INSEE: an Interconnection Network Simulation and Evaluation Environment

F.J. Ridruejo Perez, J. Miguel-Alonso

The University of the Basque Country,
Department of Computer Architecture and Technology
P.O. Box 649, 20080 San Sebastian (Spain)
{miguel, acbripef}@si.ehu.es

Abstract. In this paper we introduce INSEE, an environment to help in the design of interconnection networks for parallel computing systems. It contains two basic modules: a system to generate traffic (TrGen) and a lightweight functional simulator (FSIN). Additionally, external tools can be integrated into the environment. Examples are SICOSYS (a sophisticated network simulator that provides accurate timing information) and SIMICS (a complete, detailed computer simulator). This environment has been used to conduct some studies of interest in the design of interconnection networks, such as the effect of head-of-line blocking in the injection queues of the network routers, the effects of the injection interface in network performance, and the characteristics of topologies with skewed wraparound links.

1 Introduction

When designing and building a parallel computer, the decision of which interconnection network will be used to link all the processing elements is of crucial importance, because it greatly affects the performance of the system. In the initial design stages, most architectonic proposals (routing algorithms, deadlock-avoidance strategies, topologies, etc.) are commonly tested using fast, functional simulators that do not incorporate all the details of the hardware—while considering the most relevant ones, in such a way that a selection of promising approaches can be made. Then we can proceed to use more detailed simulators, or even develop hardware prototypes.

As important as a good model of the interconnection network is a good characterization of the workload it will have to support, in the same way that processors' design are made taking into account the programs that will run on them. The network has to be designed bearing in mind the way it will be used by parallel applications. The networks for these systems will probably be very different: **(1)** a small-sized SMP; **(2)** a larger CC-NUMA multiprocessor; **(3)** a massively parallel computer (MPP), and **(4)** a distributed system based on web services. In fact, market provides different solutions for each of these needs. We can even go further: it is not the same what we expect from the network when running master-slave applications with infrequent interchange of long messages, that what we demand when running a

fine-grained scientific application where messages are short but interchanged very often. In the first case we need high throughput, while in the second latency is the main constraint.

For these reasons, a set of tools to simulate-evaluate proposals for interconnection networks require a choice of simulators as well as a choice of traffic-generation methods to feed them. This is precisely what we introduce in this paper: INSEE, Interconnection Network Simulation and Evaluation Environment. It includes a fast, functional simulator (FSIN) and a flexible mechanism to generate traffic (TrGen). INSEE is modular, and can be augmented in many ways: with new modules, or adding capabilities to existing modules.

INSEE tools have been successfully used in our research group to carry out different studies of interest in the design of interconnection networks: the effect of head-of-line blocking in the injection queues of the network routers, the effects of the injection interface in network performance, and the performance of topologies with skewed wraparound links. Our aim is to continue improving this tool, and to make it available to other researchers.

The rest of this paper is organized as follows. Section 2 describes the overall design of INSEE and its basic tools: FSIN and TrGen. Section 3 enumerates several lines of research that are being developed with these tools. Section 4 describes tools with the same purpose than INSEE and compares them with our tools. Section 5 summarizes this paper, and indicates work in progress.

2 Structure and elements of INSEE

INSEE is organized as a collection of modules, many of which can be used as standalone applications. Fig. 1 represents the interactions between all the modules. FSIN (Functional Simulator of Interconnection Networks) and TrGen (Traffic Generator) are the core modules. SICOSYS [13], another network simulator, predates INSEE, but can be considered a part of it. The rest of the modules have been developed externally, but can (with some glue code) enhance the capabilities of INSEE, or use it as a source of traffic.

The current version of INSEE uses SIMICS [7] as a full system simulator, and a modified version of MPICH [5] as the source of traces. This will be explained with more detail in the following sections.

2.1 FSIN

FSIN (Functional Simulator of Interconnection Networks) is a flexible tool to help in the design of communication subsystems for parallel computers. Our research group already has a tool like this, SICOSYS [13]. In many aspects, FSIN could be considered a scaled-down version of SICOSYS, because their goals are the same: the general design of what an interconnection network is, and how routers for these networks are. However, FSIN is much faster, because it does not simulate the details of the hardware (that is the reason we call it a "functional" simulator) and consumes much less memory, thus allowing the simulation of large networks: with FSIN we are

routinely simulating networks of 32K nodes using off-the-self equipment, while with SICOSYS the reasonable maximum is 1024.

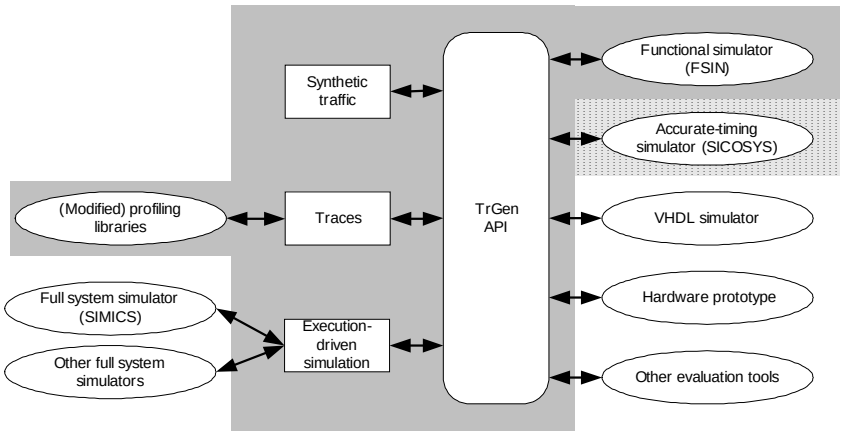

Fig. 1. Overall design of INSEE. Elements with grey background are discussed in this paper. SICOSYS, although developed independently, is considered part of this environment. The remaining parts are external modules.

The downside of FSIN is the accuracy when providing timing information. While SICOSYS includes a very detailed timing model, FSIN only has the notion of "cycle": all the relevant events in the network take exactly one cycle. Related to this, SICOSYS allows for the simulation of pipelined routers, while FSIN does not .

Fig. 2 depicts the basic elements of the simulated routers. Most of these can be parameterized: sizes of buffers and queues, number of dimensions (1, 2 or 3), number of virtual channels (VC) per physical link, uni- or bi-directional links, etc. This is, though, just a subset of the parameters the designer can modify. A non-exhaustive list follows: **(1)** packet size (measured in phits); **(2)** topology of the network: torus, mesh, midimew [2], twisted_torus; **(3)** network size; **(4)** VC management strategy: Dally [4], bubble [12]; **(5)** VC request policy (routing): several options, depending on the previous parameter; **(6)** VC assignment policy: round-robin, oldest, longest, random.

FSIN includes a built-in traffic generator that injects packet with fixed length, a Bernoulli temporal distribution, and a small choice of spatial distributions: uniform, hot-region, transpose, distribute. For more choices, it can be connected to TrGen.

This tool has been developed in ANSI C, so is fully portable to any computing environment. The verbosity of the output report can also be configured: from a short, final summary to a detailed list of all the relevant simulation events.

The most common way of using FSIN is via a batch system (it does not have a graphical interface) to run a collection of experiments in which one or several of the input parameters are changed, and then observe the impact of those changes in the performance of the simulated network. We often focus our attention on the load accepted by the network ("Load Acc.") when the offered load ("Load Prov.") is beyond saturation. This way we obtain an indicator of the maximum traffic the network can manage for a given selection of topology, VC management policies, traffic pattern, etc.

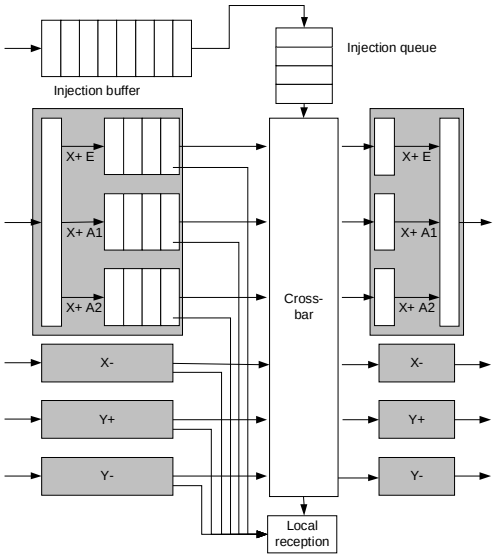

Fig. 2. Router model simulated by FSIN. This particular one is a 2-D router with 3 VCs per physical channel, and 4-phits buffers per VC.

In Fig. 3 (left) we show some results obtained by FSIN, compared to those of SICOSYS. Input parameters are: 2-D torus of 4x4, 8x8 and 16x16 nodes; 3 virtual channels per physical channel (one is the bubble-managed escape channel, using oblivious DOR routing, while the other two are adaptive); packets of 16 phits; queues of 8 packets; applied load between 0.01 and 1.0; 100.000 simulation cycles. The curves shows the load accepted by the network. Notice that results given by the two tools show exactly the same trends and very similar values. Fig. 3 (right) shows the resources used by FSIN compared to those used by SICOSYS, both in memory used and execution time for the longest experiments. Here is where FSIN shows its advantages: experiments consume much less resources.

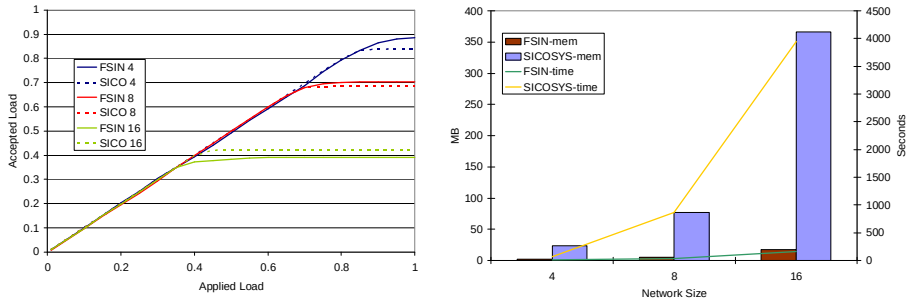

Fig. 3. Left: comparison of results provided by FSIN and SICOSYS for the same input parameters. Right: comparison of resources (memory, execution time) used by these tools.

2.2 TrGen

An actual interconnection network will not be used to randomly move packets from one node to another: it will be used to deliver actual traffic generated by actual applications. The characteristics of this traffic have an enormous impact in network performance and, for this reason, a realistic evaluation of an interconnection network must be performed with actual traffic. There are many reasons, though, to work with approximations: **(1)** actual traffic may be unknown or unavailable; **(2)** agile advance in the initial stages of network development; **(3)** specific testing of particular network characteristics, etc.

TrGen is a traffic generation tool that, via a unified API, allows us to tests our designs with a large variety of traffic sources, as depicted in Fig. 1: synthetic sources, actual traffic taken from traces, and actual traffic taken from an execution-driven simulation.

Synthetic traffic
Synthetic traffic is characterized by three distributions: temporal (that determines the packet inter-arrival times), spatial (destination of packets) and packet-size. The choice of distributions available is extensible, and they can be parameterized. Currently available options are:

- Temporal: Bernoulli, constant bursts, Markov (with several variants).
- Spatial: uniform, distribute, zipf, hot-region, constant (transpose, butterfly, perfect shuffle, inverse, etc.)
- Size: constant, uniform, polynomial.

Traces
Once we have traces taken from the actual execution of a parallel application, we can use them through the TrGen interface to feed our simulations. In order to do this, we have used the profiling capabilities of MPICH [5], a widely used MPI [8] implementation.

One of the limitations of most profiling mechanisms for MPI applications is that they log collective operations as a single event, without showing the actual packets that traversed the network. MPICH implements collective operations via the interchange of point-to-point messages, although this is performed in a hidden context, not visible to applications and not logged. We have modified MPICH in order to make this interchange visible, in such a way that events in trace files are point-to-point interchanges.

Trace files are pre-processed before being used to feed simulations. We remove all timing information, while keeping event order (and causality). We do this because event timestamps are very dependent on the computing platform in use. Without timing information, simulators are forced to work with infinitely fast CPUs—so the network itself is the element that limits execution times.

Execution-driven simulation
The third source of traffic available with TrGen is actual traffic generated and consumed by running applications. To close the simulation circle, we use full system simulators that host simulated computers (guests), which act as the computing nodes running parallel applications. In particular, we use SIMICS [7] because of its flexibility simulating different architectures. Currently we used Pentium 4 (host) machines to simulate generic x86 (guest) machines running Linux.

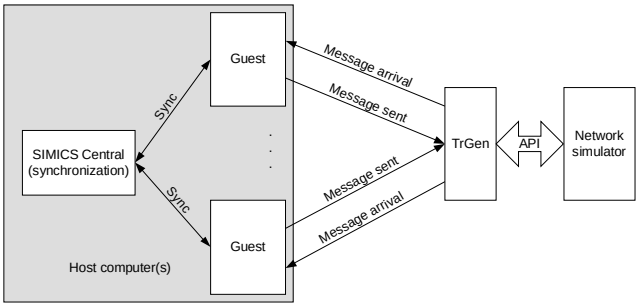

Fig. 4. Execution-driven simulation. A full system simulator (SIMICS) is connected, through TrGen, to a network simulator.

Fig. 4 depicts all the elements involved in an execution-driven simulation. The grey area (left) runs in a SIMICS environment. At the right side we have a network simulator (such as FSIN or SICOSYS) and, in the middle, TrGen acting as the interface between the two worlds. To make this setup work, it has been necessary to develop two pieces of software:

- A SIMICS module that implements a PCI network adapter. A guest sees this adapter as an additional Ethernet card. All MPI communication goes through this adapter. SIMICS is instructed to communicate all activity in this adapter to TrGen.
- A Linux kernel driver to allow the guest machines to use the (simulated) PCI network adapter to interchange messages.

SIMICS Central, a part of the SIMICS environment, also plays an important role, keeping all guest machines synchronized.

3 Research performed with INSEE

We plan to make INSEE tools available to the research community under a liberal licensing (such as GPL). However, some aspects of the code are not yet ready for distribution: user interface is not intuitive, and portions of the code are not yet fully tested. None of those aspects affect the functionality claimed in this paper.

Interested readers can obtain a copy of the software by requesting it directly to the authors. We don't have the (human) resources to provide technical support to

potential users; however, comments on possible improvements, contributed code, and bug reports will be welcomed.

The next subsections introduce work already performed and current, in the context of our research group, where INSEE tools play a crucial role.

3.1 Load unbalance in queue usage

In [10] we report a study of the effect that HOLB (Head-of-line blocking) in the packet injection queue has on the performance of bidirectional k-ary n-cubes, for values of k over a certain threshold (around 20). The HOLB causes an unbalanced use of the channels corresponding to the two directions of bidirectional links, which is responsible for a drop in the network throughput and a rise in the network delay. Simulation results obtained with FSIN (using its built-in uniform traffic generator) show that this anomaly only appears in those rings where most injections are performed (normally, those in the X axis), and that the elimination of the HOLB in the injection queue enables the network to sustain peak throughput after saturation.

This unbalance is also present under actual workloads. Using RSIM (a simulator of multiprocessor systems [11]) integrated with SICOSYS we performed an execution-driven simulation of the Radix application (part of the SPLASH-2 benchmark suite). Checking the usage of queues in the network routers, the unbalance was clearly noticeable. It cannot be attributed to the characteristics of the Radix application (it is not true that nodes in Radix send more data towards one direction that towards the other), because the application interchanges keys in a highly random, uniform way. Thus, this behavior confirms our hypothesis about the occurrence of the anomaly not only with synthetic traffic but also with actual applications.

3.2 Study of packet injection mechanisms

In [6] we analyze the impact that the injection interface has on maximum sustained throughput in an adaptive cut-through torus network. The work described in the previous section pointed out that HOLB at the network interface, due to the use of single FIFO injection queues, may prevent a network from sustaining its peak throughput at heavy loads. Meanwhile, we observed that most recent commercial parallel systems use multiple injection queues [1], but little is known about the rationale behind these design decisions, or their implementation details.

Using FSIN-TrGen we modeled and thoroughly analyzed the effect on performance of the following factors: the number of injection queues (from 1 to 4), the allocation of packets to queues (testing different selection policies, with or without pre-routing at the interface) and the mapping of queues to the available number of injection channels (virtual injection channels vs. physical injection channels). Network evaluations for medium to large size 2D tori showed that designs with multiple FIFO injection queues do not improve performance under uniform traffic, when compared with the simple, single-FIFO interface. On the contrary, for some injection policies, throughput loss increases for loads beyond the saturation point. At the hearth of this behavior was network congestion: more injection ports results in

more pressure from the injection interface to acquire the scarce network resources of an already clogged system.

We concluded that new, restrictive injection policies are required that prevent processing nodes from overflowing routers with new packets for loads beyond the network's saturation point. Interestingly, for small networks, a single injection FIFO queue, with the HOLB it entails, may actually be a good design choice as it indirectly provides the much-needed injection control. For networks with thousands of nodes, as those being implemented in current massively parallel processors, this basic form of congestion control is not enough. Regardless of the number of injectors, an injection-throttling mechanism is essential to reduce throughput losses and maintain, or even increase, maximum sustained throughput.

3.3 Study of torus-like topologies

The interconnection network literature includes many studies devoted to select the best topology for a given parallel computer, application, or combination of those. Most current systems use indirect networks (fat-trees, omega) or direct networks of the k-ary n-cube family. Examples of the later are the 3D-cube for above-mentioned BG/L, or the 3D-mesh planned for Cray's Red Storm.

Still, the work in topologies is not complete: it is still possible to obtain performance gains using the right choices of wiring. Our research group has been for a long time studying the characteristics of networks such as midimews and twisted tori (see Fig. 5), which are similar to tori, but with skewed wrap-around links [2,9].

To demonstrate the properties of this family of networks we are using mathematical analysis as well as simulation—using INSEE.

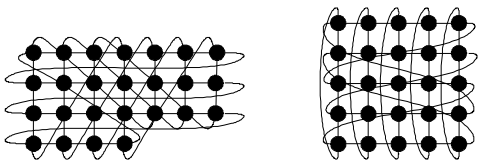

Fig. 5. Left: midimew network. Right: twisted torus with skews 3 (X-axis) and 0 (Y-axis).

4 Related work

There are several tools available from research groups with the same purpose of INSEE. Two of them FlexSim [14] and the Chaos Router Simulator (ChaosSim) [3] are very close to FSIN.

FlexSim is a powerful tool to simulate wormhole networks. Some of its most salient advantages over FSIN are: **(1)** supports full or half-duplex links; **(2)** models faults in nodes / links; **(3)** allows the specification of several delays: routing delay, cut-through delay, switch-to-switch delay, hand-shaking delay and link arbitration delay for half-duplex channels; **(4)** includes a large set of routing and selection

functions. However, it has some downsides. In particular, it neither supports virtual cut-through networks, nor bubble routing. It cannot take traffic from full system simulators, and output information is very limited.

ChaosSim is in many aspects close to the initial versions of FSIN. Its development stopped around 1993, so it does not incorporate most recent advances in interconnection network design. Positive aspects of ChaosSim are its ability of supporting full and half-duplex links, and to simulate wormhole as well as VCT networks. It is also possible to animate simulations using a graphical interface. On the downside, traffic sources are limited to the usual set (uniform, hot-spot and some permutations), and all characteristics of the simulated router have to be included in source files, so a change in a design parameter requires recompilation.

We could have decided to take one these simulators as the foundation for FSIN, adapting it to our needs. However, that was not an easy task: FlexSim is too circumscribed to wormhole routing, so it is not trivial to adjust it to work with virtual cut-through networks (the ones we are interested on). Something similar happens with ChaosSim: it was developed for a particular design of router, so we would need to change too many things to simulate our routers.

In addition to this, FSIN has evolved very rapidly from its origin to its present state; these modifications have been relatively easy because we have a thorough knowledge of its internals. It would be necessary to fully understand even the most minor detail of ChaosSim or FlexSim to adapt them to our needs—and that is not an easy task to do with external code. The design and source code of FlexSim is not documented, so it is very hard to understand; in contrast, ChaosSim is better organized and documented.

As we stated before, it is our intention to release the source code of FSIN (and all the INSEE modules) under a liberal license, so we are taking special care providing a well organized and documented product.

We would like to provide a comparison of FSIN, FlexSim and ChaosSim in terms of metrics such as execution time and memory usage for a given configuration. However, due to the differences among the tools, it has been impossible to find a network definition suitable to be simulated in the three of them.

5 Conclusions and future work

Although INSEE is still a work in progress, the modules that constitute this environment have already proven their usefulness when researching interconnection network topics. We are pleased with the performance of the standalone FSIN and integration tests with TrGen are also satisfactory.

Clearly, future work includes the improvement of each tool, as well as the integration capabilities with external tools. Some ideas already stated in this paper are summarized here. We want to improve TrGen in terms of additional sources of synthetic traffic, more accurate time modeling in trace-driven simulation, and full integration with SICOSYS. Immediate plans for FSIN include the ability of modeling pipelined routers. All these improvements will have a common goal: maximization of our capabilities to model, simulate and evaluate interconnection networks.

Acknowledgements

This work has been done with the support of the Ministerio de Educación y Ciencia, Spain, under grant TIN2004-07440-C02-02, and also by the Diputación Foral de Gipuzkoa under grant OF-846/2004.

SICOSYS has been designed by the ATC Group at the University of Cantabria (Spain). This group provided the data for Fig. 3, as well as invaluable help for the design and implementation of the tools described in this paper.

Work described in Section 4 has been carried out in collaboration with the ATC Group and with Dr. Cruz Izu (U. of Adelaide).

References

[1] N.R. Adiga et al. (2002). An overview of the BlueGene/L Supercomputer. Supercomputing 2002 Technical Papers, Available at http://sc-2002.org/paperpdfs/pap.pap207.pdf.

[2] R. Beivide E. Herrada, J.L. Balcazar, A. Arruabarrena (1991). Optimal distance networks of low degree for parallel computers. IEEE Transactions on Computers. Vol. 40, No. 10.

[3] The Chaotic Routing Project at the U. of Washington. Chaos Router Simulator. Available at http://www.cs.washington.edu/research/projects/lis/chaos/www/chaos.html

[4] W. J. Dally and C. L. Seitz (1987). Deadlock-free message routing in multiprocessor interconnection networks, IEEE Transactions on Computers, vol. 36, no.5.

[5] W. Gropp, E. Lusk, N. Doss, A. Skjellum (1996). A high-performance, portable implementation of the MPI message passing interface standard, in Parallel Computing, vol. 22, no. 6.

[6] C. Izu, J. Miguel, J.A. Gregorio, R. Beivide (2005). Packet Injection Mechanisms and their Impact on Network Throughput. Submitted.

[7] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hållberg, Johan Högberg, Fredrik Larsson, Andreas Moestedt, Bengt Werner, (2002). Simics: A Full System Simulation Platform, IEEE Computer, February.

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

[9] J. Miguel, C. Izu, A. Arruabarrena, J. García-Abajo and R. Beivide (1991). Toroidal networks for multicomputer systems. Proc. of the ISMM International Workshop on Parallel Computing. Trani (Italy).

[10] J. Miguel-Alonso, J.A. Gregorio, V. Puente, F. Vallejo and R. Beivide (2004) Load Unbalance in k-ary n-cube Networks. Lecture Notes in Computer Science 3149.

[11] V.S. Pai, P. Ranganathan, and S.V.Adve (1997). RSIM: An Execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors. IEEE TCCA New., Oct.

[12] V. Puente, C. Izu, R. Beivide, J.A. Gregorio, F. Vallejo and J.M. Prellezo (2001). The Adaptative Bubble Router. Journal of Parallel and Distributed Computing. Vol 61 - n. 9.

[13] V. Puente, J.A. Gregorio, R.Beivide (2002). SICOSYS: An Integrated Framework for studying Interconnection Network in Multiprocessor Systems, Proceedings of the IEEE 10th Euromicro Workshop on Parallel and Distributed Processing. Gran Canaria, Spain.

[14] SMART group at the U. of Southern California. FlexSim 1.2. Available at http://ceng.usc.edu/smart/FlexSim/flexsim.html