Towards Lightweight and Scalable Simulation of Large-Scale OpenSHMEM Applications

M.J. Levenhagen, S.D. Hammond, and K.S. Hemmert

Center for Computing Research
Sandia National Laboratories
Albuquerque NM 81723
United States
{mjleven, sdhammo, kshemme}@sandia.gov

Abstract. Significant changes are coming to the high-performance computing community, including unprecedented growth in the use of machine learning and large-scale high-performance data analytics. Traditionally, HPC has been dominated by the use of the Message Passing Interface (MPI), but the increasing use of smaller, fine-grained communication in these new classes of workload is now making efficient one-sided programming models like OpenSHMEM more attractive.

Historically, the accurate analysis and performance projection of parallel algorithms written to one-sided models has been challenging, in part, because of the complexity associated with the tight bounds on modeling fine grained data movement, as well as complex interactions between the network and memory subsystems. In this paper, we describe recent extensions to Sandia's high-performance, parallel Structural Simulation Toolkit (SST) which permit rapid evaluation and projection for communication patterns written to one-sided paradigms – including explicit support for OpenSHMEM like functionality and communication patterns.

In our initial work we demonstrate multi-rank random-access like communication patterns, comparing simulated results with benchmarked values from a Cray XC40 Aries-based interconnect that is known to be efficient for fine-grained communication. The models show strong predictive accuracy and trends when varying characteristic hardware parameters such as the Aries virtual page size.

We document our current approaches and the significant components of our model that allow for what-if analyses to be completed by the community, thereby establishing SST as a reliable predictive toolkit for users of OpenSHMEM.

Keywords: Performance Analysis · Simulation · SHMEM

1 Introduction

Traditionally, high-performance, parallel computing applications have been written to utilize the Message Passing Interface (MPI) standard [2] [3] – a technol-

ogy which was largely defined during the late 1990s and has been incrementally adapted since to adopt a number of more modern code constructs. The HPC community has been mostly content to pursue this path over the past two decades because of a stable, agreed standard that has gained significant vendor/implementation support and provided a performant path to parallel execution. For the simulation and performance modeling community, MPI has been the premeninent focus because of the large community interest [8] [11][10].

However, applications written in several other domains experience different requirements. For applications with low degrees of messaging locality and, typically small, fine-grained message transfers, traditional MPI two-sided semantics add considerable burden and performance overhead. MPI has adapted through its "one-sided" semantics (added in the MPI 3.0 specification) to address these concerns, but alternatives such as OpenSHMEM [1][12] have proven to be easier to adopt for some communities and have also gained interest from important classes of communities such as those pursuing large-scale graph processing and data analytics. The growing interest in supporting these communities, which is now arguably one of the main drivers for contemporary server-class hardware architecture design, is fundamentally shifting the focus of technology development. Understanding the performance of applications written to OpenSHMEM-class semantics is becoming a progressively more important aspect to modern architecture and application performance analysis.

To this end, we have begun work on adapting models in Sandia's scalable, high-performance Structural Simulation Toolkit (SST) [14],[13], [15] to support OpenSHMEM style semantics and application models. This work has required a fundamental redesign of our simulated software stack to permit the lightweight semantics required for OpenSHMEM and required thorough redesign of components to ensure that simulations are rapid to execute and do not require significant amounts of memory or computation during use. Remembering that our goal with this class of SST models (other SST models provide full cycle-accuracy at higher simulation runtime cost) is to provide accurate predictive capabilities for medium to large-scale network designs (upwards of thousands of nodes) within tractable runtimes. The reader is reminded that many tradeoffs must be made to achieve this goal. Therefore, our models are intended to capture *critical* performance concerns, allowing us to focus simulation time in the areas which affect predictive capability. To this end, not every artifact which can affect performance, or indeed, every hardware structure is fully replicated in our model. The ones selected, and documented in this paper, represent our current approach to modeling OpenSHMEM.

The contributions of this work are as follows:

- Documentation of our progress in developing a new, high-performance, scalable, open-source simulation capability for the OpenSHMEM runtime and performance analysis community;
- Presentation of the simulation parameters used in the new SST models, enabling *what-if* analyses to be completed for future machine designs;

– Validation of an initial suite of OpenSHMEM benchmarks that show the predictive capabilities of the SST OpenSHMEM simulation modules when compared to a Cray XC40.

The remainder of this paper is laid out as follows: Section 2 describes the construction of OpenSHMEM communication models in the SST simulation framework and highlights some of the important components used during model. Section 3 compares results obtained from the initial SST models against benchmark runs using the optimized Cray SHMEM runtime on our test XC40 development platforms. Finally, we conclude the paper in Section 4.

2 Modelling OpenSHMEM in the Structural Simulation Toolkit

SST is designed to be a fast parallel simulation framework which supports flexible configuration (or extension) of its various simulation models. From the ground up, SST has utilized dynamic loading of components to support highly configurable simulation models. To date, the simulation framework has been used to study a variety of next-generation architectures or architectural features that could be utilized to provide significant application or workload performance improvement.

Within the domain of modeling large-scale interconnect/communication performance, SST has historically been focused on modeling of two-sided or collective MPI communications, e.g. [4]. In our experience, the models used to project MPI behavior have driven higher fidelity for components which capture the behavior of MPI match list resolution and typically, the transfer of large messages (kilobytes or upwards). Thus, the effect has been for the authors of SST's models to direct time to increased modeling accuracy in parts of the simulation which are exercised by heavier bandwidth utilization and MPI messaging rates. Accurately replicating MPI behavior also typically requires lower fidelity models in areas such as cache and TLB performance because MPI application writers have historically manually packed communication buffers resulting in stream-like behavior when performing memory-to-network transfers.

In Figure 1, we show the components of a typical interconnect/communication simulation that are included in SST and how these map into communication patterns (the model of an application), the software runtime and the networking layers. With each model being independent, it is possible for users or vendor suppliers to change out the current implementations with optimized versions that increase modeling accuracy or permit the analysis of alternative hardware approaches. Alternatively, users can utilize the models provided and adjust the modeling parameters to represent current or next-generation platforms.

As we have adapted models within SST originally intended for simulating MPI communication, we have extended the components shown in Figure 1 to work with OpenSHMEM patterns. Specifically, we have introduced extra components for our models to capture atomic memory operations within the NIC as

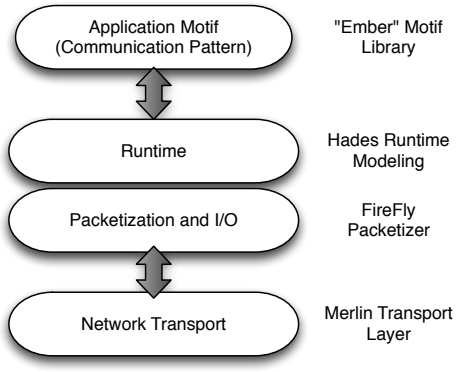

Fig. 1. Relationship between Logical Model and SST Components (SST Component Names shown on the Right-Hand Side)

well as state machines and timing models to replicate the behavior of caches and TLBs to capture the overheads associated with fine-grain or small (up to a few hundred bytes) communications where the hardware structures can dominate efficiency and operation timing.

2.1 Ember Motifs - Lightweight Communication Patterns for Scalable Simulation

The Ember component within SST [6] contains parameterizable communication patterns (which we term *motifs*), written to utilize either MPI or SHMEM-like semantics. Each motif is designed to focus on a specific communication pattern which can then be used with others during a simulation to form a model for a larger application or workflow. Since each motif is intentionally limited in scope to a specific communication sequence, and focused only on the communication aspects of the algorithm, the model is usually easy to write and the basic pattern can be scaled as a representation of our application to much larger node/instance counts than alternatives such as those that utilize trace replays or direct execution of an application binary. In previous studies, SST has been scaled to simulate over one million MPI ranks using its parallel multi-node discrete-event core.

The basic behavior of an Ember motif is that the communication pattern is repeatedly polled until it marks itself as complete. A poll will occur when there are no events left to process in the motif's queue, indicating that the last round of events provided during the previous poll has all been passed to the lower-level Hades runtime. Note, this does not mean that all previous events have completed, just that they have been processed and have either completed, or, are now in flight/being performed. The polling nature of Ember allows us to further reduce the memory overheads associated with an end-point model as

well crafted communication patterns can provide only a minimal set of events to be processed for each poll. For MPI patterns, Ember motifs typically encode one iteration worth of communication events, where an iteration relates to a period of interest in the application model – for instance, a complete set of messages associated with one round of nearest neighbor communication exchanges. For OpenSHMEM patterns, we have been able to lower the overheads substantially by making each poll return only a small number of fine-grained communication events such as one random remote atomic increment. Repeated polling of the motif adds very little to the simulation overhead.

2.2 Hades Runtime Layer

Events that are generated in an Ember motif are processed and then passed to the Hades SST component which maps each event type into a state-machine handler. Each state-machine reads the event operations and parameters, for instance, the target node and address of a remote atomic increment, and then proceeds to generate an event stream to complete the request. The state machines are therefore responsible for model timing and the semantic behavior of a specific MPI or OpenSHMEM function – note that we implement a selection of the most commonly used routines from both runtimes. Over time we intended to expand the functions available to a broader set but for the purposes of architecture exploration, only a limited subset which feature in our applications and models need implementation. State transitions generate timing which progresses the discrete event simulation forward. As each operation implements a unique state machine, timing for any individual operation can be easily replaced through SST's dynamic loading of sub-components with an alternative implementation of the same interface. Our experience with these models during development has been that vendor/academic partners have used this approach to perform analysis of hardware accelerated features or improved optimizations in their software stacks.

2.3 FireFly Packetization Layer

The FireFly model of SST performs network input/output state-machine handling. The model implements a series of low-level state machines that provide the ability to packetize data movement requests in the form of remote `put`, `get` and atomic operations. FireFly accepts data movement requests from the Hades layer and handles these by generating a series of simulated memory subsystem reads/writes, fracturing the data requests into network packet sized chunks that can then be given to the network transport layers for routing and delivery. For our adaption of SST to perform modeling of OpenSHMEM patterns, we have added interaction with NIC/node memory subsystem handlers for NIC-based TLBs and on-host caches (shown in Figure 2). These new state-machine interactions ensure that the overheads associated with address resolution and whether or not the data exists in processor caches are adequately captured.

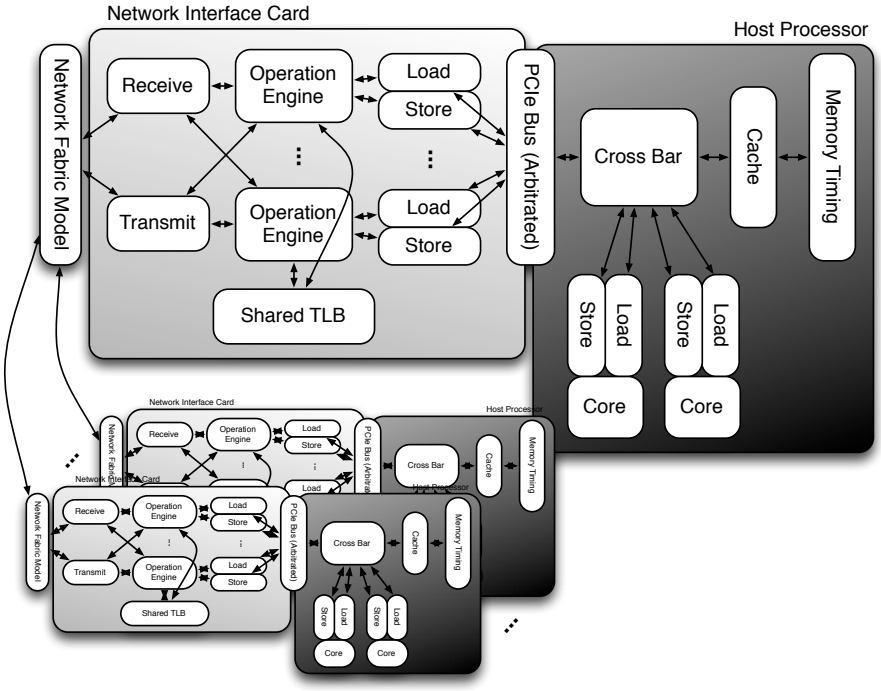

Fig. 2. High-Level Model Overview for OpenSHMEM Models in SST

2.4 Merlin Transport Layer

SST's Merlin components provides an implementation of network transport for our simulations. Both network interfaces and switches/routers are provided that allow for various topologies to be simulated (*e.g.* dragonfly, fat-tree, torus, mesh, flattened butterfly *etc.*). The transport layer accepts packets from FireFly (although any packetization scheme which utilizes the same interface can be used) and routes them to remote network interfaces modeling switch/router buffer delays, congestion management, and any adaptive routing choices. Merlin simulates network traffic at the packet level to improve simulation performance but generates timing with the knowledge of packet header/tails and how this affects router buffer utilization and timing.

2.5 SST Node Model

Since OpenSHMEM interacts deeply with the compute node, memory and various aspects of the processor, we have implemented several lightweight models of performance-relevant structures (see Figure 2). The communication pattern/application is driven by events from an Ember motif as described earlier.

In our diagram the motif conceptually runs on a core found in the host processor block. We implement one core per OpenSHMEM PE along with an instance of the Hades runtime. Hades makes requests through the other node components (arbitrated bus, PCIe connection *etc.*) until the FireFly packetizer is reached, labeled as 'Receive' and 'Transmit' in the our figure. The data movement operations generated by FireFly will then interact with the TLB and cache models found in the NIC and host. Finally, Merlin, which is shown in the image as the "Network Fabric Model", provides network transportation and delivery.

The reader should note the use of models for the TLB and load/store memory pipes, as well as a cross-bar arbitration model to control parallel access to the network interface. Our experience in model development has shown that these components are critical to achieving good validation results, as well as being places where parameterization is desired to investigate future hardware acceleration opportunities.

On-node communication is modeled without requiring transportation using Merlin, since, in most cases, this does not require participation of the NIC/interconnect. We model each PE with a unique copy of the motif and Hades stack to simplify development and permit PE to PE communication on node. When communication does occur on node, we route data transfers between the on-node instances consuming memory operations and bandwidth but by-passing the NIC structures.

3 Initial Model Validation and Assessment

In order to assess the potential accuracy of the SST models which are still under development, we have implemented several simple communication patterns, both as benchmarks written in OpenSHMEM, as well as Ember communication motifs. While these current benchmarks and motifs do not directly represent larger applications – as is our eventual goal – they have the effect of driving system runtime and hardware components extremely hard, thus ensuring we are able to assess the overheads and parameterization of our models. We will continue to use aggressive micro-benchmarks (as described, and others) to establish basic validation use cases as we build out the additional features needed for larger application models.

The micro-benchmarks used for our validation study are the following. As a basis, we use multi-node random access memory updates (often referred to as Gigaupdates-per-second (GUPs)) which was popularized as a on-node benchmark by the HPC Challenge benchmark collection [9]). We have extended this benchmark to OpenSHMEM to act as a proxy for high concurrency, fine-grained multi-node data operations.

– **Randomized Remote Increments (GUP/s)** - performs a uniform random selection of a remote PE and executes a remote atomic 32-bit integer increment of an address selected from a uniform random distribution. This

behavior mimics highly randomized remote memory access patterns in an executing job where there is no decernable locality to the accesses.
- **Randomized Remote Increments with Hot-Spot (Hot-Spot GUP/s)** - selects a weighted random selection of a remote PE (weighted towards one 'victim' node which is intended to be a hot-spot for remote traffic) and then executes a remote atomic 32-bit integer increment of an address selected from a uniform random distribution. This behavior mimics graph traversals or remote data structures where there is some degree of locality in the accesses or where some graph-vertices have high connectivity.
- **Randomized Remote Increment with Return Values (Finc-GUP/s)** - performs a uniform random selection of a remote PE and executes a remote atomic 32-bit integer increment of an address selected from a uniform random distribution with the operation requiring a return value. This behavior allows us to analyze the performance of the return path for remote atomic memory operations.

3.1 Model Validation - Cray XC40

For comparison to real hardware, we use the *Mutrino* Application Readiness Testbed (ART) system located at Sandia National Laboratories. The system is routinely used for application porting, development and optimization in support of the larger *Trinity* supercomputer [18] deployed by Los Alamos and Sandia during 2015 (one of the largest installations of its class in the world). Mutrino provides one-hundred nodes of dual-socket sixteen-core Intel Haswell server processors [5][7] which run at 2.30GHz. An additional hundred nodes of Intel's Knight Landing 7250 processor [16][17] are also available but we focus exclusively on the Haswell partition to reduce model validation complexity, leaving the Knights Landing comparison to future work. We utilize the optimized Cray SHMEM software stack for benchmarking which is provided to enable high-performance SHMEM semantics over the Cray Aries DragonFly interconnect.

The SST model for Mutrino is obtained through the specification of a Merlin dragonfly model and using the hardware parameters specified in Table 1. The reader should note that SST is extremely configurable, the parameters shown are the important characteristics required for performance but additional values can be specified as needed. In general, the SST developer group recommends that users intending to use SST consult the example models included with the distribution for additional examples. Additional examples are also found in the SST project repositories.

3.2 Model/Benchmark Validation

Table 2 shows the first of our validation studies. In this example we compare benchmarked random remote SHMEM put operations to those predicted by SST. In this benchmark the system executes a very large number of remote memory writes which are characterized by a very high rate of fine-grained communication. We show a comparison for SST when running with a single PE per node and

Host Software Timing	
Motif to Enqueue NIC Operation	140ns
NIC to Return Control to Motif (Non-Blocking Operation)	20ns
NIC to Return Control to Motif (Blocking Operations)	300ns
PCIe Bus Timing	
PCIe Latency	170ns
Link Bandwidth	7.8Gb/s
Number of Links	16
PCIe TLP Overhead	30 Bytes
PCIe DLL	16 Bytes
Network Interface Timing	
Setup a Transmit (TX)	25ns
Setup a Receive (RX)	25ns
Memory Subsystem Timing	
Memory Read	120ns
Memory Write	20ns
TLB Size (Shared)	512 Entries
TLB Miss Penalty	850ns
Number of NIC TLB Walkers	32
Network Timing	
Link Bandwidth	10GB/s
Router Crossbar Bandwidth	12.5GB/s
Link Latency	60ns
Router Input Buffer Latency	0ns
Router Output Buffer Latency	50ns
Router Input Buffer Size	8kB
Router Output Buffer Size	8kB
Network Packet Size	512 Bytes
Network Flit Size	8 Bytes

Table 1. Parameterization for SST Components (Most Significant Configurations Shown). Most SST parameters required units to be specified unless otherwise stated. For more information on parameterization, users should consult the SST examples/documentation included in the toolkit releases and project repositories.

Nodes	**1 Core/Node**			**32 Cores/Node**		
	Benchmarked	Simulated	Error (%)	Benchmarked	Simulated	Error (%)
2	0.015	0.014	7.53	0.09	0.29	238.07
4	0.030	0.027	9.69	0.17	0.52	213.59
8	0.060	0.054	9.41	0.42	0.75	78.76
16	0.114	0.108	5.65	1.13	1.69	49.86
32	0.229	0.215	5.75	2.87	3.27	13.74
64	0.449	0.430	4.22	6.70	6.44	3.95

Table 2. Benchmarked and Simulated calls to PUT operations. Values are shown for the 8MB Cray-Aries page size, higher PUT-count values are better, while lower predictive error percentages are better.

then 32 PEs per node up to 64 nodes. In the single PE case, the SST predictive accuracy is good with between 4.2 and 9.6% predictive error. This indicates the on-node model and basic network parameterization are well specified. For the 32 PEs per node case, SST over predicts the performance significantly estimating a much higher number of PUT/s than can be benchmarked. As the number of nodes increase and the number of off-node communication partners increases as a percentage of the application, the predictive accuracy improves significantly to around 4%. This result indicates that the SST model predicts performance well when more of the communication happens to be performed over the interconnect. When there is higher probability of communication on node (as is the case with small node counts), the predictive nature of the model is considerably poorer. We attribute this effect to the very simplistic modeling of contention in on-node resources. Such simple models prevent simulation time from becoming too large but have the trade-off that when communication may congest on-node resources, SST will be much more optimistic than the hardware results.

In Figure 3 we show benchmarked GUP/s implemented with remote 32-bit integer atomic increment updates compared to a simulated SST model. For this run we have included a sweep over Cray-Aries page sizes to demonstrate the variation in results that can achieved with this architecture parameter. The SST model used for our validation has a NIC-side TLB which is able to partition the address space using the Cray-Aries TLB size parameter. The figure shows strong correlation in behavioral trend with the 8MB page size performing the best (which is why other results shown in the paper utilize the 8MB page size). As in the case of the PUT results above, predictive accuracy is strongest when the number of ranks per node is lower and decreases at high rank-per-node density when contention and congestion for on-node resources increase.

Our third set of model validations are shown in Table 3. In this benchmark we perform 'hot-spot' based GUP/s where the random atomic 32-bit integer increments are selected with a weighting towards a particular 'victim' PE in the application. In our results we have adjusted the weighting so that: (1) there is no increased probability ("1x"); and, then, 4X and 8X higher probability respectively. Predictive accuracy is again, much higher when more compute nodes

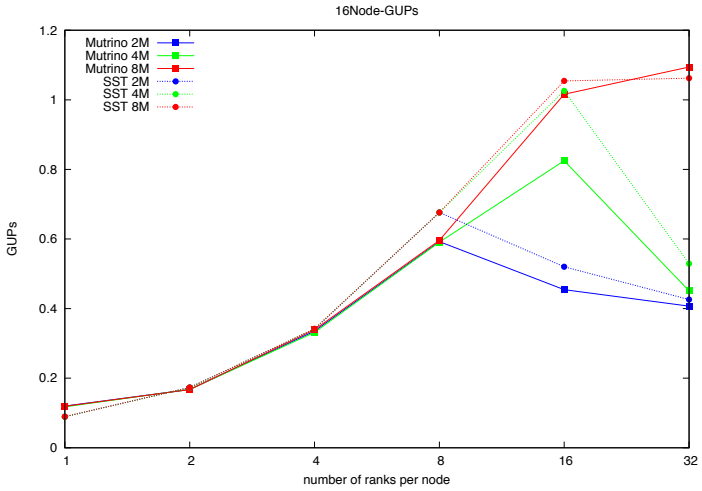

Fig. 3. Benchmarked ("Mutrino") vs. Simulated ("SST") Results for 16-Node Random Access Increments Running on the Sandia Mutrino Cray XC40 Aries System

are used reflecting the emerging trend that the network models and parameterization used have reasonable accuracy. On-node models continue to have lower predictive accuracy when used in the 2 node, 32 PE per node case as a significant number of the random communications occur on node. Predictive accuracy as the simulated system increases to between 1 and 14%.

The final set of validation results are shown in Table 4. In these results we perform benchmarking of GUP/s changing the operation type to require remote atomic 32-bit integer updates with a returned value. This comparison therefore adds the complexity of processing the return path of data from a remote PE. At a single PE per core, the predictive accuracy is very strong with between 0 and 14% modeling error for up to 64-nodes. When using 32-cores per node, we see the now familiar trend of predictive accuracy decreasing to around 50% model error. Despite the high predictive error in terms of raw GUP/s rate, the reader's attention is drawn to the high correlation of performance trends shown – *i.e.* both Mutrino and SST shown similar increase in application performance with increasing node count even if the predicted benchmark performance contains a large error. Historically, hardware designers have highly valued simulation results with good predictive trends as the use of high-level model representations has made exact performance prediction extremely challenging.

3.3 Discussion and Analysis

The predictive accuracy of the benchmarks shows a trend – that SST is more faithfully able to represent OpenSHMEM application performance as the scale of the application and system increases, or, where communication is largely reliant

	Node Count x PEs/Node					
Hot-Spot Intensity	2 x 32	4 x 32	8 x 32	16 x 32	32 x 32	64 x 32
Mutrino Benchmarked GUP/s						
1X	0.15	0.28	0.56	1.07	1.82	4.15
4X	0.14	0.28	0.54	1.00	1.99	3.90
8X	0.14	0.26	0.50	0.97	1.88	3.66
SST Simulated GUP/s						
1X	0.25	0.33	0.57	1.08	2.08	4.14
4X	0.24	0.31	0.53	0.97	1.88	3.70
8X	0.22	0.29	0.48	0.88	1.70	3.36
Predictive Error (%)						
1X	67.39	18.49	2.15	0.91	14.29	0.19
4X	68.99	10.66	2.45	3.10	5.93	5.13
8X	61.20	8.45	4.59	8.76	9.87	8.33

Table 3. Benchmarked and Simulated GUP/s for the Hot-Spot GUP/s Kernel using 32 PEs per compute node. Hot-Spot intensity refers to weighting of random selections towards the victim-PE, 4X implies 4 times higher probability of selecting the PE over alternatives. Values are shown for the 8MB Cray-Aries page size, higher GUP/s values are better, while lower predictive error percentages are better.

on the network. This is perhaps not a surprise, the complexity of modeling on-node resources with a lightweight model is extremely challenging, not least, because of the wide variety of optimizations present in contemporary systems. The network models used and the models associated with processing operations as the NIC appear to be much more favorable for predictive accuracy. We argue that modeling small applications which execute on only a handful of compute nodes, or predominantly on-node, is of less interest to architecture designers. Our goal in SST is to push the simulation toolkit capabilities to be useful for larger-scale systems and more scalable applications. Our results show that our models are more accurate in this regime.

4 Conclusion

The development of accurate, performance analysis tools for large-scale parallel applications has historically been the preserve of the MPI community – in part because of the widespread use at leadership science facilities. OpenSHMEM, however, provides a promising alternative programming model for writing applications in communities that develop algorithms for scalable graph analytics and data analysis applications. These communities are experiencing huge growth and have somewhat unique hardware/software requirements. The development of tools to support the assessment of these requirements on next-generation computing architectures is much needed.

Nodes	**1 Core/Node**			**32 Cores/Node**		
	Benchmarked	Simulated	Error (%)	Benchmarked	Simulated	Error (%)
2	0.001	0.001	3.56	0.025	0.061	142.61
4	0.002	0.003	14.09	0.048	0.097	100.85
8	0.004	0.005	1.27	0.096	0.157	64.52
16	0.009	0.009	0.05	0.190	0.288	51.83
32	0.016	0.017	5.59	0.367	0.554	51.16
64	0.032	0.034	6.41	0.724	1.088	50.25

Table 4. Benchmarked and Simulated GUP/s using calls to finc. Values are shown for the 8MB Cray-Aries page size, higher GUP/s values are better, while lower predictive error percentages are better.

In this paper, we have presented an overview of a new suite of simulation components which are being written for Sandia's Structural Simulation Toolkit (SST) – a high-performance, scalable simulation framework which is the codesign tool of choice for several industry leading vendors and academic institutions, as well as several Department of Energy laboratories.

Despite the models being early in their development, the results in this paper show the value of the high degree of parameterization which has been applied in their construction. Such an approach is required to gain accurate predictions on contemporary hardware, as well as allowing what-if comparisons to be performed on the next-generation of architectures being developed in the research laboratory today.

While our work is not yet fully complete and much more analysis and validation lies ahead, the models show good correlation with benchmarked performance from Cray's Aries optimized SHMEM implementation on several micro-benchmarks designed to stress critical runtime/hardware performance components. Our results show that as we push the scale of applications and systems to progressively large node counts, the predictive accuracy of our simulation models increase. We argue that through close collaboration with the OpenSHMEM community, we will be well placed to offer leading-edge performance projection capabilities.

In the future versions of SST our OpenSHMEM model will continue to be extended to support additional OpenSHMEM runtime/programming model features and to permit even greater levels of parameterization for codesign activities. We welcome input from the community on how best to prioritize these new features to maximize the utility of SST and look forward to showing the value of simulation tools to supporting the future generation of graph analytics and data analysis applications which will utilize SHMEM-like semantics.

Acknowledgements

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a

wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

1. Chapman, B., Curtis, T., Pophale, S., Poole, S., Kuehn, J., Koelbel, C., Smith, L.: Introducing OpenSHMEM: SHMEM for the PGAS Community. In: Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model. p. 2. ACM (2010)
2. Gabriel, E., Fagg, G.E., Bosilca, G., Angskun, T., Dongarra, J.J., Squyres, J.M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., et al.: OpenMPI: Goals, Concept, and Design of a Next Generation MPI Implementation. In: European Parallel Virtual Machine/Message Passing Interface Users Group Meeting. pp. 97–104. Springer (2004)
3. Gropp, W., Lusk, E.: Users Guide for MPICH, a Portable Implementation of MPI. Tech. rep., Argonne National Lab., IL (United States) (1996)
4. Groves, T., Grant, R., Hemmert, K., Hammond, S., Levenhagen, M., Arnold, D.: (SAI) Stalled, Active and Idle: Characterizing Power and Performance of Large-Scale Dragonfly Networks. In: Cluster Computing (CLUSTER), 2016 IEEE International Conference on. pp. 50–59. IEEE (2016)
5. Hammarlund, P., Martinez, A.J., Bajwa, A.A., Hill, D.L., Hallnor, E., Jiang, H., Dixon, M., Derr, M., Hunsaker, M., Kumar, R., et al.: Haswell: The Fourth-Generation Intel Core Processor. IEEE Micro **34**(2), 6–20 (2014)
6. Hammond, S.D., Hemmert, K.S., Levenhagen, M.J., Rodrigues, A.F., Voskuilen, G.R.: Ember: Reference Communication Patterns for Exascale. Tech. Rep. SAND2015-7019C, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States) (2015)
7. Jain, T., Agrawal, T.: The Haswell Microarchitecture - 4th Generation Processor. International Journal of Computer Science and Information Technologies **4**(3), 477–480 (2013)
8. Janssen, C.L., Adalsteinsson, H., Cranford, S., Kenny, J.P., Pinar, A., Evensky, D.A., Mayo, J.: A Simulator for Large-Scale Parallel Computer Architectures. International Journal of Distributed Systems and Technologies (IJDST) **1**(2), 57–73 (2010)
9. Luszczek, P.R., Bailey, D.H., Dongarra, J.J., Kepner, J., Lucas, R.F., Rabenseifner, R., Takahashi, D.: The HPC Challenge (HPCC) Benchmark Suite. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing. vol. 213. Citeseer (2006)
10. Mubarak, M., Carothers, C.D., Ross, R.B., Carns, P.: Enabling Parallel Simulation of Large-Scale HPC Network Systems. IEEE Transactions on Parallel and Distributed Systems **28**(1), 87–100 (2017)
11. Mudalige, G.R., Vernon, M.K., Jarvis, S.A.: A Plug-and-Play Model for Evaluating Wavefront Computations on Parallel Architectures. In: Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on. pp. 1–14. IEEE (2008)
12. Poole, S.W., Hernandez, O., Kuehn, J.A., Shipman, G.M., Curtis, A., Feind, K.: OpenSHMEM - Toward a Unified RMA Model. In: Encyclopedia of Parallel Computing, pp. 1379–1391. Springer (2011)

13. Rodrigues, A., Cooper-Balis, E., Bergman, K., Ferreira, K., Bunde, D., Hemmert, K.S.: Improvements to the Structural Simulation Toolkit. In: Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques. pp. 190–195. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering) (2012)
14. Rodrigues, A.F., Hemmert, K.S., Barrett, B.W., Kersey, C., Oldfield, R., Weston, M., Risen, R., Cook, J., Rosenfeld, P., CooperBalls, E., et al.: The Structural Simulation Toolkit. ACM SIGMETRICS Performance Evaluation Review **38**(4), 37–42 (2011)
15. Rodrigues, A.F., Murphy, R.C., Kogge, P., Underwood, K.D.: The Structural Simulation Toolkit: Exploring Novel Architectures. In: Proceedings of the 2006 ACM/IEEE conference on Supercomputing. p. 157. ACM (2006)
16. Sodani, A.: Knights Landing (KNL): 2nd Generation Intel Xeon Phi Processor. In: Hot Chips 27 Symposium (HCS), 2015 IEEE. pp. 1–24. IEEE (2015)
17. Sodani, A., Gramunt, R., Corbal, J., Kim, H.S., Vinod, K., Chinthamani, S., Hutsell, S., Agarwal, R., Liu, Y.C.: Knights Landing: Second-Generation Intel Xeon Phi Product. IEEE Micro **36**(2), 34–46 (2016)
18. Vaughan, C.T., Dinge, D., Lin, P., Hammond, S.D., Cook, J., Trott, C.R., Agelastos, A.M., Pase, D.M., Benner, R.E., Rajan, M., et al.: Early Experiences with Trinity-The First Advanced Technology Platform for the ASC Program. Tech. Rep. SAND2016-3605C, Sandia National Laboratories (SNL-NM), Albuquerque, NM (United States) (2016)