Concepts and Components of Full-System Simulation of Distributed Memory Parallel Computers

Fco. Javier Ridruejo Jose Miguel-Alonso Javier Navaridas
Dep. of Computer Architecture and Technology, The University of the Basque Country
P. Manuel de Lardizabal, 1 (20018) Donostia-San Sebastian, Spain. Telephone number +34 943018019
franciscojavier.ridruejo@ehu.es j.miguel@ehu.es javier.navaridas@ehu.es

ABSTRACT
In this work we discuss a range of approaches to full-system simulation of distributed memory parallel computers, with emphasis on the interconnection network. We present our environment, based on Simics®, and discuss how unforeseen interactions and fine tuning of components can affect results.

Categories and Subject Descriptors
C.1.2 [**Processor Architectures**]: Multiple Data Stream Architectures (Multiprocessors) - *Interconnection architectures*

C.4 [**Performance of Systems**]: Performance of systems - *Design studies, measurement techniques, modeling techniques.*

General Terms
Design, Performance.

Keywords
Full-System simulation, evaluation of interconnection networks.

1. INTRODUCTION
Simulation is a key activity in the design of a massively parallel computer (MPP). Initially it can be done using simplistic models of the computer and the applied workload, but the final, validation phases must be done with full-system simulators that accurately model every component and the workloads it will execute.

A fundamental subsystem of a distributed memory parallel computer is the Interconnection Network (IN from now on) that allows computing elements to communicate and synchronize. This IN must be of low latency and high throughput to efficiently run parallel applications. As we focus our research on IN, in this paper we describe the components needed to do full-system simulation of IN, discuss some alternatives and finally propose our own, exposing the problems arisen like unforeseen interactions, between simulation components.

2. ELEMENTS TAKING PART IN A FULL-SYSTEM SIMULATION OF INs
A full-system simulation of a MPP includes the simulation of both the computation nodes and the IN. We can see the components that take part into the simulation in Figure 1. Computation nodes are simulated using detailed full-system simulators that allow the execution of unmodified, actual software. Parallel applications run and communicate using MPI libraries, protocol stacks and drivers of the unmodified operating system through simulated NICs (Network Interface Cards).

The IN simulator must route messages between computation nodes, simulating accurately the time required to pass through the network. The Traffic Manager module gets and puts messages from computation nodes and the IN, so it is the interface that communicates both kinds of simulators. Finally, the Synchronization module keeps the synchronization among all concurrently running simulators. This is needed to obtain a time-accurate performance evaluation.

HPDC'07, June 25–29, 2007, Monterey, California, USA.
ACM 978-1-59593-673-8/07/0006.

3. INTERFACING NETWORK AND NODE SIMULATORS
There are several approaches to do the interchange of traffic between the IN and the computation node simulators, depending on where packets are extracted. One option could be to substitute the NIC driver with another that intercepts the traffic and passes it to the Traffic Manager. This would require programming a network driver that would be able to interact with the simulated hardware NIC and the protocol stack of the operating system in use. The main advantage of this approach is that we could reuse the simulated hardware NIC, provided by the full system simulator, and leave the protocol stack unmodified. Just the opposite option would be to program a whole set of components for a given IN, which include a simulated hardware NIC, its driver, a proper protocol stack and the MPI library to use. So we would have full control over the communication and results would be very realistic. The drawback of this approach is clear: it requires a huge (and error-prone) software implementation effort.

Usually, full system simulators come with default modules like Ethernet NICs, and can run unmodified OS that include appropriate drivers and protocol stacks. The reutilization of these components allows us to use proven software, reducing implementation effort. Thus, an approach in the middle of the two above is the addition of capabilities to the (simulated) hardware NIC, enabling it to send and receive packets to the Traffic Manager. In our environment we have used this approach, reusing the default Ethernet NICs provided by Simics and the drivers and TCP (UDP)/IP stacks provided by Linux, so our experiments are limited to the use of these components: all the communication is TCP/IP over Ethernet. Figure 2 shows the protocol stacks that can be used in our environment, plus an example of how a Myrinet protocol stack would be.

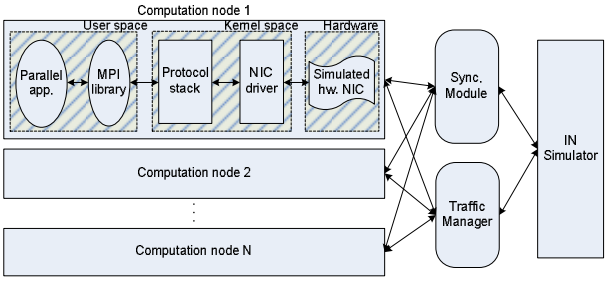

Figure 1. Elements in a full-system simulation of IN

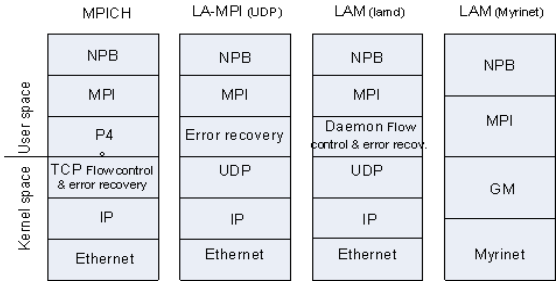

Figure 2. Communication protocol stacks for different MPI implementations and IN.

We have identified two approaches to synchronize computation nodes and IN. One way could be run them in lock-step mode, that is, first run one simulator for an interval of time, then run the other one for the same amount of time and so on. The other way could be running both simulators concurrently and stop them when a fixed amount of simulated time has passed, waiting for the slower one. Both approaches need to be fine-tuned so as not to introduce additional delays which would provide inaccurate results. If simulators are allowed to run out of synch for large periods of time the synchronization overhead is small but the delays introduced are large. On the other hand, frequent synchronization provides accurate results but at the price of longer simulation times.

4. A PROPOSAL FOR FULL-SYSTEM SIMULATION OF INs

Two simulators compose our full-system simulation environment, Simics® [1] used to simulate the computation nodes, and INSEE [2] as the IN simulator. We chose to reuse as much as possible to reduce implementation effort and to minimize errors, so we decided to use a fast Ethernet NIC module provided by Simics. As this simulator can run an actual operating system, we use a Linux distribution (RedHat 7.3) with the usual TCP(UDP)/IP/Ethernet protocol stack. The traffic extracted from the substituted simulated Ethernet is passed to the Traffic Manager that injects it into INSEE, and vice versa. INSEE simulates the passing of messages through the IN. We have also implemented synchronization modules into Simics and INSEE to keep them synchronized.

As both simulators have different perception of time (Simics is event driven and INSEE is cycle driven), the synchronization of the simulators is done in lock-step mode. Each Simics instance includes a synchronization client and INSEE includes a synchronization server. Each client allows Simics to run for a certain time, and then sends a signal to the server and stops waiting a signal to resume. When the synchronization server has received a signal from all computing nodes, INSEE runs for the equivalent number of cycles, and then sends a signal to all clients – which allows Simics instances to resume their executions.

5. EXPERIMENTAL WORK

Our environment for full-system simulation of IN has been used in experiments related to the study of the effects of network-based congestion control. To illustrate our work, we have evaluated a congestion control technique called IPR (in-transit priority) on a 64 node ring with multiple injection sources. Initially, we fed INSEE with actual traces of BT, CG and IS applications from the

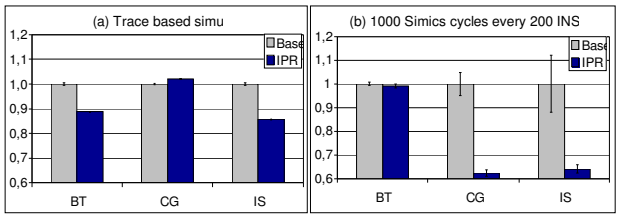

Figure 3. (a) Trace driven simulation. Times to complete a run of BT, CG and IS, relative to Base case (without IPR) and 99% confidence intervals. (b) Full-system simulation of IN with a synchronization of 1000 Simics cycles every 200 INSEE cycles. Simulated network speed is 1280Mb/s.

NAS Parallel Benchmarks (NPB), to obtain a preliminary, very optimistic estimation of achievable performance improvements. The results shown in Figure 3a predict that the reduction of execution time should be at most 15% (IS) and in some cases there should be a performance drop of 3% (CG).

Then, we ran the same benchmarks in our full-system simulator of INs using an MPICH/TCP/IP/Ethernet protocol stack. Results shown in Figure 3b differ from those from traces. We discovered unforeseen interactions between congestion control at network level (IPR) and at host level (TCP). IPR helped to reduce delay and jitter so TCP works better, and the flow of packets through the network is accelerated. However, without IPR, jitter is higher due to congestion produced at network, and to delays introduced in the synchronization of simulators, so TCP retransmits packets and activates the slow start mechanism, causing a performance drop. Note the wide confidence intervals in the Base case due to the variability of the jitter and the number of packets resent.

6. CONCLUSIONS

The full-system simulation of IN requires a large collection of interrelated components done from scratch or re-used. There are multiple factors that can lead to inaccurate results: interaction among modules, reutilization of components for other purposes than those they were designed for, synchronization between simulators, protocol stacks, MPI implementations, or drivers. Design decisions must be taken carefully to avoid wrong results. Moreover, a trade-off must be found between speed and fidelity of the simulation.

7. ACKNOWLEDGMENTS

This work has been done with the support of the Spanish Ministerio de Educación y Ciencia, grant TIN2004-07440-C02-02. Mr. J. Navaridas is supported by a doctoral grant of the UPV/EHU.

8. REFERENCES

[1] P. S. Magnusson et al. *Simics: A full system simulation platform*. IEEE Computer, 35(2):50–58, February 2002.

[2] F.J. Ridruejo, J. Miguel-Alonso. *INSEE: an Interconnection Network Simulation and Evaluation Environment*. Lecture Notes in Computer Science, Volume 3648 / 2005 (Proc. Euro-Par 2005), pp. 1014 – 102.