A Case Study of Trace-driven Simulation for Analyzing Interconnection Networks: cc-NUMAs with ILP Processors*

V. Puente, J.M. Prellezo, C. Izu†, J.A. Gregorio, R. Beivide

Abstract--

The evaluation of network performance under real application loads is carried out by detailed time-intensive and resource-intensive simulations. Moreover, the use of ILP processors in cc-NUMA architectures introduces non-deterministic memory accesses; the resulting parallel system must be modeled by a detailed execution-driven simulation, further increasing the evaluation cost.

This work introduces a simulation methodology, based on network traces, to estimate the impact that a given network has on the execution time of parallel applications. This methodology allows the study of the network design space with a level of accuracy close to that of execution-driven simulations but with much shorter simulation times. The network trace, extracted from an execution-driven simulation, is processed to substitute the temporal dependencies produced by the simulated network with an estimation of the message dependencies caused by both the application and the applied cache-coherent protocol. This methodology has been tested on two direct networks, with 16 and 64 nodes respectively, running the FFT and Radix applications of the SPLASH2 suite. The trace-driven simulation is 3 to 4 times faster than the execution-driven one with an average error of 4% in total execution time.

I. INTRODUCTION

Scalable distributed shared-memory (DSM) systems, represented by the cache coherent non-uniform memory access (cc-NUMA) systems, are emerging as the trend in building parallel systems since they provide much desired programmability. The network delay when accessing remote data is one of the basic overheads that limits parallel performance. Therefore, the design of interconnection networks that are optimized to handle DSM traffic is an important area of research.

The evaluation of interconnection network designs for a given parallel architecture is carried out via network simulation. In the first stages of design, networks are compared using synthetic loads, which provide a first approximation to real applications [2]. Notwithstanding, the analysis of the impact that the interconnection network has on system performance can hardly be forecast unless we use realistic application loads, similar to those the network is being designed to support.

The traffic in a cc-NUMA system, generated implicitly by the application's memory accesses, depends on the cache-coherent protocol used. Thus, precise loads can only be obtained by execution of the selected applications under a real or simulated system. The monitoring of a real system alters the network traffic, and it is constrained by the given network subsystem. In short, the extraction of realistic application loads can only be achieved using execution-driven simulations of the whole multiprocessor system with the high computational cost they entail.

This work tries to reduce the cost of evaluation of the interconnection network for a cc-NUMA system by introducing a new simulation methodology. Network traces are extracted from the execution driven simulation, processed to approximate the dependencies of cc-NUMA traffic, and then used by a trace-driven simulator to evaluate the network performance. This new methodology is compared with the original execution-driven simulation in terms of simulation cost and accuracy.

Our evaluation will show that the new proposal introduces an average error around 4%, and succeeds in reducing execution time by a factor of 2 to 4. We intend to use this trace-driven simulation as a partial substitute for execution-driven simulation, which should still be used in the latter stages of the design. The impact on the network design cycle will be quite large because of the cumulative speed-up in successive refinements of the router design.

The rest of the paper is organized as follows. Section 2 reviews the most relevant simulation methodologies for interconnection networks. Section 3 describes the trace-driven simulation with feedback. Section 4 describes the application of our methodology for performance evaluation and discusses the results and, finally, in Section 5 we draw some conclusions.

University of Cantabria, Spain, e-mail:{vpuente, prellezo, ja, mon}@atc.unican.es., †University of Adelaide, Australia, e-mail: cruz@cs.adelaide.edu.au

* This Work is supported in part by TI98-1162-C02-01.

II. Application loads for analyzing Interconnection Networks.

Various simulation techniques can be used to model network loads in a cc-NUMA system, depending on the accuracy of the prediction desired and the complexity of the simulator itself. On one extreme of the scale, we can generate accurate network loads from a parallel application by simulating the complete parallel system, including each processor, the memory hierarchy and the network subsystem. By simulating the application, instruction by instruction, we can generate the exact memory references, and capture the requests passed from the memory simulator to its network interface. This method provides very accurate results, but the simulator itself is very costly in terms of both development and computation.

On the other extreme, we can make simulation more affordable by using an abstract model that reflects some statistical properties of current loads such as the random traffic model or other widely used permutation patterns. However, this approach does not capture the details of a specific application, and therefore, it cannot accurately predict its performance.

Trace-driven simulation is a third approach with a level of complexity in between the above two. The trace is a time ordered sequence of events (message generation events in our case) that happened during the execution of the application. If the trace is carefully extracted and processed, it can be powerful enough to represent all the necessary characteristics of the application load. We must use a detailed simulator to obtain an accurate trace, with the associated cost, but we can reuse the trace as many times as needed during the network design cycle.

Trace-driven simulation has been widely used in studies of memory hierarchies of uniprocessor systems. This strategy is limited by the characteristics of the simulated system. For example, memory traces assume that the sequence of addresses accessed is deterministic; this is correct for simple processors but is not applicable to ILP processors with out-of-order execution and non-blocking loads, which can cause large errors [6].

Like simple processors, message-passing architectures are easier to model using traces because of the deterministic and explicit ordering in which the messages are generated by the application. There is a large body on research of interconnection networks from this point of view [2]. However, network traffic in a cc-NUMA system is not deterministic because the characteristics of the network, contention for instance, will influence on the number of messages generated by the cache-coherent protocol. In addition, the processing nodes are ILP processors. Hence, the lack of trace-driven simulations for this environment.

For this reason, our first approach to model application workloads was based on an existing execution-driven simulator of a cc-NUMA machine with ILP processors called RSIM[3]. As our research is focused on network design, we substituted its simple network module NETSIM by our own network simulator SICOSYS [9], which takes into account the key parameters from the router's low-level implementation. The resulting system, RSIM-S reflects quite accurately the impact that a particular network design will have on the application's execution time [10]. The only drawback is the simulation time, e.g. simulating the execution of Radix on a 64-node machine takes approximately 11 hours on a R10000 processor at 200 MHz for the selected problem size. The volume of simulations required to refine and thoroughly test a new network design renders this approach unworkable, even for systems with a low number of nodes.

Recent studies have attempted to reduce the computational cost of execution-driven simulators in the context of analysis of memory hierarchies, using direct execution [5] which replaces full processor emulation with a functional and timing model. Another proposal modeled the benefits of ILP by increasing the clock rate of a simple processor by a factor equal to the ILP processor's peak instruction issue rate. However, such an approximation is a source of large errors as shown in [6].

Nevertheless, the complexity of the memory subsystem cannot be reduced in a similar way so we explore another avenue to speed up network simulation based on the use of traces. Trace extraction in a cc-NUMA system with ILP processors is not a straightforward process due to the non-deterministic characteristics of memory access and the influence of the network performance on the cache coherence protocol. Next section will describe our approach to trace extraction and post-processing.

III. Trace-Driven Simulation with Network Feedback (TDS-NF).

We proposed a new methodology for modeling the impact of network design on application loads. An application's network trace is obtained from the execution-driven simulator RSIM-S, which then feeds our network simulator SYCOSYS. We called this simulation strategy Trace-driven simulation with network feedback (TDS-NF). If the trace information is correct, the resulting trace-driven simulation will model the network effect on the application's execution time with an accuracy close to that of RSIM-S but with a much lower computational cost.

The network trace should reflect the key parameters of a network load: number of messages and their distribution both in time and destinations. Most traffic in a cc-NUMA system follows a reactive pattern: a remote memory request triggers the transmission of the requested data, and the processing of the received remote data may trigger other remote accesses. Similarly, cache coherence control packets

are triggered by the movement of remote data. The application's trace should reflect this property by recording the time when a message is generated as relative to the message that triggered its generation as shown in Figure 1.

For example, Figure 1.(a) represents the communication of the processes P_1 and P_0, as observed during trace extraction. Process P_0 sends a message m_1 to process P_1 at time t_1, and, after some computations, process P_1 sends message m_2 at time t_2 as the reply to message m_1.

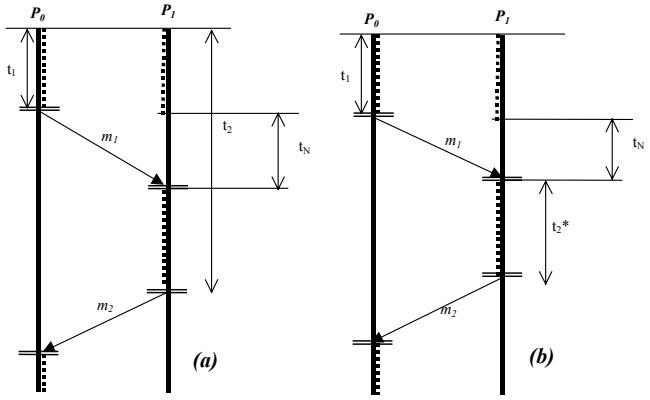

Fig. 1.- Scheme of (a) absolute temporization and (b) relative temporization.

The former trace, showing absolute times, does not encapsulate the impact of the network on the trace extraction. After the arrival of message m_1, which experimented a network latency of t_N units, process P_1 carries out some computations and generates message m_2.

Figure 1.(b) shows this dependency between network performance and the time generation for m_2, by making the time t_{2*} relative to the arrival of the message m_1. Thus t_{2*} only reflects the computation time, while t_2 includes also the communication cost.

In short, the generation time for each message depends on the current network status. Thus, we must provide a bi-directional flow of information between the network simulator and the module that interprets the trace to generate the remote memory references. This feedback mechanism distinguishes our approach from conventional trace-driven simulation of networks in which the message generator feeds messages to the simulator solely based on the trace information.

The goal of trace post-processing is to identify the time dependencies (t_Ni*)* ntroduced by the network employed when the trace was obtained. Thus, any change in network design will change the relative time at which messages are generated and delivered, and the trace-driven simulation will reflect its impact on the application's execution time.

Obviously, this temporization is oriented to reflect the changes in the communication subsystem. Severe changes in the processor architecture or in the memory hierarchy could invalidate previous traces as representative of the application loads for that new system.

A. Applying this methodology to cc-NUMA architectures.

1) Extraction of message dependencies.

Finding the correlation amongst messages described in the previous section is not an easy task in cc-NUMA architectures. It is not the application code that implicitly generates messages but the memory hierarchy subsystem. Furthermore, ILP processors that use non-blocking loads add complexity because message dependencies are then indirect.

To be able to deal with this problem we have ignored any changes in message ordering that are not causal. A first approximation consists of considering the last received message at that processing node as the cause of any outgoing messages. Figure 2 illustrates this approach. In this example, the reception of message m_{th} at node P_3 triggers this node to send three messages, two for node P_2 and one for node P_1. Let's assume that m_{th} was a request for a cache line located at P_3. Our approximation is correct in considering this reception as the trigger for m_1, the message that contains the data requested. The time t_2 models the time to access P_3's directory and locate the requested data. However, the remaining messages may or may not be related to the reception of m_{th}. They may be request messages or replies to previous requests.

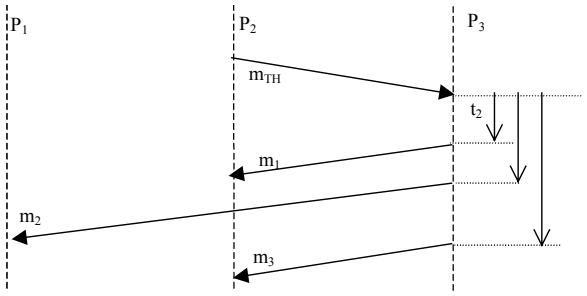

Fig. 2.- Initial approximation.

Due to the reactive nature of the DSM traffic, in which each request will trigger a reply, it is easy to identify such message pairs. A request message arriving at node *d* from node *s* will trigger, at least, one message reply from node *d* to node *s*.

Besides, we have to consider what is the trigger for the control packets that implement the cache-coherent protocol. This requires an in-depth analysis of the chosen protocol, which in our case is the invalidation-based MESI directory cache-coherent protocol.

Type of message received	*Type of messages triggered.*	
READ_SH	REPLY_SH	DATA
	REPLY_EXCL	
	COPYBACK	COHERENCY
READ_OWN	REPLY_EXCL	DATA
	RELPY_EXCLDY	
	COPYBACK_INVL	COHERENCY
	INVL	
UPGRADE	RELPY_EXCLDY	DATA
	REPLY_EXCL	
	REPLY_UPGRADE	COHERENCY
	INVL	
ANY REPLY	READ_SH / READ_OWN / UPGRADE	

Table 1.- Trigger relationships between message types.

Considering all these criteria, we have constructed a table (see Table 1) that describes the correlation between the type of messages received and the set of messages it may trigger (only appear message types seen in the network).

As well as finding the trigger message based on this type of relationships, it is also necessary to compare both the source-destination pair and the memory addresses of the trigger and triggered packet.

This approach identifies the trigger for sending a data packet, but we still have to find a trigger for the original request. We have taken a rough approximation that consists of assuming that request messages are triggered by the last data packet received at that node.

2) Trace format

The trace extraction records one entry for each packet sent, containing the basic packet information and its generation time. The message label records the memory address that the packet is referring to.

During trace post-processing, the dependencies with other messages are identified as described in the previous section and three additional fields created: the *trigger label*, the *trigger delay* and the *trigger count*, as shown in Figure 3.

The trigger label indicates the message that must be received prior to this message generation. The trigger delay specifies the time elapsed between the reception of the trigger label message and that generation (t_{2*} as shown in Figure 2.(b)). The trigger count indicates the number of messages that will be triggered upon reception of this message at its destination. Both the message label and the trigger count will travel with the packet, carrying this causal information to the destination node. When the message reaches that node, we must search the messages whose trigged labels match the received message label. All messages found are trigged after their corresponding trigger delays.

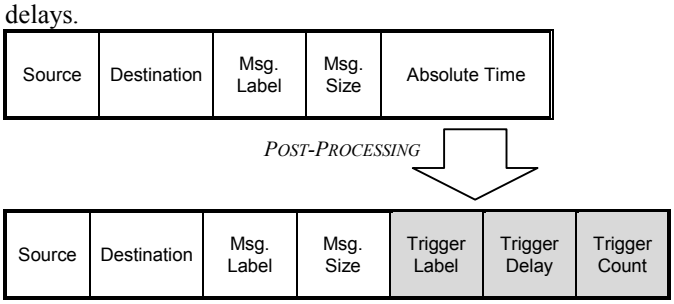

Fig. 3.- Trace entry format.

IV. VALIDATION OF TDS-NF METHODOLOGY.

A. Trace extraction.

As the previous section has described, we had to take some rough approximations in order to reduce the complexity of the cc-NUMA system with ILP processors. This fact may suggest that the level of accuracy of the network evaluation will drop when using trace files. Thus, we need to estimate the validity of these approximations by comparing the results of the trace-driven simulation with those of the accurate but time-intensive execution-driven simulation.

Figure 4 shows the process carried out to validate our methodology. As we mentioned before, we already have the simulator RSIM-S, which combines the cc-NUMA simulator RSIM with our own network module SICOSYS.

The trace is extracted from this simulator by capturing and post-processing all the references sent to the network interface as described in the previous Section.

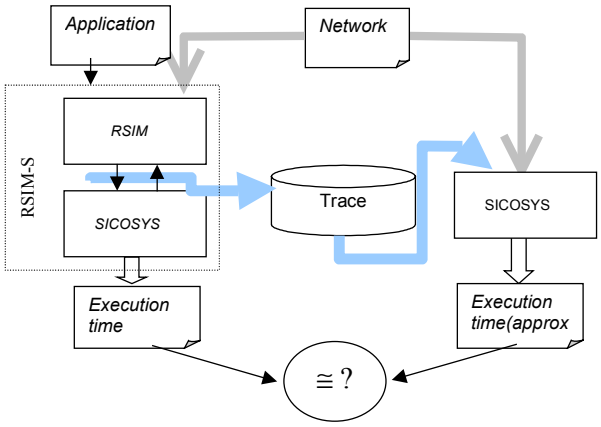

Fig. 4.- Trace extraction and validation of trace-driven results.

Once we have a trace that represents the load generated by one application, we can validate our new simulator in two phases. The first phase runs the trace-driven simulator using the same network configuration applied during trace extraction. As both simulators are emulating the same

parallel system, the execution time should be similar, the more accurate the trace, the closer the results.

As we mentioned before, the advantage of having a trace which is independent of the given network configuration is that it will reflect the impact of changes to that network on the application's execution time. Therefore, the second phase runs the TDS-NF simulation with the same trace, after changing the network configuration. Again, we run the execution-driven simulation with the new network configuration and use its results as the baseline for accuracy, as shown in Figure 5.

All simulations run on a SGI Power Challenge LX, with MIPS R10000 processor at 200 MHz, 1 GB of RAM and 2 Mb of L2 cache. Both simulators were compiled and optimized by the C/C++ MIPSPro 7.2 compiler.

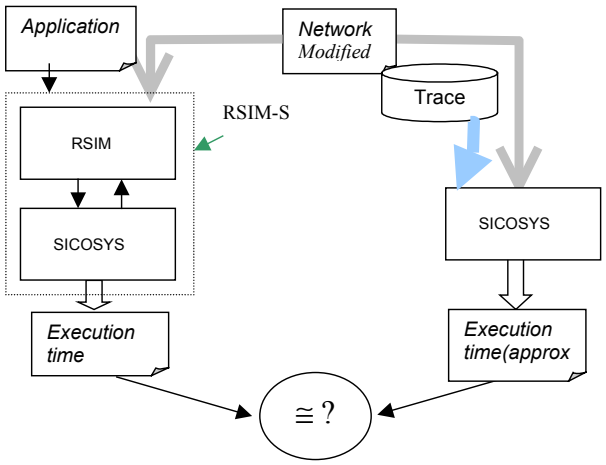

Fig. 5.- Validate the usage of the trace to analyze the impact of network changes.

B. Evaluation of TDS-NF for two applications: FFT y Radix.

We run the two-phase validation process for two applications - FFT and Radix, both from the SPLASH-2 suite [12].

We chose these applications because they are communication-intensive, so they are very sensitive to the characteristics of the interconnection network. We considered two system sizes: 16 nodes and 64 nodes. The problem size was 64K complex double points for the FFT, and 512K integer keys with a maximum radius of 512K for Radix.

The network does not play a critical role in the initialization and completion phases of parallel applications, so these two stages are eliminated to speed up testing.

The system configuration for the memory and processor subsystem is the default one provided by RSIM [3]: 600 MHz nodes with 32-byte cache lines for both L1 and L2. The size of cache coherence commands is 8 bytes.

The network configuration for the first phase is a 2D mesh with adaptive routing. Requests and replies share the same physical medium. The network channels are 4 bytes wide and the network rate is 275 MHz. The router uses Virtual Cut-Through (VCT) flow control and has two virtual channels: a deterministic one and a fully adaptive one. Each virtual channel has a buffer attached with capacity for 4 data packets (160 bytes).

Deadlock due to overflow of the consumption channel capacity is prevented by providing a large buffer size at the network interface's consumption ports (with N request-packet capacity for a network of N nodes), and restricting the number of pending memory references in face of a possible overflow.

The number of messages traced from each application under these conditions is shown in Table 2.

Radix		FFT	
16 Nodes	64 Nodes	16 Nodes	64 Nodes
564,189	1,220,718	311,985	342,107

Table 2.- Number of messages generated by each application under the two system sizes.

In the second phase of validation, we have considered three changes to the network configuration as follows:

- Change topology, from a 2D mesh to a 2D torus. The adaptive torus implementation is based on [11].
- Reduce the channel width from 4 bytes down to 2 bytes.
- Double the network clock rate.

These three changes in network configuration will have a significant impact on the application's execution time. We selected such severe changes so that any error introduced by the tracing mechanism will be magnified under such different scenarios.

Table 3 shows the set of results obtained from the validation process: the speed-up achieved by the TDS-NF simulation and the error in the execution time estimation when compared to the RSIM-S baseline.

Note that the differences in execution time between the TDS-NF simulation and the baseline are always less than 10%, for any of the 16 cases considered. As you can see in table 4, the average error is just above 4%. This level of accurracy is quite satisfactory considering the approximations discussed in section 3.

The *severe* changes in network configuration have a great impact on application execution time and this is reflected in the simulation results. For example, halving the router bandwidth resulted in a significant increment of the execution time of FFT-64 (16) by 70% (30%) as estimated by the execution-driven simulation, or by 65% (31%) as estimated by the TDS-NF simulation.

Obviously, the other two changes increased network performance and resulted in shorter execution time for all the applications. The largest average error corresponds to the networks with a clock rate of twice the original one. This may be due to our conservative correlation between messages which may not correctly model non-blocking accesses, but attributes to them false triggers that may delay their generation unnecesarily in a faster network.

The speed-up achieved by using traces ranges from 200% to 400%. For example, the time to *run* Radix in a 64 node system (RADIX-64) is reduced from 11 hours to 3 hours.
We should point out that SICOSYS, being a low-level network emulator, exhibits a higher computational cost than other functional network simulators. On the other hand, it provides the necessary accuracy to model a particular VSLI router design. In short, SICOSYS complexity limits the speed-up (%) achieved by eliminating RSIM's memory hierachy and processor modules.

Considering the number of simulations needed for a complete network design, the time reduction is very considerable and clearly outweighs the cost of trace extraction and processing.

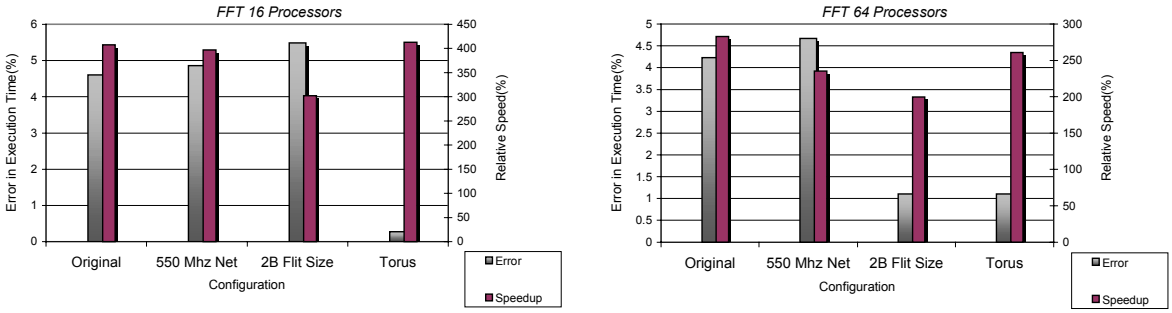

Fig. 6.- Relative error by the trace-driven simulation when compared to the execution-driven simulation for the FFT application.

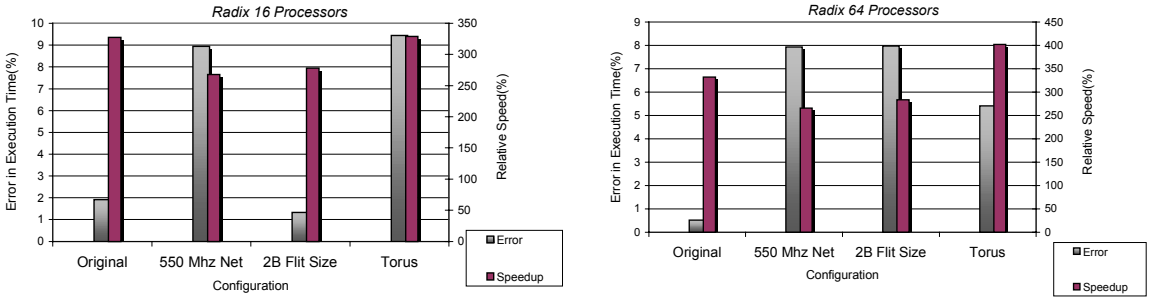

Fig. 7.- Relative error by the trace-driven simulation when compared to the execution-driven simulation for the Radix application.

Configuration	FFT16		FFT64		RADIX16		RADIX 64			
	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)	Error (%)	SpeedUp (%)
Baseline	4.6	407	4.23	283	1.99	328	0.37	332	**2.8**	**338**
Net. Speedx2	-4.85	397	4.67	235	8.94	268	7.93	265	**6.59**	**291**
Flit Size/2	5.48	302	1.1	200	1.33	278	7.97	283	**3.97**	**265**
Torus Network	0.28	412	-1.1	261	9.45	330	5.76	402	**4.14**	**351**
Average	**3.80**	**379**	**2.75**	**245**	**5.45**	**301**	**5.55**	**396**	***4.37***	***311***

Table 3.- Speed-up and error obtained by the TDS-NF simulation when compared with the execution-driven simulation (average error calculated using absolute values).

	+20%Clock		-20%Clock	
	RSIM-S	TDS-NF	RSIM-S	TDS-NF
Radix 16 Processors	7.5%	8.69%	-5.63%	-1.5%
Radix 64 Processors	8.5%	6.45%	-10.1%	-15.5%
FFT 16 Processors	7.2%	8.7%	-4.6%	-10.0%
FFT 64 Processors	14.6%	11%	-15.3%	-13.3%

Table 4.- Change (%) in execution time estimated by simulation of a faster (slower) network.

We are interested in using traces to estimate the the impact of any network redesign on the execution time of parallel applications. Thus, we consider a more realistic variation of ±20% network speed.

Table 4 shows the variations obtained by the execution-driven simulation and the trace-driven simulation after increasing (decreasing) the network clock rate.

V.-Conclusions and Future Work.

This paper has introduced a new methodology to analyze the effect that the interconnection network has on the performance of a cc-NUMA system with ILP processors.

A network trace of a given application is extracted from a execution-driven simulation, and processed to convert absolute-time references into network-relative ones. The processed trace encapsulates the reactive properties of DSM traffic. The message generator interprets this trace and the network feedback information to timely generate the remote memory references. This feedback mechanism distinguishes our approach from conventional trace-driven simulation. The same trace can be repeatedly used to evaluate the impact of different network configurations on the initial application at a much lower computational cost.

This methodology has been tested over direct networks (2D mesh and 2D torus) with 16 and 64 nodes, using the FFT and RADIX applications from the SPLASH2 suite. This approach significantly reduces computational times, 2 to 4 times, with a minimal loss of accuracy. The average error compared with the execution-driven simulation baseline is only 4.4%.

Considering the approximations made during trace processing, and the possibilities for further refining this trace extraction, the TDS-NF methodology is very promising. It can also be extended to model message-passing traffic.

VI.- References

[1] M. S. Lam and R. P. Wilson, "*Limits of Flow Control on Parallelism*", Proc. 19th Annual International Symposium on Computer Architecture, May 1992.

[2] S. Chodenekar et al. "*Towards a Communication Characterization Methodology for Parallel Applications*" Proc. 3rd High Performance Computer Architecture, Feb. 1997.

[3] V. S. Pai, P. Ranganathan, S. Adve *"Rsim: An execution-Driven Simulator for ILP-Based Shared-Memory Multiprocessors and Uniprocessors*", IEEE TCCA Newsletter, Oct. 1997.

[4] A.S. Vaidya, A. Sivasubramaniam, C.R. Das, "*Performance benefits of virtual channels and adaptive routing: An application-driven study*", in Proc. of International Conference on Supercomputing, July, 1997

[5] H. Davis, S.R. Goldschmidt, J. Hennessy, "*Multiprocessor Simulation and Tracing Using Tango*", Proc. *Intl. Conf. On Parallel Processing*, 1991.

[6] M. Durbhakula, V.S. Pai, S. Adve, "Improving the Acuracy vs. Speed Tradeoff for Simulating Shared-Memory Multiprocessor with ILP Processors", Proc. *5rd High Performance Computer Architecture*, Jan. 1999.

[7] C.Holt, J. P. Singh, J. Hennessy, "*Application and Architectural Bottlenecks in Large Scale Distributed Shared Memory Machines*", Proc. 23rd Intl. Symp. On Computer Architecture, May 1996.

[8] M. Rosemblum et al., "Using the SimOS Machine Simulator to Study Complex Computer Systems", *ACM Transactions on Modeling and Computer Simulation*, 1997.

[9] J.M. Prellezo, V. Puente, J.A. Gregorio, R. Beivide, "*SICOSYS: an interconnection network simulator for parallel computers*" Technical Report available at http://www.atc.unican.es/ REPORTS/TR-ATC2-UC98.pdf, June 1998.

[10] V. Puente, J.A. Gregorio, C. Izu, R. Beivide and F. Vallejo, "*Low-level Router Design and its Impact on Supercomputer Performance*", Proc. of International Conference on Supercomputing, June 1999.

[11] V. Puente, J.A. Gregorio, J. M. Prellezo, R. Beivide, J. Duato, C. Izu, "*Adaptive Bubble Router: a Design to Balance Latency and Throughput in Networks for Parallel Computers*", Proc. of International Conference on Parallel Processing, Sept. 1999..

[12] S. C. Woo, M. Ohara, E. Torrie, J.P. Singh, A. Gupta "*The SPLASH-2 Programs: Characterization and Methodological Considerations*". Proc. of the 22nd International Symposium on Computer Architecture, June 1995.