A NETWORK CONTENTION MODEL FOR THE EXTREME-SCALE SIMULATOR

Christian Engelmann and Thomas Naughton
Computer Science and Mathematics Division
Oak Ridge National Laboratory, Oak Ridge, TN, USA
email: engelmannc@ornl.gov and naughtont@ornl.gov

ABSTRACT
The Extreme-scale Simulator (xSim) is a performance investigation toolkit for high-performance computing (HPC) hardware/software co-design. It permits running a HPC application with millions of concurrent execution threads, while observing its performance in a simulated extreme-scale system. This paper details a newly developed network modeling feature for xSim, eliminating the shortcomings of the existing network modeling capabilities. The approach takes a different path for implementing network contention and bandwidth capacity modeling using a less synchronous and accurate enough model design. With the new network modeling feature, xSim is able to simulate on-chip and on-node networks with reasonable accuracy and overheads.

KEY WORDS
high-performance computing, performance evaluation, parallel discrete event simulation, computer network modeling

1 Introduction

The fastest supercomputers in the world are able to perform more than 10 PFlop/s (1 PFlop/s = 10^{15} floating-point operations per second) using the LINPACK benchmark [10]. For example, the "Titan" Cray XK7 system at Oak Ridge National Laboratory has 560,640 compute cores with a theoretical peak performance of 27.1 PFlop/s and a LINPACK performance of 17.1 PFlop/s. Most of the performance comes from NVIDIA graphics processing units (GPUs). It is the second fastest supercomputer in the world and has a computational efficiency for LINPACK of 63 %. In contrast, the "Sequoia" IBM BlueGene/Q system at Lawrence Livermore National Laboratory has 1.57 million IBM Power BQC processor cores with a theoretical peak of 20.1 PFlop/s and a LINPACK performance of 16.3 PFlop/s. It is the third fastest supercomputer and its computational efficiency is 81 %. Both systems employ different architectures (GPUs vs. embedded processors).

1.1 HPC Hardware/Software Co-design

The application-architecture performance gap, which is the difference between the peak capabilities of the hardware and the performance realized by applications, is in this case the computational efficiency using the LINPACK benchmark. Since different architectures provide different performance capabilities and different applications require different performance features, deploying high-performance computing (HPC) systems that fit the workload needs of computing centers requires hardware/software co-design. It closes the application-architecture performance gap by designing HPC systems to meet application requirements and HPC applications to exploit architectural features.

Investigating and understanding the performance properties of next-generation HPC systems and the performance requirements of the targeted HPC applications is an essential part of HPC hardware/software co-design. Performance profiling of compute-node architectures is typically performed using highly accurate simulation and prototyping. Full-system profiling is limited to simulation approaches only as a large-scale deployment of a next-generation HPC architecture is typically only available once the system has been delivered to a customer. An experimental prototype for testing purposes is cost prohibitive, as a single deployment can cost $50M-$200M.

Full-system simulation approaches require a significant amount of resources and a trade-off between simulation accuracy and performance. Considering that a system may be simulated that is 10-to-100 times faster than anything available today, these approaches typically employ parallel discrete event simulation (PDES) solutions and run on today's HPC systems. A combination of highly-accurate simulation at small scale and less-accurate simulation at large scale is often utilized to improve the less-accurate simulation with the results of the highly-accurate simulation in the form of tuned architectural models.

1.2 The Extreme-scale Simulator

The Extreme-scale Simulator (xSim) [12, 7, 5, 4, 3, 9, 6] is a performance investigation toolkit that permits running a HPC application with millions of concurrent execution threads, while observing its performance in a simulated extreme-scale system. Using a lightweight PDES, xSim executes a Message Passing Interface (MPI) application on a smaller system in an oversubscribed fashion with a virtual wall clock time, such that performance data can be extracted using architectural models. It currently does not support threaded programming models, task-based execution models, and accelerators.

The capabilities of xSim have been recently demonstrated [5] by (a) scaling it to 134,217,728 (2^{27}) simulated MPI ranks (each with its own process context) on a 960-

core Linux cluster (a world record in extreme-scale simulation), (b) evaluating different MPI collective communication algorithms on a simulated future-generation system with 2,097,152 (2^{21}) MPI ranks using the same cluster, and (c) investigating a Monte Carlo solver using different architectural parameters with 16,777,216 (2^{24}) simulated MPI ranks using the same cluster. xSim has a fault injection feature [7] that allows to simulate MPI process failures. Failure notifications are propagated (according to the simulated architecture) to each simulated MPI process, which in turn reacts to the fault. xSim offers full support for application-level checkpoint/restart and MPI user-level failure mitigation, i.e., a fault-tolerant MPI [2].

1.3 Contribution

xSim accounts for the execution time of each simulated MPI process using a processor model. It also accounts for the wait time incurred by simulated MPI communication using a network model. xSim's network model uses topology models with static routing and parametrized latency and bandwidth. Supported topologies include star, ring, mesh, torus, twisted torus, and tree. Hierarchical combinations, such as to simulate network-on-chip and network-on-node, as well as, rendezvous protocol and source/destination process contention simulation are supported as well. However, the network model does not provide full contention modeling due to the impact it would have on the simulators scalability and usability. A highly accurate simulation that is too slow to deliver results in a reasonable time frame does not provide much benefit.

This paper revisits this assumption and details a newly developed network contention and bandwidth capacity modeling feature for xSim. The approach uses a less synchronous and accurate enough model design. With the new feature, xSim is able to simulate on-chip and on-node networks with reasonable accuracy and overheads.

The following provides an overview of xSim's design to identify the inherent challenges in implementing a scalable network model, describes the existing network modeling capabilities and their shortcomings, and illustrates the new network modeling features that eliminate these shortcomings. The paper continues with a series of experimental results demonstrating the new network modeling features and a short discussion of related work. This paper concludes with a brief summary of the presented work.

2 Overall Simulator Design

xSim is designed like a traditional HPC application performance investigation tool, as an interposition library that sits between the MPI application and the MPI library, using the MPI performance tool interface (PMPI) to intercept MPI calls made by the application. It supports simulated point-to-point and collective MPI communication, as well as, simulated MPI data types, groups, communicators, and communication request objects. In total, xSim supports 92 simulated MPI functions for each supported programming language, C, C++, and Fortran. An application is run in the simulator using the following steps:

- Add the xSim header file to the application source code:
  - Add "#include xsim-c.h" for C applications
  - Add "#include xsim-cxx.h" for C++ applications
  - Add "#include xsim-f.h" for Fortran applications
- Recompile the application and link it with the xSim simulator library and the respective xSim programming language interface library:
  - Link with "-lxsim -lxsim-c" for C applications
  - Link with "-lxsim -lxsim-cxx" for C++ applications
  - Link with "-lxsim -lxsim-f" for Fortran applications
- Run the application with:
  - "mpirun -np <physical process cout> <application> -xsim-np <simulated process count> <simulation parameters> <application parameters>"

The simulator is implemented in C using two (pthread) threads per physical MPI process. In the communication thread, the PDES receives and enqueues simulated MPI messages from the native MPI library into an incoming message queue. This thread also receives and processes PDES control messages from the native MPI library. In the simulation thread, the core mechanism of the PDES dequeues every received simulated MPI message from the incoming message queue and switches into the context of the destined simulated MPI process for consumption. A simulated MPI process receives its MPI messages indirectly by dequeueing them from the queue and yielding to another simulated MPI process if no matching message is in the queue. A simulated MPI process sends its MPI messages directly within the simulation thread utilizing the native MPI library. A user-space process threading capability within the simulation thread and the decoupling of receiving/pre-processing and processing/sending offers optimal performance and oversubscription capability without being synchronized by the native MPI library.

3 Existing Network Modeling Capabilities

xSim utilizes a network architecture model to account for the wait time incurred by communication between simulated MPI processes. Every simulated point-to-point MPI communication has an associated cost, which reflects the time needed to transmit the message by the source and to receive it by the destination. The network model is applied upon sending and receiving a simulated MPI message.

3.1 Architectural Model

With every simulated `MPI_Isend()` call that initiates a send operation, the message transmission times are established. The point in time to send the first byte t_{s1} is initialized with the source's simulated MPI process time. The 0-byte round-trip latency l_r to the destination is added to t_{s1} if the message size meets or exceeds the MPI rendezvous protocol threshold. l_r is calculated using the statically routed network distance (hop count) h and the link latency l in the simulated network architecture: $l_r = 2 * h * l$.

Some simulated network architectures, such as mesh and torus, support different link latencies in different network dimensions. In this case, each link latency is added separately along the network route. The point in time to send the last byte t_{s2} is initialized with t_{s1} and increased by the message send time using the network bandwidth b and the message size s: $t_{s2} = t_{s1} + (s/b)$. Some simulated network architectures, such as mesh and torus, support different link bandwidths in different network dimensions. In this case, the lowest bandwidth along the network route is considered. The point in time to receive the first byte t_{r1} is initialized with t_{s1} and increased by the 0-byte one-way latency l_0 to the destination, where $l_0 = l_r/2$. The point in time to receive the last byte t_{r2} is initialized with t_{s2} and increased by l_0 as well. This basic model only considers network structure, latencies, and bandwidths.

To model source-side contention, each simulated MPI process maintains an additional time $t_{previous\_s2}$ to save the point in time the last byte of the previous simulated MPI message was sent. Each time the network model is applied upon sending a simulated MPI message, the point in time to send the first byte t_{s1} is calculated as described and compared with $t_{previous\_s2}$. t_{s1} is set to $t_{previous\_s2}$ if it is less than $t_{previous\_s2}$. All values that depend on t_{s1} are calculated after modeling source-side contention. This effectively serializes the transmission times for messages from the same source.

The message transmission times are calculated as part of the simulated `MPI_Isend()` call by the source to initiate the send operation. While t_{s2} is saved in the associated MPI request object and $t_{previous\_s2}$ is stored in the simulated MPI process object, t_{r1} and t_{r2} are transmitted with the simulated MPI message. The simulated `MPI_Wait()` call by the source to complete the send operation compares the current simulated MPI process time with the saved t_{s2} and adds a wait time to the simulated MPI process time if t_{s2} exceeds it. This properly models the source side of non-blocking point-to-point MPI communication.

To additionally model destination-side contention, each simulated MPI process maintains an additional time $t_{previous\_r2}$ to save the point in time the last byte of the previous simulated MPI message was received. Each time a message is received, the point in time to receive the first byte t_{r1} that is contained in the message is compared with $t_{previous\_r2}$. t_{r1} set to $t_{previous\_r2}$ and t_{r2} is advanced accordingly by $t_{previous\_r2} - t_{r1}$ if t_{r1} is less than $t_{previous\_r2}$. This effectively serializes the transmission times for messages to the same destination.

The simulated `MPI_Irecv()` call by the destination to initiate the receive operation saves the current simulated MPI process time as t_{post} in the associated MPI request object. The `MPI_Wait()` call by the destination to complete the receive operation applies the destination-side contention model as described and compares the previously saved simulated MPI process time t_{post} with t_{r1}. t_{r1} set to t_{post} and t_{r2} is advanced accordingly by $t_{post} - t_{r1}$ if t_{r1} is less than t_{post}. The simulated `MPI_Wait()` call also compares the current simulated MPI process time with t_{r2} and adds a wait time to the simulated MPI process time if t_{r2} exceeds it. This properly models the destination side of non-blocking point-to-point MPI communication.

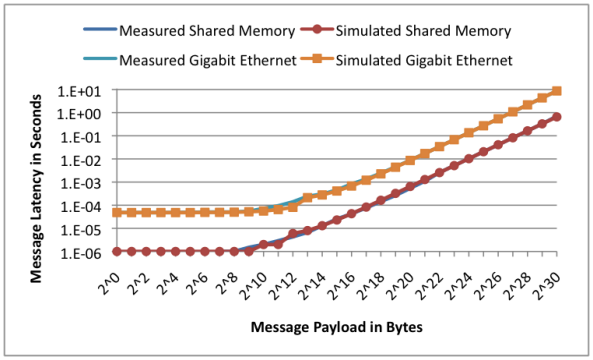


Figure 1. Shared memory and Gigabit Ethernet latency

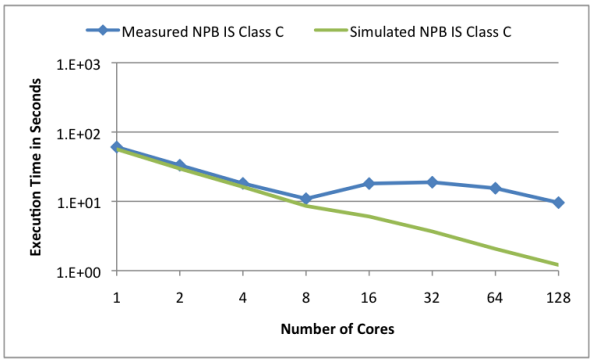


Figure 2. Native and simulated NPB IS performance

3.2 Hierarchies and Collectives

Hierarchical combinations, such as a HPC system with an on-chip, an on-node and a system network, are simulated by identifying the lowest-level network that source and destination reside in and using its model. For example, two simulated MPI processes located on the same processor would only use the on-chip network model, while two simulated MPI processes located on different compute nodes would only use the system network model. This simplification is appropriate as lower-level networks in HPC systems typically have significantly lower link latency and significantly higher link bandwidth.

Collective MPI operations, such as a broadcast or a reduction, are constructed from non-blocking point-to-point MPI communication operations. xSim supports linear implementations, as well as, log_2 variants.

3.3 Shortcomings

The described existing network modeling capabilities in xSim have been shown to be accurate enough for a number of applications [5, 9]. Figure 1, shows the payload-dependent native and simulated shared memory and Gigabit Ethernet MPI message latency in a multi-core Linux cluster, for example. xSim is able to closely simulate the performance of the native MPI library. However, the network model does not provide full contention modeling, i.e.,

the simulation does not account for the fact that a single network interface card is shared between multiple cores and that on-node core-to-core communication is limited by memory bandwidth. Figure 2 demonstrates this deficiency by executing the IS program of the NAS Parallel Benchmark (NPB) suite [11] with a class C problem size on a 16-node Linux cluster with 8 cores per node. The benchmarks stops scaling at 8 cores on the native system, but continues to scale in the simulation. The root cause of this simulation error is the lack of network contention modeling, as the NPB IS program is an integer sort with heavy communication.

4 New Network Modeling Features

To alleviate the identified shortcomings, the existing network modeling capabilities have been extended to include modeling contention in network hierarchies and modeling network bandwidth capacity. The main challenge for implementing these features is the potential for global synchronization. Recent work [8] in improving xSim's deadlock resolution protocol and simulated MPI message matching algorithm revealed that the execution time overhead introduced by xSim's PDES can surge from 238 % to 13,808 % with the global synchronization caused by the optional accurate MPI process failure simulation. This is due to the fact that sending a simulated MPI message becomes a highly synchronizing event with accurate MPI process failure simulation, significantly slowing down the otherwise asynchronous operation of the PDES. The targeted network modeling features utilize coordination and flow of information without significant synchronization to track the usage of the simulated network and to infer contention without introducing significant PDES execution time overheads.

4.1 Network Contention Modeling

The first newly implemented feature deals with network contention caused by sharing a single common access point to a higher-level network by a lower-level network. A typical example is the single network interface card that is shared between multiple processor cores on the same compute node. While the processor cores form an on-node network, its access to the higher-level system network uses a single network card. The existing network modeling capability in xSim simply simulates full bandwidth access to the system network for each processor core, ignoring the bandwidth bottleneck the network card introduces.

To add network contention modeling between different networks, a highly accurate approach would serialize and order all simulated communication going in and out of each network by their time stamps, such that the network contention model can be applied correctly to each message. This approach, however, essentially serializes all simulated communication, causing a global synchronization effect. An alternative approach is to model network contention less accurately by applying the network contention model to each message without serializing and ordering it. Due to the asynchronous operation of the PDES, physical message transmission times and order do not match simulated message transmission times and order. This introduces a simulation accuracy error as the network contention model may be applied to messages in the incorrect order, i.e., one message arrives too early and another message too late by the same time difference. Given the goal of xSim to provide a scalable simulation environment, the less accurate network contention modeling approach is more appropriate.

The existing network modeling capabilities have been extended by a mechanism for tracking the incoming and outgoing message traffic for each simulated network. This tracking mechanism are lists of incoming and outgoing message transmission times for each simulated network. For each message, the point in time to transmit the first byte t_{t1} and the point in time to transmit the last byte t_{t2} are stored in an incoming or outgoing traffic list. Every time the transmission times of a new message are added, the list is traversed to identify conflicts and the transmission times of the new message are moved accordingly to resolve conflicts. If there is no conflict or once a conflict has been resolved, there is no contention and the message transmission times are simply added to the list. A message may be fragmented due to already existing message transmission times in the list. In this case, the point in time to transmit the last byte t_{t2} is extended accordingly and the point in time to transmit the first byte t_{t1} is recalculated to maintain the bandwidth relationship between t_{t1} and t_{t2}. This is necessary as t_{t1} and t_{t2} are used by other parts of the network model that are not aware of fragmentation.

As part of the simulated `MPI_Isend()` call, the network contention model is applied once the point in time to send the first byte t_{s1} and the point in time to send the last byte t_{s2} have been calculated. t_{t1} and t_{t2} are initialized with t_{s1} and t_{s2}. The network contention model is applied for each simulated network by adding the message transmission times to the outgoing message traffic list of a lower-level network when the message is routed to a higher-level network and to the incoming message traffic list of a lower-level network when the message is routed to the lower-level network. Once the network contention model is applied, the point in time to receive the first byte t_{r1} is initialized with t_{t1} and increased by the 0-byte one-way latency l_0. The point in time to receive the last byte t_{r2} is initialized with t_{t2} and increased by l_0 as well.

The incoming and outgoing traffic lists are maintained by the lowest member of each simulated network, e.g, by the first core within each simulated compute node. In contrast to a fully centralized solution, this distributed approach enables performance gains when simulating with oversubscription. Multiple simulated MPI processes located on the same physical MPI processes are able to access some of the traffic lists directly, without the need to send/receive a native MPI message to/from a central network contention model manager. The lists are regularly cleared of old transmission times to keep memory usage low using the global virtual time of the PDES to identify the timeline all simulated MPI processes have passed. The

network contention model can be configured for each simulated network separately, i.e., it can be enabled/disabled.

4.2 Network Bandwidth Capacity Modeling

The second newly implemented feature deals with network bandwidth capacity restrictions that exist for networks with a total routable bandwidth that is less than the total possible bandwidth created through message injection. A typical example is the shared memory communication between cores on the same processor, where the main memory bandwidth limits all-to-all communication between cores. The existing network modeling capability in xSim simply simulates full all-to-all bandwidth within the on-chip network, ignoring the main memory bottleneck.

The existing network modeling capabilities have been extended by a mechanism for tracking the message traffic within each simulated network, independent from source and destination. This tracking mechanism consists of one additional message transmission times list for each simulated network. For each message, the point in time to transmit the first byte t_{x1} and the point in time to transmit the last byte t_{x2} are stored in this transfer traffic list. Before storing, t_{x2} is scaled by the difference between the network's point-to-point bandwidth b and total bandwidth capacity c: $t_{x2} = t_{x1} + (t_{x2} - t_{x1})(b/c)$. Every time the transmission times of a new message are added, the list is traversed to identify conflicts and the transmission times of the new message are moved accordingly to resolve conflicts. If there is no conflict or once a conflict has been resolved, there is no transfer contention and the message transmission times are simply added to the list. A message may be fragmented due to already existing message transmission times in the list. In this case, the point in time to transmit the last byte t_{x2} is extended accordingly and the point in time to transmit the first byte t_{x1} is recalculated to maintain the bandwidth relationship between t_{x1} and t_{x2}. Once added to the list, the scaling of t_{x1} and t_{x2} by the difference between the network's point-to-point bandwidth b and total bandwidth capacity c is reversed.

The simulated `MPI_Isend()` call applies this bandwidth capacity model for the lowest-level network that source and destination reside in, right after applying the network contention model for the outgoing traffic of the lower-level networks and before applying the network contention model for the incoming traffic of the lower-level networks. t_{x1} and t_{x2} are initialized with t_{t1} and t_{t2} from the network contention model if there is a lower-level network, or with t_{s1} and t_{s2} if not. t_{t1} and t_{t2} of the lower-level network, or t_{s1} and t_{s2} if there is none, are initialized with t_{x1} and t_{x2} afterwards. The transfer traffic lists are maintained by the lowest member of each simulated network, e.g, by the first core within each simulated compute node to maintain a distributed design approach. The transfer lists are regularly cleared. The network bandwidth capacity model can be configured for each simulated network separately as well, i.e., it can be enabled/disabled and the bandwidth capacity can be set.

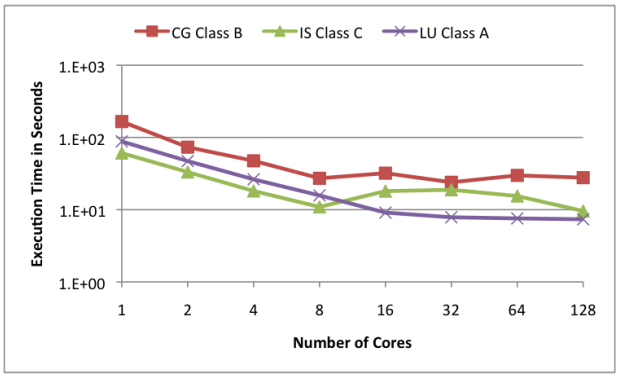


Figure 3. Native NPB CG and IS performance

5 Experimental Evaluation

The accuracy and overhead of the newly developed network modeling features for xSim have been evaluated against the existing network modeling capabilities and the physical system being simulated.

5.1 Evaluation Setup

The experiments were performed on a 128-core Linux cluster computer with 16 compute nodes, two 2.4 GHz AMD Opteron 2378 processors per node, 4 cores per processor, 8 GB RAM per node, and a non-blocking 1 Gbps Ethernet interconnect. The system is running Ubuntu 12.04 LTS, Open MPI 1.6.4, and GCC 4.6.

The experiments used a subset of the NAS Parallel Benchmark (NPB) suite [11], which was developed by the NASA Advanced Supercomputing (NAS) Division to aid in evaluating the performance of HPC systems. The benchmark programs were derived from computational fluid dynamics, unstructured adaptive mesh, parallel I/O, multizone, and computational grid applications. The input problem sizes are predefined and organized in classes, e.g., A, B, C, and D. The experiments in this paper utilize on the following benchmarks and classes:

- CG, a conjugate gradient solver (class B),
- IS, an integer sort (class C), and
- LU, a Lower-Upper Gauss-Seidel solver (class A).

These benchmarks and their classes were specifically selected as they represent particular features of interest. Figure 3 shows their execution times on the Linux cluster. NPB CG and IS scale up to the 8 processor cores located within the same compute node. The execution time of NPB CG beyond 8 processor cores stays roughly flat, while the execution time of NPB IS goes initially slightly up and then comes slightly down again at 128 cores. The execution time of NPB LU scales to 16 processor cores and then stays flat. The three benchmarks experience network contention issues.

5.2 Simulator Configuration

For the experiments using xSim, the simulated network has been configured to replicate the structure and capabilities of the physical evaluation platform. At the lowest

level, a network of 4 compute cores was configured with a shared memory core-to-core MPI message latency of 1 μs and bandwidth of 12,487.8 Mb/s, and a MPI rendezvous threshold of 4 kB. At the next level up, a network of 2 compute processors was configured with a shared memory processor-to-processor MPI message latency of 1 μs and bandwidth of 12,487.8 Mb/s, and a MPI rendezvous threshold of 4 kB. At the top level, a network of 16 compute nodes was configured with a Gigabit network node-to-node MPI message latency of 48 μs and bandwidth of 944.146 Mb/s, and a MPI rendezvous threshold of 8 kB. The network configuration values have been experimentally obtained and are a close match in the simulation (see Figure 1).

For the experiments that employ the network contention modeling feature, the network of 4 compute cores was configured with contention for accessing the network of 2 compute processors. The network of 2 compute processors was configured with contention for accessing the system network. For the experiments that employ the network bandwidth capacity modeling feature, the network of 4 compute cores was configured with a bandwidth capacity of 12,487.8 Mb/s as the shared memory is a bottleneck.

A native MPI process is placed on each physical core in the experiments, while each simulated MPI process is executed within one native MPI process. This one-to-one matching of the native and the simulated environment assures a fair evaluation.

5.3 Evaluation Results

Scaling experiments were performed from 1-128 processor cores for the NPB CG, IS and LU benchmarks running (a) on the native system without xSim, (b) within xSim using the original network model, (c) within xSim using the added network contention model, and (d) within xSim using the network contention and bandwidth capacity models.

Figure 4(a) shows the execution time of the NPB CG (class C) benchmark. The simulation error introduced by xSim using the original network model is clearly visible. Over the entire series, including 1-8 cores where it matches up, the average simulation error is 36.9 % and the maximum error is 87 %. With network contention modeling, the error is visibly reduced. The average simulation error is 10.6 % and the maximum error is 26.2 %. Adding the network bandwidth capacity model reduces the average error to 9.4 % and the maximum error is 24.3 %.

Similar results are shown in Figure 4(b) for the NPB IS (class C) benchmark. The simulation error introduced by xSim and its reduction using the added network contention and bandwidth capacity models are clearly visible. The average simulation error by xSim using the original network model of 46 % and the maximum error of 87.5 % are reduced to an average error of 14 % and a maximum error of 27.3 %. The average simulation error is reduced to 12.2 % and the maximum error is reduced to 17.5 % using the additional bandwidth capacity model.

For NPB LU (class A), as shown in Figure 4(c), the simulation error introduced by xSim using the original network model is 30.6 % on average with a maximum of 65 %. They are reduced to 16.4 % and 44 %, respectively, with the network contention model. The additional bandwidth capacity model does not yield a reduction of the average error, but of the maximum error. The average error increases to 18 %, while the maximum error decreases to 42.5 %. Even with the new network modeling capabilities, NPB LU does not fully flatten out at larger core counts. This is likely due to the less accurate (but more scalable) simulation of the new network modeling capabilities.

Figure 5(a) illustrates the overhead introduced by xSim in comparison to the native NPB CG execution. The simulation overhead average and maximum over the entire series increases only slightly from 228 % and 506 % to 230 % and 517 % with the network contention model as more messages are being processed by the PDES. The simulation overhead actually decreases dramatically to 136% and 318% with the added bandwidth capacity model. This is likely due to internal optimizations of the PDES when there is more up-to-date information communicated.

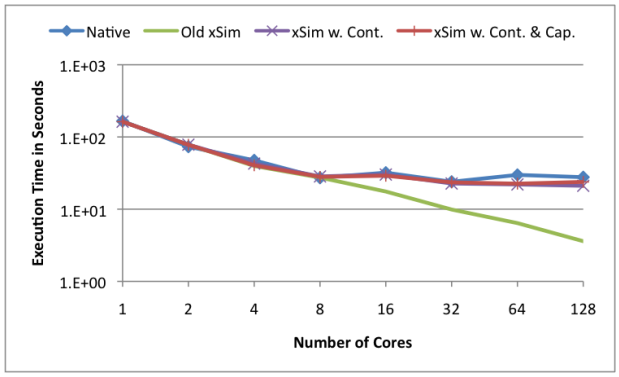
In Figure 5(b), the overhead introduced by xSim for NPB IS when going from the original network model to the network contention model is increased from an average of 56.4 % to 63.3 %, while the maximum is decreased from 283 % to 187 %. It is increased to 97.5 % and 300 % when employing the additional bandwidth capacity model.

For NPB LU (Figure 5(c)) the overhead is initially drastically increased from 22.9 % (average) and 50.8 % (maximum) to 1,825 % and 5,390%, and then further drastically increased to 4,451 % and 14,247%, respectively. This huge performance impact is likely due to the fact that the additional messages of the new network modeling features disturb the highly bulk-synchronous operation of the Lower-Upper Gauss-Seidel solver.

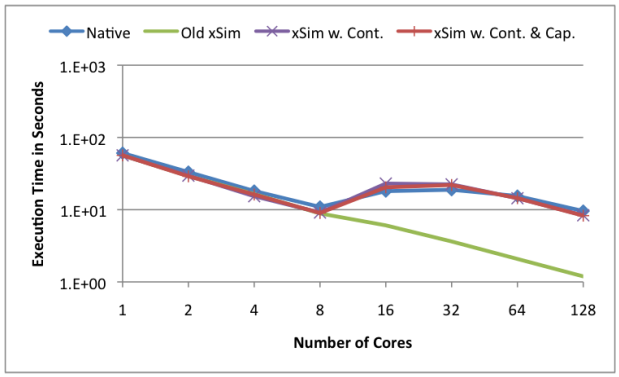
6 Related Work

The BigSim [1] project studied programming issues in large-scale HPC systems. The BigSim Emulator was developed for application testing and debugging at scale and runs atop Charm++/AMPI. It supports up to 100,000 simulated MPI processes distributed over 2,000 cores. It does not offer time-accurate simulation. The BigSim Simulator was developed for identification of performance bottlenecks and uses a trace-driven PDES that models architectural parameters of a HPC system. For time-accurate simulation, it supports a variable-resolution processor model and a detailed network model.

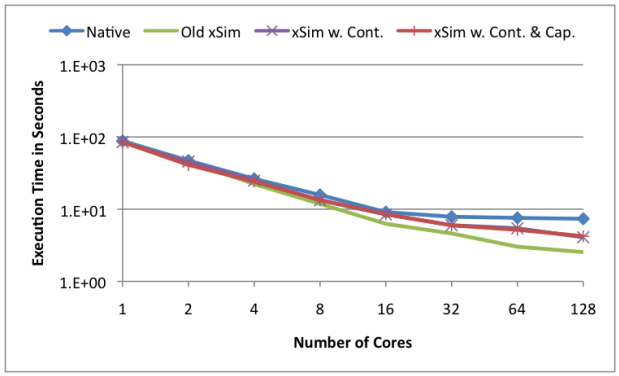
The Structural Simulation Toolkit (SST) [13] offers simulation of novel architectures, including processor, memory, and network. It is a modular PDES framework atop MPI that scales to a few hundred simulated multi-core nodes. Its value is in the ability to investigate the performance of future node architectures and to generate models for larger-scale simulations. SST/macro is a complementary simulation toolkit that processes output from the MPI tracing library DUMPI for performance evaluation. Similar to the BigSim Emulator/Simulator combination, SST and



(a) NPB CG (class C)

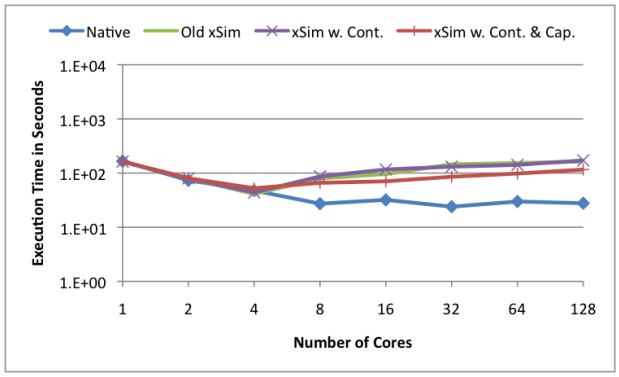


(b) NPB IS (class C)

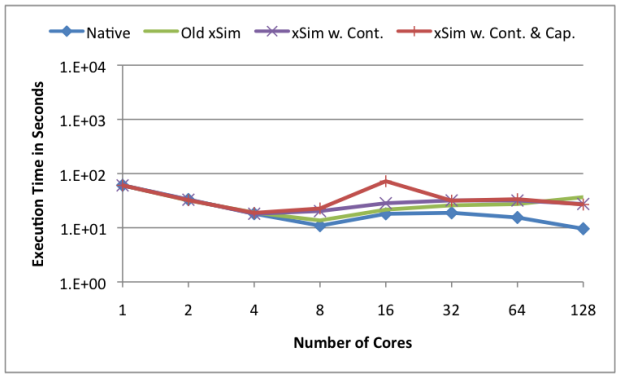


(c) NPB LU (class A)

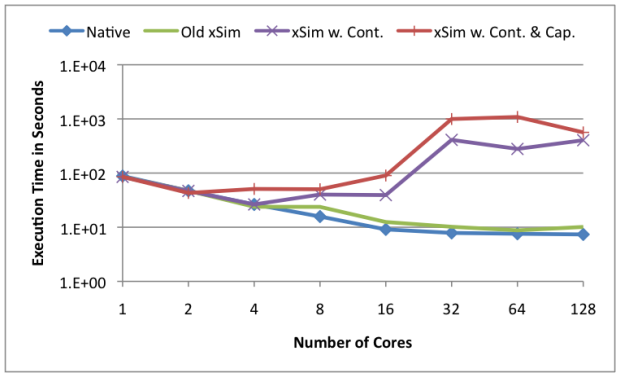
Figure 4. Native (without xSim in wall-clock time) and simulated NPB performance (in simulated time) with original network model (Old xSim), added network contention model (xSim w. Cont.), and added network bandwidth capacity model (xSim w. Cont. & Cap.)



(a) NPB CG (class C)



(b) NPB IS (class C)



(c) NPB LU (class A)

Figure 5. Native (without xSim in wall-clock time) and simulator performance (in wall-clock time) with original network model (Old xSim), added network contention model (xSim w. Cont.), and added network bandwidth capacity model (xSim w. Cont. & Cap.)

SST/macro enable the synergy between small-scale cycle-accurate and large-scale communication-accurate simulations. While SST is mature, it is quite complex to use. SST/macro is still under development.

There are a also variety of network simulators, such as ns3 (http://www.nsnam.org) and NetSim (http://tetcos.com/software.html), that are able to provide network performance metrics at various abstraction levels, such as network, sub-network, and packet traces. These detailed simulators offer high-accuracy/low-scalability results that are not compatible with the high-scalability approach needed for extreme-scale system simulation.

7 Conclusions

This paper detailed a newly developed network modeling feature for xSim, eliminating the shortcomings of the existing network modeling capabilities in simulation accuracy. The approach takes a different path for implementing network contention and bandwidth capacity modeling using a less synchronous and accurate enough model design.

The evaluation results of the implemented network contention and bandwidth capacity model show that the maximum simulation error introduced by xSim using the original network model could be reduced from 87 % to 24.3 % for NPB CG, from 87.5 % to 17.5 % for NPB IS, and from 65 % to 42.5 % for NPB LU. The additional simulation overheads for NPB CG and IS are minimal, while the simulation overhead increases for NPB LU are substantial, but manageable (in comparison to accurate MPI process failure simulation). With the new network modeling feature, xSim is able to simulate on-chip and on-node networks with reasonable accuracy and overheads.

8 Acknowledgements

This work was sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL) and the U.S. Department of Energy's Office of Advanced Scientific Computing Research.

This manuscript has been authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

References

[1] A. Bhatele, N. Jain, W. Gropp, and L. Kale. Avoiding hot-spots on two-level direct networks. In *International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 1–11, 2011.

[2] W. Bland, A. Bouteiller, T. Herault, J. Hursey, G. Bosilca, and J. J. Dongarra. An evaluation of user-level failure mitigation support in MPI. In *19th European conference on Recent Advances in the Message Passing Interface (EuroMPI)*, pages 193–203, 2012.

[3] S. Böhm and C. Engelmann. xSim: The extreme-scale simulator. In *International Conference on High Performance Computing and Simulation (HPCS)*, pages 280–286, 2011.

[4] C. Engelmann. Investigating operating system noise in extreme-scale high-performance computing systems using simulation. In *IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN)*, 2013.

[5] C. Engelmann. Scaling to a million cores and beyond: Using light-weight simulation to understand the challenges ahead on the road to exascale. *Future Generation Computer Systems (FGCS)*, 30(0):59–65, 2014.

[6] C. Engelmann and F. Lauer. Facilitating co-design for extreme-scale systems through lightweight simulation. In *Workshop on Application/Architecture Co-design for Extreme-scale Computing (AACEC)*, pages 1–8, 2010.

[7] C. Engelmann and T. Naughton. Toward a performance/resilience tool for hardware/software co-design of high-performance computing systems. In *International Conference on Parallel Processing (ICPP): International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*, pages 962–971, 2013.

[8] C. Engelmann and T. Naughton. Improving the performance of the extreme-scale simulator. In *IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, pages 198–207, 2014.

[9] I. S. Jones and C. Engelmann. Simulation of large-scale HPC architectures. In *International Conference on Parallel Processing (ICPP): International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*, pages 447–456, 2011.

[10] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon. Top 500 List of Supercomputer Sites, 2014. http://www.top500.org.

[11] National Aeronautics and Space Administration. NAS Parallel Benchmarks, 2014. http://www.nas.nasa.gov/Resources/Software/npb.html.

[12] T. Naughton, C. Engelmann, G. Vallée, and S. Böhm. Supporting the development of resilient message passing applications using simulation. In *Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP)*, pages 271–278, 2014.

[13] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38(4):37–42, 2011.