Scaling To A Million Cores And Beyond: Using Light-Weight Simulation to Understand The Challenges Ahead On The Road To Exascale

Christian Engelmann

Oak Ridge National Laboratory, P.O. Box 2008, Oak Ridge, TN 37831-6173, USA

Abstract

As supercomputers scale to 1,000 PFlop/s over the next decade, investigating the performance of parallel applications at scale on future architectures and the performance impact of different architecture choices for high-performance computing (HPC) hardware/software co-design is crucial. This paper summarizes recent efforts in designing and implementing a novel HPC hardware/software co-design toolkit. The presented Extreme-scale Simulator (xSim) permits running an HPC application in a controlled environment with millions of concurrent execution threads while observing its performance in a simulated extreme-scale HPC system using architectural models and virtual timing. This paper demonstrates the capabilities and usefulness of the xSim performance investigation toolkit, such as its scalability to 2^{27} simulated Message Passing Interface (MPI) ranks on 960 real processor cores, the capability to evaluate the performance of different MPI collective communication algorithms, and the ability to evaluate the performance of a basic Monte Carlo application with different architectural parameters.

Keywords: Parallel Discrete Event Simulation, Message Passing Interface, Collective Communication, High Performance Computing, Exascale

1. Introduction

With the recent deployment of 10-20 PFlop/s (1 PFlop/s = 10^{15} floating-point operations per second) supercomputers and the exascale roadmap targeting 100, 300, and eventually 1,000 PFlop/s over the next decade, the trend in supercomputer architecture goes clearly in only one direction. Systems will dramatically scale up in size, i.e., in compute node and processor thread counts. By 2020, an exascale system may have 1,000,000 compute nodes with 1,000-10,000 threads per node. This poses several challenges related to power consumption, performance, resilience, productivity, programmability, data movement, and data management.

The expected growth in concurrency from today's 1.57 million hardware threads in the IBM BlueGene/Q Sequoia supercomputer at Lawrence Livermore National Laboratory to 1-10 billion hardware threads at exascale, causes parallel application scalability issues due to sequential application parts, synchronizing communication, and other bottlenecks. High-performance computing (HPC) hardware/software co-design is crucial to enable extreme-scale computing by closing the gap between the peak capabilities of the hardware and the performance realized by HPC applications (application-architecture performance gap).

Investigating the performance of parallel applications at scale on future architectures and the performance impact of different architecture choices is an important component of HPC hardware/software co-design. Without having access to future architectures at scale, simulation approaches provide an alternative for estimating parallel application performance on potential architecture choices. As highly accurate simulations are extremely slow and less scalable, different solution paths exist to trade-off simulation accuracy in order to gain simulation performance and scalability.

This paper summarizes recent efforts in designing and implementing a novel HPC hardware/software co-design toolkit. The presented work focuses on a lightweight parallel discrete event simulation (PDES) solution for investigating the performance of Message Passing Interface (MPI) applications at extreme scale. The developed Extreme-scale Simulator (xSim) permits running an HPC application in a controlled environment with millions of concurrent execution threads while observing its performance in a simulated extreme-scale system using architectural models and virtual timing. This paper demonstrates the capabilities and usefulness of the xSim performance toolkit. Specifically, it shows:

- the scalability to 134,217,728 (2^{27}) simulated MPI ranks, each with its own context, on a 960-core Linux cluster (a world record in extreme-scale simulation);
- the capability to evaluate the performance of MPI collective communication algorithms on 2,097,152 (2^{21}) simulated MPI ranks using the

same cluster; and

- the ability to estimate the performance of a Monte Carlo solver on 16,777,216 (2^{24}) simulated MPI ranks with varying architectural parameters using the same cluster.

The paper is structured as follows. Section 2 briefly discusses related work, while Section 3 provides an overview of xSim's architecture and design. Section 4 demonstrates the capabilities and usefulness of xSim via a variety of experimental results. Sections 5 and 6 conclude this paper with a summary and an outlook on future work.

2. Related Work

xSim's predecessor, the *Java Cellular Architecture Simulator (JCAS)* [1], was developed in 2001 to investigate scalable fault-tolerant algorithms for large-scale systems. The prototype was able to run up to 500,000 simulated processes on a cluster with 5 native processors (using 1 for visualization) solving basic mathematical problems. While it was able to run algorithms at scale, it lacked important features, such as time-accurate simulation, high performance, support for running the simulator atop MPI, and a fully functional simulated MPI. The JCAS project sparked a new area of research in fault-tolerant algorithms [2, 3, 4].

The BigSim [5] project was a competitor to JCAS and initiated to study programming issues in large-scale HPC systems. The *BigSim Emulator* is for application testing and debugging at scale and build atop Charm++ and AMPI [6]. It supports up to 100,000 simulated MPI processes distributed over 2,000 native processors. Similar to JCAS, it does not offer time-accurate simulation. It offers more functionality, but scales worse. The *BigSim Simulator* is 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. While it uses a PDES to maintain accuracy, it does not run applications.

$\mu\pi$ [7] is a PDES-based system under development for predicting the performance of parallel programs. It uses different grafting methods for interfacing applications with the simulation, such as at the source code, library (currently implemented), and virtual machine level. It supports conservative and optimistic execution based on the μsik PDES engine (http:

//kalper.net/kp/software/musik). A prototype was tested on 216,000 cores of the Jaguar Cray XT5 system, providing over 27 million simulated MPI ranks, each with its own thread context, and all ranks synchronized by simulated time. xSim is far more scalable as $\mu\pi$ requires an extreme-scale system to simulate an extreme-scale system. $\mu\pi$'s PDES engine μsik is superior.

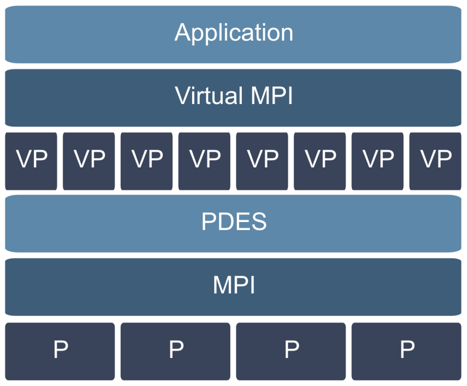
The Structural Simulation Toolkit (SST) [8] (http://www.cs.sandia.gov/sst) offers simulation of novel architectures, including processor, memory, and network. It is a modular PDES framework atop MPI that scales to a few simulated multi-core nodes. Its value is in the ability to investigate the performance 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 (http://sst.sandia.gov/about_dumpi.html) for performance evaluation. Similar to the BigSim Emulator/Simulator combination, SST and 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.

Other trace-driven PDES solutions for investigating application performance exist. For example, DIMEMAS [9] processes traces from MPIDTrace and generates trace files suitable for the two performance tools, PARAVER [10] (http://www.bsc.es/computer-sciences/performance-tools/paraver) and Vampir [11] (http://www.vampir.eu).

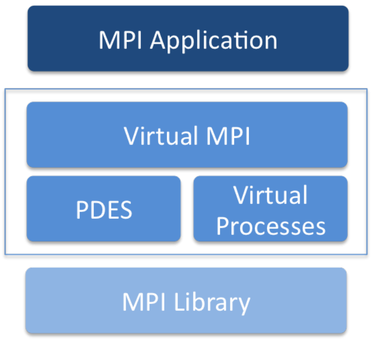
There are a also variety of network simulators, such as ns3 (http://www.nsnam.org) and NetSim (http://tetcos.com/software.html), that are able to provide network performance metrics at various abstraction levels, such as network, sub-network, and packet traces. These detailed simulators offer high-accuracy/low-scalability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

3. xSim: The Extreme-scale Simulator

The Extreme-scale Simulator (xSim) is a performance investigation toolkit that permits running HPC applications in a controlled environment with millions of concurrent execution threads. It allows observing application performance in a simulated extreme-scale HPC system for hardware/software co-design. Much of its architecture and design details has been published before [12, 13, 14, 15] and is only summarized in this paper. Additionally, a few new features have been added that will be detailed in this paper.



(a) Architecture (© 2010 IEEE [14])



(b) Design (© 2010 IEEE [14])

Figure 1: xSim's implementation architecture and design.

Using a lightweight PDES (Figure 1(a)), xSim executes a parallel application on a much smaller HPC system in an oversubscribed fashion with a simulated wall clock time, such that performance data can be extracted based on processor and network model. xSim is designed like a traditional performance tool (Figure 1(b)), as an interposition library between the MPI application and the MPI library, using the MPI performance tool interface. It has support for simulated MPI point-to-point communication, groups, communicators, and collective communication, and for native MPI data types. In total, xSim offers 81 MPI functions for each of the supported programming languages, C and Fortran.

xSim employs its own user-space process threading for optimal performance, offering each simulated MPI rank its own full thread context (CPU registers, stack, heap, and global variables) while retaining control over the thread schedule [12]. The network model offers latency and bandwidth restrictions with different network architectures, such as star, ring, mesh, torus, twisted torus and tree [13]. It also supports hierarchical combinations, such as for a network-on-chip and network-on-node, as well as, rendezvous protocol and sender/receiver network interface contention simulation. For scalability reasons, the network model does not provide full contention modeling at this point. The processor model is based on the actual single-core execution time on the real hardware platform scaled to the simulated processor core speed [14], which is estimated using external node architecture simulators, such as SST. xSim supports operating system (OS) noise simulation by injecting waste time at the processor model using the abstraction of OS noise frequency (periodic recurrence) and period (duration of each occurrence) [15].

A recent extension to the processor model is the support for heterogeneous architectures. This new feature allows to simulate cores with different relative speeds within the same processor, processors with different core counts and speeds within the same node, and node configurations with different processors. Accelerator-based architectures, such as CPU-GPGPU combinations are currently not supported.

For better simulation flexibility and scalability, xSim was recently improved with support for the execution of application models. Similar to an MPI trace replay, an application model feeds the simulator with timing and communication behavior, but does not require certain native resources to scale with the simulation. It simply advances the simulated time according to the time the application would have spent between MPI calls and executes MPI calls without actually sending message payloads and without the need for valid buffers. While no real application code is executed and only message envelopes are sent, both, the processor and network model account for the virtual time an application would have spent on these activities.

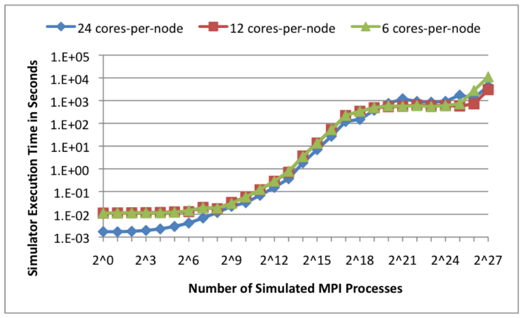
xSim was also recently enhanced to offer full support for error handling within the simulated MPI, including default MPI error handlers, user-defined MPI error handlers, and *MPI_Abort()*. xSim virtualizes the MPI error handling, such that errors caused by the application or injected by xSim for application testing purposes are handled by xSim according to the MPI standard. Any simulated abort results in a termination of the simulation with performance results and information about the source and time of the abort.

4. Experimental Results

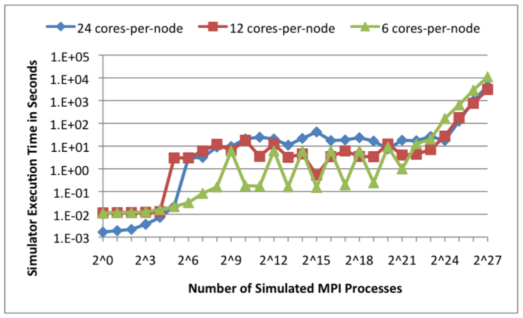
The main contribution of this paper is the demonstration of the capabilities and usefulness of xSim. Experiments were executed focusing on evaluating xSim's scalability, using it for investigating the performance of different MPI collective communication algorithms, and employing it to identify the impact of different architectural choices on the performance of a basic Monte Carlo application. The experiments were performed on a 960-core Linux cluster with 40 compute nodes, two 1.7 GHz AMD Opteron 6164 HE processors/node, 12 cores/processor, 64 GB RAM/node, and a bonded dual non-blocking 1 Gbps Ethernet interconnect. The system is running Ubuntu 10.04 LTS and Open MPI 1.5.5. Despite its small size, in comparison to the simulated systems, this system has been proven to be particularly useful due to its large amount of total RAM (2.5 TB).

4.1. Simulator Performance

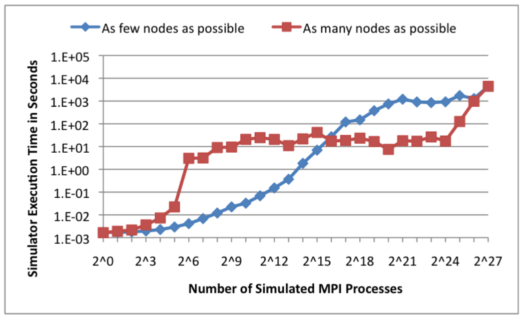
The first series of experiments evaluate the scalability of xSim itself by executing a basic MPI program that only calls *MPI_Init()* and *MPI_Finalize()* without performing any communication or input/output (I/O) operations. While it is clear that an x-times oversubscribed execution of a parallel program atop xSim would cause a corresponding x-times native execution overhead, such as by running 1,000,000 MPI ranks in xSim on a native system with 1,000 cores, the main motivation of this evaluation is to obtain the native baseline execution overhead and trade-offs involved when using xSim in the highly oversubscribed fashion.



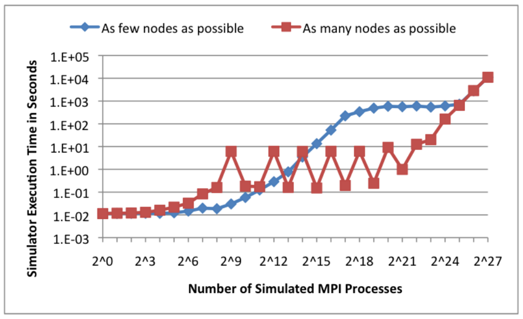
(a) Using as few nodes as possible



(b) Using as many nodes as possible



(c) Using 24 cores per node



(d) Using 6 cores per node

Figure 2: xSim performance with up to 134,217,728 simulated MPI ranks.

Figure 2 shows the wall clock execution time of xSim using 24, 12, and 6 cores/node on the 40-node cluster with up to 134,217,728 (2^{27}) simulated

MPI ranks. At 2^{27} ranks, xSim uses 2 TB of RAM as user-space stack (16 kB/rank). Most of xSim's time is spent with setting up each rank, e.g., the initial stack frame, and performing context switches between ranks, e.g., switching the stack frame. In Figure 2(a), xSim's native MPI ranks are placed to use as few nodes as possible to reduce the communication overhead of the PDES for virtual time synchronization. Memory access contention overheads from the context switches are visible at large scale. In Figure 2(b), xSim's native MPI ranks are placed on as many nodes as possible to reduce memory access contention. The higher native communication overhead is visible at small scale, as well as, the impact of imbalanced load when using 6 cores/node. The Figures 2(c) and 2(d) show side-by-side comparisons of using as few or as many nodes as possible for 24 and 6 cores/node. The trade-off between concentrating communication vs. distributing memory contention is clearly demonstrated.

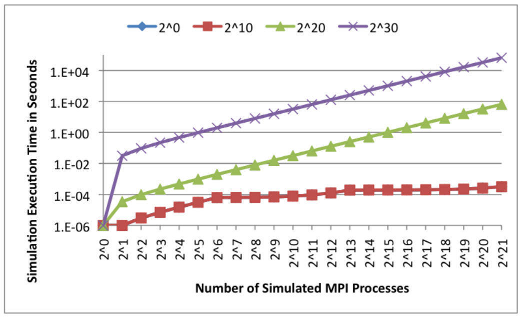
These experiments not only demonstrate the relatively low baseline execution overhead and the on-node/off-node contention trade-offs involved when using xSim, but also hint at a balance between communication-intensive simulations that should be executed on as few native compute nodes as possible and computation- or memory-intensive simulations that should exploit as many compute nodes as possible.

4.2. Simulated MPI Collective Communication Performance

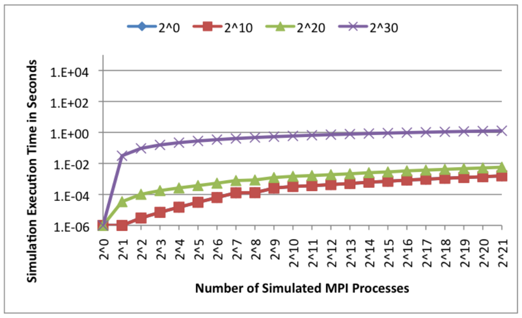
The second series of experiments investigate the performance of different collective communication algorithms for *MPI_Reduce()* and *MPI_Barrier()* in a simulated future HPC system with 2,097,152 (2^{21}) nodes organized in a 128×128×128 3-D torus with 1 μs link latency and 32 GB/s link bandwidth. The number of cores/node is not considered as an MPI+X, e.g., MPI+OpenMP, programming model is assumed. Therefore, each simulated MPI rank is placed on one simulated compute node.

The targeted collective communication algorithms are based on the virtual MPI implementation xSim offers using a point-to-point communication overlay network in a linear or tree-based fashion. For the tree-based algorithms, the trees are balanced using a simple top-down divide-and-conquer approach to create a temporary overlay topology. In every hierarchy level, the hierarchy's root is chosen and the remaining nodes are divided by the tree degree to build the next hierarchy levels. The collective communication topology type and the tree degree can be configured as part of xSim's virtual MPI configuration on startup. For *MPI_Reduce()*, the execution time of the

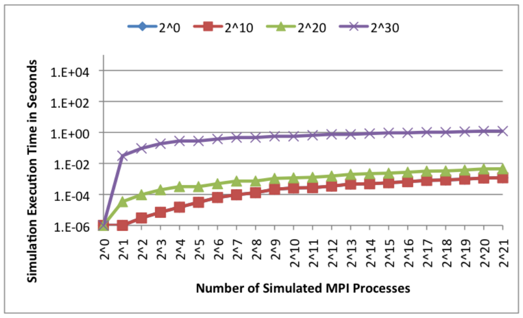
MPI_Reduce_local() operation on the data itself is explicitly not considered for this experiment for clarity. xSim uses a shortest path routing within the 3-D torus network and simulates sender- and receiver-side network interface contention, but not contention in the nodes along the routing path [13]. The simulated eager communication threshold is set to 256 kB, i.e., MPI payloads above 256 kB utilize the simulated rendezvous protocol.



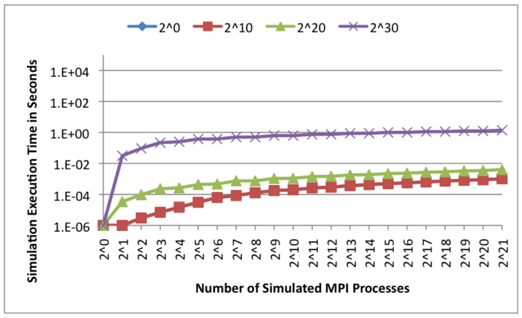
(a) Linear



(b) Tree degree of 2



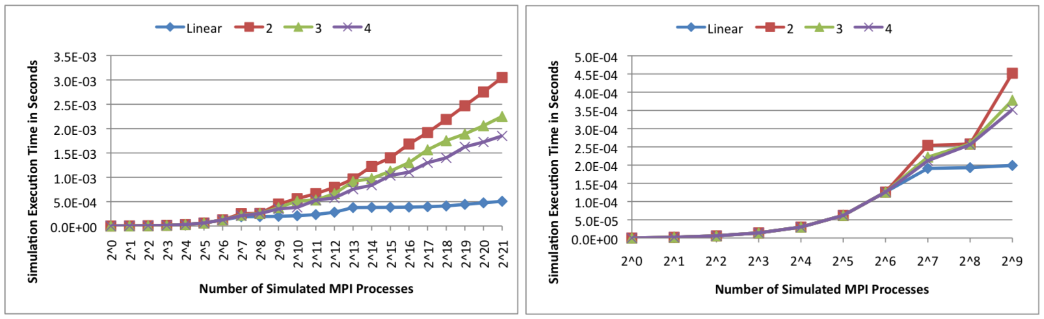
(c) Tree degree of 3



(d) Tree degree of 4

Figure 3: *MPI_Reduce()* in a 128×128×128 torus with 2^0-2^{30} bytes of payload using a linear and different tree-based collective communication algorithms. The results for payloads of 2^0 and 2^{10} bytes are too similar to differentiate.

Figure 3 describes the simulated execution time of an *MPI_Reduce()* in the 3-D torus with up to 2^{21} simulated MPI ranks and 2^0-2^{30} bytes of payload in a linear collective communication algorithm (Figure 3(a)) and in tree-based collective communication algorithms with tree degrees of 2, 3, and 4 (Figures 3(b)-3(d)). The results clearly show the performance difference between the payloads and between the linear and tree-based collectives. They also clearly demonstrate an almost negligible difference between the various tree-based collectives at high payloads (2^{20} and 2^{30} bytes), due to the increased receiver contention with higher tree degrees and the missing architecture-awareness of the overlay tree-based collective in a 3-D torus. They further illustrate the superiority of the linear collective at low payloads (2^0 and 2^{10} bytes) due to the higher 0-byte message latency of the architecture-unaware tree-based collectives.



(a) 2^0-2^{21} simulated MPI ranks (b) 2^0-2^9 simulated MPI ranks

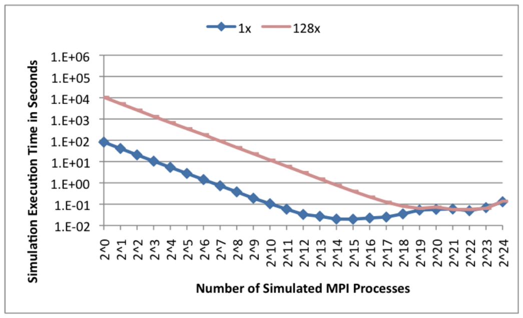
Figure 4: *MPI_Barrier()* in a 128×128×128 torus using a linear and different tree-based collective communication algorithms.

Figure 4(a) shows the simulated execution time of an *MPI_Barrier()* in the 3-D torus with up to 2^{21} simulated MPI ranks and with linear and tree-based collective communication. As the *MPI_Barrier()* is implemented in this case with a 0-byte *MPI_Gather()* (equivalent to a 0-byte *MPI_Reduce()*) followed by a 0-byte *MPI_Bcast()*, the results demonstrate the superiority of the linear collective due to the higher 0-byte message latency of the architecture-unaware tree-based collectives. Figure 4(b) further demonstrates the effect of rank placement within the 3-D torus. Scaling up from 2^7 to 2^8 simulated MPI ranks causes a change from a 1-D to a 2-D topology and a corresponding change in scaling behavior due to shortest path routing.

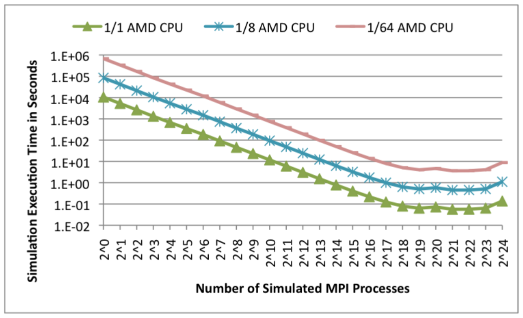
4.3. Simulated Application Performance

The last series of experiments employ xSim to identify the impact of different architectural choices on the performance of a basic Monte Carlo (MC) application. The structure of this MC application consists of an embarrassingly parallel computation phase and a final communication-intensive aggregation of the result. This makes it easy to study how architectural trade-offs affect an application's computation and communication phases differently.

For simplicity, the MC implementation for estimating π from an earlier series of experiments [14] is used. The MPI application follows the dart-board approach for computing π by creating random coordinates in a square and tests if the points are within the centered circle fitted in the square. The ratio of total points vs. the points inside the circle eventually converges to the ratio of the square area vs. the circle area, from which π can be easily computed. The computation load is set via the total number of points to create. The communication load scales with the application, as a double precision value is accumulated across all MPI ranks. The implementation uses an architecture-aware linear *MPI_Reduce()* for the accumulation based on neighbor communication in a chain.



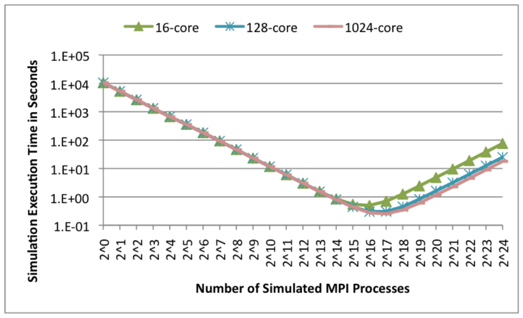
(a) Different iterations with 1/1 AMD CPU



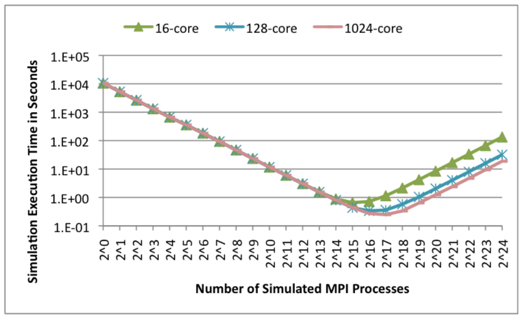
(b) Different CPU performances with 128×

Figure 5: Monte Carlo solver in a perfect network with different iterations and CPUs.

Figure 5 illustrates the simulated execution time of the π MC solver with strong scaling to 16,777,216 (2^{24}) simulated MPI ranks in a perfect network (0 latency and ∞ bandwidth). Figure 5(a) displays the execution results with different computation load (1× vs. 128×) using a 1:1 scaling model for the simulated processor speed relative to one 1.7 GHz AMD Opteron 6164 HE core of the cluster the experiment is performed on. The scaling limitation due to Amdahl's law is clearly visible as the amount of work per simulated MPI process approaches its minimum in the form of command line parsing, basic setup, and a relatively small amount of randomly generated points. The rest of the experiments are executed at problem size 128× for comparison. Figure 5(b) illustrates the effect of different simulated processor speeds on the scaling properties. An increase in execution time of a factor of 8, or 64 respectively, can be observed with a processor speed slowed by a factor of 8, or 64 respectively.



(a) In a 8×8×... torus

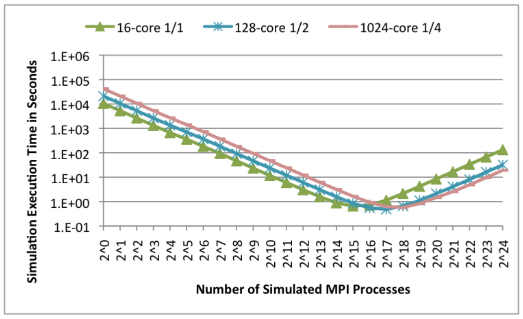


(b) In a 16×16×... torus

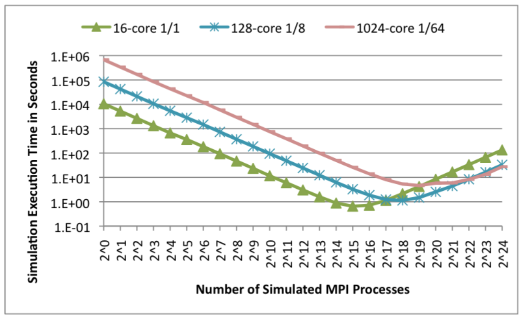
Figure 6: Monte Carlo solver in different torus networks with different cores per node.

Figure 6 shows the simulated execution time of the π MC solver with strong scaling to 2^{24} ranks in two different 3-D torus networks with different core counts per node. The simulated architecture was derived from an existing extreme-scale supercomputer and scaled up. The network interconnect is an 8×8×... or 16×16×... 3-D torus, with an open-ended 3rd dimension to accommodate different total node counts. Each compute node within the torus has 16, 128, or 1024 cores per node connected with each other over shared memory communication simulated with an on-node star network. The simulated system has an on-node latency of 0.32 μs, an off-node latency of 7 μs, an on-node bandwidth of 9600 Mbps, and an off-node bandwidth of 8800 Mbps. In comparison to Figure 5(a) (128×) the impact of the communication-intensive aggregation phase being slowed down by the network is clearly visible in Figure 6. The performance gain of using more cores per node is also clearly visible. Only slight differences exist between Figures 6(a) (using a 8×8×... 3-D torus) and 6(b) (using a 16×16×... 3-D torus). The rest of the experiments are executed using the 16×16×... 3-D torus for comparison.

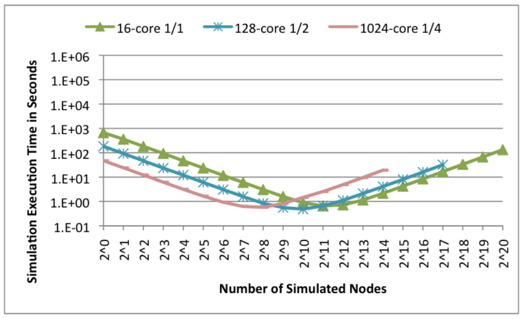
The path to exascale is limited by the power consumption envelope. 1 MW of power consumption throughout an entire year roughly costs \$1M U.S Dollars (or \$1M Euros). As it does not make much sense to spend more money on powering a system for its 5-year life time than on purchasing it, a certain power consumption envelope exists (~20 MW) in which an exascale system needs to operate. Due to this limitation, HPC architects are opting to increase energy efficiency by deploying less powerful cores in larger quantities, a well-known trade-off between operating frequency and power consumption.



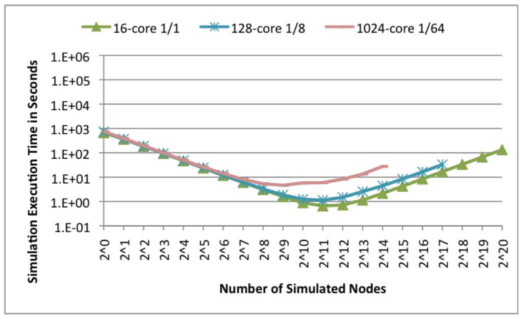
(a) 8× more cores that are 2× less powerful



(b) 8× more cores that are 8× less powerful



(c) 8× more cores that are 2× less powerful



(d) 8× more cores that are 8× less powerful

Figure 7: Monte Carlo solver in a 16×16×... torus network with different CPUs.

Figure 7 illustrates the simulated execution time of the π MC solver with strong scaling to 2^{24} simulated MPI ranks in the 16×16×... 3-D torus with 16, 128, or 1024 cores/node when decreasing the computational capability of the cores while increasing the number of cores/node. Figures 7(a) and 7(b) clearly show the impact of deploying n-times more cores per node that are m-times less powerful. As a result, the application's overall performance decreases. Figures 7(c) and 7(d) display the same results with simulated compute node counts on the x-axis, allowing a more comparative study considering node, instead of core, performance/count.

5. Conclusions

With this paper, a recent effort was summarized that focused on the design and implementation of a novel HPC hardware/software co-design toolkit. The presented Extreme-scale Simulator (xSim) focuses on a unique concept that utilizes a light-weight PDES to run an HPC application in a controlled environment with millions of concurrent execution threads. Using architectural models and virtual timing, application performance can be observed in a simulated extreme-scale HPC system.

The capabilities and usefulness of the xSim toolkit have been demonstrated by showing that it scales to 2^{27} simulated MPI ranks on a 960-core Linux-based cluster (a world record in extreme-scale simulation), different MPI collective communication algorithms could be evaluated on a simulated future-generation system with up to 2^{21} simulated MPI ranks, and the scaling properties of a basic Monte Carlo application with different architectural parameters could be investigated on a simulated HPC system with up to 2^{24} simulated MPI ranks. The results clearly demonstrate that architecture-awareness is key to enable extreme-scale computing. Whether it is collective communication algorithms within the MPI implementation, application scalability limits imposed by Amdahl's law, or energy efficiency trade-offs in hardware, HPC hardware/software co-design is essential to guarantee efficient utilization of future-generation HPC systems.

6. Future Work

While the presented work is quite novel and useful, a few limitations remain that will be addressed in the near future. First and foremost, full network contention modeling is planned as an optional part of xSim. The current sender-/receiver-based network contention modeling does not include all network nodes in the routing path and will be extended to support this as an optional feature (limiting scalability though). Another deficiency is the quite simplistic processor model. Current plans include the extension of the processor model to utilize processor counters to more accurately model future processors. Another issue that will be addressed in the near future is validation of simulation accuracy by comparing native with simulated application executions on different HPC platforms. Initial studies were performed in [12] using the NAS Parallel Benchmark suite on a real and a simulated Linux cluster. More extensive validation using large-scale HPC systems with more complex network architectures have not been done until now due to general stability issues, that have been solved now, and the missing full network contention modeling.

Future work will also target a simulation-based HPC hardware/software performance, resilience, and power consumption co-design toolkit. It focuses on enhancing xSim with (1) fault injection mechanisms, (2) fault propagation, isolation, and detection models, (3) fault avoidance, masking, and recovery simulation, and (4) power consumption models.

Acknowledgements

This research is sponsored by the Office of Advanced Scientific Computing Research, U.S. Department of Energy (DOE). This manuscript has been authored by UT-Battelle, LLC, under Contract No. DE-AC05-00OR22725 with the DOE. 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, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.

References

[1] C. Engelmann, A. Geist, Super-scalable algorithms for computing on 100,000 processors, in: Lecture Notes in Computer Science: Proceedings of the 5th International Conference on Computational Science (ICCS) 2005, Part I, volume 3514, Springer Verlag, Berlin, Germany, Atlanta, GA, USA, 2005, pp. 313–320.

[2] G. Bosilca, Z. Chen, J. Dongarra, J. Langou, Recovery patterns for iterative methods in a parallel unstable environment, SIAM Journal on Scientific Computing (SISC) 30 (2007) 102–116.

[3] Z. Chen, J. Dongarra, Algorithm-based checkpoint-free fault tolerance for parallel matrix computations on volatile resources, in: Proceedings of the 20st IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2006, IEEE Computer Society, Rhodes Island, Greece, 2006, p. 10.

[4] H. Ltaief, E. Gabriel, M. Garbey, Fault tolerant algorithms for heat transfer problems, Journal of Parallel and Distributed Computing (JPDC) 68 (2008) 663–677.

[5] G. Zheng, G. Kakulapati, L. V. Kale, BigSim: A parallel simulator for performance prediction of extremely large parallel machines, in: Proceedings of the 18^{th} IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2004, IEEE Computer Society, Santa Fe, New Mexico, 2004.

[6] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, G. Zheng, "Programming petascale applications with Charm++ and AMPI", in: Petascale Computing: Algorithms and Applications, CRC Press, 2007, pp. 421–441.

[7] K. S. Perumalla, $\mu\pi$: A highly scalable and transparent system for simulating MPI programs, in: Proceedings of the $3rd^{th}$ International ICST Conference on Simulation Tools and Techniques (SIMUTools) 2010, ACM Press, New York, NY, USA, Malaga, Spain, 2010.

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

[9] S. Girona, J. Labarta, R. M. Badia, Validation of Dimemas communication model for MPI collective operations, in: Lecture Notes in Computer Science: Proceedings of the 7^{th} European PVM/MPI Users' Group Meeting (EuroPVM/MPI) 2000, volume 1908, Springer Verlag, Berlin, Germany, Balatonfüred, Hungary, 2000, pp. 39–46.

[10] V. Pillet, J. Labarta, T. Cortes, S. Girona, PARAVER: A Tool to Visualize and Analyze Parallel Code, in: Proceedings of WoTUG-18: Transputer and occam Developments, pp. 17–31.

[11] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, W. E. Nagel, The Vampir performance analysis toolset, in: M. Resch, R. Keller, V. Himmler, B. Krammer, A. Schulz (Eds.), Tools for High Performance Computing, Springer Berlin Heidelberg, 2008, pp. 139–155.

[12] S. Böhm, C. Engelmann, xSim: The extreme-scale simulator, in: Proceedings of the International Conference on High Performance Computing and Simulation (HPCS) 2011, IEEE Computer Society, Los Alamitos, CA, USA, Istanbul, Turkey, 2011, pp. 280–286.

[13] I. S. Jones, C. Engelmann, Simulation of large-scale HPC architectures, in: Proceedings of the 40th International Conference on Parallel Processing (ICPP) 2011: 2nd International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI), IEEE Computer Society, Los Alamitos, CA, USA, Taipei, Taiwan, 2011, pp. 447–456.

[14] C. Engelmann, F. Lauer, Facilitating co-design for extreme-scale systems through lightweight simulation, in: Proceedings of the 12th IEEE International Conference on Cluster Computing (Cluster) 2010: 1st Workshop on Application/Architecture Co-design for Extreme-scale Computing (AACEC), IEEE Computer Society, Hersonissos, Crete, Greece, 2010, pp. 1–8.

[15] C. Engelmann, Investigating operating system noise in extreme-scale high-performance computing systems using simulation, in: Proceedings of the 11th IASTED International Conference on Parallel and Distributed Computing and Networks (PDCN) 2013, ACTA Press, Calgary, AB, Canada, Innsbruck, Austria, 2013.