CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2015; **00**:1–21
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe

A New Deadlock Resolution Protocol and Message Matching Algorithm for the Extreme-scale Simulator*

Christian Engelmann and Thomas Naughton

Computer Science and Mathematics Division, Oak Ridge National Laboratory, Oak Ridge, TN 37831-6016, USA

SUMMARY

Investigating the performance of parallel applications at scale on future high-performance computing (HPC) architectures and the performance impact of different HPC architecture choices is an important component of HPC hardware/software co-design. The Extreme-scale Simulator (xSim) is a simulation toolkit for investigating the performance of parallel applications at scale. xSim scales to millions of simulated Message Passing Interface (MPI) processes. The xSim toolkit strives to limit simulation overheads in order to maintain performance and productivity criteria. This paper documents two improvements to xSim: (1) a new deadlock resolution protocol to reduce the parallel discrete event simulation overhead, and (2) a new simulated MPI message matching algorithm to reduce the oversubscription management cost. These enhancements resulted in significant performance improvements. The simulation overhead for running the NAS Parallel Benchmark suite dropped from 1,020% to 238% for the conjugate gradient (CG) benchmark and 102% to 0% for the embarrassingly parallel (EP) benchmark. Additionally, the improvements were beneficial for reducing overheads in the highly accurate simulation mode of xSim, which is useful for resilience investigation studies for tracking intentional MPI process failures. In the highly accurate mode, the simulation overhead was reduced from 37,511% to 13,808% for CG and from 3,332% to 204% for EP. Copyright © 2015 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Performance Prediction; Message Passing Interface; Parallel Discrete Event Simulation; High-performance Computing;

1. INTRODUCTION

There are several difficult tasks to reaching exascale high-performance computing (HPC), which include challenges related to power, performance, resilience, productivity, programmability, data movement, and data management. In HPC hardware/software co-design, the impact of different architectural choices is studied to gain insights into the effects on parallel applications at scale. Since the future architectures are not yet available, an alternative is to employ simulation tools for estimating the performance implications of different architectural choices on parallel applications. As highly accurate simulations are extremely slow and less scalable, different approaches exist to reduce simulation accuracy in order to gain simulation scalability and performance.

Efficient simulation of large-scale distributed systems atop distributed systems is a complex endeavor. Examples range from simulation of very large-scale peer-to-peer systems deployed over

the Internet [1] to multi-scale simulation of multi-disciplinary models in scientific computing [2] and distributed simulation with real-time requirements for massively multiplayer on-line games [3]. For each specific simulation solution, the properties of the simulated distributed system and the properties of the distributed system the simulation is executed on define the design envelope. Optimizations within this design envelope improve the efficiency of the simulation, such as simulation accuracy and execution overhead.

The Extreme-scale Simulator (xSim) [4, 5, 6, 7, 8, 9] is a simulation toolkit for investigating the performance of parallel applications at scale. xSim strives to limit simulation overheads in order to maintain performance and productivity requirements. As xSim employs a conservative parallel discrete event simulation (PDES) algorithm, it has to deal with distributed deadlocks. xSim also supports highly-oversubscribed operation. The simulator is capable of running millions of simulated Message Passing Interface (MPI) processes on just a thousand physical MPI processes. However, the overhead for managing the simulated MPI processes and their associated context switches can be significant.

This paper is an extension of the original conference paper [4]. It documents two improvements to xSim. The first improvement is a new deadlock resolution protocol to reduce the overhead of the conservative PDES algorithm. The second improvement is a new simulated MPI message matching algorithm to reduce the overhead of the simulated MPI process management when running with oversubscription. This paper is structured as follows. Section 2 describes the existing work in xSim. Section 3 illustrates related work. Section 4 documents the improvements in the deadlock resolution protocol. Section 5 documents the improvements in the simulated MPI message matching algorithm. Section 6 presents experimental results and Section 7 concludes this paper.

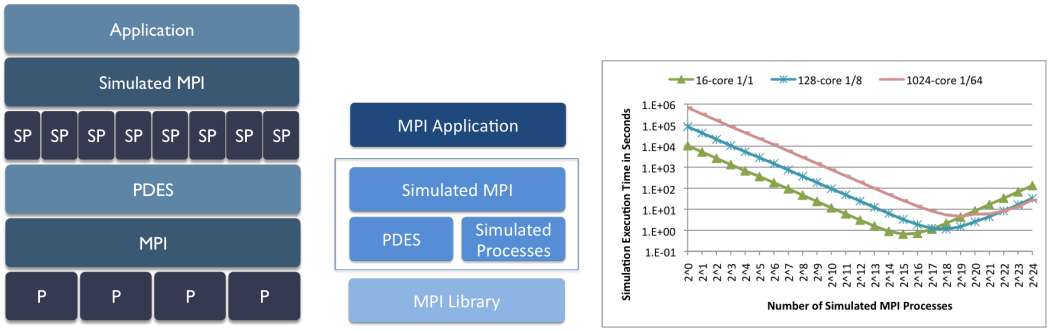
2. BACKGROUND

xSim is a performance investigation toolkit that permits running a HPC application in a controlled environment. An MPI application can be executed on a simulated HPC system with millions of concurrently executing threads. During the simulation, xSim gathers performance information for the application running on the simulated extreme-scale system, which is displayed to the end-user at the end of the run. Using a lightweight, conservative PDES algorithm, xSim executes a MPI application on a much smaller system in a highly oversubscribed fashion (Figure 1(a)). The execution uses a virtual wall clock time such that performance data can be extracted based on a processor and a network model. The design of xSim is in keeping with other MPI application performance tools, and operates as an interposition library that sits between the MPI application and the MPI library (Figure 1(b)). It uses the MPI performance tool interface (PMPI) to virtualize MPI calls. xSim currently does not support threaded execution models, such as OpenMP [10], task-based execution models, such as High Performance ParalleX (HPX) [11, 12], and accelerators, such as general-purpose graphics processing unit (GPGPUs). Recently, the usefulness and capabilities of xSim were demonstrated [7], and include:

- scaling to 134,217,728 (2^{27}) simulated MPI processes (each with its own process context) on a 960-core Linux cluster (a world record in extreme-scale simulation),
- evaluating different MPI collective communication algorithms on a simulated future-generation system with 2,097,152 (2^{21}) MPI processes using the same 960-core cluster, and
- investigating a Monte Carlo solver using different architectural parameters (Figure 1(c)) with 16,777,216 (2^{24}) simulated MPI processes using the same 960-core cluster.

xSim has a fault injection feature [6] that supports simulated MPI process failures, which can be activated based on triggered conditions or scheduled to occur at specific times during the simulation. Fault notifications are propagated (according to the simulated architecture) to each simulated MPI process, which in turn reacts to the fault. xSim also has full support for error handling within the simulated MPI, including default MPI error handlers, user-defined MPI error handlers, and `MPI_Abort()`. Under the current MPI standard, a simulated abort is triggered when a MPI process

DOI: 10.1002/cpe



(a) Architecture [9] (b) Design [9] (c) Scaling a Monte Carlo solver [7]

Figure 1. xSim: The Extreme-scale Simulator

fails if the MPI error handler is set to `MPI_ERRORS_ARE_FATAL`. A simulated abort results in the termination of the simulation with performance results and information about the source and time of the abort. xSim includes support for continuous virtual timing after a simulated abort and a subsequent restart. This enables users to investigate experiments involving application-level checkpoint/restart triggered by intentional (injected) simulated MPI process failures.

xSim further implements fault-tolerant MPI extensions [5] as proposed by the MPI Fault Tolerance Working Group. The supported functionality is based on the user-level failure mitigation (ULFM) [13] proposal, and provides the base capabilities at the simulated MPI layer to support algorithm-based fault tolerance (ABFT). The ULFM specification enables the application, via ULFM extensions, to handle MPI process failures if the MPI error handler is set to either `MPI_ERRORS_RETURN` or is user-defined. xSim permits investigating performance under failure and failure handling of ABFT solutions. It is the very first MPI application performance tool that supports ULFM and ABFT.

xSim also supports investigations using MPI communication trace logs. In lieu of replaying a trace within the simulator, the framework instead generates a benchmark from the provided trace data and runs this benchmark within the simulation. This approach offers several benefits, which include eliminating the data intensive trace replay procedure and enabling experiments at different scales via simulation. The framework uses ScalaTrace II [14, 15] to create the trace files, ScalaBenchGen II [16] to produce the benchmark, and finally xSim is used to execute the generated benchmark within a simulation. The experimental results have shown the generated benchmarks from the NAS Parallel Benchmark suite [17] perform with a mean error of 5.5% and a maximum error of 13.3%.

3. RELATED WORK

3.1. Parallel Discrete Event Simulation

Parallel discrete event simulation [18] is the execution of a single discrete event simulation with simultaneous threads. Each thread simulates one logical process or, when oversubscription is involved, multiple logical processes. Logical processes are simulated concurrently at different points in simulated time. The simulation needs to guarantee causality between logical processes to maintain correctness of the simulated system. There are generally two approaches for guaranteeing causality. Conservative approaches strictly avoid causality errors by determining when it is safe to process an event. Optimistic approaches detect causality errors and recover from them by using rollback

Conservative approaches have the risk of a distributed deadlock. In this case, one logical process is holding up the progress of all the others while it is itself waiting on the progress of another. The distributed deadlock is resolved by the logical process that is holding up the others to make progress to a simulated time where it is safe for the others to process an event. Frequent distributed deadlocks

DOI: 10.1002/cpe

may result in a lock-step simulation that is not taking advantage of concurrency. Distributed deadlocks can be reduced or entirely avoided by transmitting *null messages with time stamps* [19] between logical processes to communicate information about progress. Using this information, each logical process independently obtains a view of the global simulated time. This view is used to determine if processing an event would violate causality. Optimized PDES solutions also utilize the context of an event in the simulated system to determine if it is safe to process.

Optimistic approaches require the periodic checkpointing of the state of each logical process to enable rollback. Upon encountering a causality violation, the logical processes involved in the causality violation are rolled back to correct the error. In the worst case, all logical processes are rolled back. The *Time Warp* [18] protocol uses event time stamps to identify causality violations. An event can not be processed at a logical process if it has a time stamp that is smaller than the time stamp of the previously processed event. This protocol also utilizes the concept of a global virtual time (GVT), which is the smallest timestamp among all unprocessed events. It is used to identify the point in simulated time that is free of causality violations. No event (and no checkpoint) with a time stamp smaller than the GVT will ever be part of a rollback.

In xSim, PDES execution threads are *physical MPI processes* and logical processes are *simulated MPI processes*. Most events are *simulated MPI messages*. xSim employs a conservative approach using null messages with time stamps that are called *local virtual time (LVT) update messages*. Its novel PDES algorithm also utilizes the GVT concept of the Time Warp protocol to partially avoid distributed deadlocks and to resolve those that can not be avoided. xSim relies on the context of a simulated MPI message and the knowledge of the LVTs of all other simulated MPI processes (and therefore the GVT) to determine if it is safe to process. Communicating too many LVT update messages causes high simulation overhead for processing. Communicating not enough LVT update messages causes frequent distributed deadlocks. This paper presents a new distributed deadlock resolution protocol that improves upon the previous protocol by reducing the amount of LVT update messages and improving the resolution of distributed deadlocks itself.

3.2. Simulators

The predecessor to xSim was called the Java Cellular Architecture Simulator (JCAS) [20], which was developed in 2001 to investigate the scalability and fault-tolerance of algorithms for HPC systems with about 100,000 processor cores. The JCAS prototype is capable of running up to 500,000 simulated processes on a Linux cluster with 5 processor cores (with 1 core reserved for visualization) solving standard mathematical problems. JCAS is able to execute algorithms at scale but lacks some important features, to include: time-accurate simulation, high performance, running the simulator using MPI, and a fully functional virtualized MPI. JCAS does not implement a PDES and relies on Java threads. Oversubscription is achieved by encapsulating simulated MPI process contexts in objects.

The BigSim [21] project studied programming issues in large-scale HPC systems. The BigSim Emulator was developed for testing and debugging applications at large scale and relied upon Charm++/AMPI [22]. BigSim supports up to 100,000 simulated MPI processes spread over 2,000 processor cores. Similar to JCAS, the BigSim Emulator does not offer time-accurate simulation and does not implement a PDES. Oversubscription is achieved using the object-based parallel programming of Charm++, which encapsulates MPI process contexts in objects. Oversubscription using Charm++ has been very successful in improving the performance of scientific applications, such as NAMD (a molecular dynamics code) [23]. The BigSim Emulator offers more functionality than JCAS but scales less. In contrast, the BigSim Simulator was developed to identify performance bottlenecks and uses a trace-driven PDES that models architectural parameters of HPC systems. To support time-accurate simulations, BigSim implements a variable-resolution processor model and a detailed network model. While it uses a conservative PDES to maintain accuracy, it only supports post-mortem trace replay and does not run applications. Since a trace replay is completely deterministic, the PDES does not need to deal with distributed deadlocks.

$\mu\pi$ [24] is a PDES-based system that was developed in conjunction with research for predicting the performance of parallel programs. The $\mu\pi$ simulator is very similar to xSim, but it targets

DOI: 10.1002/cpe

different grafting methods for interfacing applications with the simulation, e.g., at the source code, library (actually implemented), and virtual machine level. It supports both optimistic and conservative execution based on the μsik PDES engine. A prototype was tested on the Jaguar Cray XT5 supercomputer using 216,000 compute cores, running over 221 million simulated MPI processes, each with a separate thread context, and all processes synchronized by simulated time. μsik uses one operating system (OS) thread per simulated MPI process. This severely limits scalability due to the thread management and context switch overhead within the OS. In the test using the Jaguar supercomputer, each of the 216,000 cores was running 1,024 simulated MPI processes. 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, however, superior. μsik supports both conservative and optimistic algorithms, e.g., null messages and Time Warp.

The Structural Simulation Toolkit (SST) [25] offers simulation of novel architectures, including processor, memory, and network. SST is a modular PDES framework that is implemented using MPI, and 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 produce models for larger-scale simulations. SST does not support oversubscription, implements a conservative algorithm, and relies on deterministic execution without distributed deadlocks. SST/macro is a complementary evaluation toolkit that processes output from the DUMPI library (i.e., MPI tracing library) to conduct performance investigations. SST and SST/macro are similar to the BigSim Emulator/Simulator pairing where experiments can explore the synergy between small-scale cycle-accurate and large-scale communication-accurate simulations. SST is a mature tool, but is very complex to use. SST/macro is still under development. Since a trace replay is completely deterministic, SST/macro does not need to deal with distributed deadlocks.

SimGrid [26] is a toolkit that provides the necessary capabilities for simulating distributed applications in heterogeneous distributed environments. It can simulate many different distributed systems, such as a Grid computing infrastructure, peer-to-peer computing environments, HPC systems running MPI, and Clouds. SimGrid is similar to xSim as it offers an MPI interface and permits running applications (online simulation). SimGrid uses UNIX98 contexts, an OS support for cooperative threading, to support oversubscription. SimGrid does not use a PDES, instead, it uses a central simulation engine. While the use of a central simulation engine removes the distributed deadlock problem entirely, it introduces a global synchronization and scalability problem.

GridSim [27] is a model-based simulation tool for investigating the characteristics of computing resources with different network configurations. Instead of the online approach of xSim that executes an application, GridSim executes models of applications, computing resources and interconnect networks. GridSim utilizes Distributed SimJava, which is a centralized simulation engine for a discrete event simulation similar to the SimGrid approach.

OMNeT++ [28] is an extensible, modular, component-based C++ PDES library and framework, primarily for building network simulators. It has a generic architecture and has been used in various problem domains, such as modeling of wired and wireless communication networks, queueing networks, and messaging-based hardware and software systems. OMNeT++ does not offer oversubscription on its own. It uses a conservative PDES with null messages [19], similar to xSim, to avoid distributed deadlocks.

There are other trace-driven PDES solutions for investigating application performance. For example, DIMEMAS [29] processes trace logs from MPIDTrace and generates trace files suitable for the two performance tools, PARAVER [30] and Vampir [31]. These PDES solutions depend on completely deterministic simulation without distributed deadlocks.

There are a variety of detailed network architecture simulators, such as ns3 [32] and NetSim [33], that are able to provide network performance metrics at various abstraction levels, such as network, sub-network, and packet traces. These detailed network architecture simulators offer high-accuracy/low-scalability results that are not compatible with the low-accuracy/high-scalability approach taken in this paper.

DOI: 10.1002/cpe

4. DEADLOCK RESOLUTION

The first xSim improvement that we discuss concerns the deadlock resolution protocol. The simulated MPI processes in xSim each maintain their own simulated process clock. Computation performed by a simulated MPI process advances the clock according to a processor model. The scaling processor model measures the time to complete an operation on the physical (real) processor, which is then scaled to the estimated performance of the simulated processor. The simulated MPI processes are executed by xSim in a piece-wise manner. The simulated clock for the xSim MPI process starts upon entry to the application's `main()` function. The simulated process clock stops when entering a call to the xSim MPI library, and prior to returning control to the application the simulated process clock is resumed. The simulated process clock halts when exiting the application's `main()` function. When simulated MPI processes reside on the same physical MPI process, context switches may occur due to oversubscription while the simulated process clock is stopped or halted. The simulator employs cooperative user-space multi-threading, and context switches take place as needed to execute multiple simulated MPI processes and to receive the simulated MPI messages in the appropriate context.

Communication performed by a simulated MPI process advances the simulated process clock based on a network model. The latency and bandwidth between the source and destination is calculated using the network model, which dictates the waiting time on the completion of a communication in the simulator. This ensures the wait and busy times are properly accounted for when sending and receiving MPI messages. The simulated wait and busy times are used to advance the simulated process clock. The network model maintains causality in simulated time as long as the communication operations are deterministic. For example, the simulated wait time for receiving messages ensures that a message can not be received at a point in simulated time prior to the message being transmitted in simulated time.

However, the MPI standard does includes a number of operations that can violate causality in the simulation, such as when the physical (actual) message ordering differs from the simulated MPI message order. This occurs because the simulation is in simulated time and may not match the actual (non-simulated) wall-clock time. Also, asynchronous operations within the simulator can effect causality as each simulated MPI process is executed piece-wise and concurrently. The MPI messages between two simulated MPI processes will always be in the same physical (actual) receive ordering for all executions of the simulation. However, the physical (actual) receive order of MPI messages from different simulated MPI process, which are addressed to the same simulated MPI process, may differ.

For example, in the case of `MPI_ANY_SOURCE` receives typically need to match the MPI message that is calling a receive first. The simulator achieves this behavior by evaluating all potential MPI messages that could match this condition. This ensures proper matching for the simulated MPI message that is received first in simulated time irrespective of the physical (actual) receive order. To maintain causality and reproducibility for `MPI_ANY_SOURCE` receives, the simulator needs to wait until all potential MPI messages have been received to make this determination. As such, xSim must wait until either: (a) all other simulated MPI processes have passed the simulated timeline for sending an MPI message that could be matched, or (b) all other simulated MPI processes that have not passed the simulated timeline are deadlocked on receiving a simulated MPI message themselves. Often this results results in a distributed deadlock of the simulation. Other simulated MPI calls that share this behavior include: `MPI_Iprobe()`, `MPI_Testall()`, `MPI_Testany()`, `MPI_Testsome()`, `MPI_Test()`, `MPI_Waitany()`, and `MPI_Waitsome()`.

In addition to these simulated MPI calls, the MPI extensions for fault-tolerance that are available in xSim can also cause errors in causality. Also, the asynchronous operation of the simulator may impact the accuracy of simulated MPI process failures. To ensure accurate simulation, a simulated MPI process that is sending needs to know that the receiving simulated MPI prcess has not yet failed (at the simulated time of the completion of the send operation). This is similar to the case with `MPI_ANY_SOURCE` receives, the sending simulated MPI process needs to wait for the receiving simulated MPI process to pass the point in the simulated time line or is deadlocked on receiving

DOI: 10.1002/cpe

a simulated MPI message. Again, similar to `MPI_ANY_SOURCE` receives, this often results in a distributed deadlock of the simulator. Since this accurate simulation feature is only needed for fault injection experiments, where simulated MPI process failures occur, it is optional and can be enabled via a command line option.

By design, the simulator enters a distributed deadlock in all these cases due to the conservative PDES implementation, which is necessary to ensure proper ordering to maintain causality and reproducibility. The deadlock resolution protocol in xSim provides a mechanism for detecting these deadlocks and subsequently resolving the deadlocks in a deterministic manner that maintains causality and reproducibility.

4.1. Previous Protocol

xSim includes a deadlock resolution protocol that is based on approaches taken by other parallel discrete event simulators in the literature. The approach relies on the management of simulated local and global time and on walking the clock of a deadlocked simulated MPI process. Since xSim supports oversubscription, at any point during the simulation the local virtual time (LVT) is the lowest simulated MPI process clock of those simulated MPI processes located the same physical MPI process. The LVT is always up to date because it is managed locally at each physical MPI process. In contrast, at any point during the simulation the global virtual time (GVT) is the lowest simulated MPI process clock of all simulated MPI processes. Since it is managed in a distributed fashion, the GVT generally lags behind and is asynchronously updated at each physical MPI process using LVT update messages. The previous implementation of the deadlock resolution protocol maintained the LVT and based on the following actions:

- When a simulated MPI message is transmitted: piggyback the LVT.
- When a simulated MPI process clock is advanced: set the LVT to the lowest simulated MPI process clock on the physical MPI process.
- When the LVT changes: broadcast it to all other physical MPI processes using an explicit LVT update message. Omit LVT update messages to physical MPI processes that have received the LVT via piggyback before.
- When a LVT is received (update or piggyback): (i) store it with the associated rank of the sending physical MPI process and (ii) set the GVT to the lowest stored LVT.

The physical MPI process is deadlocked when the incoming simulated MPI message queue is empty or contains messages that can not be matched, i.e., waiting on other physical MPI processes. To detect and resolve distributed deadlocks, the previous implementation of the deadlock resolution protocol performs the following action:

- When a physical MPI process is deadlocked, it is the lowest ranked of the deadlocked, and its value for LVT and GVT are equal: advance the LVT to the next clock value of a locally residing simulated MPI process or of a stored LVT from a remote physical MPI process, whichever is lower.

As the LVT is advanced, update messages are transmitted to the other physical MPI processes and the GVT is advanced. The deadlock is resolved by advancing the GVT, which identifies the simulated timeline that all simulated MPI processes have passed.

This protocol basically follows the concept of letting the simulation execute in a conservative fashion. The GVT is used to establish the timeline used to maintain causality and repeatability. If the simulation is in a distributed deadlock, the GVT is safely advanced from the lowest simulated MPI process clock forward until the deadlock is resolved. The weaknesses of this deadlock protocol are:

- LVT update messages are communicated after most (if not all) simulated MPI process clock updates due to the simulator's fair scheduling. The fair scheduling ensures that the simulated MPI process with the lowest clock is next in line for receiving its message and thus advancing its clock.
- The piggyback mechanism is circumvented by the simulated MPI process clock update and any resulting LVT update message that occurs upon entering a simulated MPI call.

DOI: 10.1002/cpe

- The walking of the GVT by the lowest physical MPI process often results in another physical MPI process becoming responsible for resolving the deadlock. This results in a subsequent distributed deadlock and another GVT walk, and causes message storms during deadlock resolution. This is especially true when the simulated MPI process with the highest simulated MPI process clock is causing a deadlock in all the other simulated MPI processes.

The weaknesses of the previous deadlock protocol all impact performance and have been observed while performing experiments. The primary motivation for changing to a different (better performing) protocol was the high cost (performance) impact when accurately simulating MPI process failures. In this case, each simulated send operation becomes a potential global synchronization point due a distributed deadlock. The results in Section 6 demonstrate this issue. For example, the simulation overhead for the conjugate gradient (CG) benchmark of the NAS Parallel Benchmark suite [17] increased from 1,020% (cg.C.128 in Figure 2(b)) to 37,511% (cg.C.128 in Figure 3(b)) by just switching on the capability for accurate MPI process failure simulation on the command line.

4.2. New Protocol

To eliminate the aforementioned weaknesses, the deadlock resolution protocol has been enhanced to reduce the amount of communication introduced from LVT update messages and to speed up the deadlock resolution. The new protocol still adheres to the conservative simulation design of xSim that uses the GVT to maintain causality and repeatability. However, LVT updates are performed less frequently and GVT walks are more careful to consider other deadlocked physical MPI processes. The new implementation of the deadlock resolution protocol performs the following actions to maintain LVT and GVT:

- When a simulated MPI message is transmitted: piggyback the LVT.
- When a simulated MPI process clock is advanced: set the LVT to the lowest simulated MPI process clock on the physical MPI process.
- When the incoming simulated MPI message queue is empty or contains messages that can not be matched and there is a potential for a deadlock: broadcast the LVT to all other physical MPI processes using an explicit LVT update message. Omit LVT update messages to physical MPI processes that have received the LVT via piggyback previously. Also omit LVT update messages if the LVT is equal to, or exceeds, the known LVT of the destination physical MPI process (essentially implementing a vector clock).
- When a LVT is received: (i) store it with the associated rank of the sending physical MPI process and (ii) set the GVT to the *lowest* stored LVT of the non-deadlocked physical MPI processes or, if none exist, to the *highest* stored LVT of the deadlocked physical MPI processes.

The three main improvements are that (1) LVT update messages are only communicated in the case of a potential deadlock, (2) LVT update messages are only communicated when they have the potential to change the lowest physical MPI process with the GVT, and (3) the GVT is entirely based on the LVT of the lowest non-deadlocked, or highest deadlocked, physical MPI process. In order to resolve deadlocks, the new implementation of the protocol performs the following actions:

- When a simulated MPI process enters a phase that may lead to a deadlock, e.g., when posting an `MPI_ANY_SOURCE` receive: increase a counter that indicates the deadlock potential.
- When a simulated MPI process exits such a phase, such as when completing an `MPI_ANY_SOURCE` receive: decrease the deadlock potential counter.
- When a physical MPI process is deadlocked, it is the lowest ranked of the deadlocked, and its value for LVT and GVT are equal: advance the LVT to the next clock value of a locally residing simulated MPI process or of a stored LVT from a remote non-deadlocked physical MPI process, whichever is lower.

As the LVT is advanced, LVT update messages are not immediately transmitted to the other physical MPI processes. Instead, either a simulated MPI process residing locally can proceed (i.e., resolving the deadlock), or another deadlock is encountered and resolved at the same physical MPI

DOI: 10.1002/cpe

process. Eventually, a LVT update message is transmitted and advances the GVT to resolve the deadlock.

The two main improvements are that: (1) the LVT is advanced to a non-deadlocked simulated timeline and (2) the resolution of a deadlock does not immediately cause LVT update messages to be communicated. The overall design of the new deadlock resolution protocol is much more conservative as it does not constantly update the GVT and employs an iterative approach to resolve deadlocks. Also, the new protocol is more efficient in resolving deadlocks as it skips known already deadlocks.

Another enhancement enables the behavior of the deadlock resolution protocol to be changed from the described dynamic (on-demand) mode to a static (always on) mode. In the dynamic (on-demand) case, LVT update messages are only transmitted when there is an actual deadlock potential. In contrast, LVT update messages are communicated independent from the deadlock potential in the static (always on) case. This enables faster deadlock resolution as LVT information is communicated more frequently and is more up-to-date. However, it also incurs an overhead for sending additional LVT update messages. This feature is optional and can be controlled via a command line argument.

5. MESSAGE MATCHING

The second improvement to xSim discussed in this paper concerns the matching of simulated MPI messages. As mentioned previously, xSim simulates MPI processes piece-wise, in parallel, and potentially in an oversubscribed fashion. The entire process context (stack, heap, and meta data) is replicated prior to entering the `main()` function of the application. This is used to create the simulated MPI process context when running with oversubscription. On each physical MPI process, xSim runs a simulation `pthread` thread with a user-space stack that is split between multiple simulated MPI processes using a custom user-space implementation of `fork()`. A context switch changes the stack frame for this simulation thread within this bigger user-space stack and copies out/in the static heap (`.text`, `.bss`, `.data`, and similar segments). xSim's user-space thread management capability is supported under Linux and Mac OS X.

In xSim, context switches take place when a simulated MPI process relinquishes control back to the simulator. Similar to other PDES solutions, xSim maintains an incoming message queue that contains the simulated MPI messages. The xSim incoming message queue is filled by a separate communication (`pthread`) thread that is responsible for receiving MPI messages from other physical MPI processes. The simulation thread matches a simulated MPI message in the incoming message queue with the corresponding receiving simulated MPI process. The simulator returns the message to the appropriate simulated MPI process in the proper context of the receiving call. The matching itself involves a context switch to the receiving simulated MPI process to identify the matching parameters. Since context switches can be expensive, mainly due to copying in and out of the static heap and due to cache pollution, efficient matching that limits or eliminates unnecessary context switches reduces simulation overhead.

5.1. Previous Algorithm

The original message matching algorithm was derived from the approaches taken by other PDES implementations in the literature and was relatively straight forward. Algorithm 1 shows the simple linear search of the incoming message queue where each message is checked if it matches and every check potentially involves a prior context switch. While it is simple, it has the major drawback a linear search is performed over the message queue. This likely incurs context switches to the same simulated MPI processes at different times during the search. It also incurs unnecessary context switches for messages that can not be matched. However, the advantage of this approach is that messages are matched in message queue order, which is based on the simulated timestamps of messages. Since the message with the lowest simulated timestamp is matched, fair scheduling between simulated MPI processes is guaranteed.

DOI: 10.1002/cpe

```
1  /* Search for matching message. */
2  for (index = 0; index < queue.count; index++) {
3    message = queue.array[index];
4    switch_context(message.dest);
5    /* Check matching criteria. */
6    if match(message) {
7      return message;
8    }
9  }
10 /* No message match, or queue is empty. */
11 deadlock_resolution();
```

Algorithm 1: Previous message matching algorithm

5.2. New Algorithm

The new algorithm (Algorithm 2) reduces the number of context switches to a minimum by batching up the message matching for each destination (simulated MPI process). It first traverses the incoming message queue to find a matching simulated MPI message in the context of the current simulated MPI process. If no match is found, the algorithm traverses the queue again to find the first simulated MPI message to a different destination. It then traverses the rest of the queue find a matching simulated MPI message in the context of this destination. If no match is found, the algorithm traverses the rest of the queue repeatedly until all simulated MPI messages have been checked. The algorithm performs the context switch only after a message has been matched to completely eliminate unnecessary context switches. The messages are matched by chronological order for each destination separately and the number of context switches is reduced to a minimum. However, fair sharing between simulated MPI processes is not entirely guaranteed because newer messages to one destination may be matched before older messages to another destination. This could potentially result in imbalances in the simulation, where one area is much further ahead. Such imbalances can slow down the simulation if one or more physical MPI processes need to wait for other physical MPI processes to catch up in simulating their simulated MPI processes.

6. RESULTS

Both improvements have been fully implemented and extensively tested. The following describes the performed experiments and the obtained results.

6.1. System Setup

Four sets of experiments were performed on two different Linux cluster computers, evaluating the improvements under different conditions.

The first Linux cluster computer has 128 processor cores in total in 16 compute nodes. Each compute node has two quad-core 2.4 GHz AMD Opteron 2378 processors and 8 GB RAM. The system has a bonded dual non-blocking 1 Gbps Ethernet interconnect, which offers 1.7 Gbps measured point-to-point TCP bandwidth. The software environment is Ubuntu 12.04 LTS, Open MPI 1.6.4, and GCC 4.6

The first set of experiments investigates the simulation overhead incurred by the previous version and the new version of xSim over the native performance for executing the NAS Parallel Benchmark (NPB) suite [17]. The experimental results demonstrate the improved performance of xSim when running real HPC applications that perform computation and bursty communication. The second set of experiments investigates the simulation overhead for executing a benchmark using oversubscription that was generated from the Sweep3D benchmark using ScalaTrace II [14, 15] and ScalaBenchGen II [16]. The results demonstrate the improved performance of xSim when running trace replays that perform bursty communication only.

DOI: 10.1002/cpe

```
1  /* Search for matching message (no context switch). */
2  for (index = 0; index < queue.count; index++) {
3    message = queue.array[index];
4    /* If current process is destination for message. */
5    if (current_process == message.dest) {
6      /* Check matching criteria. */
7      if match(message) {
8        return message;
9      } else {
10       message.checked = 1;
11     }
12   } else {
13     message.checked = 0;
14   }
15 }
16 /* Search for matching message (with context switch). */
17 for (index = 0; index < queue.count; index++) {
18   message = queue.array[index];
19   /* Skip previously checked messages. */
20   if (message.checked == 1) {
21     continue;
22   }
23   current_process = message.dest;
24   for (index2 = index      ;
25        index2 < queue.count;
26        index2++            ) {
27     message = queue.array[index2];
28     /* If current process is destination for message. */
29     if (current_process == message.dest) {
30       /* Check matching criteria. */
31       if match(message) {
32         switch_context(message.dest);
33         return message;
34       } else {
35         message.checked = 1;
36       }
37     }
38   }
39 }
40 /* No message match, or queue is empty. */
41 deadlock_resolution();
```

Algorithm 2: New message matching algorithm

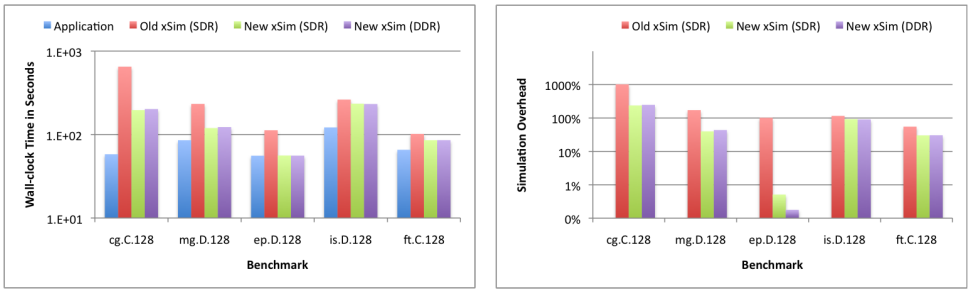
The second Linux cluster computer has 936 processor cores in total in 39 compute nodes. Each compute node has two 12-core 1.7 GHz AMD Opteron 6164 HE processors and 64 GB RAM. The system has also a bonded dual non-blocking 1 Gbps Ethernet interconnect with 1.7 Gbps measured point-to-point TCP bandwidth. The software environment is also Ubuntu 12.04 LTS, Open MPI 1.6.4, and GCC 4.6.

The third set of experiments investigates the simulation overhead for a basic Monte Carlo solver while the fourth set takes a look at the simulation overhead for a matrix-matrix multiplication. The results demonstrate the performance improvements of xSim when scaling up two different MPI applications using oversubscription and strong scaling. Only one simulator instance is put on each of the 39 compute nodes, using the significant amount of memory per compute node (64 GB RAM) for highly oversubscribed simulation.

6.2. NAS Parallel Benchmark Suite

The NPB suite [17] is a collection of benchmark programs developed by the NASA Advanced Supercomputing (NAS) Division to aid in evaluating the performance of supercomputers. They are mini applications that are derived from a number of HPC application codes, including computational

DOI: 10.1002/cpe



(a) Wall-clock execution time in seconds (b) Execution overhead relative to application

Figure 2. NPB results

fluid dynamics, unstructured adaptive mesh, parallel I/O, multi-zone, and computational grid applications. Their input/output problem sizes of are predefined and categorized in different classes, e.g., class A, B, C, and D. The experiments in this paper utilize the following benchmarks and problem sizes (classes):

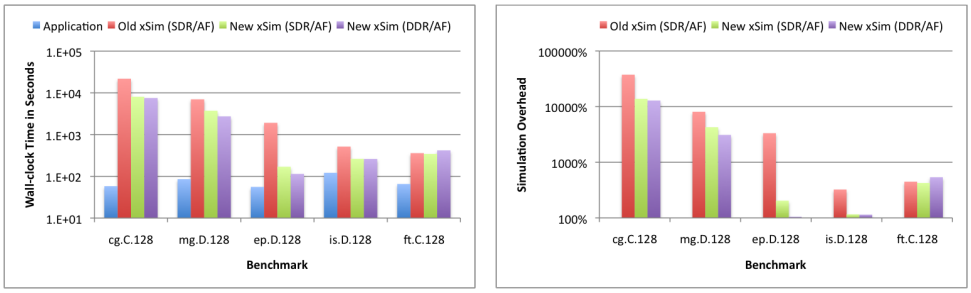
- CG, a conjugate gradient solver with irregular memory and communication patterns (class C)
- MG, a multi-grid solver on a sequence of meshes with long- and short-distance communication patterns and memory intensive operations (class D)
- EP, an embarrassingly parallel application (class D)
- IS, an integer sort with random memory access patterns (class D), and
- FT, a discrete 3D fast Fourier transform with all-to-all communication patterns (class C)

The experiments using the NPB suite included executing the selected benchmarks without xSim for obtaining a baseline, with the old version of xSim and with the new xSim. As the old version of xSim only supports static deadlock resolution (SDR), it was executed only with SDR. The new xSim supports SDR and dynamic deadlock resolution (DDR) and was executed with both, separately. Each selected benchmark was executed on 128 physical or simulated MPI processes without oversubscription. Three runs were executed and averaged for each data point in the experiment.

The wall-clock execution time of the simulation was greatly improved (Figure 2). The new deadlock resolution protocol is solely responsible for the improvements as no oversubscription is used. xSim's execution time overhead was reduced from 102% to 0% for the EP benchmark simulation and from 1,020% to 238% for the CG benchmark simulation. The EP benchmark has a long computational phase that is finished by a short communication phase. The entire simulation overhead of the deadlock resolution protocol is incurred during the single short communication phase, which is a single synchronization point. This is why the overhead could be reduced to 0%. In contrast, the CG benchmark performs irregular communication throughout the entire benchmark run. The simulation overhead of the deadlock resolution protocol with GC is incurred during the entire entire benchmark run with many synchronization points.

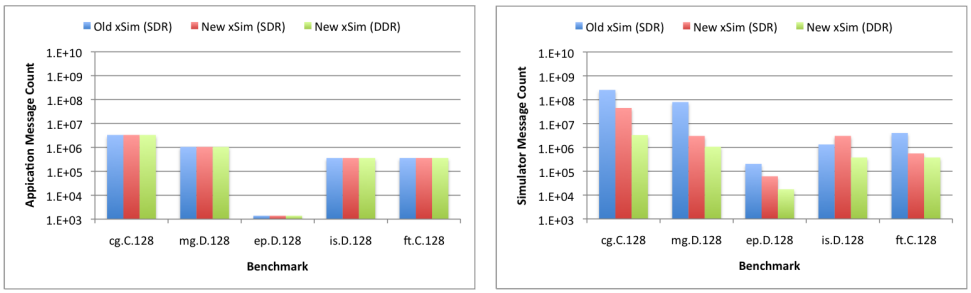
The results the NPB suite using the command line option for accurate process failure simulation are shown in Figure 3. When this option is enabled, each simulated MPI message send operation verifies at the outset that the receiving (destination) simulated MPI process has not been declared dead via an injected simulated MPI process failure. A failure notification is received in the case the receiving simulated MPI process has failed. To permit this capability, a simulated MPI message send operation synchronizes the timeline of the sending simulated MPI process with the receiving MPI process. The timeline synchronization utilizes the simulated MPI process clocks of the sender and receiver if they both reside at the same physical MPI process. If not, the sender simulated MPI process clock and the LVT of the remote physical MPI process the receiver resides at are used. Synchronization involving a remote physical MPI process typically results in a deadlock and subsequent deadlock resolution. In a communication-intensive simulation, the number of deadlocks

DOI: 10.1002/cpe



(a) Wall-clock execution time in seconds (b) Execution overhead relative to application

Figure 3. NPB results with accurate process failure simulation



(a) Application only (b) Simulator only

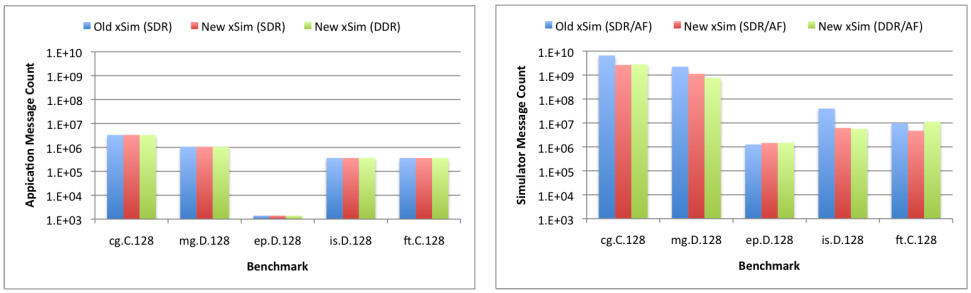
Figure 4. NPB results – Simulator MPI message counts

and subsequent deadlock resolutions can be significant as synchronization is performed with every simulated MPI message send operation.

A significant reduction in simulation overhead with accurate process failure simulation was achieved (Figure 3). The new deadlock resolution protocol is solely responsible for these improvements. xSim's the execution time overhead was reduced from 3,332% to 204% for the EP benchmark simulation and from 37,511% to 13,808% for the CG benchmark simulation, both with SDR. xSim's execution time overhead was further reduced using DDR, such as to 105% for the EP benchmark simulation and to 12,871% for the CG benchmark simulation. The different communication characteristics of EP and CG also influence the simulation overhead with accurate process failure simulation. The send/receive timeline synchronization when using accurate process failure simulation is only at a very concentrated spot with EP. Thus, the new deadlock resolution protocol has a lot of opportunity to employ its optimizations. In contrast, CG offers less opportunities due to its irregular communication pattern. The differences between SDR and DDR show that an adaptive deadlock resolution protocol that can dial-in its LVT update message rate based on the deadlock potential offers room for improvements. Only a slight improvement, however, could be observed with SDR from 448% to 425% for the FT benchmark. A slight degradation was observed with DDR to 539%. This is likely due to the all-to-all communication pattern of this particular benchmark. It has many global synchronization points and therefore offers little room for improvement. The degradation from SDR to DDR is a perfect example for the adaptive protocol creating an additional overhead by constantly enabling and disabling itself.

The differences in the total MPI message count between the application running in xSim and the simulator itself are shown in Figures 4 and 5. Figures 4(a) and 5(a) demonstrate that the simulated MPI message count of the application is constant for each application across the different simulator configurations. The total MPI message count was improved by an order of magnitude or more for the CG, MG, EP, and FT benchmarks in simulations without accurate process failures (Figure 4(b)). For example, CG's total MPI message count could be reduced from 256,668,463 to 3,280,382 with

DOI: 10.1002/cpe



(a) Application only (b) Simulator only

Figure 5. NPB results with accurate process failures – Simulator MPI message counts

DDR. The IS benchmark simulations showed improvements only when using DDR. In this case, the total MPI message count was reduced from 1,326,008 to 377,318. The shown improvements are entirely due to a reduction in LVT update messages. The new deadlock resolution protocol generally transmits less LVT update messages. It also advances the LVT to a non-deadlocked simulated timeline, skipping known deadlocks and their resolutions in-between. This further reduces the number of LVT update messages.

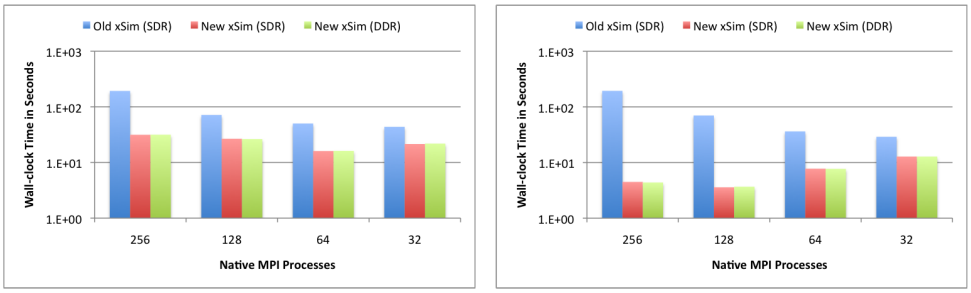
The total MPI message count was improved by up to an order of magnitude for the CG, MG and IS benchmarks in simulations with accurate process failures (Figure 5(b)). For example, IS's total MPI message count could be reduced from 39,937,855 to 5,642,266 with DDR. Although simulations with accurate process failures require send/receive timeline synchronization, the new deadlock resolution protocol was able to reduce the number of LVT update messages. For EP, the simulator's message count was not improved. This is due to the fact that EP has a single short communication phase, which becomes a single short global synchronization phase with accurate process failure simulation. This leaves the new deadlock resolution protocol little to no room for improvements. FT showed improvements only when using SDR, where the simulator's total MPI message count was reduced from 9,673,438 to 4,695,771. The all-to-all communication pattern of this particular benchmark in combination with the send/receive timeline synchronization also offers to little to no room for improvements.

6.3. Sweep3D - ScalaBenchGen

The Sweep3D trace replay benchmark is a skeleton application that simply replays MPI communication and emulates computation by putting an MPI process to sleep for a certain amount of time between MPI communication events. It is based on an earlier experiment that ran the original Sweep3D application on an InfiniBand Linux cluster with 256 MPI processes, extracted the MPI communication traces, and constructed the skeleton application from these traces. The experiment showed a high simulation overhead when running the Sweep3D trace replay benchmark within xSim, which motivated the work presented in this paper. The improvements to xSim were put to a test using the Sweep3D trace replay benchmark by executing it atop xSim in application mode as a normal MPI application and in model mode as a MPI application model. In contrast to the application mode, xSim does not transmit simulated MPI message payloads in model mode. As the simulated MPI message envelope is transmitted only, the Sweep3D trace replay simulation can be executed faster at the same accuracy. The Sweep3D trace replay benchmark was obtained from an original execution with 256 MPI processes. As its execution atop xSim requires 256 simulated MPI processes and the system xSim is executed on has only 128 cores, the experiments utilized 256, 128, 64 and 32 physical MPI processes with different native to simulated MPI process ratios (1:1, 1:2, 1:4, and 1:8). The 128-core Linux cluster was used in an oversubscribed fashion for 256 physical MPI processes. Three runs were executed and averaged for each data point.

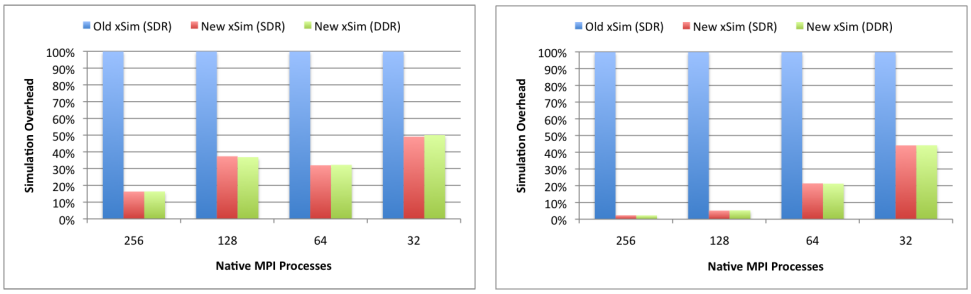
Significant improvements in simulation execution time were observed in application mode and stellar improvements were achieved in model mode (Figures 6 and 7). When executing the Sweep3D

DOI: 10.1002/cpe



(a) Trace replay in application mode (b) Trace replay in model mode

Figure 6. Sweep3D results – Wall-clock execution time in seconds



(a) Trace replay in application mode (b) Trace replay in model mode

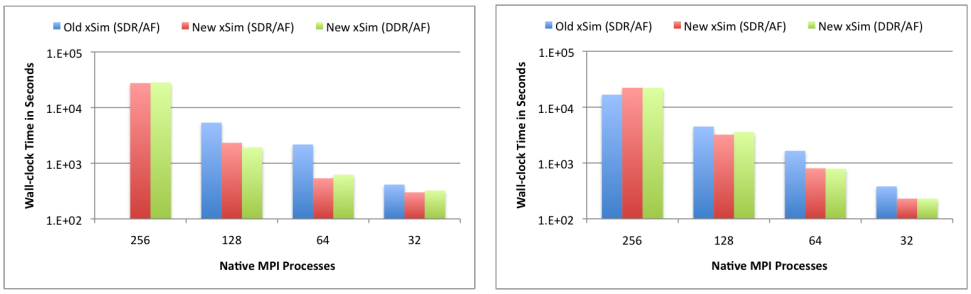
Figure 7. Sweep3D results – Execution overhead relative to application

trace replay benchmark using 128 physical MPI processes, running 2 simulated MPI processes per physical MPI process, the new xSim's simulation execution time is 37% of the old xSim's time in application mode and 5% in model mode. The improvements are due to the new deadlock resolution protocol and the new message matching algorithm. The Sweep3D trace replay benchmark is communication intensive, especially in model mode. The new deadlock resolution protocol is able to reduce the pressure on the communication infrastructure. The new message matching algorithm is able to optimize context switch management when oversubscription is employed.

With accurate process failure simulation, Figures 8 and 9, similar performance improvements were observed. The new xSim's simulation execution time is 36% of the old xSim's time in application mode and 79% in model mode. Note that the simulation overhead of the old xSim prohibited executions on 256 physical MPI processes as its simulation execution time exceeded 4 days. This was primarily due to a significant amount of deadlock resolutions. The new deadlock resolution protocol alleviates this problem by skipping known deadlocks and their resolutions. The old xSim also had a programing error, a missing synchronization point, which was discovered after implementing the new deadlock resolution protocol. This error simulated potential points of MPI process failures inaccurately and was fixed in the new xSim. On 256 physical MPI processes in model mode, the old xSim is actually faster due to the missing synchronization point than it normally should be. Only minor differences between SDR and DDR can be observed for the new xSim on less than 256 physical MPI processes. The dynamic protocol (DDR) does not offer advantages over the static protocol (SDR) in this case due to the 1:1 mapping of physical and simulated MPI processes and the communication-intensive trace replay.

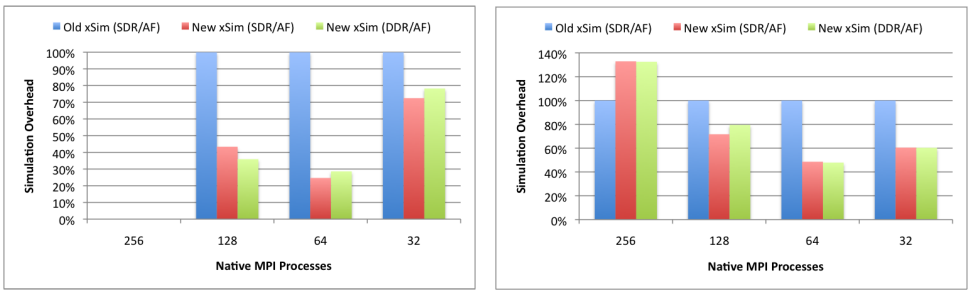
Figure 10 shows the reduction in context switches when employing oversubscription using 128, 64 and 32 physical MPI processes. A reduction by an order of magnitude or more was observed, such as from 160 million to 1.6 million on 128 physical MPI processes with SDR in application mode. It was also reduced from 160 million to 1.1 million in model mode. The new message matching algorithm enables the batching of message receives by the same simulated MPI process without

DOI: 10.1002/cpe



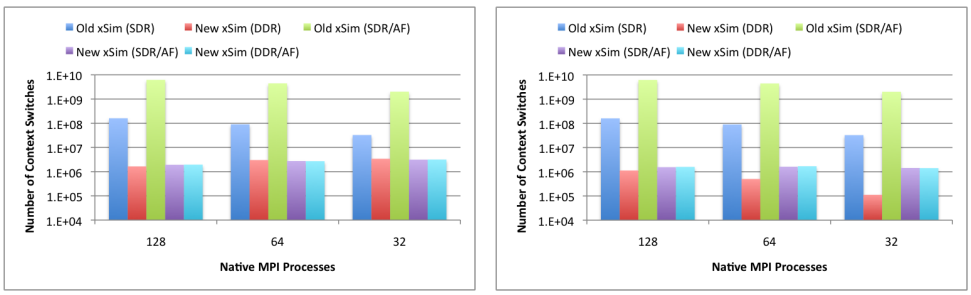
(a) Trace replay in application mode (b) Trace replay in model mode

Figure 8. Sweep3D results with accurate process failures – Wall-clock execution time in seconds



(a) Trace replay in application mode (b) Trace replay in model mode

Figure 9. Sweep3D results with accurate process failures – Execution overhead relative to application



(a) Trace replay in application mode (b) Trace replay in model mode

Figure 10. Sweep3D results – Simulator context switches

switching to another simulated MPI process. This is particular useful for communication-intensive applications, such as this benchmark. There was no significant difference between SDR and DDR with accurate process failure simulation for the new xSim. This indicates that the different variants of the deadlock resolution protocol do not influence the number of context switches in this case.

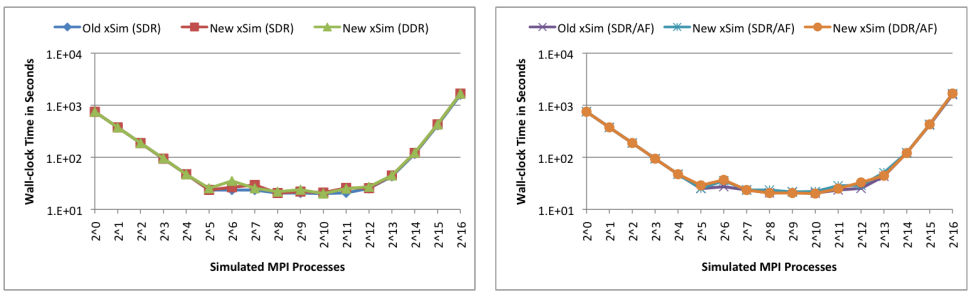
6.4. Basic Monte Carlo Solver

The basic Monte Carlo solver is an embarrassingly parallel application that has one communication-intensive data aggregation of the result. It is an implementation for estimating π from earlier experiments [7, 9] using the basic dart-board approach. The solver creates a massive number of random two-dimensional coordinates within a square. It tests if the coordinates fall within the centered circle fitted in the square or not. With enough random coordinates, the ratio of total coordinates vs. those falling in the circle eventually converges to the ratio of the square area vs.

DOI: 10.1002/cpe

the circle area. π can be computed from this ratio. To accumulate the ratios from each participating MPI process, a linear `MPI_Reduce()` is used that employs a chain of MPI processes. The computational load of this solver scales with the number of generated random coordinates, while the communication load scales with the number of MPI processes. The number of iterations is set to 10 billion and the number of simulated MPI processes is scaled from 1 (2^0) to 65,536 (2^{16}). The number of physical MPI processes was capped at 39 with each physical MPI process located on a different compute node (on the second evaluation system). This set of experiments studies oversubscription at scale (from 2^6 to 2^{16} simulated MPI processes, with 1,681 simulated MPI processes per physical MPI process at 2^{16}). Three runs were executed and averaged for each data point in the experiment.

Figure 11 shows that the basic Monte Carlo MPI application executes with roughly the same simulator wall-clock time without and with accurate process failure simulation. Figures 12(a) and Figure 13(a) demonstrate that there is no reduction of simulator MPI messages or context switches for runs without accurate process failure simulation. However, Figure 12(b) reveals a significant reduction of simulator MPI messages (by an order of magnitude for 2^8-2^{14} simulated MPI processes) with accurate process failure simulation when oversubscribing. Figure 13(b) illustrates a significant reduction of context switches (by two orders of magnitudes) with accurate process failure simulation when oversubscribing. While the improved deadlock resolution protocol and simulated MPI message matching algorithm demonstrate their capabilities, they have little impact on the simulation overhead for the basic Monte Carlo solver as its runtime is dominated by the solver's computation phase. The solver has little simulated MPI process context, so context switches are cheap. It also has only one, self-synchronizing, collective communication operation at the end.

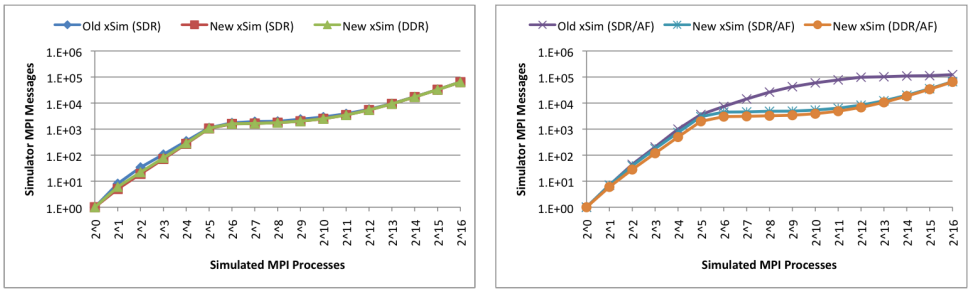


(a) Without accurate process failure simulation (b) With accurate process failure simulation

Figure 11. Basic Monte Carlo Solver – Wall-clock execution time in seconds

6.5. *Matrix-Matrix Multiplication*

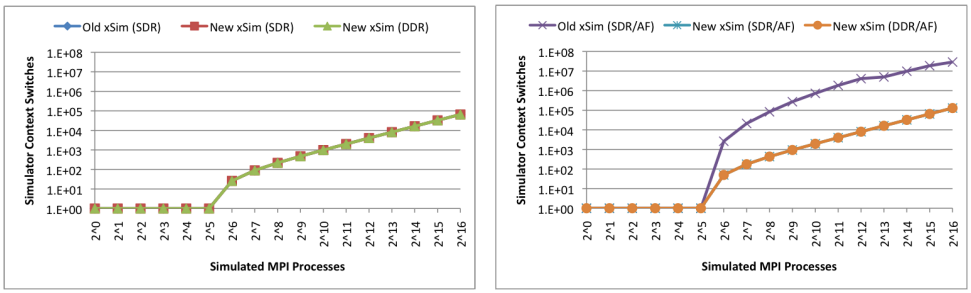
The matrix-matrix multiplication MPI application performs a number of communication steps. It scatters the rows of matrix A, broadcasts the matrix B, and gathers the rows of the result matrix



(a) Without accurate process failure simulation (b) With accurate process failure simulation

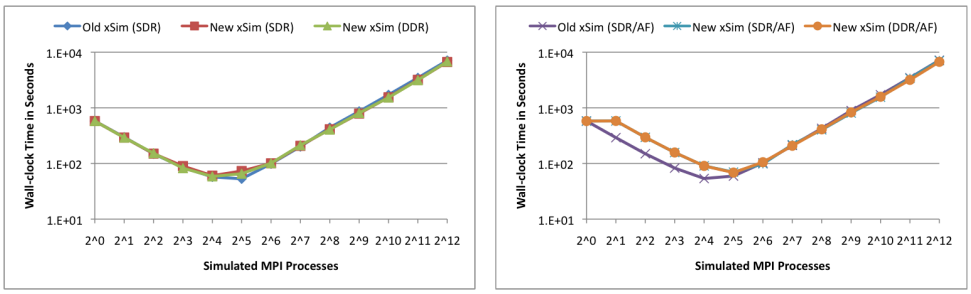
Figure 12. Basic Monte Carlo Solver – Simulator MPI message counts

DOI: 10.1002/cpe



(a) Without accurate process failure simulation (b) With accurate process failure simulation

Figure 13. Basic Monte Carlo Solver – Simulator context switches



(a) Without accurate process failure simulation (b) With accurate process failure simulation

Figure 14. Matrix-Matrix Multiplication – Wall-clock execution time in seconds

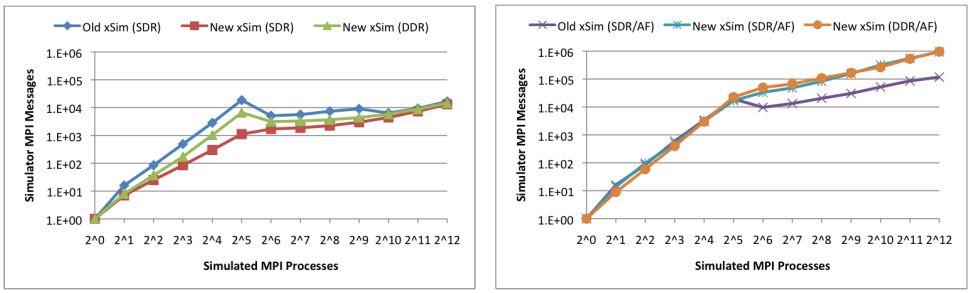
C. Each MPI process performs its local matrix-matrix multiplication and contributes to the final result. The matrix sizes were set to 4,096×4,096 and the number of simulated MPI processes was scaled from 1 (2^0) to 4,096 (2^{12}). The number of physical MPI processes was capped at 39. Each physical MPI process was located on a different compute node (on the second evaluation system) to study oversubscription at scale (from 2^6 to 2^{12} simulated MPI processes, with 106 simulated MPI processes per physical MPI process at 2^{12}). Note that at 2^{12} simulated MPI processes, each simulated MPI process performs computation on a single row of matrix A.

Figure 14(a) shows that the matrix-matrix multiplication also executes with roughly the same simulator wall-clock time without accurate process failure simulation. Figure 14(b) demonstrates a *higher* wall-clock time with accurate process failure simulation and without oversubscription (and roughly the same with oversubscription) using the new simulation performance features. However, Figure 15(a) also reveals a significant reduction of simulator MPI messages (up to a magnitude when running without oversubscription) without accurate process failure simulation. In contrast, Figure 15(b) illustrates a significant *increase* of simulator MPI messages (by an order of magnitude) with accurate process failure simulation when oversubscribing. Figure 16(a) shows no differences in context switches in runs without accurate process failure simulation. However, Figure 16(b) demonstrates a significant reduction of context switches (by two orders of magnitudes) with accurate process failure simulation when oversubscribing.

The missing performance improvements in this set of experiments are due to the fact that the matrix-matrix multiplication MPI application has little process context that is involved in a context switch. The matrices are dynamically allocated and not swapped in/out during a context switch. This application also uses a highly regular, bulk synchronous algorithm that is dominated by computation at all scales with the given problem size. The synchronization for accurate MPI process failure simulation itself has little impact as the application is already highly synchronized.

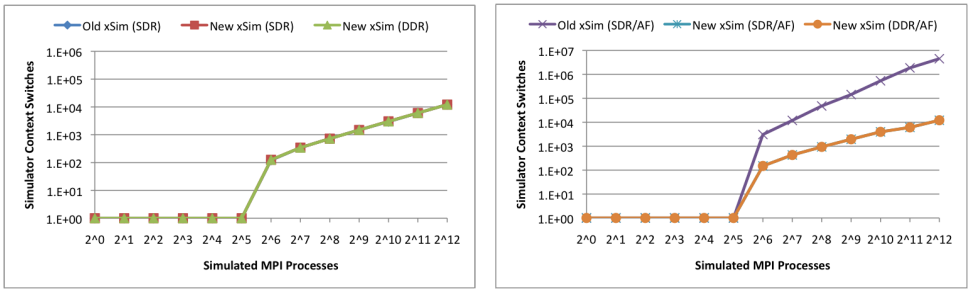
The performance degradation with accurate process failure simulation when not oversubscribing is due to the fact that the new deadlock resolution protocol is lazy, i.e., it does not send LVT update

DOI: 10.1002/cpe



(a) Without accurate process failure simulation (b) With accurate process failure simulation

Figure 15. Matrix-Matrix Multiplication – Simulator MPI message counts



(a) Without accurate process failure simulation (b) With accurate process failure simulation

Figure 16. Matrix-Matrix Multiplication – Simulator context switches

messages immediately, but rather only when there is a deadlock potential. The bulk synchronous algorithm of the application amplifies this effect as the deadlock resolution protocol has only very distinct points where a deadlock potential arises that are separated by computation only phases. The algorithm does not have any application MPI messages outside the bulk synchronous operation that could transmit the LVT between physical MPI processes via the piggyback mechanism. Once oversubscription is involved, enough LVT information is transmitted between physical MPI processes with the application MPI messages that are part of the bulk synchronous operation.

This experiment demonstrates that the deadlock resolution protocol involves a trade-off every conservative PDES algorithm has to live with: either transmit many null (LVT update messages) to maintain progress or transmit less null (LVT update messages) to reduce network and message processing load. Given the presented results, the new deadlock resolution protocol is preferable to the previous one.

7. CONCLUSION

This paper documents two improvements to xSim. A new deadlock resolution protocol was implemented and tested to reduce the parallel discrete event simulation overhead, and a new simulated MPI message matching algorithm was developed and evaluated to reduce the oversubscription management cost. The improvements resulted in a significant performance improvement. For example, the simulation overhead for running the NPB suite was reduced from 102% to 0% for the EP benchmark and from 1,020% to 238% for the CG benchmark. With highly accurate simulation, the overhead was reduced from 3,332% to 204% for EP and from 37,511% to 13,808% for CG. The results also clearly show that negative side-effects due to the new deadlock resolution protocol, which communicates less information to improve performance and therefore runs the risk of delaying deadlock resolution and decreasing performance, are rare and minimal.

DOI: 10.1002/cpe

The two improvements to xSim enhance the productivity of system architects and application scientists that use xSim to estimate application performance on future-generation HPC architectures for hardware/software co-design.

Planned future work in xSim will target the improvement of the processor model for higher simulation accuracy, a performance model for file systems, soft error (silent data corruption) injection for supporting the development of resilient applications, and power consumption modeling to study the tradeoff between performance, resilience and power consumption.

ACKNOWLEDGEMENTS

Research sponsored by the Laboratory Directed Research and Development Program of Oak Ridge National Laboratory (ORNL), managed by UT-Battelle, LLC for the U. S. Department of Energy under Contract No. De-AC05-00OR22725.

REFERENCES

1. Hanai M, Shudo K. Optimistic parallel simulation of very large-scale peer-to-peer systems. *Distributed Simulation and Real Time Applications (DS-RT), 2014 IEEE/ACM 18th International Symposium on*, 2014; 35–42.
2. Zasada S, Mamonski M, Groen D, Borgdorff J, Saverchenko I, Piontek T, Kurowski K, Coveney P. Distributed infrastructure for multiscale computing. *Distributed Simulation and Real Time Applications (DS-RT), 2012 IEEE/ACM 16th International Symposium on*, 2012; 65–74.
3. Cecin F, Geyer C, Rabello S, Barbosa J. A peer-to-peer simulation technique for instanced massively multiplayer games. *Distributed Simulation and Real-Time Applications, 2006. DS-RT'06. Tenth IEEE International Symposium on*, 2006; 43–50.
4. Engelmann C, Naughton T. Improving the performance of the extreme-scale simulator. *Proceedings of the* 18^{th} *IEEE/ACM International Symposium on Distributed Simulation and Real Time Applications (DS-RT) 2014*, IEEE Computer Society, Los Alamitos, CA, USA: Toulouse, France, 2014; 198–207.
5. Naughton T, Engelmann C, Vallée G, Böhm S. Supporting the development of resilient message passing applications using simulation. *Proceedings of the* 22^{nd} *Euromicro International Conference on Parallel, Distributed, and network-based Processing (PDP) 2014*, IEEE Computer Society, Los Alamitos, CA, USA: Turin, Italy, 2014; 271–278.
6. Engelmann C, Naughton T. Toward a performance/resilience tool for hardware/software co-design of high-performance computing systems. *Proceedings of the* 42^{nd} *International Conference on Parallel Processing (ICPP) 2013:* 4^{th} *International Workshop on Parallel Software Tools and Tool Infrastructures (PSTI)*, IEEE Computer Society, Los Alamitos, CA, USA: Lyon, France, 2013; 962–971.
7. Engelmann C. 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)* Jan 2014; **30**(0):59–65.
8. Böhm S, Engelmann C. xSim: The extreme-scale simulator. *Proceedings of the International Conference on High Performance Computing and Simulation (HPCS) 2011*, IEEE Computer Society, Los Alamitos, CA, USA: Istanbul, Turkey, 2011; 280–286.
9. Engelmann C, Lauer F. Facilitating co-design for extreme-scale systems through lightweight simulation. *Proceedings of the* 12^{th} *IEEE International Conference on Cluster Computing (Cluster) 2010:* 1^{st} *Workshop on Application/Architecture Co-design for Extreme-scale Computing (AACEC)*, IEEE Computer Society: Hersonissos, Crete, Greece, 2010; 1–8.
10. OpenMP Architecture Review Board. OpenMP Application Program Interface version 3.0 2013.
11. Dekate C, Anderson M, Brodowicz M, Kaiser H, Adelstein-Lelbach B, Sterling T. Improving the scalability of parallel n-body applications with an event-driven constraint-based execution model. *International Journal of High Performance Computing Applications (IJHPCA)* Aug 2012; **26**(3):319–332.
12. STELLAR Group, Louisiana State University. HPX version 0.9.7 2013.
13. Bland W, Bouteiller A, Herault T, Hursey J, Bosilca G, Dongarra JJ. An evaluation of user-level failure mitigation support in MPI. *Proceedings of the 19th European Conference on Recent Advances in the Message Passing Interface*, EuroMPI'12, Springer-Verlag: Berlin, Heidelberg, 2012; 193–203.
14. Noeth M, Ratn P, Mueller F, Schulz M, de Supinski BR. ScalaTrace: Scalable compression and replay of communication traces for high-performance computing. *Journal of Parallel and Distributed Computing (JPDC)* Aug 2009; **69**(8):696–710.
15. Wu X, Mueller F. Elastic and scalable tracing and accurate replay of non-deterministic events. *Proceedings of the 27th international ACM conference on International conference on Supercomputing (ICS) 2013*, ICS '13, ACM: New York, NY, USA, 2013; 59–68.
16. Wu X, Deshpande V, Mueller F. ScalaBenchGen: Auto-generation of communication benchmarks traces. *Proceedings of the 2012 IEEE 26th International Parallel and Distributed Processing Symposium (IPDPS)*, IPDPS '12, IEEE Computer Society: Washington, DC, USA, 2012; 1250–1260.
17. National Aeronautics and Space Administration. NAS Parallel Benchmarks 2014.
18. Fujimoto RM. Parallel discrete event simulation. *Commun. ACM* Oct 1990; **33**(10):30–53.

DOI: 10.1002/cpe

19. Chandy K, Misra J. Distributed simulation: A case study in design and verification of distributed programs. *Software Engineering, IEEE Transactions on* Sept 1979; **SE-5**(5):440–452.
20. Engelmann C, Geist A. Super-scalable algorithms for computing on 100,000 processors. *Lecture Notes in Computer Science: Proceedings of the* 5^{th} *International Conference on Computational Science (ICCS) 2005, Part I*, vol. 3514, Springer Verlag, Berlin, Germany: Atlanta, GA, USA, 2005; 313–320.
21. Zheng G, Kakulapati G, Kale LV. BigSim: A parallel simulator for performance prediction of extremely large parallel machines. *Proceedings of the* 18^{th} *IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2004*, IEEE Computer Society: Santa Fe, New Mexico, 2004.
22. Kale LV, Bohm E, Mendes CL, Wilmarth T, Zheng G. Programming petascale applications with Charm++ and AMPI. *Petascale Computing: Algorithms and Applications*. CRC Press, 2007; 421–441.
23. Kale LV, Bhatele A ((eds.)). *Parallel Science and Engineering Applications: The Charm++ Approach*. Taylor & Francis Group, CRC Press, 2013.
24. Perumalla KS. $\mu\pi$: A scalable and transparent system for simulating mpi programs. *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering): ICST, Brussels, Belgium, Belgium, 2010; 62:1–62:6.
25. Rodrigues AF, Hemmert KS, Barrett BW, Kersey C, Oldfield R, Weston M, Risen R, Cook J, Rosenfeld P, CooperBalls E, *et al.*. The structural simulation toolkit. *SIGMETRICS Perform. Eval. Rev.* Mar 2011; **38**(4):37–42.
26. Casanova H, Giersch A, Legrand A, Quinson M, Suter F. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing* Jun 2014; **74**(10):2899–2917.
27. Buyya R, Murshed M. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience* 2002; **14**(13-15):1175–1220.
28. Minkenberg C, Herrera GR. Trace-driven co-simulation of high-performance computing systems using omnet++. *OMNeT++ 2009: Proceedings of the 2nd International Workshop on OMNeT++ (hosted by SIMUTools 2009)*, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering): ICST, Brussels, Belgium, Belgium, 2009.
29. Girona S, Labarta J, Badia RM. "Validation of dimemas communication model for MPI collective operations". *Lecture Notes in Computer Science: Proceedings of the* 7^{th} *European PVM/MPI Users' Group Meeting (EuroPVM/MPI) 2000*, vol. 1908, Springer Verlag, Berlin, Germany: Balatonfüred, Hungary, 2000; 39–46.
30. Pillet V, Labarta J, Cortes T, Girona S. PARAVER: A Tool to Visualize and Analyze Parallel Code. *Proceedings of WoTUG-18: Transputer and occam Developments*, 1995; 17–31.
31. Knüpfer A, Brunst H, Doleschal J, Jurenz M, Lieber M, Mickler H, Müller MS, Nagel WE. The vampir performance analysis tool-set. *Tools for High Performance Computing*, Resch M, Keller R, Himmler V, Krammer B, Schulz A (eds.). Springer Berlin Heidelberg, 2008; 139–155.
32. Ns3 network simulator. Available online at http://www.nsnam.org 2015.
33. Tetcos. Netsim network simulator. Available online at http://tetcos.com/software.html 2015.

DOI: 10.1002/cpe