SMPI Courseware: Teaching Distributed-Memory Computing with MPI in Simulation

Henri Casanova
Information and Computer Sciences Dept.
University of Hawai'i at Mānoa,
Honolulu, HI, U.S.A.
henric@hawaii.edu

Arnaud Legrand
Université Grenoble Alpes,
CNRS, Inria, INP, LIG
Grenoble, France
arnaud.legrand@imag.fr

Martin Quinson
IRISA laboratory, ENS Rennes,
Université de Rennes 1, Inria, CNRS
Rennes, France
martin.quinson@ens-rennes.fr

Frédéric Suter
IN2P3 Computing Center, CNRS
Inria, LIP, ENS Lyon
Lyon-Villeurbanne, France
frederic.suter@cc.in2p3.fr

***Abstract*—It is typical in High Performance Computing (HPC) courses to give students access to HPC platforms so that they can benefit from hands-on learning opportunities. Using such platforms, however, comes with logistical and pedagogical challenges. For instance, a logistical challenge is that access to representative platforms must be granted to students, which can be difficult for some institutions or course modalities; and a pedagogical challenge is that hands-on learning opportunities are constrained by the configurations of these platforms.**

A way to address these challenges is to instead *simulate* program executions on arbitrary HPC platform configurations. In this work we focus on simulation in the specific context of distributed-memory computing and MPI programming education. While using simulation in this context has been explored in previous works, our approach offers two crucial advantages. First, students write standard MPI programs and can both debug and analyze the performance of their programs in simulation mode. Second, large-scale executions can be simulated in short amounts of time on a single standard laptop computer. This is possible thanks to SMPI, an MPI simulator provided as part of SimGrid. After detailing the challenges involved when using HPC platforms for HPC education and providing background information about SMPI, we present SMPI Courseware. SMPI Courseware is a set of in-simulation assignments that can be incorporated into HPC courses to provide students with hands-on experience for distributed-memory computing and MPI programming learning objectives. We describe some these assignments, highlighting how simulation with SMPI enhances the student learning experience.

***Index Terms*—High Performance Computing Education, Parallel Computing Education, Message Passing Interface, Simulation**

I. Introduction

Many challenges are involved in teaching High Performance Computing (HPC) concepts and practices, as indicated by the establishment of workshop series focused on HPC education. Although part of the HPC curriculum can be taught at theoretical or conceptual levels, it is widely acknowledged that a large portion of it is taught more effectively if students are provided with hands-on experience opportunities. This is typically achieved by providing students with access to one or more HPC platforms for them to design, implement, execute, and analyze the execution of programs developed as part of lab activities and/or homework assignments. Unfortunately, using real-world platforms to teach HPC courses comes with *logistical challenges* as well as, perhaps counter-intuitively, *pedagogical challenges*. We review both types of challenges hereafter.

Logistical Challenges – The most obvious logistical challenges is that it is not always possible to obtain access to a representative HPC platform in the first place. For some institutions, it may be that no HPC platform is available whatsoever. In this case, one can resort to building low-cost HPC platforms for teaching purposes [1]–[6], to using virtualization and/or container technology to emulate an HPC platform, e.g., in cloud environments [7]–[10], or to gain access to public HPC testbeds. Although a lot can be achieved pedagogically on such platforms, they are not necessarily representative of production HPC platforms (in terms of capabilities, scale, and/or performance behaviors), which can be problematic for performance-oriented learning objectives. Even if an institution hosts HPC platforms, there may be no straightforward mechanism to use these platforms for education purposes. Throughout the careers of the authors of this paper, in their past and present institutions, although production HPC resources were available for research purposes, providing access to students in a course has often times proved more challenging than expected. In many cases, access can be secured, but only to a small subset of the platform, thus making it difficult for students to experience scaling effects. If access is secured to a large platform, either hosted in-house or as part of some community HPC testbed, this platform is typically used in production, and thus is shared among students and regular platform users. This often raises social issues in which regular platform users, e.g., researchers working toward conference deadlines, are then impacted by an influx of new (student) users, which can create tensions. Furthermore, these student users are novices, and can make mistakes that lead to resource waste. For instance, when teaching HPC courses, we have seen students asking for long batch allocations for a program that does not terminate, bringing down compute nodes due to creating too many processes, filling up local disks instead of writing files to a distributed file systems, etc. Although such mistakes can occur with any user, "unleashing" dozens of novices on a production platform at once tends to, unsurprisingly, increase the number of user complaints significantly. Even if a platform is dedicated to the students in a course, another logistical challenge is that students share this platform among themselves, which leads to possibly unhealthy competition (e.g., batch queue competition in the hours before an assignment due date). The above can result in non-trivial effort and time commitments by the course instructor (e.g., platform monitoring, interaction with platform administrators, managing user interactions), which could be better invested into course preparation or delivery.

Pedagogical Challenges – Even assuming that access to one or more representative HPC platforms is provided, using these platforms still raises challenges from a pedagogical standpoint. First, students must be trained to use the platform, which entails learning how to use various software infrastructures, the most complex of which is often a batch scheduler, as well as various usage policies. Some courses could deliberately devote a large portion of the syllabus to teaching the practice and the details of the software infrastructures used in typical HPC environments. But for other courses (e.g., for a course focused on parallel algorithms and MPI programming), the time and effort spent learning the software infrastructure and usage policies, which is necessary for providing students with hands-on experience, could likely be better spent on more course-relevant learning objectives.

Second, and more importantly, students are exposed to platform configurations that are specific and limited to those of the HPC platform(s) provided to them. Unless several (large) platforms are available, many interesting phenomena simply cannot be experienced hands-on by students. The instructor is then forced to mention what would happen in hypothetical, but relevant, scenarios that are simply not available to students. For instance, in our experience instructors often find themselves make statements such as "if the network topology were different, this algorithm would behave differently", "if the platform were heterogeneous, there would be many interesting issues with data/load distribution", etc. These statements, albeit valid, fall short of deepening student interest, and thus understanding, as several important concepts may never be experienced hands-on. Finally, from a logistical perspective, when the platform in use changes from one incarnation of the course to the next, there can be a large burden placed on the instructor to ensure that previous assignments can still help students achieve the required learning objectives.

Third, real-world platforms are known to be "noisy" [11]–[14]. The observed performance of a program is subject to many effects, some deterministic and some non-deterministic, which make performance difficult to understand without deep knowledge about hardware/software infrastructures and without solid experimental and analytical skills. This is problematic since most students initially lack such knowledge and skills. Therefore, in order for them to acquire knowledge about fundamental HPC concepts and practices, the possible sophistication and depth of course assignments, and thus the learning experience, are diminished. At the simplest level, it is difficult to observe small performance variations due to platform noise, and thus it is difficult to teach effectively those techniques that result in marginal, but still desirable, performance improvements (e.g., using asynchronous communication to overlap some amount of computation with communication). In our experience, in some cases platform noise can even make it difficult for an instructor to grade programming assignments fairly based on observing executions of student's programs.

Fourth, it is by running experiments that students can develop their own understanding of the material, e.g., by validating/invalidating their own intuitions. Yet, running experiments on an HPC platform is not free as it requires time, electrical power, and in some cases funds. The number of experiments that students can perform is thus typically limited, as are their opportunities to deepen their independent understanding of the subject matter.

Fifth, a popular way to reach students is to develop and make available Massive Open Online Courses (MOOCs), so that students can acquire skills from anywhere and at their own pace. Developing an HPC course as a MOOC that provides students with valuable hands-on experience would require that access to a representative HPC platform be guaranteed for a large, diverse, and distributed student population, which can be done effectively only by a few institutions.

One approach to overcome the above logistical and pedagogical challenges is to use *simulation*, i.e., instead of running programs on a few actual HPC platforms, simulate the execution of programs on a software realization of arbitrary HPC platform configurations. Using simulation for HPC education was pioneered in the early 1990's [15]–[17]. More recently, the work in [18] demonstrates the benefit of using simulation for teaching HPC, with student surveys showing increases in both satisfaction and acquired knowledge due to the use of simulation. In this work we describe *SMPI Courseware*, a set of assignments that can be incorporated into any HPC course. These assignments are structured as progressive steps with well-defined learning objectives. The scope is distributed-memory parallel computing with the Message Passing Interface (MPI). The foundational technology is SMPI [19], an MPI simulator provided as part of the SimGrid simulation framework [20], [21]. SMPI makes it possible to simulate accurately the execution of unmodified MPI applications on arbitrary platform configurations. Furthermore, simulations can be executed quickly on a standard laptop computer. The above is made possible thanks to the scalable and validated simulation models provided by SimGrid. SMPI Courseware thus does not require access to any HPC resource, and has been used already in graduate-level HPC courses as a complement to or completely replacing the use of HPC platforms.

In this work, we describe some of the assignments in SMPI Courseware, and showcase how simulation with SMPI makes it possible to achieve popular HPC learning objectives effectively. The rest of this paper is organized as follows. Section II discusses related work. Section III provides necessary background information on SMPI. Section IV describes some of SMPI Courseware assignments and how they capitalize on the simulation capabilities of SMPI. Finally, Section V concludes with a summary and future work directions.

II. Related Work

Several authors have addressed the logistical challenge of procuring an HPC platform for educational purposes by establishing low-cost such platforms [1]–[6] or by using virtualization and/or container technology [7]–[10]. Although these solutions offer compelling capabilities, such as exposing students to real-world software infrastructures, they impose limits on platform scale and/or may not be representative of production HPC platforms, especially in terms of performance behaviors. In this work, instead, we resort to simulation so as to be completely free of the need for a hardware HPC platform and can simulate arbitrary platform configurations. In what follows we review previous works that have also used simulation for the purpose of HPC education.

Simulation is used routinely as a pedagogic tool in many areas of the computer science curriculum. For instance, it is traditional to use simulation frameworks for teaching computer architecture and network concepts, since without simulation it can be extremely challenging to create hands-on experience opportunities in these domains. By comparison, the use of simulation as a pedagogic tool is relatively rare in the HPC domain, likely because in many cases it is possible to use some HPC platform for teaching purposes, albeit facing the logistical and pedagogical challenges identified in Section I. Several works in the early 1990's proposed using simulation for the purpose of parallel computing education. In [15] the authors argue that using a simulator via which students implement and execute parallel algorithms enhances the learning experience. A parallel algorithm is described as an abstract model, and a large part of the proposed approach is a graphical user interface with execution visualization capabilities (e.g., execution Gantt charts). A similar approach is proposed in [17], but in this approach programs are written in an actual programming language (Modula-P) and compiled to run in "simulation mode", which is also what we propose in this work. The author reports on increased student motivation and satisfaction. A textbook was developed that comes with a simulator and programming exercises in simulation [22]. In [16], it is proposed to teach parallel computing using a SIMD architecture simulator as one of the options for teaching parallel computing at low cost. In all the above, the authors motivate their work by some of the same rationales as those outlined in Section I, namely: (i) acquiring a platform may not be possible (due to cost); (ii) learning is limited because experimental scenarios are limited by the platform at hand; and (iii) simulation makes it possible to easily observe and understand deterministic program executions that are not subject to any platform "noise."

More recently, the authors in [18] describe the parallel computing module of a M.S. degree in HPC, which relies on simulation as a foundational technology. They report on improvements in student grades and self perception of acquired knowledge, attributing these improvements to the use of simulation. Although there is overlap between the authors' motivations and our own motivations as outlined in Section I, there are two significant differences between this work and that in [18]. First, the work in [18] is based on the OPNET Modeler (now known as Riverbed Modeler [23]) simulation framework. This framework uses packet-level network simulation, which simulates the behavior of individual network packets in a discrete event simulation fashion. Although typically considered accurate, packet-level simulation of full-fledged applications is time-consuming (in part because the number of packets increases with the communicated data size). This problem is acknowledged by the authors in [18], and is the reason why they provide students with a software tool to run simulations on an remote HPC platform. By contrast, our approach relies on the SimGrid simulation framework, which uses scalable "macroscopic" network simulation models and provide high accuracy and orders of magnitude simulation speed improvements over packet-level network simulators [24]. As a result, full-fledged simulations of large-scale HPC applications can be executed in a as little as a few seconds on a student laptop. In our approach there is thus no need for an actual HPC platform. The second significant difference is that while with the approach in [18] students work with parameterizable predefined simulation scenarios to understand performance issues, instead with our approach students write standard MPI code that is then executed in simulation mode transparently. Thus not only can students achieve the learning objectives in [18] (and can be provided with predefined simulation scenarios as already implemented MPI applications if so desired), but they can also learn how to use MPI and can debug their MPI code in simulation mode.

Another recent work, Paralab teachware, is presented in [25], which uses the Paralab system [26] to teach parallel computing concepts and algorithms using simulation. Paralab itself is available for Windows and, based on what can be learned in [26], implements idealized simulation models that may not be representative of real-world HPC platforms. For instance, network simulation seems to be based on a "latency plus data size divided by bandwidth" model. While simple models can have pedagogic value as they make understanding performance trends easier, these models can be far from the realities of HPC platforms. By contrast, in this work we use SMPI, which implements sophisticated validated simulation models that incorporate many features encountered in real-world platforms (e.g., protocol switching based on message size, network congestion). The Paralab teachware provides students with pre-packaged applications and a set of network topologies (with configurable parameters) for students to experience application performance for learning purposes. By contrast, this work allows students to write standard MPI programs and to simulate their executions on arbitrary platform configurations. Therefore, the scope of this work, and thus of the learning objectives that can be achieved, is larger. It would be straightforward to re-implement Paralab teachware entirely using our approach.

III. SMPI: Simulated MPI

SMPI [19], which comes with the SimGrid [20], [21] distribution, makes it possible to simulate the execution of MPI applications. SMPI works seamlessly with unmodified MPI programs written in C, C++, or FORTRAN. For instance, assuming the program is written in C, one simply compiles it with `smpicc`, instead of `mpicc`, and execute it with `smpirun`, instead of `mpirun`. The only difference between `smpirun` and `mpirun` is that `smpirun` takes one extra command-line argument, `-platform`. This argument is used to specify a description of the hardware platform on which the execution of the MPI program is to be simulated. This description is in XML and allows the specification of arbitrary platform configurations [27]. At the most basic level, the platform description specifies a network topology between hosts, where network links have specified latencies and bandwidths, and hosts have specified hostnames, numbers of cores, and per-core compute speeds. The XML description schema provides higher-level constructs (e.g., a homogeneous cluster with a single switch) so that these descriptions can be easily written by hand if so desired. The SimGrid distribution comes with many example XML platform descriptions. Given an MPI program, simulating and comparing its executions on different platforms is easily done by running `smpirun` multiple times, each time with a different XML platform description file as specified by the `-platform` command-line argument. Because SMPI simulations are deterministic, differences between simulated executions of the same MPI program, provided this program is itself deterministic, are guaranteed to be solely due to the differences in the platforms. However, non-deterministic programs are subject to the same issues when simulated with SMPI as when executed with MPI on real-world platforms (e.g., race conditions caused by non-deterministic message receptions from multiple senders due to using `MPI_ANY_SOURCE`).

The basic principle behind SMPI simulations is as follows. The code of the MPI program is executed as is, but the MPI processes actually execute as threads in a single process, and thus share the same address space. This is depicted in Figure 1. Each time an MPI API function is called, control is handed off to SMPI where network operations are replaced by simulated delays. These delays are computed using the simulation models at the core of SimGrid. Each block of code

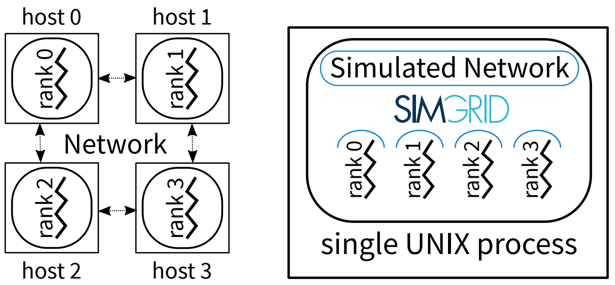

Fig. 1. Realization of a multi-process MPI execution (left) as a multi-threaded simulated execution with SMPI (right).

in between two MPI API calls is benchmarked on the machine used to execute the simulation. This is possible because, with SMPI, MPI processes execute as threads in mutual exclusion. The benchmarked execution times of each block of code can then be scaled and simulated as compute delays that correspond to the compute speeds of the hosts in the simulated platform. In this way, both communication and computation operations are simulated as computed delays by SMPI.

As described above, the SMPI simulation approach would have two severe drawbacks. First, the simulation of an MPI application with p processes in which each process computes for t seconds on machines of compute speed s, would run in time at least pts/s' on a machine with compute speed s'. Second, the simulation of an MPI application with p processes in which each process allocates m bytes of RAM would need to be executed on a machine with pm bytes of RAM. As a result, simulating long and/or large MPI application executions would be intractable.

To address the long execution time problem, SMPI can avoid executing the entire application code. While executing the entire code of an MPI application is required to obtain valid application output, the objective of most SMPI users is only to reproduce the performance behavior of the application. Once encountered again on the same or a different MPI process, the execution of a basic block of code can be simulated by SMPI without executing the code itself, but merely as a delay based on previously benchmarked executions of it. This is done by inserting code in the application at compile time. While the modified code likely produces incorrect application output, performance behavior can be preserved and simulation time is vastly reduced.

To address the large memory footprint problem SMPI makes it possible for MPI processes to share allocated memory regions (e.g., `malloc` may return a pointer to a memory region already allocated by another MPI process). Furthermore, SMPI implements a mechanism that allows the same memory page to be used multiple time as part of a single allocate zone or RAM. As a result, scenarios with large memory footprints can be simulated on a computer with small RAM capacity. Here again the application output is compromised but performance behavior can be preserved.

These two techniques, which are activated via straightforward and succinct code annotations, can only be viably applied to non-data-dependent applications, i.e., for applications in which branching does not depend on application data. In summary, SMPI makes it possible to simulate small-scale executions of any MPI application for validation/debugging purposes, but also possible to simulate large-scale executions of data-independent applications (at the expense of application output correctness) for performance/scalability analysis purposes, all on a single machine.

SMPI implements the MPI-2 standard and a portion of the MPI-3 standard. An important performance consideration when using an implementation of the MPI standard is the algorithms used for collective communication operations. This is because production MPI applications use these operations extensively, and their performance often determines overall parallel efficiency. The design space for collective communication algorithms is large, and thus many algorithms exist, exhibiting different performance characteristics depending on the network topology, the message size, and the number of communicating processes [28]. MPI implementations thus often select at runtime which algorithm to employ, which leads to relatively complex collective communication implementations. For instance, in OpenMPI, the `MPI_Allreduce` operation amounts to about 2,300 lines of code. It is necessary for an accurate MPI simulation framework to capture such complexity. To this end, SMPI implements all the collective algorithms and selection logic of both OpenMPI and MPICH [29] as well as a few other collective algorithms and selectors from Star-MPI [28], the Intel vendor implementation of MPI, and MVAPICH2 [30]. Overall, SMPI currently provides more than 120 different collective communication algorithms. Users can activate these implementations via command-line arguments passed to `smpirun`, which makes it straightforward to compare collective communication implementations for arbitrary application and platform scenarios.

To date, SMPI has been validated for representative applications and benchmarks [19], and used to simulate large-scale MPI application executions on a single machine [31]. SMPI has also been used to successfully simulate 51 unmodified MPI applications included in standard HPC application benchmarks/suites (ECP Proxy Applications, CORAL, Trinity, PRACE CodeVault, HPL, and Meteo-France), which are included as part of SMPI's nightly integration testing [32].

Given the above capabilities and accomplishments, SMPI provides a compelling foundation for addressing the challenges outlined in Section I in the context of distributed-memory parallel computing and MPI programming education.

IV. SMPI Courseware

A. Objectives and Availability

The HPC community has developed sets of learning objectives as well as entire curricula to meet these objectives. The goal of SMPI CourseWare is not to define or provide a curriculum. Rather, its goal is to provide ready-to-use assignments through which students can achieve learning objectives in the context of distributed-memory parallel computing and MPI programming. Capitalizing on the capabilities of SMPI (see Section III), these assignments can be integrated into an HPC course without any need for an HPC platform. These assignments can and have been used with several modalities. These include in-class sessions during which instructors and teaching assistants provide instructional scaffolding, students working in groups outside of the classroom, or students working individually outside of the classroom.

To date, SMPI Courseware is simply available as a Web site [33]. Instructors can either point their students to that site, or download and modify assignments as they wish. A first section of the site provides instructions on how to install and test SMPI. In the near future, this site will also provide pre-installed Docker containers with all necessary software and assignment files. Currently no mechanisms is available for retrieving assignment solutions, but they can be requested directly via e-mail to *henric@hawaii.edu*.

B. Sample Assignments and Usefulness of Simulation

In this section we showcase four of the assignments provided in SMPI Courseware. While we hope that these assignments have some pedagogic merit, our main focus here is to explain how, in the context of these assignments, simulation with SMPI enhances the student learning experience. In several cases, the use of simulation unlocks learning opportunities that would remain out of reach otherwise. We refer readers interested in the details of the assignments to the SMPI Courseware site [33]. In what follows we also report on our own experience using SMPI in our courses. No formal user studies have been conducted to date. But we believe that what we observed in the classroom, although for now anecdotal, is nevertheless enlightening.

1) Parallel Julia set computation: SMPI Courseware provides an introductory assignment in which students implement a parallel Julia set computation. The learning objectives include (i) to understand the principle of SPMD (Single Program Multiple Data) programming; (ii) to understand and implement the concept of data distribution in distributed memory programming; (iii) to understand and implement some process communication/synchronization feature; (iv) to achieve mastery of essential MPI calls.

The first (optional) step of the assignment is to implement a sequential program that generates a jpeg image of a Julia set with dimensions provided as command-line arguments. The rest of the assignment consists in parallelizing this program in a sequence of baby steps. Students are led to discover the concepts of partitioning the work across MPI ranks in an SPMD fashion.

In this assignment, students must understand and implement the notion of data distribution across MPI ranks. As expected, the assignment imposes the constraint that if the Julia set consists of n pixels, then each process in a p-process execution can allocate memory for no more than $\lceil n/p \rceil$ pixels. As a result, students are exposed to the (initially quite difficult) notions of local indices and global indices: computing a pixel in the Julia set is based on the pixel's coordinates in the global image, but these pixels must be stored in a local array using some local indexing scheme. Students first use a 1-D strip data distribution and then a 2-D block distribution. This is likely the most challenging learning objective in this assignment. In fact, the rationale for having the program compute a Julia set is that students can detect and understand data distribution bugs quickly by visually inspecting the output image. As part of this assignment, therefore, the students must generate the output image. This requires that students use some communication in their MPI program since pixels blocks must be written in the right order to a single file. Students are free to implement some token-passing scheme in which each process writes its "piece" of the image to the file, or have a master process do all the writing after receiving image pieces from other processes. This assignment places very little emphasis on performance (beyond common-sense "it goes faster with more processors" or "it goes faster with faster processors" observations), and students mostly focus on producing correct programs.

Even for this introductory assignment, the use of SMPI affords some pedagogical advantages. First, students can hit the deck running without having to learn all the logistics of using an HPC resource. Learning these logistics can be completely decoupled from learning parallel computing and MPI programming concepts (it can come later in the course if desired, or not be part of the course at all). Furthermore, even if acquired by the students, these logistics tend to lengthen the implement-test-debug cycle, which is particularly frustrating for an introductory assignment. For this particular assignment for instance, the image of the Julia file is created on the student's own laptop. It thus can be opened in some image viewer that automatically updates the image when it changes, i.e., each time the MPI program execution is simulated. This is to be contrasted with having students view the image on a remote HPC platform (often requires, e.g., multiple scp hops, ssh X forwarding). In our experience, these seemingly small details can end up getting in the way of the initial learning experience, requiring extra time and effort. Furthermore, "technical difficulties" do occur on HPC platforms in spite of the instructor's best effort, especially for a large class size.

A consequence of the above is that it is possible to hold an effective, efficient, and relatively short in-class development session in which students perform the full assignment under the guidance of the instructor. This makes it possible to address the inevitable initial difficulties with MPI programming in the classroom, focusing 100% on MPI programming issues. This is by contrast with our experience using real-world HPC platforms at the introductory level, in which attempts at holding such sessions are hindered by the logistics of using these platforms. One option that is sometimes used for introductory assignment is to have each student run MPI on their laptop in class (using a hostfile with only `localhost` entries). But in this case no performance benefit due to parallelism can be observed beyond the number of cores, which we have found to be an underwhelming first parallel computing experience for the students. With SMPI, instead, students are able to experiment right away with using arbitrary numbers of hosts, and witness the performance benefits. As a result, we have observed that well-prepared students who go through the assignment relatively quickly are able to discover on their own several parallel computing principles simply by modifying platform descriptions and execution conditions. For instance, in the last instance of the graduate-level "Principles of High Performance Computing" course at the University of Hawai'i at Mānoa (UHM), during the 1h15min in-class development session in which students go through this assignment with help from the instructor, several students were able to independently discover the fundamental notions of weak scaling and strong scaling. We contend that such invaluable independent learning would have been difficult to achieve if an actual HPC platform had been used for this session.

2) Physical topologies and communication patterns: One assignment in SMPI Courseware has students implement several versions of a broadcast. The learning objectives include: (i) to be proficient with a large subset of the MPI API for point-to-point communication; (ii) to be able to implement moderately complex communication patterns efficiently; (iii) to experience and understand the effect of network topology on communication patterns; (iv) to experience and understand the trade-offs between message size, network bandwidth, and network latency; (iv) to be able to conduct experimental performance comparisons of competing implementations and draw sensible conclusions; and (v) to become aware of the large size of the design space for collective communication operations and of the challenges involved.

After implementing a naive broadcast in which a message is sent by one process to n other processes via n `MPI_Send` calls, the students implement a broadcast first using a ring communication pattern, and then using a binary tree communication pattern. This assignment, if done entirely, is challenging as many students are not necessarily used to thinking about, let alone implementing, communication patterns. Throughout the assignment students compare their implementations with each other, and with `MPI_Bcast` as provided by several implementations of the MPI standard. They perform these comparisons using various network topologies, varying message size, and varying network bandwidth and latency values. They thus get to experience relevant performance effects (e.g., for small messages and/or high bandwidth the broadcast is latency-bound, if the communication pattern matches the physical topology then the broadcast is more efficient).

The obvious way in which simulation makes this assignment effective, or even possible, is that the HPC platform, and more specifically its network topology, is entirely simulated. As a result, it is possible to provide students with many network topology configurations. These configurations can be hypothetical and used for confirming expectations (e.g., the ring, resp. tree, broadcast perform well on a platform whose physical topology is an actual ring, resp. tree), or can be representative of actual network topologies used in HPC platforms (e.g., a single switch, a shared backplane, a fat-tree structure, a k-ary n-cube). Also, thanks to simulation, students can tweak platform descriptions to test hypotheses. For instance, it is possible to vary network latencies to understand

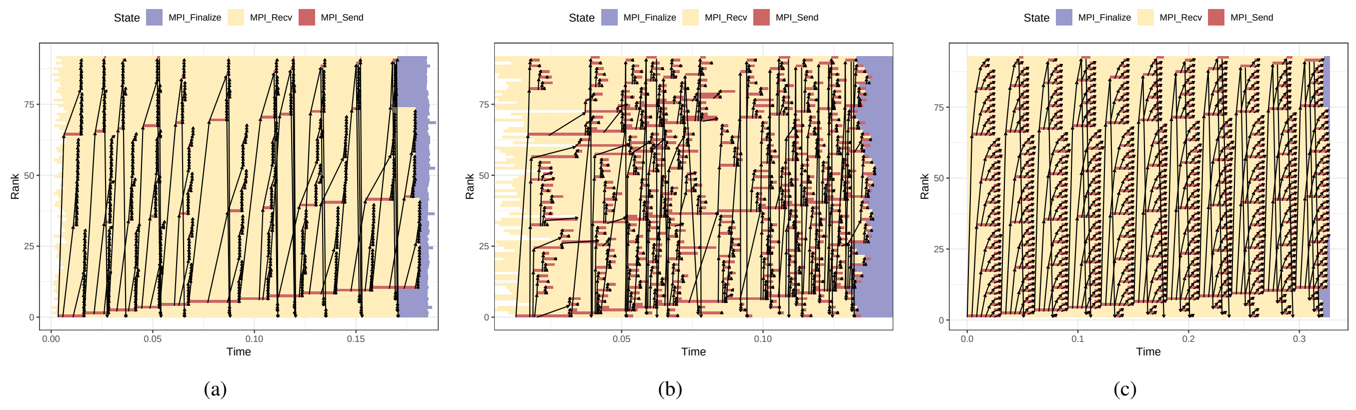

Fig. 2. Execution Gantt charts of 11 consecutive 92-process binomial tree broadcasts of 400,000-byte messages implemented using MPI point-to-point communication primitives and with different broadcast roots. (a) Execution on six 16-core nodes (2.40GHz cores) connected via 10Gbps Ethernet in the Paravance cluster of the Grid'5000 testbed [34] using OpenMPI 2.0.2; (b) Execution on one 4-core laptop computer (1.30GHz cores) using OpenMPI 2.1.1; and (c) Execution on 92 nodes of the Griffon cluster of the Grid'5000 testbed as simulated using SMPI. Time spent in `MPI_Send`, `MPI_Recv`, and `MPI_Finalize` are indicated with different colors, and black arrows match sends to their respective receives.

the impact of latency as message size varies. The scale of the platform can also be changed at will to explore scalability limits of the implementations. Finally, because SMPI provides many implementations of collective operations from popular implementations of the MPI standard (see Section III), students can experiment with all these implementations. This allows students to see first-hand that these implementations are not all equal, and that some of them work better than others in some regimes. This brings home the message that the design space for collective communication algorithms and their implementations is large.

All the above would be at best very difficult if teaching with real-world HPC platforms (i.e., changing network topologies is not possible, installing several MPI versions on a production platform is typically problematic, scale is limited). Instead, simulation makes it possible to provide students with a wide range of relevant scenarios, makes it possible for students to design their own scenarios, and in the end makes the learning objectives easier to achieve because the entire gamut of relevant effects can still be experienced hands-on.

3) Pipelined and asynchronous communication: One sobering (by design) conclusion from the assignment above is that popular MPI distributions provide broadcast implementations that are orders of magnitude faster that the implementations developed by the students. Another part of that assignment is thus focused on boosting broadcast performance. The additional learning objectives include: (i) to understand how splitting a message into chunks can increase throughput; (ii) to understand how asynchronous communication can increase throughput; (iii) to be able to enhance a communication pattern and implement a version of it that achieves high throughput using message splitting and asynchronous communication; and (iv) to be suitably impressed by the performance of production broadcast implementations provided by various MPI implementations.

In this phase of the assignment the students enhance their implementations by first splitting the message into chunks. Via experiments they draw conclusions about the best chunk size to use. They then use asynchronous communications to increase throughput further by overlapping sending and receiving operations. This is typically a challenging part of the assignment, as many students find it difficult to use asynchronous communication primitives in a way that actually achieves increase in throughput as well as correctness. In this assignment, student can also witness first-hand how impressive some production `MPI_Bcast` implementations can be performance-wise. For instance, once they have developed a ring-based broadcast that splits the message into chunks and uses asynchronous communication to benefit from communication pipelining, students experimentally determine the best chunk size to use for a given message size, a given total number of processes, and a physical platform that is an actual ring. In this case, their naive implementation does outperform the best `MPI_Bcast` implementations, but this improvement is typically marginal. And yet, the `MPI_Bcast` implementations do not know the topology structure and do not get to pick a good chunk size based on experiments. Furthermore, when changing the topology, the number of processes, and the message size, students then see that their own implementation can be outperformed by the adaptive `MPI_Bcast` implementations by orders of magnitude. In the graduate-level "Principles of High Performance Computing" at UHM, this assignment provides the motivation to explore what algorithmic and implementation techniques are used by these "Swiss army knife" `MPI_Bcast` implementations, often resulting in some students extending this assignment into their capstone course project, all in simulation with SMPI.

Most of the considerations in Section IV-B2 on why simulation is key to achieving learning objectives apply for the above learning objectives as well. Another consideration here, however, is that simulated executions are "clean". In other terms, they are reproducible and well-structured because they do not suffer from idiosyncratic effects seen in real-world platforms [11]–[14]. Therefore, it is possible to observe clearly performance effects of variations in algorithms, implementations, and platform configurations. As an example, consider Figure 2, which depicts Gantt charts of 2 real-world executions of an MPI program that performs a sequence of 92-process binomial tree broadcasts, on 92 cores of a real-world cluster and on 4 cores of a single laptop computer, and one simulated execution of the same program on a specification of another real-world cluster. As seen in the figure, while the simulated execution shows clear, expected patterns, the real-world executions experience software/hardware effects that make the pattern more chaotic and difficult to understand. Furthermore, not shown on the figure, there are differences between multiple runs of the same program on the same platform. We include the results on a single laptop since this is one option when teaching an HPC course without giving access to an actual HPC platform and have student instead run MPI programs on their own computers. Even on a single computer, due to time-sharing of cores, the patterns are also complex and non-deterministic. Understanding the effects the impact real-world executions is interesting from a research and development perspective. But they impede learning for novice students by obfuscating fundamental principles. Finally, another problem with these platform effect is that even if an instructor verifies that some learning objective is easily achievable on a specific platform used for a course, in spite of platform effects, there is no guarantee their the same holds on other platforms. This makes it difficult to share assignments among instructors and institutions, and places a large burden on the instructor if the platform changes from one course instance to the next.

4) Scalability studies: SMPI Courseware provides an assignment in which students study the performance of a matrix multiplication application. The learning objectives include: (i) to be able to determine an application's performance bottleneck; (ii) to understand both strong and weak scaling concepts and be able to perform an experimental scalability study; (iii) to be able to develop algorithmic and implementation techniques for pushing scalability further.

Student actually implement the matrix multiplication implementation in another assignment (they can of course be provided with the implementation directly for this assignment, if so desired). That implementation is based on an outer product algorithm and uses a 2-D (non-cyclic) block distribution. The algorithm entails performing multicasts between rows and columns of processes, which are logically arranged in a 2-D grid. This is done by creating MPI communicators and performing broadcasts within these communicators. In terms of learning objectives, that assignment is a step up from the assignment described in SectionIV-B1.

In this assignment students first instrument their MPI program to measure communication and computation times separately. They then perform various scalability studies by keeping the matrix sizes fixed and increasing the number of processors, and by keeping the number of processors fixed and increasing the matrix sizes. They also experiment with various platform configurations that are more or less network-bound. Throughout, students draw conclusions regarding the weak and strong scaling properties of the program. Given the $O(n^3)$ computation and $O(n^2)$ data features of matrix multiplication, students can observe good strong scaling behavior but poor weak scaling behavior in several relevant scenarios (e.g., slow network, poor broadcast implementations).

The benefit of using simulation for the above is that large-scale executions can be simulated on each student's laptop. This is because, as explained in Section III, SMPI provides techniques to reduce the CPU and RAM footprints of MPI applications. These techniques compromise the output of the simulated application (i.e., the computed matrix product is erroneous) but preserve the performance behavior when the application execution is non-data-dependent, which is the case for a matrix multiplication. As a result, students can push their scalability studies up to large scales, well beyond scales that could be achieved on HPC platforms used for teaching purposes in the majority of institutions.

V. Conclusion

Teaching HPC topics on a real-world HPC platform comes with many logistical and pedagogical challenges. In this work we have described SMPI Courseware, a set of ready-to-use assignments that target part of the HPC curriculum, namely distributed-memory parallel computing and MPI programming, and uses simulation to enhance the learning experience. More specifically, using SMPI, SimGrid's MPI application simulator, it is possible to have students implement MPI programs and run them unmodified in simulation mode for arbitrary platform configurations on their own computers. We have illustrated the pedagogic benefits of these capabilities by describing the content of and our experience with some of the assignments included in SMPI Courseware. SMPI Courseware has already been used in university courses, namely in several instance of the "Principles of High Performance Computing" graduate course at the University of Hawai'i at Manoa (USA) and of the "Parallel Systems" graduate course at the University of Grenoble (France). In some instances in-simulation assignments were used in addition to assignments on a real-world HPC platform, while in other instances all assignments in the course were in simulation. Although no formal user studies have been conducted, feedback in the classroom has been overwhelmingly positive. Simple questionnaires revealed that students did not find it frustrating to perform assignments in a "fake" environment. No student has complained that, instead of experiencing actual execution times first-hand, they merely see in the simulation output what the execution time would have been in the real-world. In those courses in which students used both real-world and simulated HPC platforms, students felt they learned a lot in simulation because they could tweak experimental conditions at will to explore ideas. Students also acknowledged that although the typical frustrations with real-world platforms (e.g., being "stuck" in the batch queue for a log time, being subjected to announced but regular platform down times, being forced to carefully plan job submissions) are valuable experiences, they do get in the way of learning if encountered for every assignment. A clear future direction for this work is to perform a thorough pedagogic evaluation in the classroom to obtain evidence-based qualitative and quantitative measures of the merit of using our approach for teaching parallel computing and MPI programming.

To date, SMPI Courseware contains seven assignments, most of which have already been used in university courses. While current assignments hit fundamental MPI programming and parallel computing learning objectives, many more assignments can be designed that capitalize on SMPI for the same or other learning objectives. For instance, current assignments do not include any master-worker or stencil applications, and only have students use the most basic collective communication operations. There are also several ways in which to further capitalize on simulation to achieve learning objectives that are often out of reach in traditional non-simulation-based courses. For instance, one important topic is heterogeneous computing. Many parallel algorithms for heterogeneous architectures are known, and in a traditional course one often merely describes their principles without students gaining any hands-on experience. With SMPI it is straightforward to have students experiment with arbitrarily heterogeneous (simulated) platform configurations. More generally, many compelling assignments are proposed/used in the HPC education community that could be part of SMPI Courseware (e.g., those presented at the eduPar and eduHPC workshop series, those available in popular HPC training opportunities such as Blue Waters and XSEDE). Developing in-simulation versions of such assignments would be straightforward in many cases and SMPI Courseware welcomes contributors. Although currently modest in size and scope, with both our own and outside contributions SMPI Courseware has the potential to become an invaluable resource for many HPC curricula world-wide.

A broader future direction is to use simulation for HPC education beyond the scope of MPI programming and traditional parallel computing. For instance, we are involved in a simulation project focused on scientific workflows [35]. In a nutshell, this project uses SimGrid as a foundation to develop a simulation framework for quickly prototyping and simulating the execution of Workflow Management Systems. We are currently developing a set of pedagogic modules with reusable in-simulation assignments that target learning objectives centered around fundamental distributed computing principles (data locality, load balancing, resource management and scheduling, fault-tolerance). The use of simulation in this context is if anything even more compelling than in the context of distributed-memory parallel computing and MPI programming. Ultimately, we aim at developing a large pedagogic corpus that provides a wealth of reusable assignments that can be integrated across the HPC curriculum for both undergraduate and graduate courses.

Acknowledgments

Some of the experiments in this work were carried out on the Grid'5000 experimental testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER, and other Universities and organizations (see https://www.grid5000.fr). Part of this work was supported through the HAC SPECIS Inria Project Laboratory. Part of this work was supported by NSF under contracts #1642369 and #1642335, and by CNRS under grant #PICS07239.

References

[1] M. Ludin, A. Weeden, J. Houchins, S. Thompson, C. Peck, I. Babic, K. Muterspaw, and E. Sergienko, "LittleFe: The high performance computing education appliance," in *Proc. of the International Conference on Cluster Computing*, 2013.

[2] S. Holt, A. Meaux, J. Roth, and D. Toth, "Making the One Cluster Per Student Method of Teaching Parallel Computing Financially Practical," *Journal of Computing Sciences in Colleges*, vol. 33, no. 4, pp. 106–113, 2018.

[3] R. Brown, J. Adams, S. Matthews, and E. Shoop, "Teaching Parallel and Distributed Computing with MPI on Raspberry Pi Clusters," in *Proc. of the 49th ACM Technical Symposium on Computer Science Education*, 2018, pp. 1054–1054.

[4] A. M. Pfalzgraf and J. A. Driscoll, "A low-cost computer cluster for high-performance computing education," in *Proc. of the International Conference on Electro/Information Technology*, 2014, pp. 362–366.

[5] K. Doucet and J. Zhang, "Learning Cluster Computing by Creating a Raspberry Pi Cluster," in *Proc. of the SouthEast Conference*, 2017, pp. 191–194.

[6] O. Abuzaghleh, K. Goldschmidt, Y. Elleithy, and J. Lee, "Implementing an Affordable High-performance Computing for Teaching-oriented Computer Science Curriculum," *ACM Transactions on Computing Education*, vol. 13, no. 1, pp. 3:1–3:14, 2013.

[7] C. Ivica, J. T. Riley, and C. Shubert, "StarHPC – Teaching parallel programming within elastic compute cloud," in *Proc. if the 31st International Conference on Information Technology Interfaces*, 2009, pp. 353–356.

[8] P. Marshall, M. Oberg, N. Rini, T. Voran, and M. Woitaszek, "Virtual Clusters for Hands-on Linux Cluster Construction Education," in *Proc. of the 11th LCI International Conference on High-Performance Clustered Computing*, 2010.

[9] N. A. Robison and T. J. Hacker, "Comparison of VM Deployment Methods for HPC Education," in *Proc. of the 1st Annual Conference on Research in Information Technology*, 2012, pp. 43–48.

[10] D. Johnson, S. Mason, and B. Hartpence, "Designing, Constructing and Implementing a Low-Cost Virtualization Cluster for Education," in *Proc. of International Multi-Conference on Society, Cybernetics and Informatics*, 2013.

[11] S. Hunold and A. Carpen-Amarie, "Reproducible MPI Benchmarking Is Still Not As Easy As You Think," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 12, pp. 3617–3630, 2016.

[12] A. Bhatele, K. Mohror, S. H. Langer, and K. E. Isaacs, "There Goes the Neighborhood: Performance Degradation Due to Nearby Jobs," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, 2013, pp. 1–41:12.

[13] Y. Inadomi, T. Patki, K. Inoue, M. Aoyagi, B. Rountree, M. Schulz, D. Lowenthal, Y. Wada, K. Fukazawa, M. Ueda, M. Kondo, and I. Miyoshi, "Analyzing and Mitigating the Impact of Manufacturing Variability in Power-constrained Supercomputing," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–78:12.

[14] O. Tuncer, E. Ates, Y. Zhang, A. Turk, J. Brandt, V. J. Leung, M. Egele, and A. K. Coskun, "Diagnosing Performance Variations in HPC Applications Using Machine Learning," in *Proc. of International Supercomputing Conference*, 2017, pp. 355–373.

[15] E. Luque, R. Suppi, and J. Sorribes, "A Quantitative Approach for Teaching Parallel Computing," in *Proc. of the 23rd SIGCSE Technical Symposium on Computer Science Education*, 1992, pp. 286–298.

[16] J. Hartman and D. Sanders, "Teaching parallel processing using free resources," in *Proc. 26th IEEE Frontiers in Education Conference*, vol. 3, 1996, pp. 1483–1486.

[17] A. N. Pears, "Using the DiST Simulator to Teach Parallel Computing Concepts," in *Proc. of the 1st International Forum on Parallel Computing Curricula*, 1995.

[18] G. Zarza, D. Lugones, D. Franco, and E. Luque, "An Innovative Teaching Strategy to Understand High-Performance Systems through Performance Evaluation," in *Proc. of International Comference on Computational Science*, 2012.

[19] A. Degomme, A. Legrand, G. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI applications: the SMPI approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 2387–2400, 2017.

[20] 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, 2014.

[21] "The SimGrid Project," Available at http://simgrid.org/, 2018.

[22] B. Lester, *The Art of Parallel Programming*. Prentice Hall, 1993.

[23] "Riverbed Modeler," Available at https://www.riverbed.com/fr/products/steelcentral/steelcentral-riverbed-modeler.html, 2018.

[24] P. Velho, L. Mello Schnorr, H. Casanova, and A. Legrand, "On the Validity of Flow-level TCP Network Models for Grid and Cloud Simulations," *ACM Transactions on Modeling and Computer Simulation*, vol. 23, no. 4, 2013.

[25] A. Kozinov, E.and Shtanyuk, "Learning Parallel Computations with ParaLab," in *Proc. of the 1st Ural Workshop on Parallel, Distributed, and Cloud Computing for Young Scientists*, 2015, pp. 11–20.

[26] V. Gergel and A. Labutina, "ParaLab System for Investigating the Parallel Algorithms," in *Proc. of the Russia-Taiwan Symposium on Methods and Tools of Parallel Processing*, 2010, pp. 95–104.

[27] L. Bobelin, A. Legrand, D. A. G. Márquez, P. Navarro, M. Quinson, F. Suter, and C. Thiery, "Scalable Multi-Purpose Network Representation for Large Scale Distributed System Simulation," in *Proceedings of the 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, Ottawa, Canada, May 2012, pp. 220–227.

[28] A. Faraj, X. Yuan, and D. Lowenthal, "STAR-MPI: self tuned adaptive routines for MPI collective operations," in *Proc. of the 20th ACM Intl. Conf. on Supercomputing*, 2006, pp. 199–208.

[29] R. Thakur, R. Rabenseifner, and W. Gropp, "Optimization of Collective Communication Operations in MPICH," *International Journal of High Performance Computing Applications*, vol. 19, no. 1, pp. 49–66, 2005.

[30] D.-K. Panda, K. Tomko, K. Schulz, and M. A., "The MVAPICH Project: Evolution and Sustainability of an Open Source Production Quality MPI Library for HPC," in *Proc. of the Workshop on Sustainable Software for Science: Practice and Experiences*, 2013.

[31] T. Cornebize, "Capacity Planning of Supercomputers: Simulating MPI Applications at Scale," Master's thesis, Grenoble INP ; Université Grenoble - Alpes, 2017. [Online]. Available: https://hal.inria.fr/hal-01544827

[32] "SMPI Integration Testing of MPI Proxy applications," https://github.com/simgrid/SMPI-proxy-apps, 2018.

[33] "SMPI Courseware," https://simgrid.github.io/SMPI_CourseWare/, 2018.

[34] "Grid'5000 Testbed," www.grid5000.fr, 2018.

[35] "The WRENCH Project," http://wrench-project.org, 2018.