Proceedings of the 2005 Winter Simulation Conference
M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.

Scaling an Optimistic Parallel Simulation of Large-scale Interconnection Networks

Nilesh Choudhury, Yogesh Mehta, Terry L. Wilmarth, Eric J. Bohm and Laxmikant V. Kalé

Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, IL 61801, U.S.A.

ABSTRACT

Parallel computers today are designed with larger number of processors than ever before, connected by large scale Interconnection Networks (INs). Communication is the key to achieving high performance on such machines, making the study of Interconnection Networks more important. Parallel simulations of Interconnection Networks present a unique problem characterized by fine-grained computation and a strong dependence among events. The absence of large lookaheads makes it unsuitable to use a conservative simulation. Using an optimistic Parallel Discrete Event Simulation (PDES) allows us to extract reasonable parallelism from this simulation. In this paper we present BigNetSim, an Interconnection Network simulator. We analyze its performance and present techniques related to enhancing performance and scaling it to a large number of processors on different artificial traffic patterns and real application logs. In spite of the overheads of a parallel optimistic simulation, we have achieved a breakeven point with sequential simulation at 4 processors and demonstrate perfect scaling to 128 processors. [1]

1 Introduction

New parallel computers with hundreds of thousands of processors, capable of achieving hundreds of teraflops of peak speed have been built recently. For example, BlueGene (BG/L) developed by IBM, when completed will have 128K processors and is expected to achieve 360 teraflops at peak speed. Development of new applications and porting existing applications to such large machines is a challenging task. Few applications today efficiently scale to thousands of processors. Increasing machine size by an order of magnitude presents greater challenges.

Recent history has shown years of lag time between the appearance of new parallel hardware and the deployment of scientific applications which can efficiently use such hardware. Performance prediction of applications via simulation can provide insights to help optimize applications so that they are ready to be run on actual machines as soon as they become available. Even for existing large parallel machines, time for tuning applications on large machines as well as the queueing time before allocation of nodes can be very long. A simulator presents a more available alternative so that minimal supercomputing time is consumed by debugging and performance optimization. Furthermore, if we can forecast the behavior of target applications on a particular hypothetical machine design, we can use the simulation to inform the development process of future machines built from that design.

The BigSim (Zheng et al. 2005) project aims at developing techniques to help develop efficient scalable applications on very large parallel machines via accurate parallel simulation. Sequential simulation is inadequate for extreme scale applications as the data involved is too large to fit in the memory of a single machine. An important component of the overall simulation is BigNetSim, a parallel simulator for large-scale interconnection networks. It simulates packet level communication on detailed contention-based network models for large parallel computers. Parallel simulation of networks with standard traffic patterns is a difficult task. It becomes even more challenging when the network is loaded with communication transactions generated by real applications. The strong dependencies between events can result in limited available work in segments of the network.

In BigNetSim, simulated application computation is performed by running the application on the BigSim emulator. The application's event dependency and com-

[1]This work was supported in part by the National Science Foundation (NGS 0103645), the National Institutes of Health (PHS 5 P41 RR05969-04), the Defense Advanced Research Project Agency (NBCH30390004) and the Department of Energy (B341494).

munication activities are recorded and stored in log files. BigNetSim reads these log files and models the communication load on the interconnection network model. This results in fine-grained simulation since the computational overhead of modeling packet transmission is negligible. The nature of the simulation makes event safety extremely difficult to predict with sufficiently large lookahead to utilize conservative simulation strategies efficiently. Thus, it is useful to use optimistic synchronization for such a network simulation. We use POSE (Wilmarth and Kalé 2004) as the simulation environment to develop BigNetSim.

The paper is organized as follows. Related work in parallel network simulation is reviewed in Section 2. Section 4 discusses the modelling of the network. Section 5 presents some performance optimization techniques such as more fine-grained decomposition, increased virtualization, immediate GVT calculation and breaking application depedencies to avoid transient load imbalance. The performance results with the standard traffic patterns and with application work loads are discussed in Section 6. This section also illustrates the impact of the performance enhancements in the network simulator. We present our conclusions in Section 7 and discuss our plans for augmenting BigNetSim in near future.

2 Related Work

Simulation has proved to be a useful technique for network analysis over the years. Network simulator(NS) (Information Sciences Institute) is a widely used simulator which uses packet-level simulation and supports simulation of TCP, routing, and multicast protocols over general-purpose wired networks like the Internet and wireless networks. However, sequential simulation does not scale for large-scale networks due to increasing memory requirements and simulation time. Parallel/Distributed NS (COMPASS group) includes extensions and enhancements to the original NS to allow it to run in parallel and benefit from the advantages of parallel simulation. GTSNetS (Fujimoto et al. 2003), a parallel packet-level simulation of a general-purpose network like the Internet, achieves millions of packet transmissions per second. GloMoSim (Zeng, Bagrodia, and Gerla) is a scalable simulation framework for wireless networks and is based on the PARSEC (Bagrodia et al. 1998) simulation environment which adopts the process interaction approach to PDES.

High performance interconnection networks present certain unique issues. Achieving such performance requires highly connected topologies, carefully chosen routing algorithms, and particular buffering and flow control decisions. Diverse traffic patterns have been simulated to study and analyze different network architectures in SMART (Petrini and Vanneschi 1997). Recent simulation efforts involve parallel simulation for better scalability, a conservative parallel simulation of the IBM SP2 network (Benveniste and Heidelberger 1995).

Á la carte (Berkbigler et al. 2003) is a Los Alamos computer architecture toolkit for extreme-scale architecture simulation which is based on the DaSSF (Liu and Nicol 2002) framework. Another simulation (Nicol et al. 2003) of large scale communication networks based on DaSSF achieves faster than real-time simulation. Our work in interconnection network simulation differs in the fact that we use optimistic strategies and virtualization for simulation. We use the POSE (Wilmarth and Kalé 2004) simulation environment, an optimistically-synchronized PDES environment based on the Time Warp (Das et al. 1994) mechanism that uses a variety of adaptive (Das 1996) synchronization protocols.

3 Simulation Environment

POSE is a general-purpose optimistically-synchronized PDES environment designed for simulations with fine computation granularity and a low degree of parallelism. POSE (Wilmarth and Kalé 2004) is implemented in CHARM++ (Kalé and Krishnan 1993), a C++-based parallel programming system that supports *virtualization*. Virtualization involves the decomposition of a problem into N asynchronous *chares* or objects that execute on P processors (Kalé 2004). For best performance, N should be much greater than P. An advantage of this approach is that no object can hold a processor idle while it is waiting for a message. Since $N>>P$, there are other chares on a processor that can execute in the interim. Thus, using virtualization maximizes the degree of parallelism. This is discussed in further detail in Wilmarth et al. (2005).

Posers represent sequential entities in the simulation model, behaving as tiny LPs which encapsulate very small portions of state as illustrated in Figure 1. A poser encapsulates an *object virtual time* (OVT), *event methods* which receive *timestamped* messages, a state, a local event queue and a synchronization strategy instance.

POSE requires the programmer to decompose the simulation model into the smallest posers possible to achieve the best performance. The benefits of this decomposition are many. The local event queue limits the scope of simulation overhead to the poser itself. A finer decomposition enables less frequent checkpointing on smaller states, reduces the likelihood and effects of rollbacks, reduces the complexity of object migrations, and makes it possible to use adaptive synchronization strategies that are fine-tuned to an LP's behavior. The

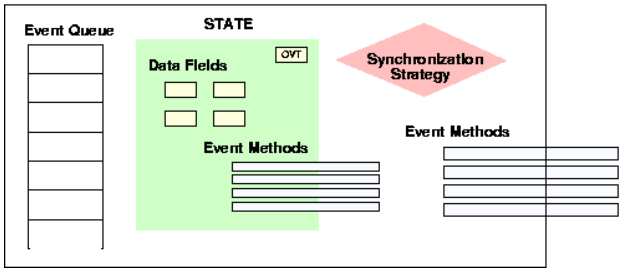

Figure 1: Components of a poser

drawbacks of a high degree of virtualization are the costs of management of per-object information and the cost of more frequent context-switching between entities for each event. These trade-offs are studied in detail in Wilmarth et al. (2005) where we found that the benefits of higher degrees of virtualization strongly outweighed these costs, with finer decompositions consistently outperforming coarser ones as problem size increases. Higher degrees of virtualization also enable programs to scale to more processors than do lower degrees of virtualization.

POSE uses an *adaptive synchronization strategy* to control how events are executed on posers. The strategy adapts to a poser's behavior and ranges from cautiously to aggressively optimistic. Traditionally, optimistic approaches sort arriving events and execute the earliest. In our approach, a *speculative window* governs how far into the virtual time future a poser may proceed. When a poser has control, *all* the events within its window are executed as a *multi-event*. Multi-events reduce scheduling and context switching overhead and benefit from a warmed cache, compensating for some of the drawbacks like the startup time taken to construct the additional objects on the additional processors and a small additional overhead for object management associated with the high degree of virtualization. Adaptive synchronization and multi-events are discussed in detail in Wilmarth (2005).

POSE's current strategy, Adept, is a general-purpose strategy flexible enough to apply to a variety of simulations. Adept adapts a poser's speculation to its past, current and likely future behavior. It outperforms traditional optimistic strategies as described in detail in Wilmarth et al. (2005).

POSE makes use of a fully asynchronous GVT algorithm invoked in a distributed fashion as processors run out of sufficiently early events to execute and then transmit processor-level data via a reduction to a central processor which computes the latest GVT. Should idle processors receive additional work, they may execute it in the interim while waiting for the GVT to complete.

4 Simulation Model

BigNetSim is an effort to simulate large current and future computer systems to study the behavior of applications developed for those systems. It simulates with reasonable detail an integrated model for computation (processors) and communication (interconnection networks). Our earlier work on computation simulation for performance prediction, BigSim (Zheng et al. 2004, Zheng et al. 2005), assumed fixed message latencies. The BigSim emulator (Saboo et al. 2001) was the first phase of our performance prediction system. CHARM++ and AMPI (Huang, Lawlor, and Kalé 2003) applications can be compiled to run on this emulator as though it were the target architecture. The emulator captures a collection of tasks (blocks of computation and communication) on a number of processors (objects) along with their dependencies and writes these tasks to log files. These application tasks are translated into discrete events. Each event has a timestamp and originating and destination objects. The logs are read by BigNetSim which simulates the execution of the original tasks by elapsing time, satisfying dependencies, and spawning additional tasks by passing messages through a detailed network contention model. This generates corrected times for each event which can be used to analyze its performance on the target machine.

In this paper, we focus on the simulation of interconnection networks in BigNetSim. The logical components of our abstract interconnection network model are implemented with posers. The hardware modelling of each node has two posers:

- A computation unit which manages the execution of computational tasks on the node
- A communication unit which manages incoming and outgoing messages of the node

Each node has an additional virtual unit, a traffic generator, which generates artificial message traffic on the Interconnection Network. This allows us to study the interconnection network under a variety of conditions without using application task logs. The traffic generator can send point-to-point messages, reductions, multicasts, broadcasts and other collective traffic. It supports k-shift, ring, bit-transpose, bit-reversal, bit-complement and uniform random traffic. These are based on common communication patterns found in real applications. The frequency of message generation is determined by a uniform or Poisson distribution.

Each computation node has a network interface card (NIC) that collects messages from the computation node,

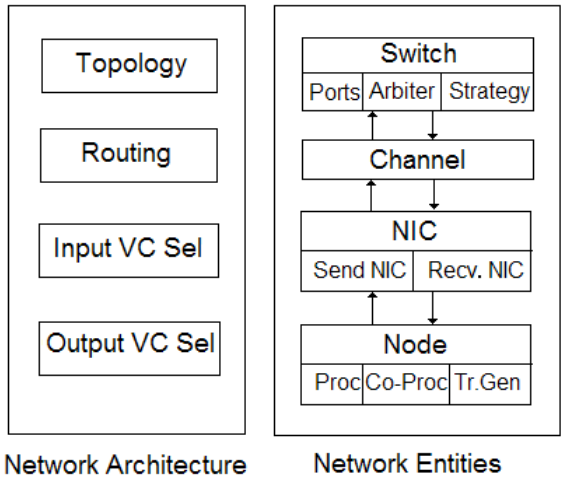

Figure 2: BigNetSim conceptual model

packetizes and puts them out onto the network. The NIC is modeled with two posers:

- A send NIC which packetizes and sends messages as packets. It models DMA and HCA delays. The delays are categorized for small and large messages and added to the message send times. It responds to excessive load with an injection threshold that models deteriorating caching effects as it gets overloaded.
- A receive NIC which receives packets, combines them into messages and passes the messages to the node. Similar delay modeling is done to simulate precise NIC behaviour. These parameters can be set at runtime, allowing us to simulate different types of NICs.

Channels in the Interconnection Network, modeled as posers, are duplex channels. They model channel delays. A channel connects a switch and a NIC or two switches. The Interconnection Network has switches connected as specified by the topology. Switches can be distinguished as:

- *Input Buffered (IB):* A packet in a switch is stored at the input port until its next route is decided and leaves the switch if it finds available space on the next switch in the route.
- *Output Buffered (OB):* A packet in a switch decides beforehand on the next route to take and is buffered at the output port until space is available on the next switch along the route.

IB switches are mostly popular in current architectures, but research (Kumar, Kale, and Stunkel 2005) shows the potential that OB switches offer. We believe that OB can be the choice for future switch designs. Switches are modeled in much detail. Ports, buffers and virtual channels at ports to avoid head-of-the-line blocking are modeled. Hardware collectives are implemented on the switch to enable broadcasts, multicasts and other collective operations efficiently. These are configurable and can be used if the system being simulated supports them. We also support configurable strategies for arbitration, input virtual channel selection and output virtual channel selection. The configurability of the switch provides a flexible design, satisfying the requirements of a large number of networks.

At a higher level the entire design is extremely modular. New topologies and routing algorithms can be easily plugged into the system. We use virtual cut-through packet switching with a credit-based flow control to keep track of packets in the network. The system supports virtual topologies for virtual channel routing which is essential for deadlock-free routing algorithms on most topologies.

Topologies implemented include N-dimensional meshes and Tori, N-dimensional Hypercubes and K-ary N-trees and Hybrid topologies. All topologies have physical and virtual channel routing algorithms. Most routing algorithms are adaptive. To support adaptivity based on the network load, we developed a contention model and a load model for the Interconnection Network. Each port of a switch has information which is dynamically updated and fed to the routing engine to make informed decisions to minimize contention. The load model maintains load information on each of the neighbors while the contention model maintains information about the number of packets contending for a particular output port of a switch.

5 Performance Optimization and Scaling

This section discusses performance optimization techniques to enhance performance and scalability of the simulation.

Some important factors related to performance are:

- *Number of GVT synchronizations:* This metric gives an estimate of how much parallelism the simulation has within the threshold controlled by the simulation. A large number of synchronizations implies there is very little speculative work within allowable limits.
- *Phase time:* This is the real time elapsed between two GVT synchronizations. Phase time

is directly proportional to the amount of parallelism in the simulation.

- *Rollback fraction:* The proportion of total time taken for undoing speculative execution which could not be committed due to some dependence violation.
- *Communication fraction:* The fraction of the total time spent communicating between processors. Since this application involves a large amount of communication with very small message sizes, communication becomes an important factor to consider.
- *Speedup:* The speedup of the simulation compared to sequential execution of the same problem, if it fits in the memory of a single node. Sequential simulation is more efficient as it avoids synchronization. Events are simply executed in non-decreasing timestamp order.

One throttling mechanism used by POSE's adaptive synchronization strategy is the speculative window. When rollbacks occur, the window size is reduced to the average rollback offset from the current GVT. In the absence of rollbacks, this window is allowed to expand. In almost all runs the rollback fraction is under control and speculation remains below 20%. Problems arise with ill-designed simulations which exhibit a high rollback fraction in spite of a tightly constrained speculative window. This motivates the need for a finely decomposed simulation model to reduce the likelihood of rollbacks and maximize computation and communication overlap.

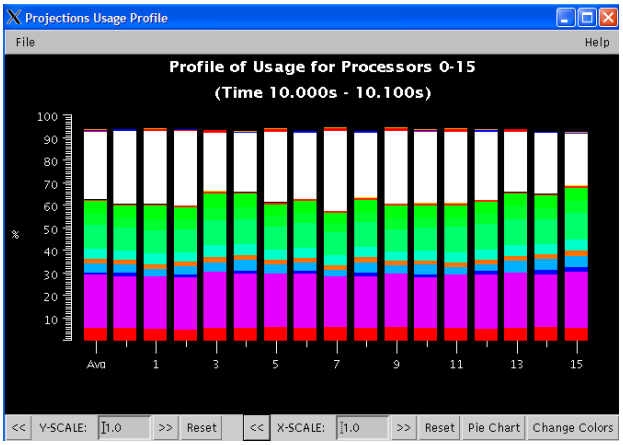

Figure 3: Usage Profile before optimization

Initial analysis revealed that the phase time was very small, at 5ms, and that it decreased with larger problem size. Real speedup was poor, with a breakeven point (the number of parallel processors needed to equal the performance of a sequential execution) at close to twelve (Wilmarth et al. 2005). Increasing the number of processors worsened the problem as each synchronization grew more expensive. Idle time on each processor increased as there was a greater probability of a small load imbalance on any one of the processors causing all processors to wait for the last processor to finish its work. Thus, the simulation's ability to scale was also limited.

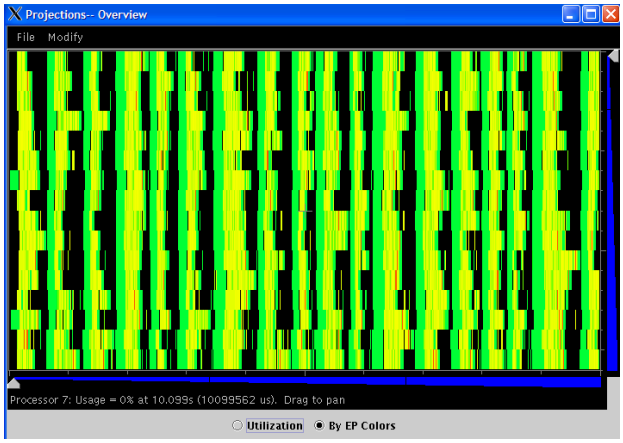

Figure 4: Overview before optimization

Figures 3, 4, 5 and 6 are projections (Kale et al. 2005) generated graphs for a 100ms time interval for a 16 processor simulation of a 2048 node hypercube network. The former two graphs cater to the analysis of the simulation before the optimizations, while the latter two depict the improvements to the simulation that we finally achieved.

Figure 3 presents a plot of processor utilization against the processors. The first bar shows the average utilization. Each entry method (function call) is represented by a different color on the plot, with the pink and red colors representing the simulation environment book-keeping tasks, while the white region represents idle time on a processor. The other colors are the entry methods which do the real work. It should be noted that not all the work here is forward execution, a significant part of it is work that is later cancelled.

Figure 4 shows an overview of the actual execution. Each horizontal bar represents a processor. The green color represents the work done by POSE while the yellow represents actual simulation methods. The black area represents idle time on a processor. Each phase is clearly visible with a reduction followed by broadcasting the new GVT across all processors at the end of every phase.

Frequent synchronizations were caused by the switch poser, which was large and complex with many locally parallel events. Tracing the number of events on each object revealed that the switch handled many more events than any other poser. This caused it to rollback more often, rarely allowing the GVT to advance. It also had

the largest state-size, making each checkpoint expensive. The solution was to **decompose the switch into more fine-grained posers.** Each poser representing a parallel entity in the switch.

Ports are logical parallel entities in a switch. Each port can be modelled as a separate poser that maintains its own state and is connected to channels or NICs. The refactoring of the switch into ports was meticulously done, so as to avoid adding new events to the simulation. It was essential to keep the number of events per packet hop low, so that the refactoring does not degrade the speed of the interconnection network simulation. Currently, we have a refactored version for an Output Buffered (OB) switch. An Input Buffered (IB) switch shares much more information between ports at each arbitration, making it difficult to decompose. We hope to find efficient ways to refactor the IB switch without stripping it of functionality.

Our results showed that the phase time had increased and the number of GVT iterations had decreased. The rollback fraction remained within tolerable limits even for a large speculative window, On average the simulation time was reduced by half, however, good scaling eluded us. With larger problem sizes on a large number of processors, the CPU utilization was low. The problem manifested as fine load imbalances in the phases between each GVT calculation. The POSE GVT algorithm was fully asynchronous (i.e. GVT calculation proceeds concurrently with forward execution of events), but relied on idling processors for initiation of the GVT reduction and the most loaded processor for the completion of a phase. When a processor ran out of work, it would start a GVT update, but other processors were not required to send their updated information until they too ran out of work to do. Once all processors supplied this information, the GVT reduction could reach completion. Thus, subtle variations in available work caused idle time for some processors during a phase. The solution was **expedited GVT calculation**.

The first idle processor triggers all processors to provide updated information immediately, after which they could continue with forward execution of available work. This reduced the transient load imbalance but increased the number of GVT calculations. This is evident if we compare the figure 4 with figure 5. The former shows a green line at the end of each phase for each processor, while the latter shows a synchronized thin green line across all processors at the same time. This represents a new calculation of GVT so that the least loaded processor does not have to wait for long before the phase ends. This optimization improved the performance of larger sized problems on a large numbers of processors by another 50% where these subtle load imbalances between processors within phases were more likely to occur.

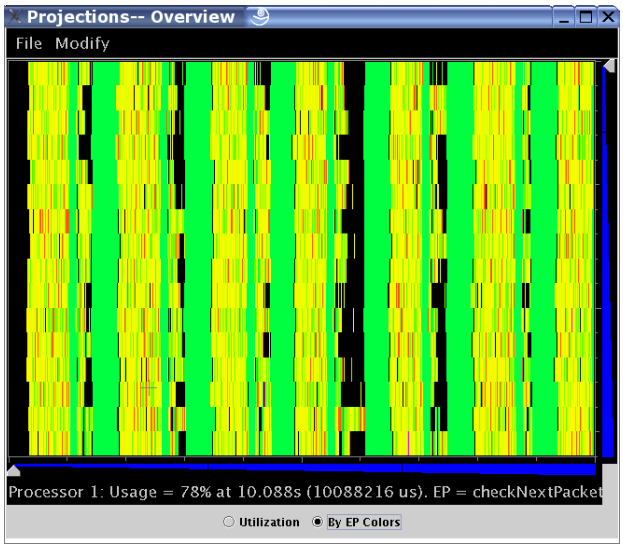

Figure 5: Overview after optimization

There still remained a wide load imbalance. Load seemed to be shifting from one processor to another in different phases following a fixed pattern based on a particular traffic pattern on a particular network. This hinted at the existence of application specific dependencies in workload on the posers. We had been partitioning the input problem based on the communication graph of the application to minimize communication. This seemed logical as the application is largely dominated by communication. However, this caused posers that communicate more often to be placed on the same processor, causing a strong dependence relation. Work from one processor was passed to another in a fixed pattern in spite of randomness in the input. We used various initial placements of objects to try to break the dependencies. Partial random, partially METIS-based (Karypis and Kumar 1996), and fully random placement were used, while maintaining computation load balance.

Fully random initial placement was substantially better than all other placements. This could be attributed to it being able to **break all control dependences** in the problem more effectively and use virtualization to the maximum. The improvement provided was substantial. This solution, in spite of being counter-intuitive, as it inherently increases communication, minimizes the overall runtime. We intend to perform further studies to find a better initial placement of objects, if possible, as this would be important to reduce communication, which is the bottleneck for large problem sizes.

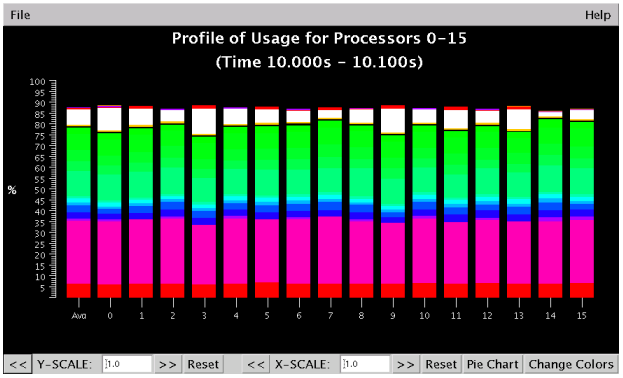

Figure 6: Usage Profile after optimization

The final version with all the above techniques incorporated in the simulation performed nearly 3 times better than what we started with (Wilmarth et al. 2005) and exhibits very good scaling to large problem sizes and substantially large number of processors. Figure 6 shows that the processor utilization is now much higher than previously, with little idle time on each processor. In addition, the percentage of rollbacks decreased as the large switch poser responsible for most rollbacks was taken out. This results in a larger proportion of forward execution in the simulation.

6 Results

This section presents performance results for BigNetSim on artificial and application generated traffic loads.

6.1 Standard Traffic Patterns

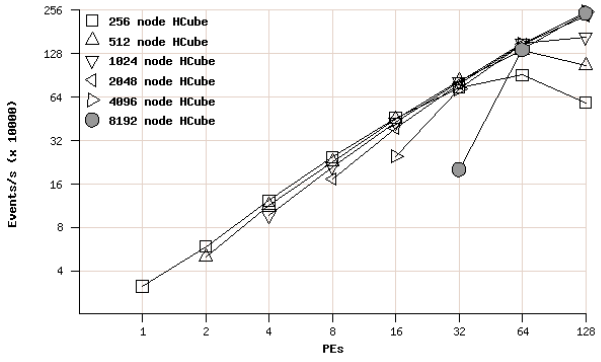

Figure 7: BigNetSim events/sec with TrafficGen

A uniform random traffic pattern using a Poisson traffic generation frequency is used to generate packets in the network. This pattern was selected because it would closely resemble the behavior of an arbitrary collection of applications running on a supercomputer. The random selection of destinations result in a repeatable but random asymmetric load on the network. Network sizes range from 256 to 8192 nodes. Each run has each node in the simulated network generating 1000 packets. All runs were made on Turing[2].

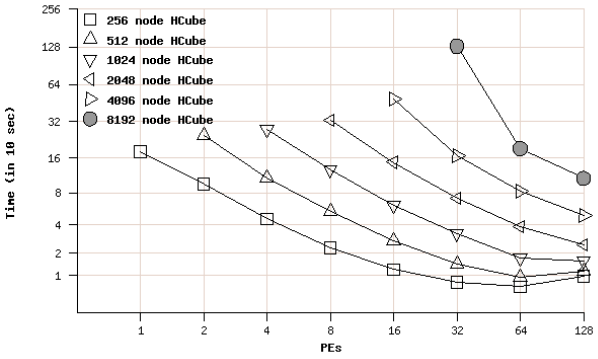

Figure 8: BigNetSim time with TrafficGen

We present some runs on Turing for the N-dimensional Hypercube mentioned earlier in this section. We plot events per second, time of simulation and real speedup as a function of the number of processors for input network sizes from 256 nodes to 8192 nodes. Figure 7 shows that for various problem sizes the number of events/sec it can simulate is nearly constant for a fixed number of processors. The simulation scales well as the number of processors increase. For smaller problem sizes, the performance dips down on large number of processors because there is not enough work per processor to maintain a reasonable overlap of communication and computation. Very large problem sizes on small numbers of processors run out of virtual memory and physical memory, leading to poor performance. Points in the graphs for large problem sizes on small number of processors are missing for this reason.

Figure 8 reiterates the same facts that Figure 7 shows. The simulation demonstrates good self scaling. The execution time for most problem sizes decreases linearly with increasing number of processors. The same explanation as above explains the points which do not follow this pattern.

Figure 9 is a plot of speedup over sequential simulation. The breakeven point is approximately 4, when parallel performance executes faster than sequential simulation and the scaling seems to be near perfect as the number of processors increases.

Table 1 gives an estimate of the number of packet hops in the simulated networks. The average ratio of events to packet hops varies from 4.5 for small networks to closer to 4 events per packet hop for larger networks

[2] *Turing* is a cluster of 640 dual 2GHz G5 processors 4 GB RAM Apple Xserves connected by Myrinet.

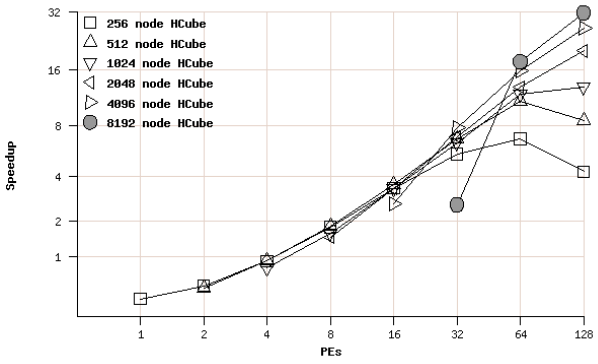

Figure 9: BigNetSim speedup with TrafficGen

simulated above. It also presents the number of posers for each network size.

Table 1: Interconnection Network simulation parameters

	Network size					
	256	512	1024	2048	4096	8192
Packet Hops (in millions)	1.2	2.82	6.15	13.3	28.7	61.5
Posers (in thousands)	5.1	11.3	24.6	53.3	115	246

We analyze the communication requirements of the parallel simulation next. For the 8192 node simulation on 128 processors, the problem fits in memory comfortably but communication reaches close to 50MB/s per processor. Most of these are small messages, with message sizes between 250 and 300 bytes. Communication performance is poor for most interconnection networks for frequent small messages. The Myrinet communication layer handles this huge volume of communication, but for a 16384 node network, the communication layer starts dropping and resending packets at an alarming rate, freezing the network cards, causing the simulation to slow down considerably and in most cases not ending within the specified time. Thus, tuning communication strategies to handle small message communication on this scale is our next challenge.

6.2 Application Generated Traffic

Next we evaluate the communication requirements of real applications using BigNetSim. One such application is the simulation of biomolecules using molecular dynamics. Molecular dynamics simulations comprise a substantial fraction of current supercomputing time, and therefore represent a highly typical workload for a high performance interconnection network. NAMD (Phillips, Zheng, Kumar, and Kalé 2002), a state of the art parallel molecular dynamics application, was chosen for its unprecedented scalability and widespread usage in the molecular dynamics community.

To demonstrate BigNetSim performance at a modest scale, representing a more typical application use than the extreme scale example in Wilmarth et al. (2005), we simulated a small 64 node run for 1000 timesteps of a single GlpF aquaporin channel (4210 atoms) on 32 real nodes of Turing. BigNetSim was then run using those application trace files to simulate runs with a hypercube interconnection network with contention modelling on and off. The running time prediction accuracy of this simulator has been presented earlier in Wilmarth et al. (2005). Here we will evaluate the performance of the simulator.

Table 2: Namd simulation Performance Scaling

	Number of processors						
	Seq	1	2	4	8	16	32
with IN	35.8	135	69.9	33.4	19.9	15	22
w/o IN	12.2	23.2	16.7	8.48	5.97	5	6.1

Table 2 presents a scaling set of runs for the NAMD GlpF simulation. It shows simulation execution time for BigNetSim with NAMD from 1 to 32 processors. The current simulation model for application simulations is log file based. Log file based post mortem simulation presents additional constraints as the log size becomes large. File system and memory constraints can dominate overall simulation performance.

The above experiment is on a small log size (megabytes per simulated processor) which easily fits in the memory of each processor. With file performance thereby factored out, we achieve good performance scaling as well as a breakeven with sequential simulation at 4 processors, which reiterates our earlier results. This demonstrates the ability of BigNetSim to perform with sufficient accuracy and good performance for real applications as well as simulated ones. But as the logs are small, the problem size we are trying to simulate is also small, not allowing the simulation to have enough parallelism to scale to a large number of processors, as there is not enough work. Some optimizations, like replicating data on all the nodes performing the simulation, improve the simulation performance for small log sizes, however, these simple optimizations are not scalable to large log sizes. To scale this, we need to get rid of log-based simulations and integrate BigSim and the Interconnection Network Simulator more tightly, so that they interact at runtime. There are a number of issues with this integration which we are analyzing carefully.

7 Conclusions and Future Research

Simulation of detailed contention-based interconnection network models for predicting parallel performance is still quite challenging. This problem is characterized by the fine granularity of computation and abundant amount of communication, as the entire state is largely shared across a number of processing units. Furthermore, traces of iterative applications frequently evince a low degree of parallelism when only their network load is being modeled. Despite these challenges, we have had significant break-throughs using our performance optimization techniques to achieve a breakeven with sequential simulation at four processors and very good scaling until 128 processors.

Our optimistic simulation environment, POSE, introduces a new object model based on virtualization. Judicious use of the advantages this simulation environment presents fosters the development of applications, which demonstrate extremely good scaling and a very good overall performance. We have successfully demonstrated the use of optimizations, such as fine-grained decomposition, greater virtualization, and immediate GVT calculation to improve performance and scalability. BigNetSim has achieved 2.5 million events/sec for a 245,000 modelled entities (posers) on 128 processors. This translates to approximately 2000 posers, contributing an average of 20,000 events/sec per processor simulating a total of approximately 0.625 million packet hops/sec.

A challenge that remains to be solved is to optimize the handling of small messages. Achieving better performance for small messages in the Charm++ runtime system is critical for improving the overall performance of BigNetSim. We expect to optimize overall communication through adaptive message combining, smaller message envelopes, and performance optimizations in the Charm++ messaging layer. Some improvements are possible with POSE reduction handling to distribute messaging load across all processors. This would balance out the communication load and allow us to simulate larger networks.

BigNetSim itself will be enhanced to include higher fidelity NIC modelling. Further performance predictions for other applications like FEM, CPAIMD, leanMD will be undertaken in more detail. Performance prediction design currently involves log based simulations which ultimately constrains scaling to file system performance with exploding log sizes. Tighter integration with BigSim to perform simultaneous emulation and detailed interconnect modelling will also be explored.

REFERENCES

Bagrodia, R., R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, B. Park, and H. Song. 1998. Parsec: A parallel simulation environment for complex systems. In *Computer, Vol. 31(10)*, 77–85.

Benveniste, C., and P. Heidelberger. 1995. Parallel simulation of the IBM SP2 interconnection network. In *Winter Simulation Conference*.

Berkbigler, K., G. Booker, B. Bush, K. Davis, and N. Moss. 2003, April. Simulating the Quadrics Interconnection Network. In *HPCS'03, Advance Simulation Technologies Conference 2003*. Orlando, Florida.

COMPASS group, G. T. Parallel/distributed ns. URL: http://www.cc.gatech.edu/computing/compass/pdns/.

Das, S. R. 1996. Adaptive protocols for parallel discrete event simulation. In *Winter Simulation Conference*, 186–193.

Das, S. R., R. Fujimoto, K. S. Panesar, D. Allison, and M. Hybinette. 1994. GTW: a time warp system for shared memory multiprocessors. In *Winter Simulation Conference*, 1332–1339.

Fujimoto, R., K. Perumalla, A. Park, H. Wu, M. Ammar, and G. Riley. 2003. Large-scale network simulation – how big? how fast? In *MASCOTS*.

Huang, C., O. Lawlor, and L. V. Kalé. 2003, October. Adaptive MPI. In *Proceedings of the 16th International Workshop on Languages and Compilers for Parallel Computing (LCPC 03)*. College Station, Texas.

Information Sciences Institute, U. o. S. C. Network simulator ns-2. URL: http://www.isi.edu/nsnam/ns/.

Kalé, L., and S. Krishnan. 1993, September. CHARM++: A Portable Concurrent Object Oriented System Based on C++. In *Proceedings of OOPSLA'93*, ed. A. Paepcke, 91–108: ACM Press.

Kalé, L. V. 2004, February. Performance and productivity in parallel programming via processor virtualization. In *Proc. of the First Intl. Workshop on Productivity and Performance in High-End Computing (at HPCA 10)*. Madrid, Spain.

Kale, L. V., G. Zheng, C. W. Lee, and S. Kumar. 2005. Scaling applications to massively parallel machines using projections performance analysis tool. In *Future Generation Computer Systems Special Issue on: Large-Scale System Performance Modeling and Analysis*, Number to appear.

Karypis, G., and V. Kumar. 1996. Parallel multilevel k-way partitioning scheme for irregular graphs. In *Supercomputing '96: Proceedings of the 1996 ACM/IEEE conference on Supercomputing (CDROM)*, 35.

Kumar, S., L. V. Kale, and C. Stunkel. 2005, Mar. Architecture for supporting hardware collectives in output-queued high-radix routers. Technical Report 05-02, Parallel Programming Laboratory, Department of Computer Science, University of Illinois at Urbana-Champaign.

Liu, J., and D. Nicol. 2002, February. *Dartmouth scalable simulation framework user's manual*. Dept. of Computer Science, Dartmouth College, Hanover, NH.

Nicol, D. M., J. Liu, M. Liljenstam, and G. Yan. 2003. Simulation of large scale networks i: simulation of large-scale networks using ssf. In *WSC '03: Proceedings of the 35th conference on Winter simulation*, 650–657: Winter Simulation Conference.

Petrini, F., and M. Vanneschi. 1997, June. SMART: a Simulator of Massive ARchitectures and Topologies. In *International Conference on Parallel and Distributed Systems Euro-PDS'97*. Barcelona, Spain.

Phillips, J. C., G. Zheng, S. Kumar, and L. V. Kalé. 2002, September. NAMD: Biomolecular simulation on thousands of processors. In *Proceedings of SC 2002*. Baltimore, MD.

Saboo, N., A. K. Singla, J. M. Unger, and L. V. Kalé. 2001, April. Emulating petaflops machines and blue gene. In *Workshop on Massively Parallel Processing (IPDPS'01)*. San Francisco, CA.

Wilmarth, T., and L. V. Kalé. 2004, August. Pose: Getting over grainsize in parallel discrete event simulation. In *2004 International Conference on Parallel Processing*, 12–19.

Wilmarth, T. L. 2005. *Pose: Scalable general-purpose parallel discrete event simulation*. Ph. D. thesis, Department of Computer Science, University of Illinois at Urbana-Champaign.

Wilmarth, T. L., G. Zheng, E. J. Bohm, Y. Mehta, N. Choudhury, P. Jagadishprasad, and L. V. Kalé. 2005. Performance prediction using simulation of large-scale interconnection networks in pose. In *Proceedings of the Workshop on Principles of Advanced and Distributed Simulation*, 109–118.

Zeng, X., R. Bagrodia, and M. Gerla. Glomosim: a library for parallel simulation of large-scale wireless networks. In *PADS '98*, 154–161.

Zheng, G., T. Wilmarth, P. Jagadishprasad, and L. V. Kalé. 2005. Simulation-based performance prediction for large parallel machines. In *International Journal of Parallel Programming*, Number to appear.

Zheng, G., T. Wilmarth, O. S. Lawlor, L. V. Kalé, S. Adve, D. Padua, and P. Geubelle. 2004, April. Performance modeling and programming environments for petaflops computers and the blue gene machine. In *NSF Next Generation Systems Program Workshop, 18th International Parallel and Distributed Processing Symposium(IPDPS)*. Santa Fe, New Mexico: IEEE Press.

AUTHOR BIOGRAPHY

Nilesh Choudhury is a PhD student at the Parallel Programming Lab in the Computer Science Department at the University of Illinois, Urbana-Champaign. His research interests include communication optimization for large machines, parallel simulations, interconnection networks and parallel run-time environments. His e-mail address is <nchoudh2@uiuc.edu> and his web address is <http://charm.cs.uiuc.edu/~nilesh>.

Yogesh Mehta is a MS student at the Parallel Programming Lab in the Computer Science Department at the University of Illinois, Urbana-Champaign. His research interests include interconnection networks, performance optimization and parallel simulation. His e-mail address is <ymehta@uiuc.edu> and his web address is <http://charm.cs.uiuc.edu/people/yogesh>.

Terry Wilmarth is a Post-doctoral Research Associate in Computational Science and Engineering at the University of Illinois, Urbana-Champaign. Her research interests include parallel discrete event simulation and parallel environments for highly adaptive unstructured meshes. Her e-mail address is <wilmarth@uiuc.edu> and her web address is <http://charm.cs.uiuc.edu/people/wilmarth>.

Eric Bohm is a research programmer at the Parallel Programming Lab in the Computer Science Department at the University of Illinois, Urbana-Champaign. His research interests include parallel simulation, quantum chemistry and molecular dynamics. His e-mail address is <ebohm@uiuc.edu> and his web address is <http://charm.cs.uiuc.edu/~ebohm>.

L.V. Kalé is a Professor at the Computer Science Department at the University of Illinois, Urbana-Champaign. His research interests span the broad area of parallel computing and run-time environments and its applications. His e-mail address is <kale@uiuc.edu> and his web address is <http://charm.cs.uiuc.edu/~kale>.