Simulating a communication middleware for one-sided and collective operations

Camille Coti, Martin Quinson

Simulating a communication middleware for one-sided and collective operations

Camille Coti
École de Technologie Supérieure
Université du Québec
Montréal, Canada
camille.coti@etsmtl.ca

Martin Quinson
Univ Rennes, Inria, CNRS, IRISA
Rennes, France
martin.quinson@ens-rennes.fr

***Abstract*—Measuring the performance of one-sided and collective operations on a set of distributed resources is not trivial because of the complexity of distributed time measurement on non-synchronized hardware. Simulation is attractive because distributed applications can be executed on a controlled set of resources and in a reproducible manner. Writing a fully calibrated simulator for a new software library represents however a time- and labor-intensive effort that is usually only undertaken for well established software solutions such as the MPI standard.**

This paper introduces an approach leveraging the SimGrid simulation framework for the study of an existing standard for high-performance computing parallel applications, featuring one-sided communications and atomic and collective operations. We present the several mechanisms that are mandatory to integrate the behavior of the real distributed applications into the simulated platform. Leveraging this simulation framework greatly simplifies this endeavor, opening the way to the study of more distributed middleware and libraries through realistic simulations.

I. Introduction

Measuring the performance of one-sided and collective operations on a set of distributed resources is not trivial because of the complexity of distributed time measurement on non-synchronized hardware [1], [2]. Simulation is attractive because distributed applications can be executed on a controlled set of resources and in a reproducible manner, including a fully controllable clock on the simulated resources with reproducible skew and drift. Many simulation frameworks were proposed in the literature [3], many of them enabling the study of abstract algorithms. Some solutions were proposed for the study of concrete systems, but only for well-established communication frameworks such as the MPI standard [4], [5]. The scarity of solutions to study other concrete communication libraries can be explained by the fact that developping a complete and accurately calibrated simulator is tedious and very labor-intensive [6]. One classical solution is to simulate offline execution traces [7], but this approach forbids the study of adaptative applications that modify their behavior according to the network conditions.

This paper presents an approach leveraging the SimGrid simulation framework [8] for the study of an existing standard for high-performance computing parallel applications called OpenSHMem [9]. This library features one-sided communications and atomic and collective operations, which performance is notoriously difficult to study on real platforms [1].

This paper is organized as follows. Section II gives an overview of simulation techniques for parallel and distributed systems, and how one-sided communications are implemented in high-performance computing libraries. Section III presents the design and implementation of a one-sided communication interface in a simulator. Some applications that can be executed are presented in Section IV. Finally, VI concludes the paper and discusses some open perspectives.

II. Review of the literature

A. Simulating distributed applications

The simulation of distributed systems usually faces a trade-off between speed and accuracy. Some simulators reproduce every operation made by the system: cycle-accurate CPU simulator play each CPU cycle [10], packet-accurate network simulators play every packet [11]. These simulators are useful to study such components in particular, since they are highly accurate. However, distributed systems combine a potentially large number of such components, and this approach is not scalable enough.

One solution consists in using coarser-grain simulation, with fewer details taken into account by the simulation. The challenge is then to find models that capture enough details of the simulated systems to reach an acceptable accuracy, and scalable enough. It is followed by simulators such as SimGrid [8] and GridSim [12].

SimGrid provides high-level definitions of the hardware components, such as CPUs, network links, and disks. Each component has characteristics, such as a number of cores that have a given compute speed for CPUs. It provides several network-level simulation approaches, such as fluid-based models that capture some effects from the TCP protocol or using the ns-3 simulator [13].

SimGrid provides an internal interface called S4U, but also an MPI implementation [14]. SimGrid can therefore be used as a drop-in replacement of the MPI library to execute parallel applications on the simulated platforms and allow precise performance analysis [15], or to implement communication protocols [16], large-scale task-based platforms [17] or other parallel computation models [18].

SimGrid can be extended by overriding methods to specify the model's behavior, and by defining plugins, for instance using the Functional Mock-up Interface (FMI) standard [19] and for energy consumption estimation [20].

While other frameworks can be used as drop-in replacements of the MPI library to enable the live simulation of the applications [21], we do not know any such solution exists for distributed applications that are not based on the MPI standard. Some solutions such as Netrix [22] exist to control the behavior of arbitrary distributed applications to allow their testing, but they do not given the experimental control of the network conditions as we do in this work.

B. Implementing one-sided communications

One-sided communications are critical for Parallel Global Address Space (PGAS) languages such as Unified Parallel C (UPC) [23] and UPC++ [24]. [25] noticed that MPI 1.1 and MPI 2.0's one-sided communications were not a satisfying target to implement UPC's one-sided communication needs. Later, a new RMA interface in MPI 3.0 and aggressive optimization efforts made it possible to implement a one-sided communication interface with good performance [26].

The OpenSHMEM interface [9], implemented in the aforementioned paper, is a standard for one-sided, atomic and collective communications. It finds its roots in the original SHMEM library from Cray Research in 1993 [27] and later variants such as SGI SHMEM in 1996 when Cray Research was merged with Silicon Graphics (SGI) and Quadrics SHMEM (Q-SHMEM), which later was the basics for HP SHMEM. IBM made IBM SHMEM for internal use, and TurboSHMEM using its low-level API (LAPI). Since the release of the OpenSHMEM standard [9], multiple approaches have been explored to get the best possible performance for its one-sided communications.

In [28] and [29], the authors propose the UCCS framework to close the increasing gap between the capabilities provided by the hardware and the communication models implemented by the communication libraries such as MPI and OpenSHMEM. In [30], the authors show that the cost of traversing the software stack can have a significant cost. The UCX framework aims at providing portability across network layers while maintaining a short API-to-network path [31].

Implementations have been made using low-level network drivers directly, such as Sandia OpenSHMEM [32] or MVAPICH [33], and using UCX, such as OpenMPI [34].

III. Simulating the evolution of distributed applications

A. Simulating time: the SimGrid approach

SimGrid relies on three abstractions:

- A *resource* is an element of the simulated platform (CPU, network link, disk)
- An *activity* is something happening on the platform and that we want to measure in some way. For instance, it can be a computation on a CPU or a communication on a network link.
- An *actor* can be seen as a process which is executing an activity on a resource. Actors execute their own activities sequentially, but they execute them independently of each other.

SimGrid measures the simulated time during the execution, but it can also be extended to use different metrics such as energy consumption. To measure time, it needs to keep track of the (simulated) moment when each activity starts, its progress, and when it completes. Each activity is defined by an amount of work to be done, and the chosen model computes how a given activity uses a given resource. When concurrent activities compete for the same resource (for instance, when two computations run on the same CPU core, or when two communications use the same network link), a linear min-max solver computes the speed of each activity.

An operation involving SimGrid is called a *simcall* and can be seen like what a system call is to the operating system. SimGrid considers each process of a distributed application as a sequence of activities that do not involve SimGrid and simcalls. SimGrid uses dedicated models to compute the simulated time of an activity on a given resource using its hardware specification.

B. Process folding

The processes of a distributed application simulated by SimGrid are folded into a single process, which is then used to simulate the execution and measure time. This mechanism ensures, among other properties, reproducible executions, and a global view of the simulated time.

a) Execution: SimGrid folds the processes of the distributed application into a single process. A maestro schedules the execution between these (simulated) processes. A process is chosen and executed until the next simcall. At this point, the maestro stops the execution of the process and switches to another process, which is executed until the next simcall. Once all the processes are waiting on a simcall, the maestro executes them.

The maestro schedules the execution between the processes in a reproducible way, and since their executions are made sequential, there is no concurrency between the executions of the simulated processes. However, the simulated operations can be concurrent.

b) Implementation: To achieve this sequential execution, SimGrid folds all the processes into a single process. For sake of clarity, we call *rank* a process of the distributed application that is to be folded as a thread in the simulation process, while the word *process* is used for the system process hosting the whole simulation (that is, all ranks plus the platform simulation).

To ensure that the global variables are privatized for each rank despite the folding, SimGrid loads the binary object corresponding to each rank and the libraries it depends upon using `dlopen`. This requires the binaries to be compiled as Position-Independent Executable with the `-fPIE` compiler flag, but this is a common practice in Linux anyway as it enables security counter-measures such as Address-Space

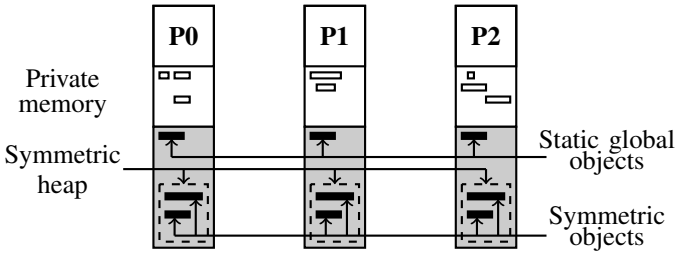

Fig. 1. OpenSHMEM memory model.

Layout Randomization (ASLR) [35]. Usually, the dynamic linker ensures that each object is loaded only once in memory even if it is used by several other objects. We circumvent this protection by first copying the object in another file on disk to force the dynamic linker to load a new copy of the same object for each *rank*. This approach has the disadvantage of duplicating not only the global variables but also the code of each object in memory, but from our experience, this is much more robust and portable than the `dlmopen` call that is supposedly intended for this usage.

Once the binary object is loaded in memory, we intercept the `main` function with `dlsym`. Hooks can be added at this point to execute specific operations on the library, such as the mechanism we added to share global and static variables with OpenSHMEM as discussed in the next Section.

C. Shared distributed memory model

a) OpenSHMEM's memory model: Figure 1 depicts OpenSHMEM's distributed memory model: each process has its own (private memory), and shares a heap that can be accessed remotely by other processes. This heap is *symmetric*, *i.e.,*allocations made on this heap are made by *all the processes of the application*. The corresponding function, `shemm_malloc`, is a collective operation. As a consequence, objects located in this heap are located at the same offset in each process's heap. Moreover, static global variables are located in this shared heap.

```
void shmem_put(TYPE *dest,
               const TYPE *source,
               size_t nelems, int pe);
```

Listing 1. Prototype of an OpenSHMEM RMA operation.

Listing 1 presents the prototype of a remote memory access function in OpenSHMEM: `shemm_put` puts the `nelems*sizeof(TYPE)` bytes from address `source` into `pe`'s shared heap, at the same place as `dest`. Since every process has a `dest` in its shared heap, the offset between the beginning of the shared heap and `dest` is the same (with architectural variations) on every process.

b) The symmetric heap: Since the simulated distributed processes are folded into a single process, the simulated processes have access to each other's memory. However, to obtain semantically realistic executions, we cannot let processes access all the memory of the other processes. For instance, in our simulation, process x might be able to access any address of process's y memory space, but in real life, it is not possible. Therefore, we put the remotely-accessible memory space into a `Buffer` object that contains enough information to verify that a remote access is actually happening in public space, and computes offset arithmetic operations to translate the local address used in the OpenSHMEM operations into offset-based addressing and into actual addresses.

c) Shared objects: In OpenSHMEM, global and static variables are accessible remotely by other processes. In the Elf format, such variables are located in the data segment of the executable (if they are initialized) and in the BSS segment (if they are not). In executable binaries, we can find these segments between symbols such as `__data_start`, `__bss_start` and `_end`. However, we have seen in Section III-B that the simulated processes are executed from libraries and not from executables. When the library corresponding to each process is opened by `dlopen` (see section III-B), we find the addresses of the bounds of these segments using the `dl_phdr_info` structure obtained from `dl_iterate_phdr` after the `dlopen`. We create a buffer for these memory areas and make them accessible similarly to the shared heap.

D. Implementing one-sided communications in SimGrid

We implemented the OpenSHMEM interface in SimGrid, making it possible to run OpenSHMEM applications on a simulated platform. All the OpenSMEM communications are implemented to achieve two goals: 1) perform the inter-process communication according to the OpenSHMEM specification and 2) take the communication into account in the simulation.

Specifically, OpenSHMEM's peer-to-peer communications are *one-sided*, *asynchronous* and *non-blocking*. However, a distinction must be made between *outgoing* operations such as `shemm_put` and *incoming* operations such as `shmem_get`: when the former returns, there is no guarantee that the data has been delivered to the target processes, whereas when the latter returns, the data is available.

a) The SimGrid communication model: SimGrid uses a mailbox-based communication model. A process that initiates a communication puts it in a mailbox, the destination process reads it from the mailbox. This model needs some adaptation to be used to implement one-sided communications, since the mailbox cannot be directly the remote process's memory.

We used a *communication daemon* that polls for messages in this mailbox. An outgoing communication is an *Activity*, in terms of SimGrid abstractions. When a process puts a communication in this mailbox, the daemon reads it and delivers it into the target process's memory.

We used SimGrid's internal primitives to take the communication into account into the simulation, and `memcpy` to perform the actual data movement.

b) Synchronization operations: OpenSHMEM provides several synchronizations to ensure completion and ordering of events.

shemm_quiet ensures completion of all the shmem_put operations issued before the call to shmem_quiet. We implemented it by waiting until all the outgoing local activities have completed.

shmem_fence is an ordering routing: all the shmem_put communications issued before shemm_fence are completed before any of the communications issued after shemm_fence can start. We implemented it using an *epoch*: the number of messages issued during an epoch is counted and they must be completed before the epoch number is incremented and subsequent communications can happen.

c) Atomic operations: OpenSHMEM defines atomic fetch-and-operate operations and atomic memory operations. We used a lock to ensure atomicity of the operations. In the SimGrid execution model (see Section III-B), these operations cannot be interrupted.

E. Measuring time

SimGrid provides mechanisms to insert probes triggered upon events happening on the simulated platform and in the application. The data collected during the execution can be used to visualize simulated resource usage, such as CPU load or how much of the links' capacity is used [36]. The probe mechanisms can be used by simulation platforms built on top of SimGrid, such as a BOINC simulator [17] or the StarPU task-based parallel environment [18], [37].

In addition to generic probes, SimGrid provides instrumentation functions to trace the execution of communications. We extended this interface to implement OpenSHMEM-specific functions, with functions corresponding to when a shmem_put is initiated on the source process and when the data arrives on the target process, and to when a shmem_get is initiated on the source process, when the target process starts transferring the data, and when the data has arrived on the source process.

Figure 2 represents the instrumentation points inserted in OpenSHMEM communications. The mechanisms provided by SimGrid to have a unified view of the simulated time make it possible to have a unique clock for these operations, which would require very accurate distributed clock synchronization in a distributed environment.

F. Miscellaneous implementation issues

A particularity of OpenSHMEM is that it provides type-specific routines: for instance, shmem_TYPE_put exists for TYPE being char, signed char, unsigned char, short, unsigned short, int, unsigned int, int8_t, int16_t, int32_t, int64_t, uint8_t, uint16_t, uint32_t, uint64_t, size_t, ptrdiff_t, double, long double, float, long, unsigned long, long long, and unsigned long long. There is also shmem_putSIZE with SIZE being one of 8, 16, 32, 64 and 128, shmem_putmem, and if C11 is supported, a generic shmem_put.

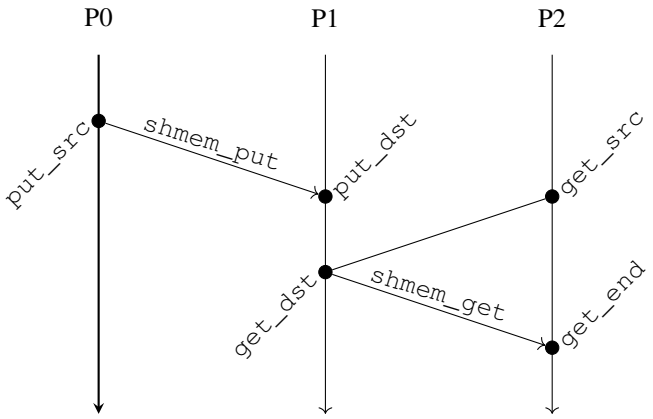

Fig. 2. Measuring OpenSHMEM communication operations.

Since SimGrid is implemented in C++, we used templates to implement these functions. However, we are providing a C interface that provides each of these functions.

IV. Simulation of some applications

We executed a set of open-source OpenSHMEM applications:

- The NAS Parallel Benchmarks [38] are a set of benchmarks performing well-characterized operations. The IS kernel (Integer Sort) performs random memory accesses and was implemented in OpenSHMEM[1]
- Shmem-Mandel computes a Mandelbrot fractal using OpenSHMEM[2]
- We used a sample program that solves the heat equation problem in parallel using OpenSHMEM[3]

The output obtained by the Shmem-Mandel program is shown Figure 3.

V. Current limitations and open perspectives

a) Contexts: OpenSHMEM 1.4 introduced contexts, which are operation streams in which operations are ordered, but independently from the other events of the application. They can be seen as Cuda streams: operations made in the same context are ordered, whereas operations made in different contexts might or might not be ordered. They can be used for thread safety and to obtain better overlap between communications [39].

SimGrid has some support for programs using POSIX threads through the *sthread* interface [40]. Combining one-sided communications and threads is not trivial and left for future works.

[1]https://github.com/openshmem-org/openshmem-npbs
[2]https://github.com/intijk/shmem-mandel
[3]https://github.com/blastmaster/heat_eaquation_openshmem

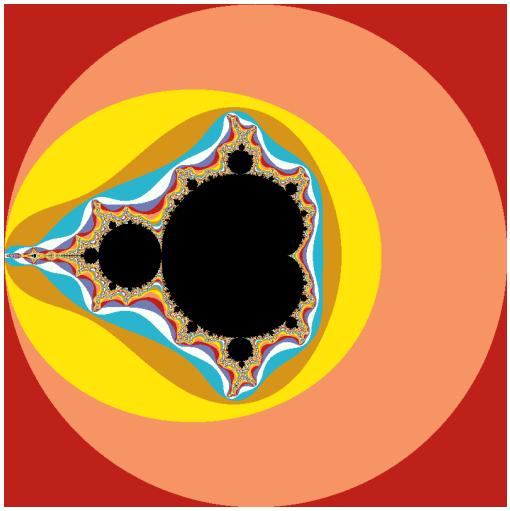

Fig. 3. Output of the simulated Shemm-Mandel program.

b) Hardware-specific performance considerations: The OpenSHMEM standard was designed to take advantage of fast RDMA interconnects [41], [42]. SimGrid provides different models for its network links, such as a TCP-fluid approximation, the NS-3 simulator, or use a plugin such as the WiFi network load and energy modelling plugin. An implementation of the Portals 4 protocol was made as a standalone SimGrid module [16]. It would be interesting to get a more fine-grain model of the performance of RDMA interconnects and how the OpenSHMEM library takes advantage of its semantics.

c) Memory accesses: SimGrid intercepts network operations through the calls to the library it implements (in our case, OpenSHMEM). It makes it possible to perform some safety checks, for instance by detecting concurrent communications or deadlocks. However, this does not include memory accesses. For instance, if a process performs a remote *put* on another process and this process performs a local read or write operation with no causal ordering of these two operations, there will be a race condition. The remote *put* is performed by SimGrid, but the local memory access is performed directly and SimGrid has no control over it.

VI. Conclusion and perspectives

This paper details an approach to simulate a concrete communication library featuring one-sided communications as well as atomic and collective operations, which performance is notoriously difficult to study on real platforms [1]. Such simulation is usually a time- and labor-intensive endeavor to capture both the behavior of the application running on top of the communication library and the semantic of the library itself, and integrate these elements into the simulated platform. Trace replay simplify the capture of the applicative behavior, but cannot be applied to reactive applications that adapt to the network conditions.

Instead, we present several mechanisms to fold the many nodes of the distributed application into a single process for sake of simulation performance and simplicity, and to trick the unmodified application into using the simulated platform instead of the real one. Our approach relies only on standard and portable mechanisms of the dynamic library linker. Communications are captured with a thin reimplementation of the library semantic on top of the SimGrid framework, enabling the study of advanced communication patterns such as one-sided or group communications.

We exemplify our approach on the OpenSHMEM library, that implements a growing standard for advanced communications in HPC context, but we believe that this approach could be used to ease the realistic simulation of other advanced communication libraries such as Remote Direct Memory Access (RDMA) standards, ZeroMQ, disaggregated architectures, and Parallel Global Address Spaces (PGAS).

In the future, this would enable the clairvoyant study of these applicative stacks, making it possible to extract information such as (simulated) time at any moment of the execution. This would permit advanced tracing solutions, and could even enable the study of these applications through fuzzing by modifying the behavior of the simulated platform accordingly.

References

[1] C. Coti and A. D. Malony, "SKaMPI-OpenSHMEM: Measuring openshmem communication routines," in *Workshop on OpenSHMEM and Related Technologies*. Springer, 2021, pp. 63–80.

[2] Z. Kang, R. Canady, A. Dubey, A. Gokhale, S. Shekhar, and M. Sedlacek, "A study of publish/subscribe middleware under different iot traffic conditions," in *Proceedings of the International Workshop on Middleware and Applications for the Internet of Things*, ser. M4IoT'20. New York, NY, USA: Association for Computing Machinery, 2021, p. 7–12. [Online]. Available: https://doi.org/10.1145/3429881.3430109

[3] M. Kumar, G. Kaur, and P. S. Rana, "Performance, portability, productivity, and security in hpc cloud: a systematic literature review," *The Journal of Supercomputing*, vol. 81, no. 11, p. 1197, Jul 2025. [Online]. Available: https://doi.org/10.1007/s11227-025-07685-x

[4] T. Hoefler, T. Schneider, and A. Lumsdaine, "Loggopsim: simulating large-scale applications in the loggops model," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 597–604. [Online]. Available: https://doi.org/10.1145/1851476.1851564

[5] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating mpi applications: The smpi approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2387–2400, 2017.

[6] J. McDonald, M. Horzela, F. Suter, and H. Casanova, "Automated calibration of parallel and distributed computing simulators: A case study," in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2024, pp. 1026–1035.

[7] M. Noeth, P. Ratn, F. Mueller, M. Schulz, and B. R. de Supinski, "Scalatrace: Scalable compression and replay of communication traces for high-performance computing," *Journal of Parallel and Distributed Computing*, vol. 69, no. 8, pp. 696–710, 2009, best Paper Awards: 21st International Parallel and Distributed Processing Symposium (IPDPS 2007). [Online]. Available: https://www.sciencedirect.com/science/article/pii/S074373150800169X

[8] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Lowering entry barriers to developing custom simulators of distributed applications and platforms with SimGrid," *Parallel Computing*, vol. 123, p. 103125, 2025.

[9] S. W. Poole, O. Hernandez, J. A. Kuehn, G. M. Shipman, A. Curtis, and K. Feind, "OpenSHMEM-toward a unified RMA model," in *Encyclopedia of Parallel Computing*. Springer, 2011, pp. 1379–1391.

[10] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti *et al.*, "The gem5 simulator," *ACM SIGARCH computer architecture news*, vol. 39, no. 2, pp. 1–7, 2011.

[11] G. F. Riley and T. R. Henderson, "The ns-3 network simulator," in *Modeling and tools for network simulation*. Springer, 2010, pp. 15–34.

[12] R. Buyya and M. Murshed, "Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing," *Concurrency and computation: practice and experience*, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[13] A. Legrand and P. Velho, "Accuracy study and improvement of network simulation in the simgrid framework," *The ReScience journal*, 2021.

[14] A. Degomme, A. Legrand, G. S. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI applications: the SMPI approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 8, pp. 2387–2400, 2017.

[15] V. Honoré, T. M. A. Do, L. Pottier, R. F. Da Silva, E. Deelman, and F. Suter, "SIM-SITU: A framework for the faithful simulation of in situ processing," in *2022 IEEE 18th International Conference on e-Science (e-Science)*. IEEE, 2022, pp. 182–191.

[16] J. Emmanuel, "A full stack simulator for HPC: multi-level modelling of the BXI interconnect to predict the performance of MPI applications," Ph.D. dissertation, Université Claude Bernard-Lyon I, 2023.

[17] R. Keller Tesser, L. Mello Schnorr, A. Legrand, F. Dupros, and P. Olivier Alexandre Navaux, "Using simulation to evaluate and tune the performance of dynamic load balancing of an over-decomposed geophysics application," in *European Conference on Parallel Processing*. Springer, 2017, pp. 192–205.

[18] L. L. Nesi, L. M. Schnorr, and A. Legrand, "Multi-phase task-based hpc applications: Quickly learning how to run fast," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 357–367.

[19] A. Gougeon, B. Camus, F. Lemercier, M. Quinson, A. Blavette, and A.-C. Orgerie, "Co-simulation of power systems and computing systems using the FMI standard," in *2021 IFIP/IEEE international symposium on integrated network management (IM)*. IEEE, 2021, pp. 730–731.

[20] F. C. Heinrich, T. Cornebize, A. Degomme, A. Legrand, A. Carpen-Amarie, S. Hunold, A.-C. Orgerie, and M. Quinson, "Predicting the energy-consumption of MPI applications at scale using only a single node," in *2017 IEEE international conference on cluster computing (CLUSTER)*. IEEE, 2017, pp. 92–102.

[21] C. L. Janssen, H. Adalsteinsson, S. Cranford, J. P. Kenny, A. Pinar, D. A. Evensky, and J. Mayo, "A simulator for large-scale parallel architectures," *International Journal of Parallel and Distributed Systems*, vol. 1, no. 2, pp. 57–73, 2010, http://dx.doi.org/10.4018/jdst.2010040104.

[22] C. Dragoi, S. Nagendra, and M. Srivas, "A domain specific language for testing distributed protocol implementations," in *Networked Systems*, A. Castañeda, C. Enea, and N. Gupta, Eds. Cham: Springer Nature Switzerland, 2024, pp. 100–117.

[23] W. W. Carlson, J. M. Draper, D. E. Culler, K. Yelick, E. Brooks, and K. Warren, "Introduction to UPC and language specification," Technical Report CCS-TR-99-157, IDA Center for Computing Sciences, Tech. Rep., 1999.

[24] Y. Zheng, A. Kamil, M. B. Driscoll, H. Shan, and K. Yelick, "UPC++: a PGAS extension for C++," in *2014 IEEE 28th international parallel and distributed processing symposium*. IEEE, 2014, pp. 1105–1114.

[25] D. Bonachea and J. Duell, "Problems with using MPI 1.1 and 2.0 as compilation targets for parallel language implementations," *International Journal of High Performance Computing and Networking*, vol. 1, no. 1-3, pp. 91–99, 2004.

[26] M. Si, H. Fu, J. R. Hammond, and P. Balaji, "OpenSHMEM over MPI as a performance contender: Thorough analysis and optimizations," in *Workshop on OpenSHMEM and Related Technologies*. Springer, 2021, pp. 39–60.

[27] A. J. Wallcraft, "A comparison of several scalable programming models," Tech. Rep., 1998.

[28] A. Bouteiller, T. Herault, and G. Bosilca, "A multithreaded communication substrate for OpenSHMEM," in *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models*, 2014, pp. 1–2.

[29] P. Shamis, M. G. Venkata, S. Poole, A. Welch, and T. Curtis, "Designing a high performance OpenSHMEM implementation using universal common communication substrate as a communication middleware," in *Workshop on OpenSHMEM and Related Technologies*. Springer, 2014, pp. 1–13.

[30] N. Papadopoulou, L. Oden, and P. Balaji, "A performance study of UCX over InfiniBand," in *2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 2017, pp. 345–354.

[31] P. Shamis, M. G. Venkata, M. G. Lopez, M. B. Baker, O. Hernandez, Y. Itigin, M. Dubman, G. Shainer, R. L. Graham, L. Liss *et al.*, "UCX: an open source framework for HPC network APIs and beyond," in *2015 IEEE 23rd Annual Symposium on High-Performance Interconnects*. IEEE, 2015, pp. 40–43.

[32] Sandia OpenSHMEM (SOS. [Online]. Available: https://github.com/Sandia-OpenSHMEM/SOS

[33] M. Li, K. Hamidouche, X. Lu, J. Zhang, J. Lin, and D. K. Panda, "High performance OpenSHMEM strided communication support with infiniband UMR," in *2015 IEEE 22nd International Conference on High Performance Computing (HiPC)*. IEEE, 2015, pp. 244–253.

[34] OpenMPI. [Online]. Available: https://www.open-mpi.org/

[35] L. Binosi, G. Barzasi, M. Carminati, S. Zanero, and M. Polino, "The illusion of randomness: An empirical analysis of address space layout randomization implementations," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 1360–1374. [Online]. Available: https://doi.org/10.1145/3658644.3690239

[36] L. M. Schnorr, A. Legrand, and J.-M. Vincent, "Interactive analysis of large distributed systems with scalable topology-based visualization," in *2013 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 2013, pp. 64–73.

[37] L. L. Nesi, L. M. Schnorr, and A. Legrand, "Communication-aware load balancing of the LU factorization over heterogeneous clusters," in *2020 IEEE 26th International Conference on Parallel and Distributed Systems (ICPADS)*. IEEE, 2020, pp. 54–63.

[38] S. Pophale, R. Nanjegowda, T. Curtis, B. Chapman, H. Jin, S. Poole, and J. Kuehn, "Openshmem performance and potential: A npb experimental study," in *Proceedings of the 6th Conference on Partitioned Global Address Space Programming Models (PGAS'12)*. Citeseer, 2012.

[39] A. Bouteiller, S. Pophale, S. Boehm, M. B. Baker, and M. G. Venkata, "Evaluating Contexts in OpenSHMEM-X Reference Implementation," in *OpenSHMEM and Related Technologies. Big Compute and Big Data Convergence*, M. Gorentla Venkata, N. Imam, and S. Pophale, Eds. Cham: Springer International Publishing, 2018, vol. 10679, pp. 50–62, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-319-73814-7_4

[40] G. Cooperman and M. Quinson, "Sthread: In-Vivo Model Checking of Multithreaded Programs," *The Art, Science, and Engineering of Programming*, 2020. [Online]. Available: https://inria.hal.science/hal-02449080

[41] K. Seager, S.-E. Choi, J. Dinan, H. Pritchard, and S. Sur, "Design and Implementation of OpenSHMEM Using OFI on the Aries Interconnect," in *OpenSHMEM and Related Technologies. Enhancing OpenSHMEM for Hybrid Environments*, M. Gorentla Venkata, N. Imam, S. Pophale, and T. M. Mintz, Eds. Cham: Springer International Publishing, 2016, pp. 97–113.

[42] S. Bhattacharya, S. Salman, M. Gorentla Venkata, H. Kundnani, N. Imam, and W. Yu, "An Initial Implementation of Libfabric Conduit for OpenSHMEM-X," in *OpenSHMEM and Related Technologies. OpenSHMEM in the Era of Extreme Heterogeneity*, S. Pophale, N. Imam, F. Aderholdt, and M. Gorentla Venkata, Eds. Cham: Springer International Publishing, 2019, vol. 11283, pp. 56–69, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/978-3-030-04918-8_4