM Research. “The Cell project”. Available at: http://www.research.ibm.com/cell/

[IHB] IOR HPC Benchmark project. “IOR HPC Benchmark”. Available at: http://sourceforge.net/projects/ior-sio

[IMG05] C Izu, J Miguel-Alonso, J.A Gregorio. “Evaluation of Interconnection Network Performance under Heavy Nonuniform Loads”. Lecture Notes in Computer Science, Volume 3719 / 2005 (Proc. ICA3PP 2005), pp. 396 - 405.

[IMG06] C Izu, J Miguel-Alonso, JA Gregorio. “Effects of Injection Pressure on Network Throughput”, in Proc. PDP 2006 14th Euromicro Conference on Parallel, Distributed and Network based Processing. Montbéliard-Sochaux, France- February 15-17 2006. DOI: 10.1109/PDP.2006.32

[ISI] Information Science Institute. “Network Simulator ns-2”. Available at: http://www.isi.edu/nsnam/ns/

[ISN] Intel® Software Network. “Intel® MPI Benchmarks 3.2”. Available at: http://www.intel.com/cd/software/products/asmo-na/eng/219848.htm

[Jone86] DW Jones. “An empirical comparison of priority queue and event-set implementations”. Communications of the ACM Volume 29, Issue 4, April 1986, pp. 300-311. DOI: 10.1145/5684.5686

[KCR94] V Kotov, L Cherkasova, T Rokicki and G Ciardo. “Modeling A Scalable High-Speed Interconnect with Stochastic Petri Nets”. HP Labs Technical Report HPL-94-106, 1994.

[Kend53] DG Kendall. “Stochastic Processes Occurring in the Theory of Queues and their Analysis by the Method of the Imbedded Markov Chain”. The Annals of Mathematical Statistics Volume 24, Number 3, 1953, pp. 338-354. DOI:10.1214/aoms/1177728975

[KJA02] M Koibuchi, A Jouraku and H Amano. “The Impact of Path Selection Algorithm of Adaptive Routing for Implementing Deterministic Routing”. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, Volume 3, 24-27 June, 2002, pp. 1431-1437.

[KK79] P Kermani and L Kleinrock, “Virtual cut-through: A new computer communication switching technique”, Journal on Computer Networks, Volume 3, pp. 267-286, 1979.

[KL94] S Konstantinidou and L Snyder. “The Chaos Router”. IEEE Transactions on Computers Volume 43, Issue 12, December 1994, pp. 1386-1397. DOI: 10.1109/12.338098

[KLP08] MM Khan, DR Lester, LA Plana, A Rast, X Jin, E Painkras and SB Furber. "SpiNNaker: Mapping Neural Networks onto a Massively-Parallel Chip Multiprocessor". Proc. 2008 International Joint Conference on Neural Networks, 1-8 June, 2008, pp. 2849-2856. DOI: 10.1109/IJCNN.2008.4634199

[KLS88] M Kunde, HW Lang, M Schimmler, H Schmeck, H Schröder. “The Instruction Systolic Array and its Relation to Other Models of Parallel Computers”. Parallel Computing, Volume 7, Issue 1, April 1988, pp. 25-39. DOI: 10.1016/0167-8191(88)90095-6

[KNR09] MM Khan, J Navaridas, AD Rast, X Jin, LA Plana, M Luján, JV Woods, J Miguel-Alonso and SB Furber. “Event-Driven Configuration of a Neural Network CMP System over a Homogeneous Interconnect Fabric”. 8th International Symposium on Parallel and Distributed Computing. June 30-July 4 2009. Lisbon, Portugal.

[KON06] A Khonsari, M Ould-Khaoua, A Nayebi and H Sarbazi-azad. “The impacts of timing constraints on virtual channels multiplexing in interconnect networks”. 25th IEEE International Performance, Computing, and Communications Conference, Phoenix, AZ, USA, April 10-12, 2006. DOI: 10.1109/.2006.1629390

[KPK00] SK Raman, V Pentkovski and J Keshava. “Implementing streaming SIMD extensions on the Pentium III processor”. IEEE Micro, Volume 20, Issue 4, July 2000, pp. 47-57. DOI: 10.1109/40.865866

[KRM01] S Kannan, M Roberts, P Mayes, D Brelsford and JF Skovira “Workload Management with LoadLeveler”. IBM Red Books SG24-6038-00. ISBN 0738422096. November 2001.

[KS82] DJ Kuck and RA Stokes. “The Burroughs Scientific Processor (BSP)”. IEEE Transactions on Computers, Volume 31, Issue 5, May 1982, pp. 363-376. DOI: 10.1109/TC.1982.1676014

[KTG06] VS Kumar, MJ Thazhuthaveetil and R Govindarajan. “Exploiting programmable network interfaces for parallel query execution in workstation clusters” 20th International Parallel and Distributed Processing Symposium, Rhodes Island, USA, 25-29 April 2006. DOI: 10.1109/IPDPS.2006.1639314

[KWO05] M Koibuchi, K Watanabe, T Otsuka and H Amano. “Performance Evaluation of Deterministic Routings, Multicasts, and Topologies on RHiNET-2 Cluster”. IEEE Transactions on Parallel and Distributed Systems, Volume 16, Issue 8, August 2005, pp. 747-759. DOI: 10.1109/TPDS.2005.97

[Labo01] S Labour, "MPICH-G2 Collective Operations, Performance Evaluation, optimizations", Internship report, Magistère d’informatique et modélisation (MIM), ENS Lyon, MCS Division, Argonne Natl. Labs, USA, September 2001, available at http://www-unix.mcs.anl.gov/~lacour/argonne2001/report.ps

[LAD96] CE Leiserson, ZS Abuhamdeh, DC Douglas, CR Feynman, MN Ganmukhi, JV Hill, WD Hillis, BC Kuszmaul, MA St. Pierre, DS Wells, MC Wong-Chan, SW Yang and R Zak. "The Network Architecture of the Connection Machine CM-5". Journal of Parallel and Distributed Computing, Volume 33, Issue 2, March 1996, pp. 145-158. DOI: 10.1006/jpdc.1996.0033

[LAN] Los Alamos National Laboratory. "Sweep3D Benchmark" .Available at: http://www.c3.lanl.gov/pal/software/sweep3d/

[LDK05] P Luszczek, J Dongarra, D Koester, R Rabenseifner, B Lucas, J Kepner, J McCalpin, D Bailey, D Takahashi, "Introduction to the HPC Challenge Benchmark Suite," Lawrence Berkeley National Laboratory Technical report LBNL-57493, April 2005.

[LLNa] Lawrence Livermore National Laboratory. "ASC Sequoia Benchmark Codes". Available at: https://asc.llnl.gov/sequoia/benchmarks/

[LLNb] Lawrence Livermore National Laboratory. "Simple Linux Utility for Resource Management". Available at: https://computing.llnl.gov/linux/slurm/

[LWL97] V Lo, KJ Windisch, W Liu and B Nitzberg "Noncontiguous Processor Allocation Algorithms for Mesh-Connected Multicomputers", IEEE Transactions, on Parallel and Distributed Systems, Volume 8, Issue 7, July 1997, pp. 712-726. DOI: 10.1109/71.598346

[LZL07] Y Liu, X Zhang, H Li and D Qian. "Allocating Tasks in Multi-core Processor based Parallel System". 2007 IFIP International Conference on Network and Parallel Computing Work-shops, September 18-21, 2007, pp. 748-753. DOI: 10.1109/NPC.2007.26

[MAB96] J Miguel, A Arruabarrena, R Beivide and JA Gregorio. "Assessing the Performance of the New IBM SP2 Communication Subsystem". IEEE Parallel & Distributed Technology: Systems & Technology, Volume 4, Issue 4, December 1996, pp. 12-22. DOI: 10.1109/88.544433

[Mark07] AA Markov. "Investigation of a noteworthy case of dependent trials". Izv Ros Akad Nauk, 1907

[MBS08] C Martinez, R Beivide, E Stafford, M Moretó and EM Gabidulin. "Modeling Toroidal Networks with the Gaussian Integers". IEEE Transactions on Computers, Volume 57. Issue 8, August 2008, pp. 1046-1056. DOI: 10.1109/TC.2008.57

[MCB84] MA Marsan, G Conte, and G Balbo. "A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems". ACM Transactions on Computer Systems Volume 2, Issue 2, May 1984, pp. 93-122. DOI: 10.1145/190.191

[McCa95] JD McCalpin. "Memory bandwidth and machine balance in current high performance computers". IEEE Technical Committee on Computer Architecture Newsletter, Dec 1995.

[MCE02] PS Magnusson, M Christensson, J Eskilson, D Forsgren, G Hallberg, J Hogberg, F Larsson, A Moestedt and B Werner. "Simics: A full system simulation platform". IEEE Computer, Volume 35, Issue 2, February 2002, pp.50-58. DOI: 10.1109/2.982916

[MCK08] CC Minh, JW Chung, C Kozyrakis and K Olukotun. "STAMP: Stanford Transactional Applications for Multi-Processing". IEEE International Symposium on Workload Characterization, Seattle, WA, USA, 14-16 September, 2008, pp. 35-46. DOI: 10.1109/IISWC.2008.4636089

[MDB] Microelectronics Division T.U. of Berlin. "Design and implementation of spiking neural networks". Available at: http://mikro.ee.tuberlin.de/spinn.

[MHW02] CJ Mauer, MD Hill and DA Wood. "Full-system timing-first simulation". Proceedings of the Joint International Conference on Measurement and Modeling of Computer Systems, Marina Del Rey, California, June 15 - 19, 2002, pp. 108-116. DOI: 10.1145/511334.511349

[MIG08] J Miguel-Alonso, C Izu and JA Gregorio. "Improving the performance of large interconnection networks using congestion-control mechanisms". Performance Evaluation, Volume 65, Issue 3, March 2008, pp. 203-211. DOI:10.1016/j.peva.2007.05.001

[MKH07] M Mirza-Aghatabar, S Koohi, S Hessabi and M Pedram. "An Empirical Investigation of Mesh and Torus NoC Topologies Under Different Routing Algorithms and Traffic Models" Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, August 29-31, 2007, Lübeck, Germany, pp. 19-26. DOI: 10.1109/DSD.2007.28.

[MNR09] J Miguel-Alonso, J Navaridas, FJ Ridruejo. "Interconnection network simulation using traces of MPI applications". International Journal of Parallel Programming. Volume 37, Issue 2, 2009, pp. 153-174. DOI: 10.1007/s10766-008-0089-y.

[MO02] G Min and M Ould-Khaoua. "A Comparative Study of Switching Methods in Multicomputer Networks". The Journal of Supercomputing, Volume 21, Issue 3, March 2002, pp. 227-238. DOI: 10.1023/A:1014143310001

[MPI] Message Passing Interface Forum. MPI: A Message-Passing Interface Standard. Available at http://www-unix.mcs.anl.gov/mpi/standard.html

[MSB05] MK Martin, DJ Sorin, BM Beckmann, MR Marty, M Xu, AR Alameldeen, KE Moore, MD Hill and DA Wood. "Multifacet's General Execution-Driven Multiprocessor Simulator (GEMS) Toolset," ACM SIGARCH Computer Architecture Newsletter, Volume 33, Issue 4, November 2005, pp. 92-99. DOI: 10.1145/1105734.1105747

[MSD06] DH Miriam, T Srinivasan and R Deepa. “An Efficient SRA Based Isomorphic Task Allocation Scheme for k-ary n-cube Massively Parallel Processors”. International Symposium on Parallel Computing in Electrical Engineering (PARELEC'06), Bialystok, Poland, September 13-17, 2006, pp. 37-42. DOI: 10.1109/PARELEC.2006.13

[NAS] NASA Advanced Supercomputing (NAS) division. “NAS Parallel Benchmarks” Available at http://www.nas.nasa.gov/Resources/Software/npb.html

[NCR] NAS Computing Resources. “SGI ICE – Pleiades supercomputer”. Available (July 2009) at: http://www.nas.nasa.gov/Resources/Systems/pleiades.html

[NLM09] J Navaridas, M Luján, J Miguel-Alonso, LA Plana, SB Furber. “Understanding the Interconnection Network of SpiNNaker”. In proceedings of the 23rd International Conference on Supercomputing, June 8-12, 2009,Yorktown Heights, NY, USA, pp. 286-295. DOI: 10.1145/1542275.1542317

[NM95] LM Ni and PK McKinley. “A Survey of Wormhole Routing Techniques in Direct Networks". IEEE Computer, Volume 26, Issue 2, Feb. 1993, pp. 62-76. DOI:10.1109/2.191995

[NM09] J Navaridas and J Miguel-Alonso. “Realistic Evaluation of Interconnection Networks Using Synthetic Traffic”. 8th International Symposium on Parallel and Distributed Computing. June 30 to July 4 2009. Lisbon, Portugal.

[NMR] J Navaridas, J Miguel-Alonso, FJ Ridruejo, W Denzel. "Reducing Complexity in Tree-like Computer Interconnection Networks". Technical report EHU-KAT-IK-06-07. Department of Computer Architecture and Technology, UPV/EHU. Submitted to Journal on Parallel Computing.

[NMR08] J Navaridas, J Miguel-Alonso, FJ Ridruejo. "On synthesizing workloads emulating MPI applications". The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, Miami, Florida, USA, April 14-18, 2008. DOI: 10.1109/IPDPS.2008.4536473

[NPM09] J Navaridas, JA Pascual, J Miguel-Alonso. "Effects of Job and Task Placement on Parallel Scientific Applications Performance". Proc 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. Weimar, Germany, February 18-20, 2009, pp. 55-61. DOI: 10.1109/PDP.2009.53

[NRM07] J Navaridas, FJ Ridruejo, J Miguel-Alonso. "Evaluation of Interconnection Networks Using Full-System Simulators: Lessons Learned". Proceedings 40th Annual Simulation Symposium, Norfolk, VA, USA, March 26-28, 2007, pp. 155-162. DOI: 10.1109/ANSS.2007.19

[OA99] SH Oh and JS Ahn. “Dynamic Calendar Queue”. Proceedings of the 32nd Annual Simulation Symposium, San Diego, CA, USA, 11-15 April, 1999, pp. 20-25. DOI: 10.1109/SIMSYM.1999.766449

[OCC08] L Oliker, A Canning, J Carter, J Shalf and S Ethier. “Scientific Application Performance on Leading Scalar and Vector Supercomputering Platforms”. International Journal of High Performance Computing Applications. Volume 22, Issue 1, February 2008, pp. 5-20. DOI: 0.1177/1094342006085020

[OCS] OMNeT++ Community site. “InfiniBand”. Available at: http://www.omnetpp.org/models/catalog/doc_details/2070-infiniband

[OC] OMNeT++ Community. "OMNeT++ Community Site". Available at: http://www.omnetpp.org/

[OFW99] S Oberman, G Favor and F Weber. “AMD 3DNow! technology: architecture and implementations”. IEEE Micro, Volume 19, Issue 2, March 1999, pp. 37-48. DOI: 10.1109/40.755466

[ORN] Oak Ridge National Laboratory, “Jaguar Supercomputer”. Available (august 2007) at: http://info.nccs.gov/resources/jaguar

[OSD04] JM Orduña, F Silla and J Duato. “On the development of a communication-aware task mapping technique”. Journal of Systems Architecture, Volume 50, Issue 4, March 2004, pp. 207-220. DOI: 10.1016/j.sysarc.2003.09.002

[OT] OPNET Technologies “Making Networks and Applications Perform”. Available at: http://www.opnet.com/

[PBF08] LA Plana, J Bainbridge, SB Furber, S Salisbury, Y Shi, J Wu. “An on-chip and inter-chip communications network for the spinnaker massively-parallel neural net simulator”. In Proceedings of the Second ACM/IEEE International Symposium on Networks-on-Chip, Newcastle, UK, 7-11 April 2008, pp. 215-216. DOI: 10.1109/NOCS.2008.30

[Petr62] CA Petri, “Kommunikation mit Automaten.” Bonn: Institut fur lnstrumentelle Mathematik, Schriften des IIM Nr. 3, 1962. English translation: ”Communication with Automata.” Rome Air Develop Center. Tech. Rep. RADC-TR-65-377, vol. 1, SUPPI. 1, New York, 1966.

[PFH02] F Petrini, W Feng, A Hoisie, S Coll and E Frachtenberg. “The Quadrics Network: High-Performance Clustering Technology”. IEEE Micro Volume 22, Issue 1, January 2002, pp. 46-57. DOI: 10.1109/40.988689

[PFT07] LA Plana, SB Furber, S Temple, MM Khan, Y Shi, J Wu and S Yang. “A GALS Infrastructure for a Massively Parallel Multiprocessor”. IEEE Design & Test of Computers, Volume 24, Issue 5, September 2007, pp. 454-463. DOI: 10.1109/MDT.2007.149

[PG07] V Puente, JA Gregorio, "Immucube: Scalable Fault-Tolerant Routing for k-ary n-cube Networks," IEEE Transactions on Parallel and Distributed Systems, Volume 18, Issue 6, June 2007, pp. 776-788. DOI: 10.1109/TPDS.2007.1047

[PGB02] V Puente, JA Gregorio and R Beivide. “SICOSYS: an integrated framework for studying interconnectionnetwork performance in multiprocessor systems”. Proceedings of the 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing, Canary Islands, Spain, 9-11 January, 2002, pp. 15-22. DOI: 10.1109/EMPDP.2002.994207

[PGB03] V Puente, JA Gregorio, R. Beivide and C Izu. “On the Design of a High-Performance Adaptive Router for CC-NUMA Multiprocessors”. IEEE Trans. on Parallel and Distributed Systems, Volume 14, Issue 5, May 2003. DOI: 10.1109/TPDS.2003.1199066

[PGV08] V Puente, JA Gregorio, F Vallejo and R Beivide. “Immunet: Dependable Routing for Interconnection Networks with Arbitrary Topology”. IEEE Transactions on Computers, Volume 57, Issue 12, December 2008, pp. 1676-1689. DOI: 10.1109/TC.2008.95

[PGW] PBS GridWorks. “PBS Pro”. Available at: http://www.pbsgridworks.com/

[PIB01] V Puente, C Izu, R Beivide, JA Gregorio, F Vallejo and JM Prellezo. “The adaptive bubble router”. Journal of Parallel and Distributed Computing, Volume 61, Issue 9, September 2001, pp. 1180-1208. DOI: 10.1006/jpdc.2001.1746

[PLB04] M Pirretti, GM Link, RR Brooks, N Vijaykrishnan, M Kandemir and MJ Irwin. “Fault tolerant algorithms for network-on-chip interconnect”. Proceedings of the IEEE Computer society Annual Symposium on VLSI, Lafayette, LA, USA, 19-20 February, 2004, pp. 46-51. DOI: 10.1109/ISVLSI.2004.1339507

[PN99] J Patton-Jones and B Nitzberg. “Scheduling for Parallel Supercomputing: A Historical Perspective of Achievable Utilization”. In Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science 1659, Springer-Verlag, 1999, pp. 1-16. DOI: 10.1007/3-540-47954-6_1

[PNM09] JA Pascual, J Navaridas, and J Miguel-Alonso. "Effects of Topology-Aware Allocation Policies on Scheduling Performance". Proc. 4th Workshop on Job Scheduling Strategies for Parallel Processing In Conjunction with IPDPS 2009. Rome, Italy, May 29, 2009. To be published in Elsevier’s Lecture Notes in Computer Sciences.

[PRA97] VS Pai, P Ranganathan and SV Adve. “RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors”. IEEE Technical Committee on Computer Architecture Newsletter, October 1997.

[PS] Palisade Software, “BestFit”. Available at: http://www.palisade-europe.com/bestfit/

[PV97] F Petrini and M Vanneschi. “k-ary n-trees: High Performance Networks for Massively Parallel Architectures”. Proceedings of the 11th International Parallel Processing Symposium, Geneva, Switzerland, 1-5 April, 1997, pp. 87-93. DOI: 10.1109/IPPS.1997.580853

[PW96] A Peleg and U Weiser. “MMX technology extension to the Intel architecture”. IEEE Micro, Volume 16, Issue 4, August 1996, pp. 42-50. DOI: 10.1109/40.526924

[PWD] A Petitet, RC Whaley, JJ Dongarra, A Cleary. “HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers”. Available at: http://www.netlib.org/benchmark/hpl/

[Raic09] I Raicu. “Many-Task Computing: Bridging the Gap between High Throughput Computing and High Performance Computing”, VDM Verlag Dr. Muller Publisher, 2009. ISBN: 978-3-639-15614-0

[RBL08] G Rodriguez, RM Badia and J Labarta. “An Evaluation of Marenostrum Performance”. The International Journal of High Performance Computing Applications, Volume 22, Issue 1, February 2008, pp. 81-96. DOI: 10.1177/1094342006085022

[RHW95] M Rosenblum, SA Herrod, E Witchel and A Gupta. “Complete computer system simulation: the SimOS approach”. Parallel & Distributed Technology: Systems & Applications. Volume 3, Issue 4, Winter 1995, pp. 34-43. DOI: 10.1109/88.473612 [RM05] FJ Ridruejo, J Miguel-Alonso. "INSEE: an Interconnection Network Simulation and Evaluation Environment". Lecture Notes in Computer Science, Volume 3648 (Proc. Euro-Par), 2005, pp. 1014-1023. DOI: 10.1007/11549468_111

[RMN07] FJ Ridruejo, J Miguel-Alonso, J Navaridas. "Concepts and Components of Full-System Simulation of Distributed Memory Parallel Computers". Proceedings of the 16th International Symposium on High Performance Distributed Computing, Monterey, CA, USA, June 25-29, 2007, pp. 225-226. DOI: 10.1145/1272366.1272402

[RMN09] FJ Ridruejo, J Miguel-Alonso, J Navaridas. "Full-System Simulation of Distributed Memory Multicomputers". Cluster Computing, Published online, March 28, 2009, DOI 10.1007/s10586-009-0086-y

[RNM07] FJ Ridruejo, J Navaridas, J Miguel-Alonso and C Izu, "Realistic Evaluation of Interconnection Network Performance at High Loads", 8th International Conference on Parallel and Distributed Computing Applications and Technologies, Adelaide, Australia, December 3-6, 2007, pp.97-104. DOI: 10.1109/PDCAT.2007.73

[RRA91] R Ronngren, J Riboe, and R Ayani. “Lazy queue: An efficient implementation of the pending event set”. Proceeding of the 24th Annual Simulation Symposium, New Orleans, LA, USA, 1-5 April, 1991, pp. 194-204. DOI: 10.1109/SIMSYM.1991.151506

[RYK08] AD Rast, S Yang, MM Khan, SB Furber. “Virtual synaptic interconnect using an asynchronous network-on-chip”. In Proc. 2008 International Joint Conference on Neural Networks, Hong Kong, China, 1-8 June 2008, pp. 2727-2734. DOI: 10.1109/IJCNN.2008.4634181

[SAK06] S Scott, D Abts, J Kim and WJ Dally. “The BlackWidow High-Radix Clos Network”, Proceedings of the 33rd annual International Symposium on Computer Architecture 2006, Boston, MA, USA, June 17-21, pp. 16-28. DOI: 10.1109/ISCA.2006.40

[SBB90] MD Schroeder, AD Birrell, M Burrows, H Murray, RM Needham, TL Rodeheffer, EH Satterthwaite and CP Thacker. “Autonet: a High-speed, Self-configuring Local Area Network Using Point-to-point Links” SRC research report 59, DEC, April 21, 1990.

[SCG] Supercomputing Center of Galicia (CESGA). “FinisTerrae: architecture”. Available at: http://www.cesga.es/File/Computacion/ARCHITECTURE_FINIS_TERRAE(1).pdf

[SCG08] S Saini, R Ciotti, BTN Gunney, TE Spelce, A Koniges, D Dossa, P Adamidis, R Rabenseifner, SR Tiyyagura and M Mueller. “Performance evaluation of supercomputers using HPCC and IMB Benchmarks”. Journal of Computer and System Sciences, Volume 74, Issue 6, September 2008, pp. 965-982. DOI:10.1016/j.jcss.2007.07.002

[SCH06] S Saini, J Chang, R Hood and H Jin. “A Scalability Study of Columbia using the NAS Parallel Benchmarks”. Computational Methods in Science and Technology, Special Issue 2006, pp. 33-45

[SCS08] L Seiler, D Carmean, E Sprangle, T Forsyth, M Abrash, P Dubey, S Junkins, A Lake, J Sugerman, R Cavin, R Espasa, E Grochowski, T Juan and P Hanrahan. “Larrabee: a many-core x86 architecture for visual computing”. International Conference on Computer Graphics and Interactive Techniques, Los Angeles, California, 2008. DOI: 10.1145/1399504.1360617

[Sequ81] CH Sequin, “Doubly Twisted Torus Networks for VLSI processor arrays”, Proc. 8th Annual International Symposium on Computer Architecture, 1981, Minneapolis, Minnesota, USA, pp. 471-480.

[Shan02] T Shanley. “InfiniBand Network Architecture”. Addison-Wesley, November 2002. ISBN: 978-0-321-11765-6

[SI] Simulcraft Inc. “Omnest Simulation Environment”. Available at: http://www.omnest.com/

[SK75] KV Sastry and RY Kain “On the Performance of Certain Multiprocessor Computer Organizations”. IEEE Transactions on Computers Volume 24, Issue 11, November 1975, pp. 1066-1074. DOI: 10.1109/T-C.1975.224134

[SKS02] V Subramani, R Kettimuthu, S Srinivasan, J Johnson and, P Sadayappan “Selective Buddy Allocation for Scheduling Parallel Jobs on Clusters”. Fourth IEEE International Conference on Cluster Computing, (CLUSTER'02), September, 2002, pp. 107. DOI: 10.1109/CLUSTR.2002.1137735

[SMS] Sun Micro Systems, Inc. “N1 Grid Engine 6 User’s Guide”. Available at: http://docs.sun.com/app/docs/coll/1017.3

[SNLa] Sandia National Labs, “ASCI Red”. Available (august 2007) at: http://www.sandia.gov/ASCI/Red/

[SNLb] Sandia National Labs, “RedStorm”. Available (august 2007) at: http://www.cs.sandia.gov/platforms/RedStorm.html

[SOM01] H. Sarbazi-Azad, M. Ould-Khaoua and LM Mackenzie “An Accurate Analytical Model of Adaptive Wormhole Routing in k-ary n-cubes Interconnection Networks”. Performance Evaluation Volume 43, Issues 2-3, February 2001, pp. 165-179. DOI: 10.1016/S0166-5316(00)00049-3

[SP06] L Schaelicke and M Parker. “The design and utility of the ML-RSIM system simulator”. Journal of Systems Architecture: the EUROMICRO Journal, Volume 52, Issue 5, May 2006, pp. 283-297. DOI: 10.1016/j.sysarc.2005.07.001

[SPE] Standard Performance Evaluation Corporation. “Spec Benchmarks”. Available at: http://www.spec.org

[SS07] H Shan and J Shalf, "Using IOR to analyze the I/O Performance for HPC Platforms" June 8, 2007). Lawrence Berkeley National Laboratory. Technical Report LBNL-62647. Available at: http://repositories.cdlib.org/lbnl/LBNL-62647

[SWG92] JP Singh, DW Weber and A Gupta. “SPLASH: Stanford Parallel Applications for Shared-Memory”. Technical Report No. CSL-TR-92-526. June 1992. Available at: http://historical.ncstrl.org/litesite-data/stan/CSL-TR-92-526.pdf

[TF88] Y Tamir and G Frazier. “High performance multi-queue buffers for VLSI communication switches”. Proc.of 15th Annual Symposium on Computer Architecture, June 1988, pp.343-354. DOI: 10.1109/ISCA.1988.5245

[TG03] R Thakur and W Gropp, “Improving the Performance of Collective Operations in MPICH”. Lecture Notes in Computer Science, Volume 2840/2003, pp. 257-267. DOI: 10.1007/b14070

[TH97] D Tutsch and G Hommel. “Performance of buffered multistage interconnection networks in caseof packet multicasting”. Proceedings of Advances in Parallel and Distributed Computing, Shanghai, China, 19-21 March, 1997, pp. 50-57. DOI: 10.1109/APDC.1997.574013

[TS99] T Tabe and QF Stout. “The use of the MPI communication library in the NAS parallel benchmarks”. Technical Report CSE-TR-386-99, Department of Computer Science, University of Michigan, November 1999.

[UC] University of Cantabria. “SICOSYS”. Available at: http://www.atc.unican.es/SICOSYS/

[UF] University of Florida. “SimPack Toolkit”. Available at: http://www.cis.ufl.edu/~fishwick/simpack/simpack.html

[UP] University of Princetown. “The PARSEC Benchmark Suite”. Available at: http://parsec.cs.princeton.edu/

[USa] University of Stanford. “Home page for the Stanford Parallel Applications for Shared Memory (SPLASH)”. Available at: http://www-flash.stanford.edu/apps/SPLASH/

[USb] University of Stanford. “STAMP: Stanford Transactional Applications for Multi-Processing”. Available at: http://stamp.stanford.edu/

[USC] University of Southern California “Information on FlexSim1.2.” Available at: http://ceng.usc.edu/smart/FlexSim/flexsim.html

[UVa] University of Virginia. “Stream Home Page”. Available at: http://www.cs.virginia.edu/stream/

[UVb] University of Virginia. “STREAM2 Home Page”. Available at: http://www.cs.virginia.edu/stream/stream2/

[UW] University of Washington. "The Chaos Router Simulator." Available at: http://www.cs.washington.edu/research/projects/lis/chaos/www/simulator.html

[VAD06] JS Vetter, SR Alam, TH Dunigan, MR Fahey, PC Roth and PH Worley. "Early evaluation of the Cray XT3". Proceedings of the 20th International Parallel and Distributed Processing Symposium, 2006, 25-29 April. DOI: 10.1109/IPDPS.2006.1639300

[Varg01] A Varga. "The Omnet++ Discrete Event Simulation System". European Simulation Multiconference, Prague, Czech Republic, June 2001.

[VRV07] DC Vasiliadis, GE Rizos, C Vassilakis and E Glavas. "Performance Evaluation of Two-Priority Network Schema for Single-Buffered Delta Networks". The 18th Annual IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07), Athens, Greece, 3-7 September, 2007. DOI: 10.1109/PIMRC.2007.4394153

[WL08] DH Woo, and HHS Lee. "Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era". IEEE Computer Volume 41, Issue 12, December 2008, pp. 24-31. DOI: 10.1109/MC.2008.494

[WOT95] SC Woo, M Ohara, E Torrie, JP Singh and A Gupta. "The SPLASH-2 Programs: Characterization and Methodological Considerations". Proceedings of the 22nd Annual International Symposium on Computer Architecture, Santa Margherita Ligure, Italy, 22-24 June, 1995, pp. 24-36. DOI: 10.1145/223982.223990

[WWF06] TF Wenisch, RE Wunderlich, M Ferdman, A Ailamaki, B Falsafi and JC Hoe, "SimFlex: Statistical Sampling of Computer System Simulation", IEEE Micro, Volume 26, Issue 4, July 2006, pp. 18-31. DOI: 10.1109/MM.2006.79

[YFJ01] Y Yang, A Funahashi, A Jouraku, H Nishi, H Amano and T Sueyoshi. "Recursive diagonal torus: an interconnection network for massively parallel computers". IEEE Transactions on Parallel and Distributed Systems Volume 12, Issue 7, July 2001, pp. 701-715. DOI: 10.1109/71.940745

Annex A. List of Publications

International Journals

JM Cámara, M Moreto, E Vallejo, R Beivide, J Miguel-Alonso, C Martinez, J Navaridas. “Twisted Torus Topologies for Enhanced Interconnection Networks”. Accepted in IEEE Transactions on Parallel and Distributed Systems. To be published.

FJ Ridruejo, J Miguel-Alonso and J Navaridas. "Full-System Simulation of Distributed Memory Multicomputers". Cluster Computing. DOI: 10.1007/s10586-009-0086-y

J Miguel-Alonso, J Navaridas, FJ Ridruejo. "Interconnection network simulation using traces of MPI applications". International Journal of Parallel Programming. Volume 37, Issue 2, 2009, pp. 153-174. DOI: 10.1007/s10766-008-0089-y

International Conferences

J Navaridas, M Luján, J Miguel-Alonso, LA Plana, SB Furber. “Understanding the Interconnection Network of SpiNNaker”. In proceedings of the 23rd International Conference on Supercomputing, June 8-12, 2009,Yorktown Heights, NY, USA, pp. 286-295. DOI: 10.1145/1542275.1542317

MM Khan, J Navaridas, AD Rast, X Jin, LA Plana, M Luján, JV Woods, J Miguel-Alonso and SB Furber. “Event-Driven Configuration of a Neural Network CMP System over a Homogeneous Interconnect Fabric”. 8th International Symposium on Parallel and Distributed Computing. June 30-July 4 2009. Lisbon, Portugal.

J Navaridas and J Miguel-Alonso. “Realistic Evaluation of Interconnection Networks Using Synthetic Traffic”. 8th International Symposium on Parallel and Distributed Computing. June 30 to July 4 2009. Lisbon, Portugal.

JA Pascual, J Navaridas and J Miguel-Alonso. “Effects of Topology-Aware Allocation Policies on Scheduling Performance”. 14th Workshop on Job Scheduling Strategies for Parallel Processing. Rome, Italy May 29, 2009. In Conjunction with IPDPS 2009

J Navaridas, JA Pascual, J Miguel-Alonso. "Effects of Job and Task Placement on Parallel Scientific Applications Performance". Proc 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. Weimar, Germany, February 18-20, 2009, pp. 55-61. DOI: 10.1109/PDP.2009.53

J Navaridas, J Miguel-Alonso, FJ Ridruejo. "On synthesizing workloads emulating MPI applications". The 9th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, Miami, Florida, USA, April 14-18, 2008. DOI: 10.1109/IPDPS.2008.4536473

FJ Ridruejo, J Navaridas, J Miguel-Alonso and C Izu, "Realistic Evaluation of Interconnection Network Performance at High Loads", 8th International Conference on Parallel and Distributed Computing Applications and Technologies, Adelaide, Australia, December 3-6, 2007, pp.97-104. DOI: 10.1109/PDCAT.2007.73

FJ Ridruejo, J Miguel-Alonso, J Navaridas. "Concepts and Components of Full-System Simulation of Distributed Memory Parallel Computers". Proceedings of the 16th International Symposium on High Performance Distributed Computing, Monterey, CA, USA, June 25–29, 2007, pp. 225-226. DOI: 10.1145/1272366.1272402

J Navaridas, FJ Ridruejo, J Miguel-Alonso. "Evaluation of Interconnection Networks Using Full-System Simulators: Lessons Learned". Proceedings 40th Annual Simulation Symposium, Norfolk, VA, USA, March 26-28, 2007, pp. 155-162. DOI: 10.1109/ANSS.2007.19

JM Cámara, Miquel Moreto, E Vallejo, R Beivide, J Miguel-Alonso, C Martinez, J Navaridas. "Mixed-radix Twisted Torus Interconnection Networks". Proceedings 21st IEEE International Parallel & Distributed Processing Symposium, Long Beach, CA, USA, March 26-30, 2007. DOI: 10.1109/IPDPS.2007.370270

National Conferences

J Navaridas, J Miguel-Alonso, W Denzel. "Thin trees: Cost-Effective Tree-like Networks". Actas de las XX Jornadas de Paralelismo. A Coruña, 16-18 Septiembre 2009.

J Navaridas, M Lujan, J Miguel-Alonso, LA Plana and SB Furber. "Evaluation of a Large-Scale SpiNNaker System". Actas de las XIX Jornadas de Paralelismo. Castellón, 17-19 Septiembre 2008.

MM Khan, J Navaridas, LA Plana, M Luján, JV Woods, J Miguel-Alonso and SB Furber. “Configuring a Large-Scale GALS System”. 20th United Kingdom Asynchronous Forum, Manchester, 1st-2nd September 2008

MM Khan, J Navaridas, X Jin, LA Plana, JV Woods and SB Furber. “Real-time Application Support for a Novel SoC Architecture”. 4th United Kingdom Embedded Forum, Southampton, 8-9 September 2008

FJ Ridruejo, J Miguel-Alonso y J Navaridas."Full-system simulation of distributed memory parallel computers using Simics". Actas XVIII Jornadas de Paralelismo. Zaragoza, Septiembre 2007.

FJ Ridruejo, J Navaridas, J Miguel-Alonso, C. Izu. "Evaluation of Congestion Control Mechanisms using Realistic Loads". Actas de las XVII Jornadas de Paralelismo. Albacete, Septiembre 2006.

Technical Reports

FJ Ridruejo, J Miguel-Alonso, J Navaridas. "Full-System Simulation of Distributed Memory Multicomputers". Technical report EHU-KAT-IK-01-09. Department of Computer Architecture and Technology, The University of the Basque Country.

J Navaridas, JA Pascual, J Miguel-Alonso. "Effects of Job and Task Placement on the Performance of Parallel Scientific Applications". Technical Report EHU-KAT-IK-04-08. Department of Computer Architecture and Technology, The University of the Basque Country.

J Miguel-Alonso, J Navaridas, FJ Ridruejo. "Interconnection network simulation using traces of MPI applications". Technical report EHU-KAT-IK-01-08. Department of Computer Architecture and Technology, The University of the Basque Country.

J Navaridas, J Miguel-Alonso, FJ Ridruejo, Wolfgang Denzel. "Reducing Complexity in Tree-like Computer Interconnection Networks". Technical report EHU-KAT-IK-06-07. Department of Computer Architecture and Technology, The University of the Basque Country.