S4BXI: the MPI-ready Portals 4 Simulator

Julien Emmanuel, Matthieu Moy, Ludovic Henrio, Grégoire Pichon

S4BXI: the MPI-ready Portals 4 Simulator

Julien Emmanuel*†, Matthieu Moy*, Ludovic Henrio*, Grégoire Pichon†

* Univ Lyon, EnsL, UCBL, CNRS, Inria, LIP, F-69342, LYON Cedex 07, France
first.last@ens-lyon.fr

† Atos, Échirolles, France
first.last@atos.net

***Abstract*—We present a simulator for High Performance Computing (HPC) interconnection networks. It models Portals 4, a standard low-level API for communication, and it allows running unmodified applications that use higher-level network APIs such as the Message Passing Interface (MPI). It is based on SimGrid, a framework used to build single-threaded simulators based on a cooperative actor model. Unlike existing tools like SMPI, we rely on an actual MPI implementation, hence our simulation takes into account MPI's implementation details in the performance. This paper also presents a case study using the BullSequana eXascale Interconnect (BXI) made by Atos, which highlights how such a simulator can help design space exploration (DSE) for new interconnects.**

Keywords*—*Simulation, SimGrid, HPC, MPI, Interconnect, Portals 4, BXI

I. Introduction

While High Performance Computing (HPC) clusters are often used to run models of real-world objects in a lot of scientific fields, these systems are themselves so complex that there is valuable information to be learned by simulating them. In particular, having models of performance for HPC hardware enables performing some optimizations in the code or the configuration of the applications using simulation, in order to make the execution on a real cluster as fast as possible. This is important since clusters are limited resources that are expensive to use, both from an economic and ecologic point of view, so performing many test runs on real machines is often problematic. Another important use of simulation results of HPC systems is for the co-design of new hardware: since it is not realistic to create many physical prototypes at every stage of the development of a new chip, the best way to make design decisions is to evaluate new ideas in simulation.

Many simulators already exist for HPC systems (we present some of them in Section II), especially for network interconnects which allow the machines to communicate. This is especially important since clusters keep getting larger, with many servers (nodes) connected together, and a big amount of time is spent in communications. To get an accurate timing for communication, we need to model both low-level network primitives (typically hardware-accelerated) and algorithms used to implement high-level communication primitives. Our simulator, S4BXI[1], models the Portals 4 low-level API, a standard network API specified by Sandia and entirely implemented in hardware by the BXI interconnect developed by Atos [1], used in some of the fastest European supercomputers [2]. S4BXI allows running applications using the Message Passing Interface (MPI) standard API, that offers in particular a wide range of collective operations while Portals 4 only offers point-to-point semantics. We do not require any modification to the application, and take into account the implementation details of the Atos implementation of MPI running on top of BXI.

The closest related works are [3], which provides a model of Portals 4 as implemented in BXI, and SMPI [4]. [3] allows simulating unmodified applications using the Portals API directly, but does not allow running MPI applications. Typical HPC applications call MPI primitives, and rely on this very widely used library to call the lower-level Portals primitives, hence this is an important limitation to run real-life applications on the simulator. SMPI allows running unmodified applications using MPI. It does so by catching calls to MPI primitives and abstracts away the details of the hardware, leading to a lack of accuracy for some workloads. Also, it only allows a choice between a fixed set of MPI models corresponding to particular MPI implementations. Our work is based on [3], and follows a path different from SMPI to offer an MPI model. Instead of providing MPI *models* running on top of the SimGrid [5] network model, we allow a specific real-world MPI implementation to run directly on top of a more precise Portals 4 model, itself relying on SimGrid. We achieved this by providing a patch for OpenMPI to make it compatible with our simulator. The patch is relatively small because the API of the portals simulator we designed mimics faithfully the interface of the real network interconnect. We also adapted SMPI's performance model for computation to be usable in our simulator, which allows us to get accurate timing when modeling realistic applications.

After presenting related work in Section II, we present the details of how we implemented MPI support in Section III, and the results of our validation experiments in Section IV. Finally, Section V presents some ongoing experiments conducted with our simulators to help the design of next generations of hardware for the BXI interconnect.

II. Related Work

Network models for HPC can generally be categorized in three groups: packet-level models, flow models, and analytical

[1]Open-source under LGPLv2 licence, https://s4bxi.julien-emmanuel.com

models, which all have their own purpose.

Packet-level models are usually close to emulating the hardware, as they model the processing of each individual packet. While this makes for a very accurate model, it is usually very slow to run, which means that it is very impractical, if at all possible, to simulate several machines running a realistic application. These simulators are well suited to help the design of specific circuits when creating new networking hardware, and they are usually written in frameworks like SystemC [6], Omnet++ [7], ns3 [8], or the Structural Simulation Toolkit (SST) [9].

At the other end of the spectrum, analytical models enable very fast simulations by using very abstract methods for timing network transfers. This allows them to model thousands of processes very quickly, while sacrificing a lot of accuracy. Indeed these models often ignore important aspects of the network, which have a huge impact on performance, such as congestion. For this reason, they are more suited to simulate very large scale applications in order to get a very quick but rough estimate of their performance. They include all the LogP family of models, such as LogGOPSim [10] for example. Another use of these types of models is worst-case timing analysis, which is enabled by techniques such as Network Calculus [11].

The last category of simulators tries to find a middle ground between extreme precision and very good speed, by using a flow model to represent resource sharing: this enables building simulators at message-level, with different messages sharing the bandwidth of the links that they go trough. This way a wide spectrum of effects can be taken into account, depending on the purpose of the simulator. In this category, the simulation framework SimGrid [5] and its MPI simulator SMPI [4] are widely used. While SMPI has been shown to give very good results when modeling real-world applications on large-scale clusters, it has a few issues:

1. Since SMPI is a full re-implementation of the MPI standard, it only gives good results when modeling an MPI variant that is supported. While this isn't a problem when using an MPI implementation already available in SMPI (such as OpenMPI or MPICH), it cannot model faithfully an MPI version that has been tuned to use different collective algorithms for example.

2. Similarly, since function calls are intercepted by the simulator at the MPI level, SMPI doesn't allow any fine tuning of the MPI implementation itself, neither in its code nor in most of the numerous parameters that MPI typically has. Therefore, while SMPI is a good tool for developers of MPI applications, it isn't usable by developers who work on optimizing MPI itself, or by advanced MPI users which might want to fine-tune their implementation's parameters. It isn't usable to explore any lower-level changes either, such as different hardware behaviors, and therefore it can't be used to simulate interconnection networks that are not available.

3. While SMPI models completely the network topology of the interconnect, the intra-node communications are modeled in a very simplistic way. In particular, transfers between the memory and the the network controller (NIC) can only be accounted for using multiplying factors applied to the latencies or bandwidths of network links. This isn't a problem when modeling a relatively slow interconnect, because intra-node transfers' overhead is then very small, and modeling them as a small latency is a good heuristic. On the other hand, on HPC clusters this model isn't as good. Indeed most NICs use a PCIe network to communicate with the host memory, and the inter-node cables have speeds of the same order of magnitude as PCIe (usually somewhere between 100 and 200 Gbits/s). Therefore having a more detailed model of the PCIe network is better for the accuracy of the simulator, and enables studying various effects (such as congestion) on this network too.

4. Because SMPI's model is quite simple, a complex tuning procedure is mandatory to find bandwidth and latency factors that should be applied for different message sizes. Thankfully most of this is automated in various scripts [12], but it is still an additional step that is required to get realistic results, and it is not entirely trivial either.

While most of these characteristics contribute to making SMPI more performant, it leaves some space for new simulators that provide a lower-level model, and therefore a better accuracy (at the cost of some performance).

Even though network communications are important when simulating an HPC application, it isn't possible to get a good performance estimate without a model for computation phases too. The approaches can be categorized similarly to network models: some simulators go for cycle-accurate models, which emulate very precisely (but slowly) the hardware. These are usually made with SystemC [6] or gem5 [13] for example. On the other hand, some simulators have faster but more approximate ways of timing computations: for example, by default SMPI benchmarks the time between network operations as the application gets executed on the host machine (that is running the simulation), and the measured time is then injected into the simulated world (potentially with a constant multiplying factor). Another option available in SMPI is to disable this automatic benchmarking, and to manually describe computation time in the application's code itself, which has the downside that the application no longer runs unmodified in simulation.

S4BXI's approach: SMPI's model of computations is sufficient for our needs, and we reuse it with very little modifications. On the other hand, our network model is different: instead of reimplementing high-level MPI primitives in the model like SMPI, we rely on a real-world MPI implementation, which uses our simulated Portals implementation as a transport, solving points **1.** and **2.**. Thus, for communication operations, our simulator will run the same code as machines on the real-world clusters, as shown in Figure 1. To address point **3.**, our simulator includes a simplified model of the PCI network inside each machine [3]. Finally, our simulator still requires some configuration (in particular the bandwidth and latency of PCI and inter-node links), but it is easier to setup than SMPI, since it doesn't rely as extensively on empirical coefficient to adapt the speed of operations (point **4.**). As a

result, our simulator is more specific than SMPI because it specifically targets interconnects compliant with the Portals API, but it is more accurate. It is also slower, but still faster than most packet-level simulators.

III. MPI SIMULATION

To model MPI applications on top of our simulator we used Atos' version of OpenMPI, which adds an optimized transport (Byte Transfer Layer, abbreviated BTL) for the BXI interconnect to the community version of OpenMPI. This is the version that is traditionally used in production on BXI clusters. As shown in Figure 1, the S4BXI's workflow is similar to the real-world execution. The main advantage, additionally to the accuracy of the simulation, is that the solution can easily be adapted to a new implementation of MPI, to a new communication library (preliminary experiments on OpenSHMEM [14] gave promising results), or to a new hardware design of the network transport layer (see experiments in SectionV). As shown in the figure, S4BXI required small adaptations of the OpenMPI library. We present these adaptations below.

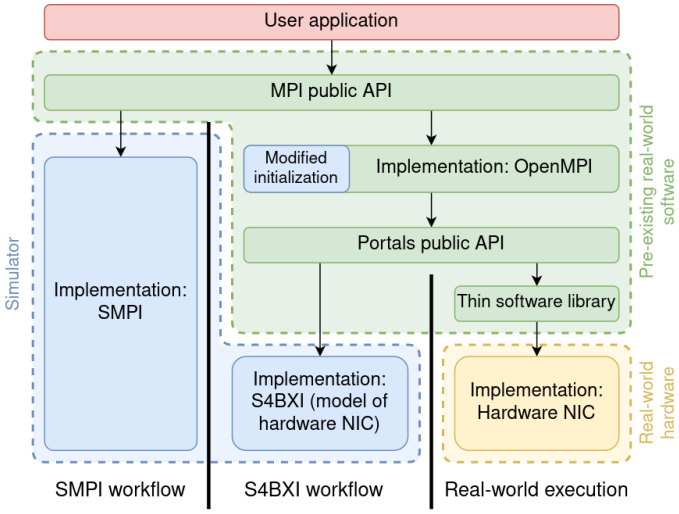

Fig. 1. High level view of simulations workflows and real-world execution

Initializing the MPI library for simulation: Although S4BXI is able to run classical applications written using Portals unmodified, allowing a library such as MPI to run on top of it is still challenging. The main reason for this is that during its initialization, OpenMPI exchanges meta-data about the different processes in the job through the Process Management Interface (PMI). These communications don't use Portals (since they are used to setup the Portals interface), and they usually occur across an out-of-band network, which can be Ethernet for example. While S4BXI provides a Portals implementation, it doesn't have a PMI implementation compatible with the simulated world, nor a model for the out-of-band network. Even though making an implementation of PMI in simulation seems feasible, it would be very time-consuming, which is why we directly modified OpenMPI's code to remove PMI calls, and instead inject values directly from the simulator. This data mainly includes the rank associated with each process, and other identifiers that processes need to communicate with each other (Network IDentifier and Process IDentifier in the case of Portals). These PMI calls are required in the execution on a real-world cluster, and they influence the duration of MPI's initialization, but on realistic applications the timing of this phase is negligible compared to the actual computations performed by the application.

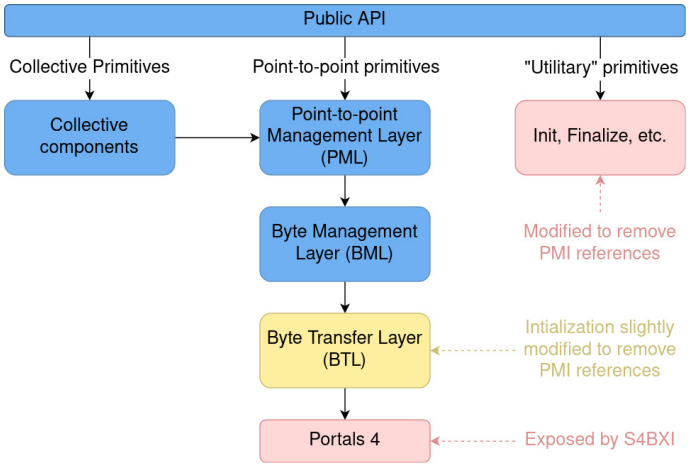

Fig. 2. Simplified view of relevant MPI components

Modifying OpenMPI components: A simplified view of OpenMPI's architecture and our modifications is depicted in Figure 2. Because MPI is a very large library with many components, this representation is very simplified, and many components that are not commonly used with Atos' version are not depicted. The darker boxes are components that run completely unmodified, and lighter ones are those which have been modified to some extent (or that are completely implemented by S4BXI). These modifications are really minor: in total around 400 lines of code were modified, although OpenMPI is composed of several hundred thousand lines of C code. Also, for the vast majority of them they only change the initialization of MPI, which means that they are very easy to maintain, in particular as other developers work on the components that are truly interesting to study: the Collective components, the Point-to-point Management Layer (abbreviated PML) and the BTL. This was confirmed as rebasing our modifications on top of the work of the MPI team at Atos was always trivial and without conflict, for several months now.

Potential optimizations and experimental setup: Our simulation is slower compared to SMPI, this is partially because we have an accurate simulation of the network layer. Our approach consists in runing the MPI library modified only where necessary, we thus decide not to try to gain performance by modifying this library. However, we can still obtain different trade-offs between accuracy and performance by modifying the network transport layer. For that purpose, several options exist in S4BXI to simplify the model. These options include "quick acknowledgements (ACK)", which makes ACK events instantaneous instead of requiring a very small transfer on

the network, and also several levels of detail for the PCI model (to ignore small commands for example). Using these options we can improve the performance of the simulator of up to 30%, while loosing some accuracy but remaining on average significantly more accurate than SMPI. These different trade-offs need to be further investigated and next section will present our experimental validation without using the existing shortcuts (in other words we focus on the gain in accuracy in our experiments).

The fact that all OpenMPI primitives are able to run on top of our simulator with such minor changes demonstrates the versatility of S4BXI, and shows that it is possible to model APIs that have a Portals transport with a relatively small effort.

IV. Experimental Validation

Since our low-level Portals model has already been validated for point-to-point operations in [3], our experiments focus on collective operations using MPI. We show here experimental results, first in Section IV-A on OSU Micro-Benchmarks [15], and then in Section IV-B on a realistic application, LULESH [16]. In our experience we will evaluate the accuracy of our simulator and the duration of the simulation, we will also compare our simulator to SMPI (the closest existing simulator). We couldn't perform a comparison with a packet-level simulator since none exist for the BXI interconnect.

A. OSU Benchmarks

OSU Micro-Benchmarks are a complete suite of tests that are used to evaluate the performance of individual MPI primitives. Because there are a lot of collective operations (a few dozens), we do not present every single one of them, but we instead synthesize our results into a few different categories[2].

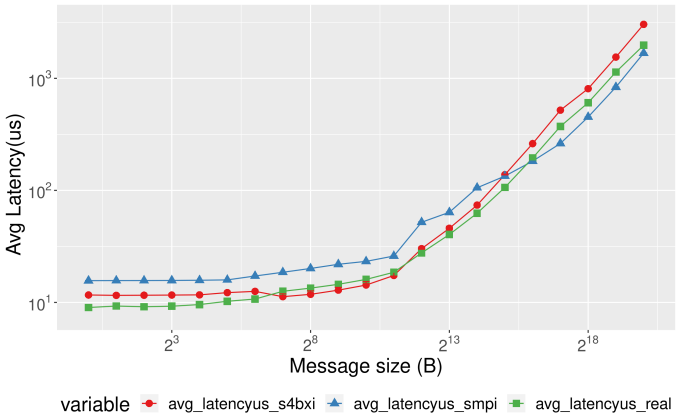

Fig. 3. AllgatherV collective on 16 nodes

Each graph that we present shows our simulation results in terms of simulated timing, as well as a comparison with the same benchmark executed on a real-world cluster equipped with BXI v2 hardware and AMD EPYC™ 7763 64-Core processors. Even though our simulator supports running several processes on each simulated node, we focus on simulations with one process per node only, to maximize the usage of the BXI network and have a better estimate of how our network model performs. Experiments were executed five times each (for both simulations and the real-world comparison), and error bars show the minimal and maximal values obtained in each case (although the difference is often so small that they merge with the median data point).

The graphs also show the comparison with the same benchmark simulated in SMPI, which has been calibrated using tools described earlier (point **4.** in Section II).

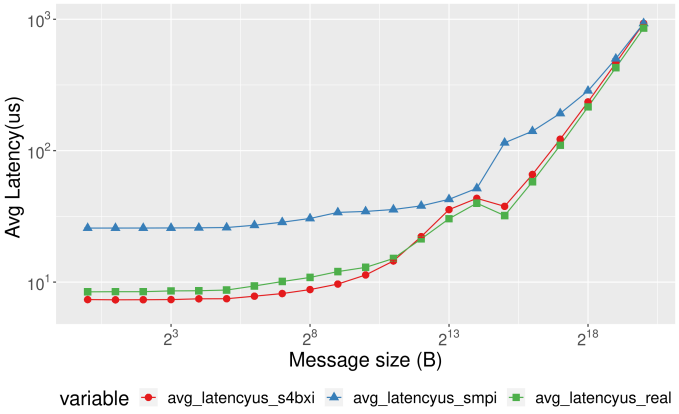

Fig. 4. Scatter collective on 16 nodes

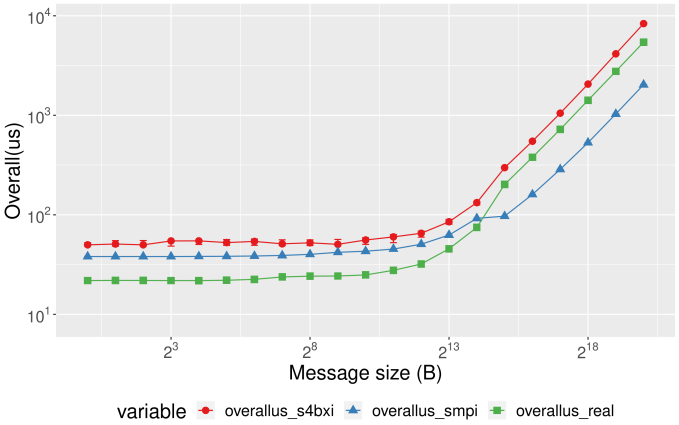

Fig. 5. IAllgather collective on 16 nodes

The first type of results that we obtain is the favorable case for both S4BXI and SMPI: when the algorithm for a collective operation matches what is used in Atos' variant, both simulators manage to get a very good estimate of the performance for the collective operation. This can happen for operations which don't have a lot of different algorithms (for example if it relies heavily on a root process with a behavior different from the others), or if by chance the default OpenMPI's algorithm is optimal on the BXI interconnect for the considered number of processes. An example can be seen when running the *AllgatherV* benchmark on 16 machines, which is showed on Figure 3. *AllgatherV* allows all processes to send non-contiguous chunks from a local buffer to all the other processes in the job.

[2]Data and graphs for all benchmarks are available at https://framagit.org/s4bxi/s4bxi-mpi-paper-data

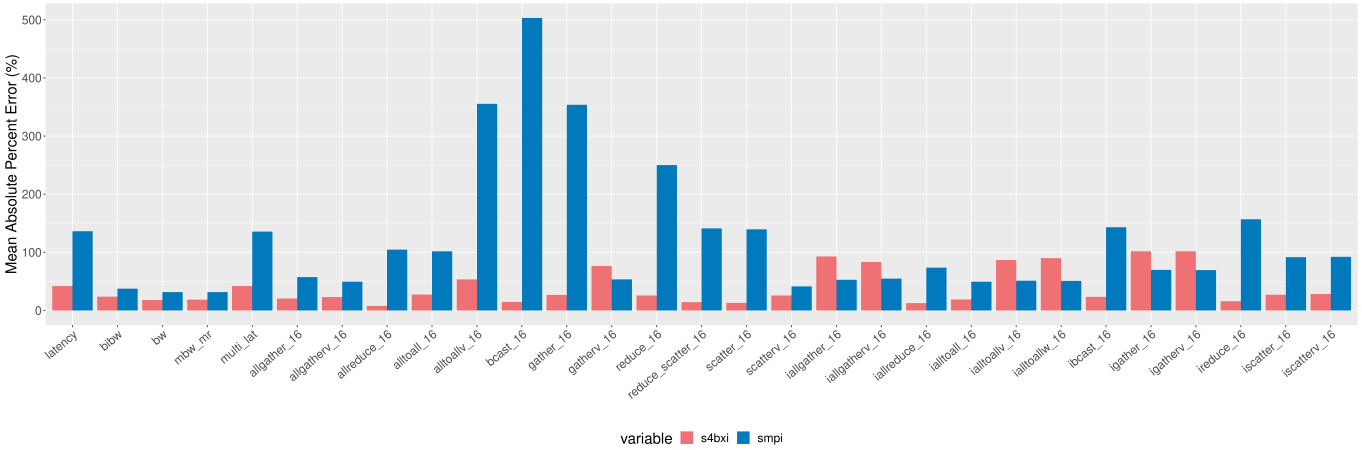

Fig. 6. Mean relative error for all OSU benchmarks: S4BXI and SMPI

In this case SMPI seems like a better solution than S4BXI, since it simulates faster and has a lower memory footprint.

On the other hand, when the algorithm used for a collective operation has been finely tuned by Atos, and results in different choices than OpenMPI's community version, SMPI would model a different algorithm than the one used in our implementation, at least for some message sizes and node numbers. On the other hand, S4BXI models the correct algorithm by design, since the complete real-world MPI implementation runs in simulation. This leads to a better accuracy of S4BXI as can be seen in Figure 4, which shows a *Scatter* benchmark on 16 machines. *Scatter* spreads data from a root process to every other processes in the job. For this operation, OpenMPI uses two algorithms: *basic*, which is the simplest implementation where the root node sends a message to every other process in the job, and *binomial tree*, where data is passed along a tree structure, which causes bigger data transfers but lowers congestion. Experiments show that for small job sizes, the default choice of using mostly *basic* is not optimal on BXI, and that the *binomial tree* algorithm is optimal for most message sizes. SMPI does not properly model the switch from *basic* to *binomial tree* implemented in Atos' version of OpenMPI.

Unfortunately there is a last category of benchmarks where both simulators struggle to give accurate results, which corresponds to asynchronous collective operations (benchmarks which start with an "i"). In this case S4BXI gives better results than SMPI in the majority of benchmarks, in particular thanks to the algorithm selection explained previously, but the accuracy of both simulators is not as good as for synchronous collective operations, as can be seen on Figure 5, which shows the *IAllgather* benchmark running on 16 machines. *IAllgather* allows all processes to send a piece of data to all the other processes in the job in an asynchronous way.

Thankfully, these operations seem more rarely used than synchronous ones, and we haven't come across realistic applications that make a heavy use of them. In particular, the application that we study in the next section, LULESH, doesn't use them at all.

Finally, the relative error of S4BXI and SMPI is depicted for all benchmarks on Figure 6 (for asynchronous collective operations the value is the average of the *compute* error and the *communication* error, since these benchmarks measure both, as well as the overlap between communication and computation). We can see that for the most common MPI operations S4BXI performs better than SMPI, and that the asynchronous collectives are indeed the worst case for us, as S4BXI produces approximately the same error as SMPI.

B. LULESH

While OSU micro-benchmarks are useful to evaluate individual primitives, they are not the most representative workload of a real-world application. We now present results on LULESH [16], an hydrodynamics proxy application commonly used in HPC. This application is an interesting case study, as it alternates between intensive computation phases and collective operations for data exchanges, which will stress both our network and computation models.

Our setup is as follows: we run the benchmark on a variable number of nodes (with one process per node), with different problem sizes (which has a big impact on the execution time of the application). We were able to run the real-world execution on two clusters: the system that we used for OSU benchmarks, equipped with AMD CPUs, as well as a cluster with more machines, equipped with Intel®'s Knight Landing CPUs (Xeon Phi™ 7250), which allows our results to go up to 27 nodes (a requirement of LULESH is that the number of processes must be the cube of an integer).

Our results are shown on Table I for the AMD cluster (including a comparison with SMPI), and on Table II for the Intel cluster (with no SMPI comparison because we don't have a proper calibration of SMPI for this cluster). No error is reported on the tables because the results are extremely consistent across multiple executions, and we always obtain the same values both in simulation and real-world executions (which might be in part due to the fact that LULESH only gives us two significant digits for performance).

Several conclusions can be drawn from this data. First, the executions on one node do not perform any communication,

TABLE I
TIMING ACCURACY OF SIMULATORS FOR LULESH (AMD CLUSTER)

Problem size	Nodes	S4BXI	SMPI	Real-world
10	1	0.13s	0.13s	0.15s
10	8	0.39s	0.37s	0.41s
20	1	3.5s	3.5s	3.5s
20	8	7.7s	8.4s	8.0s
30	1	19s	20s	18s
30	8	41s	44s	40s

TABLE II
TIMING ACCURACY OF S4BXI FOR LULESH (INTEL CLUSTER)

Problem size	Nodes	S4BXI	Real-world
10	1	1.6s	1.7s
10	8	4.8s	4.6s
10	27	12s	9.5s
20	1	41s	44s
20	8	91s	99s
20	27	150s	150s
30	1	240s	230s
30	8	490s	500s
30	27	720s	790s

and therefore we used them to calibrate the computation models for both simulators. For the smaller AMD cluster (Table I), we can see that both simulators are very accurate, with a small advantage for S4BXI (average relative error of 5%) over SMPI (average relative error of 8%). On the bigger cluster (Table II), we can also see that the accuracy of our simulator is very good for bigger problems (Problem size = 30), and for small problems when they are not too distributed (Problem size = 30, Nodes = 8), and it is still reasonable (around 25% longer in simulation than real-life) in unfavorable cases where a lot of small computations are distributed on the largest number of nodes (Problem size = 10, Nodes = 27). Our explanation for this is that when the application runs very fast (in less than ten seconds) on many machines, it puts a lot of stress on the network model, as there is little computation between calls to MPI collective operations. Therefore it makes sense that we get an error that is comparable with what we obtain for OSU benchmarks (which are designed to stress the network). Even though we will investigate this difference in the future, to provide a model as accurate as possible, the error is not too concerning: this type of workload is not very representative of a realistic use of HPC clusters (as such short executions would probably never run on many nodes).

Overall, the accuracy of our simulator on LULESH application is convincing: on realistic workloads we provide a very accurate estimation, and our simulator is always more accurate than SMPI.

C. A Word on Performance

We have shown that our simulator can accurately model a variety of workloads, but another important aspect of simulation is its performance. Indeed, as explained in Section II, one of the most important characteristics of a simulator is the trade-off between accuracy and performance.

In this regard, we observe different behaviors for network intensive applications like OSU benchmarks, and more realistic ones like LULESH: in the first case, our detailed Portals model is costly compared to other flow models like SMPI, both in terms of execution speed and memory usage of the simulation. On an Intel® Core™ i9-10850K CPU with 4 simulations running in parallel (on different CPU cores) we can simulate all 31 OSU benchmarks in 12 minutes with S4BXI, and about 3 minutes with SMPI. The detail for each benchmark is shown on Figure 7, where the slowdown factor of each simulator is depicted (relative to the real-world execution). We can see that S4BXI is significantly slower than SMPI, as expected, with similar simulation times in the best cases, and one order of magnitude slower in the worst cases. Regarding memory, S4BXI requires around 300MB of memory for each simulated process, regardless of the OSU benchmark, which means that on a very powerful machine with a lot of memory we can expect to be able to simulate up to a few hundred processes, but not thousands as SMPI has been shown to support [17]. To scale past this point in the future, it will be necessary to reduce MPI's memory consumption. Making the simulation distributed is not realistic because of the sequential nature of SimGrid.

TABLE III
PERFORMANCE OF SIMULATORS FOR LULESH (AMD CLUSTER)

Problem size	Nodes	S4BXI	SMPI	Real-world
10	1	0.25s	0.11s	0.15s
10	8	6.9s	3.2s	0.41s
20	1	3.1s	2.9s	3.5s
20	8	51s	45s	8.0s
30	1	16s	16s	18s
30	8	250s	250s	40s

The time needed to run simulations of LULESH is shown on Table III, where S4BXI is compared to SMPI and to the duration of the real-world experiment. For most realistic application, the biggest cost for performance is the execution of the computations: as the complete application runs in the simulator, the cost of modeling the network becomes negligible compared to intensive computation phases. Both S4BXI and SMPI run every simulated process sequentially (using SimGrid's scheduler), which means that the simulation performance (i.e. the "wall-clock" time duration) is approximately N times the execution time of one process in real life, where N is the number of simulated processes (assuming the CPUs on simulated machines are approximately as powerful as the CPU running the simulation, which can be tuned in the configuration of the computation model). Although S4BXI is slower for small problem sizes (because of the small number of computational phases), for bigger problems the differences in the network's model performance becomes negligible, and the value is the same for a problem size of 30. This shows that even though our network model is more complex and costly, the performance loss is only visible for highly network-intensive workloads, and that it is competitive with other simulators such as SMPI when modeling realistic applications with significant computational phase. In the SMPI community, studies have shown that this simulation performance can be

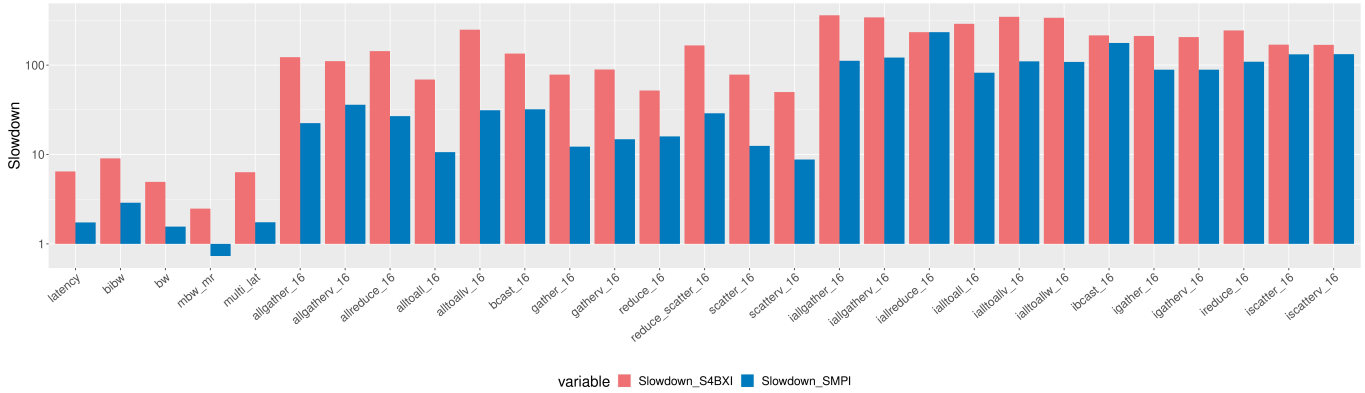

Fig. 7. Slowdown of simulation for OSU benchmarks (factor relative to real-world execution)

greatly optimized [12], but it requires both a deep knowledge of the modeled application, and the modification of the application code. Such analysis could be re-used in S4BXI, as we used the same model for computation as SMPI.

V. Co-Design of Next-Generation Hardware

Since Portals is implemented directly in hardware in the BXI interconnect, our simulator is a good tool to experiment with new potential designs for the next-generation Network Controllers (NIC). Because we use a flow model at message-level (and therefore have no representation of individual packets), it isn't a substitute for more detailed models that are typically built with SystemC for example, but it is complementary because it enables experiments on the processing at message level to be evaluated at a larger scale.

The study that we present is based on flow control, which is a feature that is lacking in the current generation of BXI hardware. There are many ways to implement flow control [18], so the approach we are studying is to limit the number of messages in-flight between each pairs of machines (node-level flow control), or each pair of processes (process-level flow control) at the sender side. We implemented this feature in simulation, which is tunable using environment variables at runtime to test various scenarios. This required little effort, as it represents only about 250 lines of modified code (the total size of the simulator is around 5500 lines of C++ code), which shows how much easier it is to implement message-level processing in our model than in a low-level emulator.

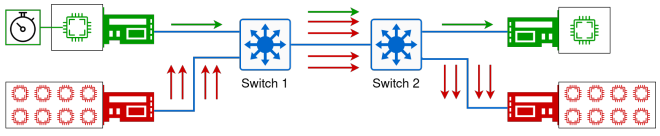

Fig. 8. Experimental setup

The experimental setup that we use to validate the behavior of our model is as depicted in Figure 8: our simulated platform consists of two switches connected together, with two machines connected to each. This way machines connected to switch 1 that communicate with machines connected to switch 2 will share a common BXI link. While this is a very simple topology, it is representative of real-world scenarios as pruning is very common in HPC clusters, which means that most of the time there will be shared links of this sort between switches (this is especially common in fat-tree topologies). The workload that we simulate is simple: a pair of machines runs eight processes each, which will flood the network by sending as many 1MB messages as possible to each other. While this isn't what a realistic application would do, it does emulate realistic situations where an application running on many nodes would have an intense communication phase. On the other hand, the remaining pair of machines simply runs one process each, which sends a fixed amount of 1MB messages sequentially. We measure the latency between the second pair of machines (top pair of Figure 8), which gives us an estimate of the congestion on the shared BXI link.

We run this experiment in two scenarios: node-level flow-control and process-level flow control. The result is shown on Figure 9, where the average latency of a message is represented as a function of the number of messages authorized inflight, and the horizontal dashed line represents the latency when flow control is completely disabled.

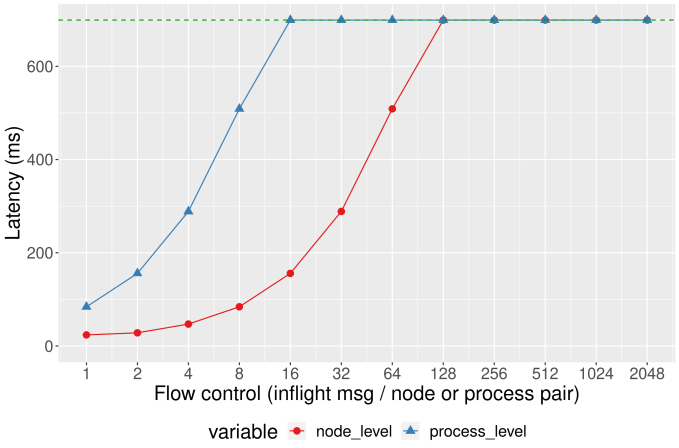

Fig. 9. Latency of messages when using flow control

We can see that the results are as expected: as the flow control gets more strict, the nodes flooding the network are slowed down, which allows the other node pair to exchange

messages with a far lower latency. When the flow control configurations reaches the maximum number of messages that nodes are able to send in parallel, the latency becomes constant as the control-flow has no effect. We can also see that small values of flow control have a greater impact at the node-level than process-level, which is also expected since there are eight processes on the machines that flood the network.

These results confirm that flow control can be a useful feature in the presented type of workload, and they help us quantify the decrease in congestion that we could expect if it was implemented in the future generation of interconnect. We also executed LULESH with different levels of flow-control, and we observed that this configuration has a negligible influence on the performance of the application. This shows that different workloads can be affected very differently by this feature. To go further, it is now necessary to use lower-level simulators in order to take the final decision and decide how to implement this feature.

VI. Conclusion and Future Work

We presented our approach for MPI simulation, using the low-level Portals model that S4BXI provides. We showed that it gives an accurate prediction for a variety of workloads, whether on micro-benchmarks or on more realistic applications. We also quantified the cost of this accuracy in terms of performance, and compared our results to a state-of-the art simulator, SMPI. Finally, we described a practical use of our simulator to study potential improvements in the next-generation BXI hardware, by implementing hardware-level flow control. This allowed us to quantify the benefits of using different algorithms, with a small effort in the adaptation of our simulator.

The next step to improve our simulation is to focus on the performance of our model while ensuring a good accuracy of the simulation. A first set of optimisations of the network transport layer are already available in S4BXI (see Section III). We plan to extend these options further, so that the performance of our lowest-precision model is as close as possible to SMPI's performance. We also need to investigate variable-precision simulation, where part of the network is modeled using a precise but performance-costly model, and the rest uses a faster but more abstract model, which could be especially usefull for workloads that heavily rely on a "root" process doing more work than the others.

Another strength of our simulator that hasn't been described in detail is its versatility: since it models a low-level communication API, it should be usable under any high-level network API that has a Portals transport. In particular, we are working on running OpenSHMEM [14] (a Partitioned Global Address Space library, which is used to transparently share memory across machines) on top of S4BXI. Most primitives already work with few efforts, so we should be able to get performance results soon using this workflow.

Acknowledgements: We would like to end by thanking the BXI low-level and MPI teams at Atos for their help in understanding Portals and OpenMPI, as well as the SimGrid team for their valuable help throughout the development of our simulator, especially Tom Cornebize for his help on the tuning of SMPI, and Arnaud Legrand for his advices.

References

[1] S. Derradji, T. Palfer-Sollier, J. P. Panziera, A. Poudes, and F. Wellenreiter, "The BXI Interconnect Architecture," *Proceedings - 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects, HOTI 2015*, pp. 18–25, 2015.

[2] (2020) Top500 list - november 2020. [Online]. Available: https://www.top500.org/lists/top500/list/2020/11/

[3] J. Emmanuel, M. Moy, L. Henrio, and G. Pichon, "Simulation of the Portals 4 protocol, and case study on the BXI interconnect," *Proceedings - 2020 International Conference on High Performance Computing & Simulation (HPCS)*, dec 2020. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02972297

[4] 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.

[5] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, scalable, and accurate simulation of distributed applications and platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899–2917, Jun. 2014. [Online]. Available: http://hal.inria.fr/hal-01017319

[6] *IEEE Standard for Standard SystemC ® Language Reference Manual IEEE Computer Society*, 2012, vol. 2011, no. January.

[7] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," 2008. [Online]. Available: https://doc.omnetpp.org/workshop2008/omnetpp40-paper.pdf

[8] G. F. Riley and T. R. Henderson, "The ns–3 Network Simulator," *Modeling and Tools for Network Simulation*, pp. 15–34, 2010.

[9] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "The structural simulation toolkit," *ACM SIGMETRICS Performance Evaluation Review*, vol. 38, no. 4, pp. 37–42, mar 2011. [Online]. Available: https://dl.acm.org/doi/10.1145/1964218.1964225

[10] T. Hoefler, T. Schneider, and A. Lumsdaine, "LogGOPSim - Simulating large-scale applications in the LogGOPS model," *HPDC 2010 - Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, pp. 597–604, 2010.

[11] J.-Y. Le Boudec and P. Thiran, *Network calculus: a theory of deterministic queuing systems for the internet*. Springer Science & Business Media, 2001, vol. 2050.

[12] T. Cornebize, "High performance computing: Towards better performance predictions and experiments," Ph.D. dissertation, Université de Grenoble, 2021. [Online]. Available: https://cornebize.net/thesis/build/thesis.pdf

[13] 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.

[14] B. Chapman, T. Curtis, S. Pophale, S. Poole, J. Kuehn, C. Koelbel, and L. Smith, "Introducing OpenSHMEM," in *Proceedings of the Fourth Conference on Partitioned Global Address Space Programming Model - PGAS '10*, no. October. New York, New York, USA: ACM Press, 2010, pp. 1–3. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2020373.2020375

[15] "Mvapich::benchmarks," accessed: 2021-06-10. [Online]. Available: http://mvapich.cse.ohio-state.edu/benchmarks/

[16] I. Karlin, "LULESH Programming Model and Performance Ports Overview," Lawrence Livermore National Laboratory (LLNL), Livermore, CA (United States), Tech. Rep. LLNL-TR-608824, dec 2012. [Online]. Available: https://asc.llnl.gov/sites/asc/files/2021-01/lulesh_ports1.pdf

[17] T. Cornebize, F. Heinrich, A. Legrand, and J. Vienne, "Emulating High Performance Linpack on a Commodity Server at the Scale of a Supercomputer," 2017.

[18] M. Gerla and L. Kleinrock, "Flow control: A comparative survey," *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 553–574, 1980.