Trace-driven Co-simulation of High-Performance Computing Systems using OMNeT++

Cyriel Minkenberg
IBM Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon, Switzerland
sil@zurich.ibm.com

Germán Rodriguez Herrera
IBM Zurich Research Laboratory
Säumerstrasse 4
8803 Rüschlikon, Switzerland
rod@zurich.ibm.com

ABSTRACT

In the context of developing next-generation high-performance computing systems, there is often a need for an "end-to-end" simulation tool that can simulate the behaviour of a full application on a reasonably faithful model of the actual system. Considering the ever-increasing levels of parallelism, we take a communication-centric view of the system based on collecting application traces at the message-passing interface level. We present an integrated toolchain that enables the evaluation of the impact of all interconnection network aspects on the performance of parallel applications. The network simulator, based on OMNeT++, provides a socket-based co-simulation interface to the MPI task simulator, which replays traces obtained using an instrumentation package. Both simulators generate output that can be evaluated with a visualization tool. A set of additional tools is provided to translate generic topology files to OMNeT's ned format, import route files at run time, perform routing optimizations, and generate particular topologies. We also present several examples of results obtained that provide insights that would not have been possible without this integrated environment.

Keywords

High-performance computing, interconnection network, PDES.

1. INTRODUCTION

The design of high-performance computing (HPC) systems relies to a large extent on simulations to optimize the various components of such a complex system. To evaluate processor performance, tools such as MAMBO [1] or SIMICS [2] are used, which can perform cycle-accurate simulations of entire applications at the instruction level. However, such a level of detail prevents scaling of this type of simulation to systems with more than a handful of processors.

The interconnection network of a HPC system is usually modelled at a higher level of abstraction, resorting to discrete-event simulation to enable scaling to systems with hundreds or thousands of network ports. The purpose of interconnection network simulation is to optimize the network topology, switch and adapter architectures and parameters, scheduling and routing policies, link-level flow control mechanism, and end-to-end congestion control mechanism. This type of simulation is commonly of the "Monte Carlo" variety, i.e., applying a synthetically generated workload with random destination and interarrival-time distributions, rather than the load from a real application.

Although such simulations are useful in determining load–throughput and load–delay characteristics, they are not necessarily a reliable performance indicator for the communication phases of specific applications. Therefore, we have the situation that instruction-level processor simulation, although accurate, does not scale to the desired system sizes, whereas interconnect simulation does scale, but suffers from unrealistic stimuli. This gap needs to be bridged to enable true end-to-end full-system simulation.

One class of approaches to bridge this gap employs *trace-driven* simulation. Instead of by an exact model of their behavior, computing nodes are represented by a trace that contains two basic kinds of records, namely computation and communication. Computations are not actually performed, but simply represented by the amount of CPU time they would consume in reality. Communications are transformed into data messages that are fed to a model of the interconnection network. To ensure accurate results, the simulation should preserve causal dependencies between records, e.g., when a particular computation depends on data to be delivered by a preceding communication, the start of the computation must wait for the communication to complete. As many scientific HPC applications are based on the Message Passing Interface (MPI), tracing MPI calls is a suitable method to characterize the communication patterns of an important class of HPC workloads. An example of this approach is the MARS simulator presented in [3].

The simulation framework described here is the result of joint project between the Barcelona Supercomputer Center (BSC) and IBM, in which a follow-on machine to the currently installed *MareNostrum* system is being designed under the working title of *MareIncognito*. MareNostrum is a 2,560-node cluster of IBM JS21 blades, each having two dual-core IBM 64-bit PowerPC® 970MP processors running at 2.3 GHz for a total of 10,240 CPUs. The computing nodes are interconnected by means of a high-bandwidth, low-latency Myrinet® network [5], with each blade having one integrated Myrinet adapter. The switch fabric com-

OMNeT++ 2009, Rome, Italy.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

prises ten 512-port and two 1280-port Myrinet switches arranged in a two-level fat-tree topology. However, as these switches are constructed internally using 32-port switch elements (again arranged in a fat-tree topology), the network really has five levels. The nodes are also connected to a Gigabit Ethernet local area network. MareNostrum has 20 TB of RAM and 280 TB of external disk storage. Peak performance with respect to the LINPACK benchmark is 94.21 teraflops, which ranked it as the world's 26th fastest supercomputer on the Top500 list of June 2008.

Several teams at IBM and BSC are cooperating on key aspects of the design of MareIncognito, including applications, programming models, tools, load balancing, interconnection network, processor, and system modeling. This paper describes part of the effort responsible for the design of the interconnection network.

The remainder of this paper is organized as follows. In Sec. 3 we describe the existing BSC tools Dimemas and Paraver and the IBM interconnection network simulator MARS. Section 4 explains how we integrated these tools to form a co-simulation platform for MPI applications using realistic interconnection network models. Section 5 provides examples of the type of new insights that this new environment can help generate. Finally, we conclude in Sec. 6.

2. INTERCONNECTION NETWORK

The role of the interconnection network in scientific as well as commercial computer systems is of increasing importance. The underlying trend is that the growing demand for computing capacity will be met through parallelism at the instruction, thread, core, and machine levels.

The scale, speed, and efficiency of interconnects must grow significantly, because of (i) increasing parallelism, (ii) the replacement of computer busses by switched networks, (iii) consolidation of heterogeneous communication infrastructure (storage area, local area, and clustering networks; IO and memory busses) onto a single physical network, and (iv) virtualization of computing resources that leads to increased variability and unpredictability in the network load.

As neither busses nor legacy LAN/WANs can meet all datacenter and HPC requirements—notably low latency, high bandwidth, high reliability, and low cost—a number of networking technologies designed specifically for the datacenter environment have emerged. Examples are Fibre Channel, commonly employed in storage area networks, Myrinet [5] and QsNet (Quadrics®), used for low-latency, high-bandwidth inter-process communication in HPC or clustering environments, and InfiniBand, designed as a comprehensive datacenter interconnect.

A consequence of these trends is that the interconnection network is becoming a more significant factor in both the overall system performance and the overall system cost. Accordingly, it is receiving more attention in the modelling of such systems to determine the optimum cost-performance trade-off. This is not possible without a detailed, faithful representation of all aspects pertaining to the network.

3. TOOLS

To perform a detailed performance analysis of parallel programs, BSC developed a simulation tool, Dimemas, and a visualization tool, Paraver. To study the performance of interconnection networks, IBM developed a tool referred to as the *MPI Application Replay network Simulator* (MARS) [3]. As these tools form the basis of our work, we briefly describe them below.

3.1 Dimemas

Dimemas is a tool developed at BSC to analyze the performance of message-passing programs. It is an event-driven simulator that reconstructs the time behavior of message-passing applications on a machine modelled by a set of performance parameters.

The input to Dimemas is a trace containing a sequence of operations for each thread of each task. Each operation can be classified as either computation or communication. Such traces are usually generated by instrumenting an MPI application, although they can also be generated synthetically. During instrumentation, each computation is translated into a trace record indicating a "busy time" for a specific CPU, whereas the actual computation performed is not recorded. Communication operations are recorded as send, receive, or collective operation records, including the sender, receiver, message size, and type of operation.

Dimemas replays such a trace using an architectural machine model consisting of a network of SMP nodes. The model is highly parametrizable, allowing the specification of parameters such as number of nodes, number of processors per node, relative CPU speed, memory bandwidth, memory latency, number of communication buses, communication bus latency, etc. Dimemas outputs various statistics as well as a Paraver trace file.

3.2 Paraver

Paraver is a tool, also developed at BSC, to create visual representations of the behavior of parallel programs. One of the outputs of Dimemas is a Paraver trace that represents the state of each thread at every time during the simulation, all communications between threads, and occurrences of punctual events.

A Paraver trace is a series of records, each one being associated with a specific thread. There are three basic kinds of records: A *state record* specifies the state of a particular thread in a particular time interval. A *communication record* specifies a point-to-point communication between two threads, including the physical and logical start and end times, the size of the communication, and a tag. An *event record* specifies the occurrence of particular event at a particular thread, including the type of event, the time of occurrence, and an associated value.

Although Paraver was developed to analyze the performance of parallel programs, its input trace format is highly generic and can easily be adopted for other uses, see Sec. 4.2.

3.3 MARS

MARS [3] is a simulation framework based on OMNeT++ [4] for end-to-end simulation of HPC systems. It comprises two key components: a computing node model and a detailed interconnection network model. The computing node model comprises submodules representing tasks, the system kernel, processors, and the system bus. Each task is modelled by an MPI task replay engine that reads its input from a trace file containing a sequence of MPI and compute operations, akin to a Dimemas input trace. Tasks can be placed onto nodes in an arbitrary fashion. As an alternative to

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

the trace-driven simulation mode, traffic can be generated randomly by built-in generator nodes, which mainly serves to determine the maximum throughput of the system for GUPS-like (Giga-Updates Per Second) benchmarks.

The network model includes models of the adapter and switch modules. The switch modules are arranged in a fat-tree topology with a configurable number of levels. The main objective of MARS is to optimize the design of the interconnect, including (i) the network topology, (ii) routing policies (iii) the switch hardware implementation, and (iv) the adapter hardware implementation.

MARS also takes advantage of OMNeT++'s built-in support for parallel execution, enabling simulations of at least 65,536 processors on a 32-node cluster.

4. INTEGRATION

System design is tightly coupled to the workload that will be executed on the machine. Accurately simulating entire parallel applications with detailed models of the hardware is a complicated task mainly because of the difficulty of writing a single simulator combining the capability of simulating the sofware and hardware stacks in sufficient detail. One trend has therefore been to simulate the behaviour of an application with simplified models (such as the bus-based model used by Dimemas) and then estimate the parameters of this simplified models to match them either with real components' parameters or with more detailed simulations using cycle-accurate simulators of such components. Another complementary trend has been to feed the detailed hardware simulators with random traffic or synthetic traffic, and drawing conclusions about the hardware design under the assumption that they also apply to the applications.

To optimize the design of the interconnection network of MareIncognito, we need to be able to replay traces from a set of key applications over a detailed network model. Unfortunately, the existing tools did not meet these needs. The network model employed by Dimemas employs a (configurable) set of busses that each connect to all nodes. As such, the effects of network topologies, routing policies, traffic contention, and anything relating to switch implementation could not be studied with Dimemas alone.

Although MARS did provide the necessary, much more detailed, network abstraction level, its trace in- and output capabilities are not compatible with Dimemas and Paraver. In addition, as it was designed to simulate a specific system, it does not provide support for arbitrary network topologies.

To meet the needs of the MareIncognito project we started with the MARS simulator as a basis, and extended it with the following capabilities:

- A *server mode* to support co-simulation with Dimemas via a standard Unix socket interface.
- Output of paraver-compatible trace files to enable detailed observation of network behavior.
- A translation tool to convert Myrinet map files to OMNeT++ *ned* topology description files.
- Import facility to load Myrinet route files at simulation runtime.
- Detailed models of Myrinet switch and adapter hardware.
- A tool to generate any topology belonging to the class of Extended Generalized Fat Tree (XGFT) topologies [8].
- Built-in support for three-dimensional Torus networks and n-dimensional Hypercubes.
- Support for multi-rail networks.
- A flexible mechanism to map Dimemas tasks to network nodes.

Figure 1 depicts the complete toolchain of our simulation environment. We refer to the enhanced network simulator as *Venus*. The following sections describe each of the extensions included in Venus in some detail.

4.1 Server mode

To achieve interoperability between Venus and Dimemas, we implemented a hybrid approach between a *Parallel Discrete Event Simulation* (PDES) and a server/client model. The PDES approach enables a global simulation in which the independent simulators are distributed components. The natural boundary of the co-simulation lies between the detailed simulation of the network (Venus) and the replaying of an application's trace (Dimemas). We extended the PDES framework to make the Venus side act as server and the Dimemas side as client. We defined a communication interface between the two sides that allows one or more Dimemas instances to be plugged in to one or more instances of Venus.

To synchronize the simulators' event schedulers we adopted a conservative version of the "Null Message Algorithm" [10, 11]. We assume that the *earliest input time* of each of the simulators is 0, so that each of the parallel simulators can expect an event from the other one at the current timestamp. The *earliest output time* is set to the next event in the local event queue. Although the algorithm is borrowed from a PDES technique, the actual lookahead settings make it run in a serial way; the simulations take turns to perform at least one action at a time, so that they always progress. To reduce the communication overhead due to the "null messages" between the simulators, these are only exchanged when there are messages in flight. Otherwise, Dimemas runs without synchronizing with Venus until some communication event is reached. On the other hand, when the next event in the Dimemas queue is at a time t strictly greater than or equal to the current time Venus runs without exchanging any messages until that time t, unless an event is processed that could change the state of Dimemas, such as the arrival of a message at the output of the network.

Venus has been extended with a module that acts as a server receiving commands from Dimemas. Upon initialization, a listening socket is opened and Venus awaits incoming connections. Once a client connects to Venus, it can send new-line separated commands in plain text. Venus understands several types of commands, including `STOP` and `SEND`. `STOP` is the actual "null message" exchange: it only serves to inform Venus of the timestamp of the next relevant event in the Dimemas queue. The `SEND` command will force the server module to send a message through the network simulated by Venus. When a message has been received by an adapter, Venus passes it to the server module, which in turn sends a corresponding `COMPLETED SEND` message to Dimemas.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

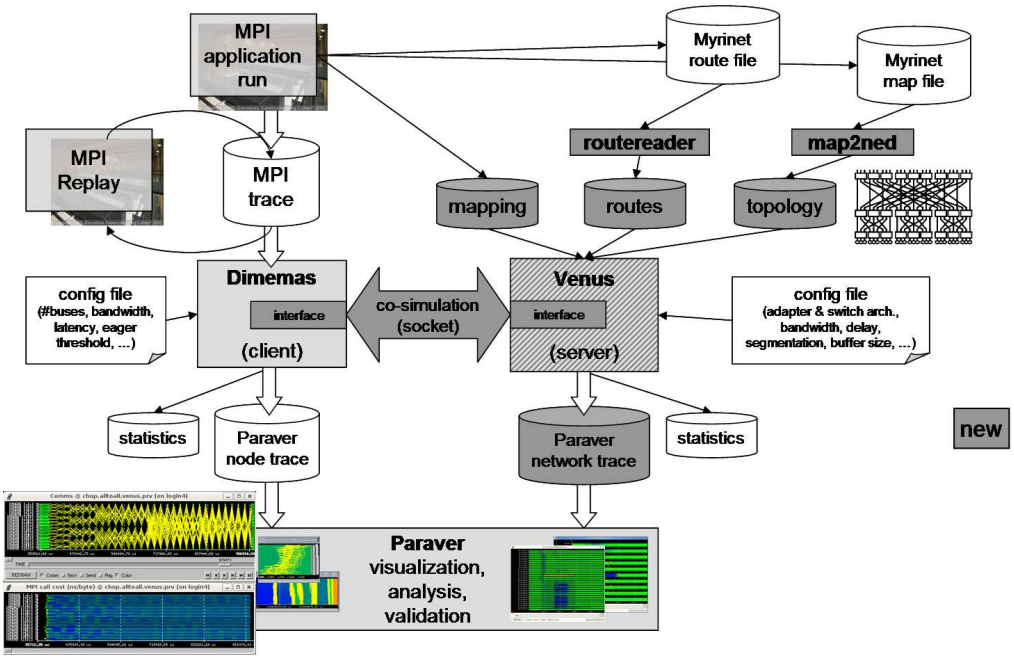

Figure 1: Integrated tool chain.

One of the difficulties we encountered is related to the dynamic precision of floating point numbers. Both OMNeT++ and Dimemas use floating point numbers to store the simulation time. To facilitate human interaction, debugging and extensibility of the integration, we took the implementation decision of using plain text for the communication exchanges between the simulators. When a double is printed in a base 10 representation and then read back, a rounding error can arise. When this error is negative, the simulators' kernels will interpret this as an attempt to insert events in the past. This will cause the simulation to bail out with an error, or, more insidiously, introduce hard-to-find bugs due to these events being ignored. Therefore, we have to check for "time drifts" and advance the scheduling of the event to the current simulation time.

We also observed that the simulation of the two simulators together took much longer than either simulator running independently with the same traffic. This turned out to be caused by the use of Nagle's algorithm by the TCP stack, which introduced large delays (up to 500 ms) at every synchronization point. Using the `TCP_NODELAY` option solved this particular problem by forcing immediate transmission.

4.2 Paraver output

As pointed out in Sec. 3.2, Paraver was originally intended to represent the state of and communication between MPI threads. However, owing to its generic trace format and high level of configurability (association of semantic labels with threads, states, and events) and the myriad ways in which data contained in trace records can be translated to numbers and colors for visual rendering, Paraver is also highly suitable to represent the state of the interconnection network.

We chose the following, natural mapping from network entities to Paraver tasks and threads: Each adapter and each switch is represented by one task. Each port within each adapter and switch is represented as a thread belonging to the corresponding task.

Table 1 shows the structure of all Paraver records, including their semantics, both as originally intended at the MPI level and as newly assigned in the context of the network level. Regarding communication records, the main difference is that the logical and the physical send time now correspond to the first and the last flit of a message. At the network level, there will be one communication record for each link traversal of a given message. In each record, the sending entity corresponds to the transmitting port of the switch, whereas the receiving entity corresponds to the peer port of the receiving switch. The size corresponds in both cases to the size of the MPI message in bytes, whereas the tag uniquely identifies the message.

In each state record the entity identifies the specific switch and port to which the record applies. The state value indicates which state the entity was in from begin to end time. Note that the begin and end times pertain to the state record, but not necessarily to the state per se: there may be subsequent state records for the same object with the same state value. The key difference between state records at the MPI and at the network level is that at the MPI level the states correspond to certain MPI thread (in)activities (idle, running, waiting, blocked, send, receive, etc.), whereas at the network level it represents a buffer-filling level. The actual state value is quantized with respect to a configurable buffer quantum. Also configurable is whether the backlog should be counted per input or per output port.

An event record marks the occurrence of a punctual event. At the network level, we implemented events to flag the issuance of stop and go flow-control signals, the start and end of head-of-line blocking, and the start and end of transmis-

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

Table 1: Paraver state, event, and communication record structures.

field	content	MPI-level meaning	network-level meaning
0	'1'	state record type	same
1	entity	app., task, and thread ID of sending thread	switch/adapter and port ID of port
2	begin time	starting time of state record	same
3	end time	ending time of state record	same
4	state	activity carried out by thread	quantized buffer backlog at port
0	'2'	event record type	same
1	entity	app., task, and thread ID of sending thread	switch/adapter and port ID of port
2	time	time at which event occurred	same
3	event type	type of event	same (but different set of events)
4	event value	value associated with event	same (but different semantics)
0	'3'	communication record type	same
1	sending entity	app., task, and thread ID of sending thread	switch/adapter and port ID of sending port
2	logical send time	time at which send is posted	sender's reception time of first flit of message
3	physical send time	actual sending time of message	sending time of first flit of message
4	recv. entity	app., task, and thread ID of recv. thread	switch/adapter and port ID of recv. port
5	logical recv. time	time at which recv. is posted	reception time of first flit of message
6	physical recv. time	actual message reception time	reception time of last flit of message
7	size	message size in bytes	same
8	tag	message type (MPI operation)	unique message identifier

sion of individual message segments, all at the port level. The semantics of the value depend on the specific type of event.

4.3 map2ned

As MareNostrum uses a Myrinet interconnect, our environment also had to support this kind of network. To describe an arbitrary network topology comprising hosts and switches, Myrinet defined a quite simple, yet very generic, ASCII-based topology file format referred to as a *map* file.

We implemented a translation tool to convert such a map file to an OMNeT++ network description (ned) file. This *map2ned* tool assumes generic base module definitions for both host (`AbstractHost`) and switch (`AbstractSwitch`). It creates a compound module that comprises the entire network, with input and output gates for every host to connect to other parts of the simulation environment. Within this compound module, a host array is created using the ned language's polymorphism construct (`like AbstractHost`) to allow different kinds of hosts to be instantiated via a `hostType` parameter. In addition, a switch array of `AbstractSwitch` modules is created. These are also polymorphic, but because we wanted to support different types of switches within the same system, we needed to resort to one level of indirection by passing a `switchType` parameter to the `AbstractSwitch` module. This is in fact a compound module that just instantiates the desired switch type. For convenience, the gate sizes of the arrayed hosts and switches are simply set to the maximum size encountered for each kind.

The connections section reflects all of the connections specified in the source map file. As not all ports may be connected, it uses the `nocheck` option. Link latencies can be passed to the compound network module via two parameters for host-to-switch and switch-to-switch delays.

In addition to the ned file, map2ned generates an initialization file that specifies the network address (which is assigned automatically by map2ned) name, and label for every host and the number of ports and label for every switch and adapter, as well as some top-level parameters.

4.4 routereader

As a companion to the map file format, there is also a Myrinet format to specify routes between any pair of hosts. Myrinet networks use turn-based source-routing, meaning that each sender determines the full route and the message carries it in its header. The route consists of one *relative* port index ("turn") for every switch on the path. In each switch, the corresponding turn is added to the index of the port on which the message arrived to obtain the output port index. Turns can therefore also be negative numbers. There may be multiple routes between any pair of hosts to support multi-path routing. To be useful, a route file must match a given map file, both in terms of topology and in terms of host naming.

We implemented a library to import a route file into the simulator at run time, to be able to utilize the routes corresponding to given map file.

4.5 Myrinet hardware models

In addition to support for the Myrinet topology and routing file formats, we also implemented switch and adapter models that accurately represent the real Myrinet hardware. This includes such aspects as message segmentation and interleaving in the adapters and wormhole routing and stop/go flow control in the switches.

In addition to the Myrinet-specific models, the environment also includes more generic output-queued and combined input-output-queued switch models.

4.6 Extended generalized fat trees

The property of full bisectional bandwidth provided by the well-established class of k-ary n-tree networks [6] generally ensures good performance, but incurs significant cost in terms of switch hardware and cabling. As these costs represent an increasing fraction of the overall system cost, the prospect of trading off a modest reduction in performance against a significant *slimming* of the topology is quite attractive [7].

Perfectly suited to this task is the class of *extended gen-*

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

eralized fat tree (XGFT) topologies [8], which includes any kind of slimmed (or fattened) tree topology. An XGFT is completely described by its parameters $(h; m_1, \ldots, m_h; w_1, \ldots, w_h)$ where h equals the number of switch levels; the leaves count as separate level (0). Each node at level i has m_i children for $1 \leq i \leq h$ and w_{i+1} parents for $0 \leq i \leq h-1$.

We implemented a tool that can generate a map file representing any XGFT, which can then be converted to ned using map2ned and loaded into the simulator.

4.7 Direct topologies

To be able to compare indirect network topologies such as fat trees and XGFTs with direct networks, we provided built-in support for three-dimensional Torus networks and n-dimensional Hypercubes. The dimensions of the Torus and the number of dimensions of the Hypercube are configurable, as is the number of hosts attached to each switch.

4.8 Multi-rail networks

HPC systems often feature multiple networks in parallel, each network carrying a different kind of traffic. Sometimes multiple instances of the same network are used in parallel to achieve higher overall throughput. In either case, a single host is connected to multiple independent networks within the same system; we refer to such networks as *multi-rail networks*. Our environment provides explicit support for such systems in two ways.

First, the map2ned tool accepts map files that specify multiple connections per host. map2ned also recognizes an optional parameter for each switch and adapter to select the type to instantiate. The output ini file specifies the number of networks and computing nodes per host, as well as the switch and adapter types selected.

Alternatively, a built-in `MultiRail` network type is provided, which takes the number of networks as a parameter. The type of network to instantiate can be specified for every network individually for full flexibility.

In multi-rail networks, the model instantiates an alternate host implementation, `MultiHost`, which supports multiple adapters. Furthermore, the `MultiHost` host type also provides support for multi-core nodes: the number of computing nodes (processors) per host can be specified with a parameter. The computing nodes and adapters are interconnected within the host using a router. The nodes need to attach a routing tag to their outgoing messages so that the router knows to which network to route the messages. Vice versa, the adapters need to inform the router to which node to route each incoming message.

4.9 Task mapping

To enable the evaluation of different mapping policies, our environment allows arbitrary mappings of Dimemas tasks to Venus hosts. This is accomplished by means of a simple configuration file that contains one hostname (as known to Venus) per line; task n is mapped to the host corresponding to the hostname specified on line n. In principle, multiple tasks can be mapped to the same host. The format could also be extended to map one task to multiple hosts by allowing multiple hostnames per line, but this is currently not supported.

4.10 Remarks

Myrinet software on any real installation can generate a matching map file. Being able to translate this file to an OMNeT-compatible format enabled us, for instance, to simulate the real topology of MareNostrum. Moreover, the matching route files reflecting the actual routes can also be generated, enabling an accurate simulation of both topology and routing of real-world installations.

Although originating from Myrinet networks, the map and route file formats are highly generic and can be used to describe basically any kind of topology and routing. In fact, we also implemented a conversion utility to translate the built-in topologies to the map format. This enables, for instance, the use of map-based offline route optimization tools to compute a set of routes with minimum conflicts for a given topology, mapping, and traffic pattern.

5. CASE STUDY: MareIncognito

The combination of Dimemas and Venus enabled us to gain new insights at various levels: (i) the application and MPI library level, (ii) the MPI protocol level, (iii) topology and routing, and (iv) switch and adapter hardware.

The space available does not permit us to go into detail or even touch upon all of the issues, but the next sections provide at least one example from each of these categories.

5.1 Application and MPI library level

Dimemas allows us to simulate changes in the application level or MPI level and to obtain a Paraver trace that we can analyze and compare with the trace obtained with the real execution. Combined with the capabilities of Venus, this enables a quantitative study of the potential benefits of such a change in the software stack under specific hardware configurations that match the target architecture.

We can, for instance, evaluate the gain of substituting blocking MPI sends/receives by their non-blocking (asynchronous) counterparts, or assess the benefit of changes in the MPI library implementation. Such experiments provide valuable feedback to the application and library developers when taking a decision on whether to implement potentially expensive changes.

As an example, one experiment demonstrated that changing the blocking (sendrecv) implementation of the exchanges in a finite element solver program (ALYA) by non-blocking sends and receives reduced the communication time by a factor of two.

5.2 MPI protocol level

In typical MPI implementations, a so-called *eager* threshold can be set to control how messages are injected into the network: Messages smaller than this threshold are sent eagerly, i.e., without first performing a rendez-vous exchange with the receiving node. Larger messages need to wait for the rendez-vous protocol to complete: the sender first sends a request (REQ) to the receiver, which, if it has room to store the message, will reply with an acknowledgment (ACK). Only when the sender gets the acknowledgment will it actually send the message. As the protocol messages (REQs and ACKs) share the network infrastructure with (potentially very large) data messages, this can have a notable impact on performance.

To test the effect, we used a trace of a 32-way all-to-all personalized exchange of 32,679-byte messages. When sending all messages eagerly, the exchange's run time was 4.3 ms. With the eager threshold set to 32,768 bytes, the run time

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

increased to 4.8 ms, an increase of just under 12%. However, when artifically introducing a small random imbalance of less than 10 μs at the start of every thread, the run time increased to 6.3 ms, an increase of more than 46%. The reason is that the imbalance causes protocol messages to be delayed by segments of data messages. Figure 2 shows per-thread progress timelines in the first and the last case. The color encodes the number of the message currently being received.

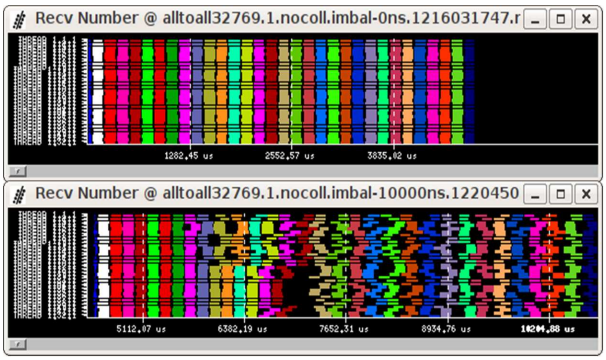

Figure 2: All-to-all progress without imbalance and rdvz (top), with imbalance and rendez-vous (bottom).

5.3 Topology and routing

Deciding upon a particular topology is a crucial design issue that has a big impact on the performance and cost of a machine. Venus is able to model a wide range of topologies. Together with Dimemas, it can quantitatively estimate the global performance that an application could achieve with each topology. Using the Paraver traces generated by Dimemas or Venus we can also understand the reasons of a performance difference between two architectures.

For the MareIncognito project we evaluated several applications in detail across a set of topologies from the XGFT family. We found that the routing policy can have a noticeable impact on the performance of communication phases of several applications in these network topologies. We observed that random routing in k-ary n-trees has, contrary to widespread belief, in general worse behaviour than simple regular routings. We also explored several communication-pattern-aware techniques by means of an offline tool that, by analyzing the communication pattern and the network graph, produces a set of routes with minimal routing conflicts for a particular pattern.

Figure 3 shows the relative performances that can be obtained with different routing policies across a particular family of XGFTs with decreasing connectivity (x axis). We can classify these routing policies according to their performance: some are severely affected by a decreasing connectivity (brute force, naive), some remain close to the optimum performance for the decreased connectivity (mod-k, colored), and some do not show a particular trend (random routings).

5.4 Switch and adapter hardware

An example of insights gained at the hardware level is illustrated in Fig. 4. The trace used here is a 256-way WRF

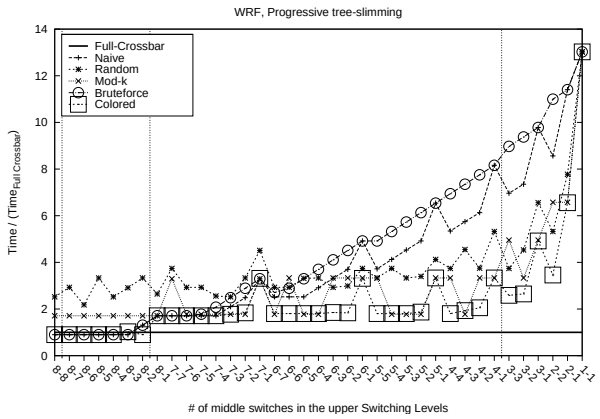

Figure 3: Performance of 256-node WRF on slimmed 3-level XGFTs.

(weather research and forecast) pattern on a two-level fat-tree network, in which each node n first sends one message to node $n + 16$ and then one message to node $n - 16$. The exceptions are nodes 0–15, which only send to $n + 16$ and nodes 240–255, which only send to $n - 16$.

Most nodes completed after about 63 μs, whereas three nodes took 24 μs longer than the rest (circled in Fig. 4). Deep inspection of the Paraver traces revealed that this was caused by head-of-line blocking: In the first phase, the last 16 nodes caused destination contention between communications from, e.g., nodes 253 and 221 to node 237. The message from 253 was served first, forcing the message from 221 to wait. The first message from node 221 was still blocked when the second message (to node 205) arrived on the same input port of the second-level crossbar: this second message was HOL blocked! The same thing happened for messages from two other nodes. Additional instrumentation to explicitly detect HOL-blocking yielded the results shown in Fig. 5, which shows the affected ports on all 16 second-level switches. Three switches exhibit HOL blocking; it turned out that this depends on the initial setting of the round-robin pointer used by the switches to select which port to serve in case of contention. As these pointers were initialized randomly, blocking occurred on only a few ports. By initially setting all pointer to the same value, we were able to induce blocking on all or none of the ports. Simulations with an output-queued switch architecture did not exhibit this behavior, as expected.

6. CONCLUSION

We presented a communication-centric simulation environment for HPC systems based on trace-driven co-simulation between an MPI task simulator and a detailed network simulator. This combination builds on the strengths of these two tools—and of an array of auxiliary ones—to enable a thorough exploration of the impact of the interconnection network on parallel application performance with an unprecedented accuracy, depth, and flexibility. Moreover, the extension of the visualization part of the toolchain to incorporate network communications, states, and events provides a new level of evaluation power.

The purpose of the environment is to aid the design of

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521

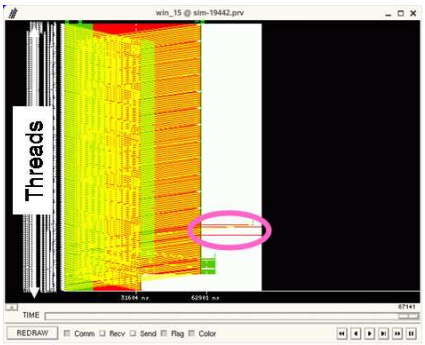

Figure 4: 256-way WRF pattern; communication among threads.

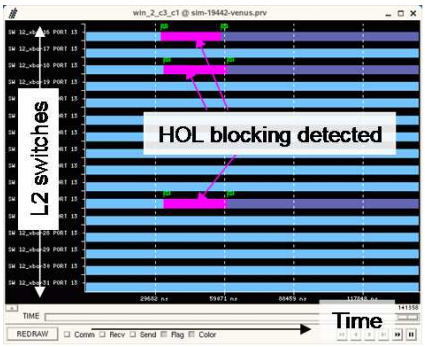

Figure 5: 256-way WRF pattern; HOL blocking on L2 switch ports.

next-generation HPC machines. We presented some examples of the kinds of insight the new environment can provide. Most notably, these insights range across all four levels: application, MPI library, topology, and hardware. We expect these insights to help point the way towards scalable, cost-effective, yet high-performance interconnection networks for the next generation of supercomputers.

7. REFERENCES

[1] Peterson, J. L., Bohrer, P. J., Chen, L., Elnozahy, E. N., Gheith, A., Jewell, R. H., Kistler, M. D., Maeurer, T. R., Malone, S. A., Murrell, D. B., Needel, N., Rajamani, K., Rinaldi, M. A., Simpson, R. O., Sudeep, K., Zhang, L. Application of full-system simulation in exploratory system design and development. *IBM Journal of Research and Development*, Vol. 50, No. 2/3, March/May 2006, pp. 321–332.

[2] Magnusson, P. S., Christensson, M., Eskilson, J., Forsgren, D., Hallberg, G., Hogberg, J., Larsson, F., Moestedt, A., Werner, B. Simics: A full system simulation platform. *IEEE Computer*, Vol. 35, No. 2, Feb. 2002, pp. 50–58.

[3] Denzel, W. E., Li, J., Walker, P., Jin, Y. A framework for end-to-end simulation of high-performance computing systems. In *Proceedings of the First International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTools 2008)*, Marseille, France, March 3–7, 2008, article No. 21..

[4] Varga, A. The OMNeT++ discrete event simulation system. In *Proceedings of the European Simulation Multiconference (ESM' 01)*, Prague, Czech Republic, June 2001.

[5] Boden, N. J., Cohen, D., Felderman, R. E., Kulawik, A. E., Seitz, C. L., Seizovic, J. N., Su, W.-K. Myrinet: A gigabit-per-second local area network. *IEEE Micro*, Vol. 15, No. 1, 1995, pp. 29–36.

[6] Petrini, F., Vanneschi, M. k-ary n-trees: High-performance networks for massively parallel architectures. In *Proceedings of the 11th International Symposium on Parallel Processing (IPPS '97)*, Apr. 1–5, 1997, Geneva, Switzerland, pp. 87–93.

[7] Desai, N., Balaji, P., Sadayappan, P., Islam, M. Are nonblocking networks really needed for high-end-computing workloads? In *Proceedings of the 2008 IEEE International Conference on Cluster Computing (Cluster '08)*, Tsukuba, Japan, Sep. 29–Oct. 1, 2008, pp. 152–159.

[8] Öhring, S., Ibel, M., Das, S. K., Kumar, M. J. On generalized fat trees. In *Proceedings of the 9th International Symposium on Parallel Processing (IPPS '95)*, Santa Barbara, CA, April 25–28, 1995, pp. 37–44.

[9] Geoffray, P., Hoefler, T. Adaptive routing strategies for modern high performance networks. In *Proceedings of the 16th IEEE Symposium on High Performance Interconnects (HOTI '08)*, Stanford, CA, Aug. 27–28, 2008, pp. 165–172.

[10] Bagrodia, R., Takai, M. Performance evaluation of conservative algorithms in parallel simulation languages. *IEEE Transactions Parallel Distributed Systems*, Vol. 11, No. 4, 2000, pp. 395–411.

[11] Varga, A., Sekercioglu, Y. A., Egan, G. K. A practical efficiency criterion for the null message algorithm. In *Proceedings of the European Simulation Symposium (ESS 2003)*, Oct. 26–29, 2003, Delft, The Netherlands.

Digital Object Identifier: 10.4108/ICST.SIMUTOOLS2009.5521
http://dx.doi.org/10.4108/ICST.SIMUTOOLS2009.5521