2021 IEEE/ACM Ninth Workshop on Education for High Performance Computing (EduHPC) | 978-1-6654-1116-5/21/$31.00 ©2021 IEEE | DOI: 10.1109/EduHPC54835.2021.00012

Peachy Parallel Assignments (EduHPC 2021)

Henri Casanova*, Rafael Ferreira da Silva† Arturo Gonzalez-Escribano‡,
Herman Li*, Yuri Torres‡, David P. Bunde§

* *Information and Computer Sciences, University of Hawaii*, Honolulu, HI, USA
{henric,herm8888}@hawaii.edu

†*National Center for Computational Sciences, Oak Ridge National Laboratory*, Oak Ridge, TN, USA
silvarf@ornl.gov

‡*Dept. Informatica, Universidad de Valladolid*, Valladolid, Spain
{arturo,yuri.torres}@infor.uva.es

§*Dept. Computer Science, Knox College*, Galesburg, IL, USA
dbunde@knox.edu

***Abstract*—Peachy Parallel Assignments are high-quality assignments that are easy for other instructors to adopt and use in their own classes. They are selected competitively for presentation at the Edu* workshops based on ease of adoption and how "cool and inspirational" they are for students. The goals are to excite students about PDC, to save faculty the time and risk associated with creating new assignments, and to recognize faculty who create awesome assignments for their students.**

In this paper, we present two assignments. The first assignment is a simulation of air flow in a wind tunnel, which students parallelize using OpenMP, MPI, and CUDA to illustrate the different techniques needed for these paradigms. The second assignment is a series of exercises to teach students the principles of batch scheduling and how to interact with a batch scheduler to submit parallel jobs. It uses simulation to allow students to quickly see the results of their decisions and to support revisiting an earlier decision.

I. Introduction

Peachy Parallel Assignments are a collection of assignments for use when teaching Parallel and Distributed Computing and/or High-Performance Computing. The goal is to disseminate exceptional assignments that will excite students about the content while teaching them important principles and skills. These assignments are designed for easy adoption by other instructors, who are saved the effort of developing their own assignments and also the risk of an assignment going awry due to an unforeseen issue. They are presented at the Edu* series of workshops on parallel computing education [1]–[3], [6] and also on the Peachy Assignments webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy).

Peachy Assignments are selected via a competitive process. All of them must have been successfully used in class. After this consideration, they are selected based on the following criteria:

- Adoptable: A Peachy Parallel Assignment should be easily adopted by a variety of instructors. The assignment should be well-described, including a discussion of the context in which it was used and how it might be adapted to other classes, and provide the needed materials (e.g. assignment handout for students and given code). This criteria also includes how broadly applicable the assignment is to others; ideally, the assignment should have students practice widely-taught concepts using commonly-used programming languages and hardware, have few prerequisites, and (with variations) be appropriate for different levels of students.
- Cool and inspirational: A Peachy Parallel Assignment should excite students through the problem being solved and/or the artifact(s) that students create. This will encourage students to spend time on the assignment and ideally tell others about it.

This paper presents two Peachy Parallel Assignments. In Section II, we present a series of assignments that ask students to parallelize a wind tunnel simulation using OpenMP, MPI, and CUDA. These could be used separately, but repeating the parallelization in multiple paradigms helps students understand the similarities and differences between the paradigms. In Section III, we present activities to teach students the principles and practical considerations of batch scheduling. The activities use Docker for portability and present students with a simulated environment to learn and practice commands for the Slurm scheduler. Because the activities use simulation, the students can quickly see the results of their commands and also "rewind" time to experiment with other decisions.

II. Wind tunnel simulation

Our first assignment was used in a Parallel Computing course to show how different approaches are needed for the same problem in different parallel programming models. It targets concepts of shared-memory programming with OpenMP, distributed-memory programming with MPI, and/or GPU programming with CUDA or OpenCL. This assignment is based on a simulation of air pressure and particle movement

978-1-6654-1116-5/21/$31.00 ©2021 IEEE
DOI 10.1109/EduHPC54835.2021.00012

inside a wind tunnel with fixed obstacles. The program is designed to be simple, easy to understand by students, and to include specific parallelization and optimization opportunities. Although there are quite direct parallel solutions in the three programming models, the program has plenty of opportunities for further improvements. It maintains the same core concepts used in three previously presented assignments, introducing a different parallel structure based on a pipeline approach, and new relevant optimization challenges with high performance impact. It also introduces a progressive approach with optional levels of difficulty. This assignment has been successfully used in parallel programming contests during an optional Parallel Programming course in the third year of Computer Engineering degree. The assignment material can be found at https://trasgo.infor.uva.es/peachy-assignments/.

A. Idea and context

Different programming models use different approaches for the parallelization of application structures. Understanding these differences is key for students to get into more advanced techniques, and to face parallel programming in current heterogeneous platforms. For several years, we have been teaching an optional course of Parallel Programming in the Computer Engineering degree at Universidad de Valladolid. The course introduces the basics of OpenMP, MPI, and CUDA or OpenCL. Three previous Peachy Parallel Assignments have been presented in this series [1]–[3]. All of them are designed to be parallelized by the students during three one-week programming contests, where they work to obtain the best performance with a mixed competitive and collaborative strategy [5]. Although this kind of assignment can be used to teach a single programming model, here we also use it to show which concepts and techniques can be reused across different models, and which cannot, exposing the approach differences and the conceptual shift between them. For example, the students learn the differences between controlling race-conditions in shared-memory vs. using distributed data structures with explicit communications, or dealing with tiling and memory hierarchies in GPU coprocessors.

This new assignment keeps the main core concepts used in the previous assignments, but it is built around a different synchronization structure based on a macro-pipeline with wave fronts. It introduces different possibilities to deal with the workload balance and interesting structural optimizations. It can be optimized in many different ways depending on the parallelization and detail level, with a high potential impact on performance. It maintains a clear focus on simple but effective code parallelizations and optimizations, while introducing more opportunities for the advanced students and new and different choices in the three programming models considered.

The assignment is based on a simulation of the propagation of air pressure through a wind tunnel, in which obstacles modify the air flow, and moving particles are pushed. A lattice represents a 2-dimensional cut of the tunnel space along the flow axis. In the upper row of the lattice there is a fan inlet. The air pressure is propagated in downward wave fronts at each

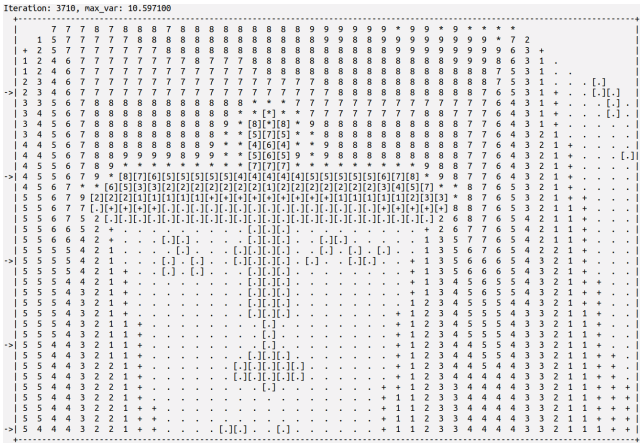

Fig. 1. Graphical representation of the wind tunnel at a given step, provided as output by the simulation program. Square brackets represents obstacles and particles. Numbers and symbols represent the air pressure level. Arrows in the left margin indicate the current position of the wave fronts.

simulation step using a directional stencil operator, leading to a pipeline computation. Fixed and flying particles can be added randomly at the start of the computation, or in chosen positions to form complex obstacles. Each particle has a given mass and air flow resistance. Different scenarios, with different obstacles and particle densities can be easily created with the program parameters. During the simulation, we can observe how the air flow is spread across the lattice and how the obstacles modify the air pressure. We can also observe how the flying particles are pushed by the air flow with simplified turbulence effects around the obstacles. The simulation results are determined by the initialization arguments that include random seeds. Thus, they are reproducible. The arguments can be chosen to generate specific situations with different load distributions, different ratios of concurrency problems when particles collide in a lattice position, etc.

The program has been organized to be modular. Thus, parallelization and tests can be done in three stages, helping the students to follow a progressive approach. The first stage requires the parallelization of only the air pressure propagation. In the second stage, we add the computation of air flow modifications due to fixed particles. In the third stage, the students should also deal with the code that controls the flying particles. This also allows the introduction of a mixed classification criteria of the solutions, using both scores (related to the number of tests passed) and performance.

B. Using the assignment

As in the previous assignments of this series, the provided material includes a sequential code, a test-bed of input arguments, and a handout explaining the assignment. The students can use common compilers and PC platforms to develop and test their codes. An automatic judge tool with an on-line public ranking is used to provide a fair arena, and to keep the students engaged during the contests with competitive and collaborative rewards [4], [5]. The judge configuration is done by simply

providing tuples of input arguments (representing the scenarios chosen by the teacher), and the expected output results. The tool executes the programs on a real parallel system. In order to rank the students, it measures the total number of tests passed and the program performance. The sections of the sequential code that should be parallelized and optimized by the students are clearly marked to help them skip argument processing, scenario initialization, OpenMP/MPI/CUDA setup, time measurement commands, and results output. Thus, the original code can be directly compiled and run by the students, or submitted to the judge tool even before starting to parallelize it.

The students in the course where we tested this assignment have already studied concepts of operating systems and concurrency, and they have used the C programming language in a couple of previous courses. There were 68 students enrolled. The degree of participation was high, with more than 11,900 requests of program execution in our parallel cluster, including both tests and judgment requests. A survey conducted at the end of the course shows that the students have a good degree of satisfaction with the learning experience. The evaluation of the assignments is 90% of the final grade in the course. For each programming model, the students present an essay along with their final code, do a short video presentation, and answer live questions from the teachers for 15 minutes. The position in the leaderboard is also taken into account for the assignment grade. To the question: "Are you satisfied with the overall experience of the course, activity types, evaluation method, etc.?", using a Likert scale from 1 to 5, the average is 4.08. The assignment illustrates the main concepts of the course and provides opportunities to deepen in the subject. For example, some students optimized their OpenMP codes run 6.8 times faster while obtaining the same results as other students who did the minimum amount of work. The modular organization, with three progressive steps, helps the students to have a functional parallel program in any of the programming models in a reasonable time, without facing all the complications tackled in the more advanced steps. The results show a smoother grading of the students than with previous assignments in the series, improving the grades in the programming model that the students find more demanding (MPI).

C. Concepts covered

The stages on the simulation steps of the program are: (1) Generating the new input inlet value with added random noise; (2) Computing the air pressure alterations caused by obstacles; (3) Computing the movements of the flying particles; (4) Propagating air pressure at the advancing wave fronts; (5) A reduction to check minimum stability. Stages 1 to 3 only happen each 8 iterations, and the distance between waves is 8 rows. This simplifies the load balancing when creating scenarios, and opens different approaches to parallelize the pipeline. The output of the program includes the number of iterations executed, the result of the reduction and a sample of chosen array positions. If desired, the program can also write a text-mode graphical representation each simulation step that can be used to visualize the evolution of the simulation (see Fig. 1).

The basic concepts covered in OpenMP are parallelization of loops, reductions, atomic operations, and schedulings. In MPI, the students work with array partitions, variable size communications, reductions, asynchronous operations, communicators, and load balance. For GPU programming, the main ideas are embarrassingly parallel kernels, thread-block geometries and sizes, non-trivial atomic operations, simple reductions, minimizing communication operations, overlapping of kernels, and host computing. The program also shows how to use fixed-point arithmetic to avoid precision and concurrency problems in all models. Several advanced optimizations can be discovered and applied. For example, aggressive code reorderings can allow easier parallelization or better operation overlapping, clever load-balancing techniques adapted to the the scenario features, memoization techniques and pipeline stage reorganization to expose more parallelism in OpenMP, taking decisions about replicated vs. distributed computing in MPI, fusing kernels, new uses for the shared memory or non-trivial reductions on GPUs, etc.

D. Variants

This assignment covers an important class of parallel programs based on non-basic stencil operations and interactions of particles or agents with an environment represented with a grid. Many different scenarios can be chosen by the teacher by selecting the proper input arguments. The assignment already presents three levels of difficulty, introducing new code and challenges at each level. Most of the code is modular. The pipeline structure can be easily modified to introduce new stages. The particles' movement or stencil functions can be changed to produce different communication structures, etc. Finally, better graphical and online interfaces can be devised to enrich the learning experience.

III. Batch Scheduling Concepts and Practices

Our second assignment targets the basic concepts and practices that are necessary for using a batch-scheduled platform effectively. Active learning is achieved via interactive pedagogic activities. Specifically, these activities provide students with an in-the-browser, command-line shell running on the head node of a cluster managed by Slurm, all in simulation. Each activity guides students toward specific learning objectives via a series of questions, each of which requires hands-on experimentation in the shell. Because the execution is simulated, it is possible for students to reset time or to advance time at will, which makes it possible for them to explore and compare alternative job submission strategies quickly and conveniently. This assignment has few prerequisites, does not require any particular hardware resources, and only requires that Docker be installed on the computer. To date, this assignment has been used successfully in one offering of a graduate-level HPC course, with overwhelmingly positive student feedback.

A. Context and Objectives

This assignment provides undergraduate or graduate students with a gentle introduction to the concepts and practices that are relevant to the use of batch-scheduled compute platforms. The various concepts (batch queue, job submission, job cancellation, job queue wait time, job turnaround time, etc.) are explained in a pedagogic narrative. These concepts are put in practice by presenting students with a simulated, in-the-browser, command-line terminal in which they can use (simplified versions of) the most fundamental Slurm commands.

This assignment can be used stand-alone or as a complement to assignments that require the use of a batch-scheduled platform, in which case it provides "in-simulation training" so as to better prepare students for using and understanding the behavior of the real platform. It is available at: https://eduwrench.org/pedagogic_modules/batch_scheduling/.

B. Prerequisites, Hardware, and Software

This assignment only requires minimal knowledge of the Linux command-line and basic knowledge of the concept of parallelism. Students in a course that involves batch scheduling most likely already have the required knowledge. But if this is not the case, this assignment is hosted on the EduWRENCH Web site (https://eduwrench.org/), which provides introductory on-line pedagogic modules for all this background.

No hardware besides the student's own computer is needed since the assignment is done in simulation in the browser. The only required software is Docker. The assignment runs on the student's computer in a Docker container that exposes a local Web server to which the student connects using any Web browser. This ensures that all students can run the assignment regardless of their operating systems.

C. Overview

The assignment is presented as a Web page with six tabs, each with a set of learning objectives listed at the top. The first tab presents an overview of batch scheduling and of the role of a batch scheduler. Each of the next five tabs explains a concept, and then presents students with a pedagogic activity. Specifically, in the browser, students interact with a Linux shell that is running on the (simulated) head node of a (simulated) Slurm-managed cluster. In this shell, students can type standard UNIX commands as well as the `squeue`, `sbatch`, and `scancel` Slurm commands. They can also edit "batch scripts" in which they specify batch jobs they want to execute, passing these scripts to the `sbatch` command. Each tab then asks a series of questions, which require students to interact with the shell, observe what happens, draw conclusions, and, in later tabs, come up with strategies to reduce job turn-around time. Fig. 2 shows a screenshot of the simulated shell, and Fig. 3 shows a screenshot of the simulated text editor.

The use of simulation makes it possible to *manipulate time*. As seen in the top-right corner of Fig. 2, there is a "Reset Time" button, which students can click to rewind time to the origin. This makes it possible to try something, say "never-mind", and try something else in the exact same conditions. Also, the simulated shell provides a `sleep` command, which instantaneously advances time. Say a student types a `sleep 3600` command. This command returns immediately (in real time), but the simulated time advances by one hour. This is crucial for students to observe what happens in the future without having to actually wait for that future to happen. For instance in the "Job Size" tab, students are exposed to the concept of picking appropriate job sizes based on the state of the queue so as to reduce turn-around time. They submit jobs that request different numbers of compute nodes for the same parallel program, each option corresponding to a different trade-off between wait time and execution time. By resetting time and moving forward in time, students can quickly compare these options, draw conclusions based on observation, and actively learn important lessons regarding job turn-around time optimizations. This same kind of active learning would be difficult and time-consuming to achieve if using a real-world platform.

D. Assignment Description

Beyond the initial "Basics" tabs, which does not include an in-simulation activity, the next 5 tabs do. They are as follows:

- "Job Submission": the `sbatch` Slurm command;
- "Batch Queue": the `squeue` Slurm command;
- "Job Cancellation": the `scancel` Slurm command;
- "Job Duration": the impact of a job's requested time;
- "Job Size": the impact of a job's requested number of nodes.

Due to lack of space, we cannot describe all the above, but instead focus on the "Job Cancellation" tab. In this tab, students are introduced to the `scancel` Slurm command and presented with a scenario in which a few jobs by other users are running on the cluster and many other jobs are in the queue. The assumption is that the batch scheduler uses conservative backfilling (which is explained in the "Basics" and the "Batch Queue" tabs).

Students are told that they need to submit a job for a parallel program whose execution time when executed on n compute nodes is $2 + 20/n$ hours. They are instructed to submit a job asking for 16 compute nodes, and then to right away use `squeue` to determine whether their job is running. The batch queue is crafted so that their job is not running. Students are then asked to advance time and determine their job's turn-around time, which is over 40 hours.

Students are then instructed to reset the time to zero and to inspect the state of the batch queue to determine how many competing jobs are running and how many compute nodes are idle. They are then told that their goal is to submit the largest job that can begin execution right away. The relevant excerpt from the tab is: "Your goal is to submit a job asking for as many nodes as possible but so that your job can run **right away**! Note because n nodes are idle right now it does not mean that any job that asks for n nodes will start immediately. If the job requests too much time, then starting it right now may postpone previously submitted jobs, which is disallowed

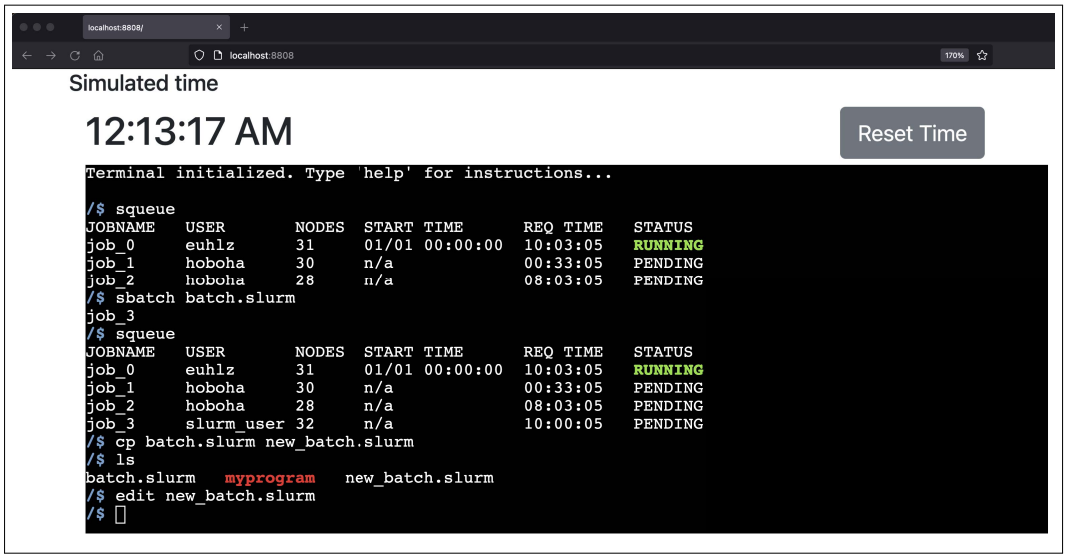

Fig. 2. In-the-browser simulated shell that supports UNIX and Slurm commands.

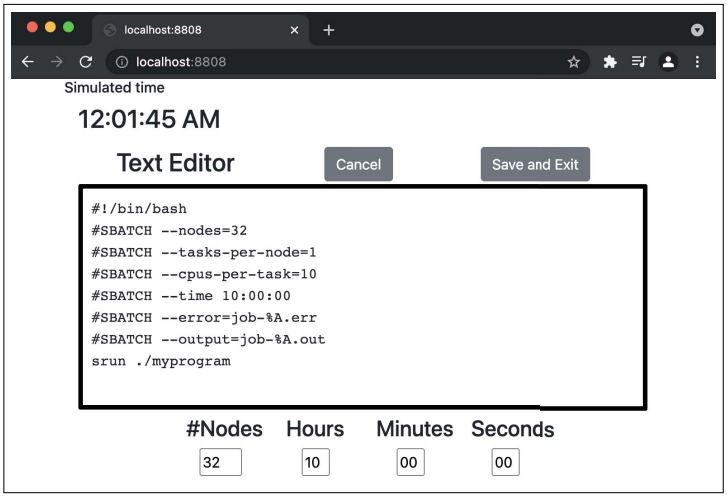

Fig. 3. In-the-browser simulated text editor (currently editing a Slurm batch file).

TABLE I
SURVEY OF SELF-ASSESSMENT OF LEARNING RESULTS.

To what extent did the assignment help you learn new things?	• 80%: to a great extent • 20%: to some extent
How confident do you feel about your understanding of the main concepts behind batch scheduling?	• 40%: very confident • 40%: condfident • 20%: somewhat condfident
To which extent would you say that simulation is a valuable addition to the overall process of learning about batch scheduling?	• 75%: to a large extent • 20%: to some extent • 5%: don't know

in our cluster. So answering this question is not as simple as it seems." This is essentially re-explaining conservative backfilling to students.

Students are then suggested to use an exploration algorithm

in which they submit a job asking for $n = 1$ node and just enough time to run the program successfully (22 hours), and see if it starts right away. Then they immediately cancel that job, and repeat with $n = 2$ (and 12 hours), and so on. In this way, they can find the largest n that works via series of submissions and cancellations. They can of course use a binary research instead of a linear search, or whatever approach they choose. Once they have determined the largest number of nodes that can be used to that job start immediately (which is 4 nodes), they perform that submission. They then advance time to determine the job's turn-around time, which is 7 hours, which they then contrast with the much longer turn-around time when asking for 16 nodes.

Finally, students are asked whether it would have been possible to determine the best number of nodes purely via reasoning based on the initial state of the batch queue. It turns out the answer to that is "yes". Note that this tab states explicitly that all other user jobs are assumed to ask for exactly the amount time they need (i.e., no job completes prematurely). Subsequent tabs in the assignment mention this is not the case for most jobs and that premature job completions are commonplace, thus making the schedule less predictable.

E. Strengths and Weaknesses

The key strength of this assignment is that the use of simulation affords many pedagogic advantages for hands-on learning. First, it is possible to present students with interesting, relevant, and perfectly analyzable and reproducible scenarios, which cannot be done on a real-world platform. Second, running actual jobs on a real batch-scheduled platform can take a very long time, which would negatively impact learning. Third, as described earlier, simulation makes it possible for students to manipulate time at will (resetting time and jumping ahead in time), which is key for active learning.

The main weakness of this assignment is that it is only in simulation. As a result, students do not run anything real, which some may find disconcerting. Given the aforementioned strengths afforded by simulation, and given collected student feedback, we contend that the pedagogic benefits brought about by simulation far outweigh this one drawback.

F. Previous Uses

This assignment was used in a 600-level graduate HPC course (ICS 632) at the University of Hawai'i at Mānoa in Fall 2021. Twenty students completed the assignment, filled out a self-assessment questionnaire, and provided feedback on the pedagogic material. Sixteen of these students are Computer Science graduate students, 2 are Computer Science undergraduate students, and 2 are non-Computer Science graduate students.

Table I shows answers to key self-assessment questions. Open-ended comments from students were overwhelmingly positive and included: "Simulations are a great way to practice", "This approach provides the freedom to learn and spend as much time as needed to get comfortable with the process", "I really liked this simulation! It was a great way to get hands-on practice without having to actually be on a live cluster", "I had access to a cluster in the past, but I had no idea how everything worked since it all looked so complicated. But this simulation helped to clear a lot of things up.", "It made me feel like I was actually submitting jobs", "A sense of open-endedness caused me to try and reverse engineer the system, by submitting different inputs and seeing the responses of the system", "I would love to see more of this type of content/simulations!"

Finally, students provided constructive feedback on the software (bug reports, suggestions for interface improvements) and the pedagogic narrative (typos, suggestions for clarification), which has been used to update the assignment's Web site and Docker container.

Acknowledgments. The wind tunnel assignment was partially funded by Universidad de Valladolid (Spain), Proyecto Innovación docente PID 20-21_63, Spanish Ministerio de Economía, Industria y Competitividad with the ERDF program of the European Union, project PCAS (TIN2017-88614-R); and *Nos Impulsa* Junta de Castilla y León - FEDER Grants, projects PROPHET and PROPHET-2 (VA082P17, VA226P20). The batch scheduling assignment is funded by NSF contracts #1923539, #1923621, #2103489, and #2103508.

REFERENCES

[1] Mulya Agung, Muhammad Alfian Amrizal, Steven Bogaerts, Ryusuke Egawa, Daniel A. Ellsworth, Jorge Fernandez-Fabeiro, Arturo Gonzalez-Escribano, Sukhamay Kundu, Alina Lazar, Allen Malony, Hiroyuki Takizawa, and David P. Bunde. Peachy parallel assignments (EduHPC 2019). In *IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC 2019)*, Denver (CO), USA, 2019. IEEE.

[2] E. Ayguadé, L. Alvarez, F. Banchelli, M. Burtscher, A. Gonzalez-Escribano, J. Gutierrez, D.A. Joiner, D. Kaeli, F. Previlon, E. Rodriguez-Gutiez, and D.P. Bunde. Peachy parallel assignments (EduHPC 2018). In *IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC 2018)*, Dallas (TX), USA, 2018. IEEE.

[3] Henri Casanova, Rafael Ferreira da Silva, Arturo Gonzalez-Escribano, William Koch, Yuri Torres, and David P. Bunde. Peachy parallel assignments (EduHPC 2020). In *IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC 2020)*, Atlanta (GE), USA, 2020. IEEE.

[4] J. Fresno, A. Ortega-Arranz, H. Ortega-Arranz, A. Gonzalez-Escribano, and D.R. Llanos. *Gamification-Based E-Learning Strategies for Computer Programming Education*, chapter 6. Applying Gamification in a Parallel Programming Course. IGI Global, 2017.

[5] Arturo Gonzalez-Escribano, Victor Lara-Mongil, Eduardo Rodriguez-Gutiez, and Yuri Torres. Toward improving collaborative behaviour during competitive programming assignments. In *IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC 2019)*, Denver (CO), USA, 2019. IEEE.

[6] O. Ozturk, B. Glick, J. Mache, and D.P. Bunde. Peachy parallel assignments (EduPar 2019). In *Proc. 9th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*, 2019.

APPENDIX: REPRODUCIBILITY

The wind tunnel assignment has been used in the context of a Parallel Computing course, in the third year of the Computing Engineering grade at the University of Valladolid (Spain).

The material of the assignment, including a handout, the starting sequential code, and some input data sets to be used

as examples will be made publicly available through the CDER courseware repository.

The on-line judge program used in the programming contests is named *Tablon*, and it was developed by the Trasgo research group at the University of Valladolid (https://trasgo.infor.uva.es/tablon/). The contest software uses the Slurm queue management software to interact with the machines in the cluster of our research group. During the course we used the Slurm 18.08.3 release.

The machine of the cluster used for the OpenMP contest is named *heracles*. It is a server with four AMD Opteron 6376 @ 2.3Ghz CPUs, with a total of 64 cores, and 128 GB of RAM.

The machine used in the CUDA/OpenCL contests is named *hydra*. It is a server with two Intel Xeon E5-2609v3 @1.9 GHz CPUs, with 12 physical cores, and 64 GB of RAM. It is equipped with 4 NVIDIA's GPUs (CUDA 3.5), GTX Titan Black, 2880 cores @980 MHz, and 6 GB of RAM.

During the MPI contest, we use *heracles* and *hydra* in combination with two other servers to create a heterogeneous cluster. The other two machines are: *thunderbird*, with an Intel i5-3330 @2.4 GHz CPU and 8 GB of RAM; and *phoenix*, with and Intel QCore @2.4 Ghz CPU with 6 GB of RAM.

All machines are managed by a CentOS 7 operating system. The compilers and system software used are GCC v7.2, and CUDA v10.2.

The assignment provides the sequential code, program arguments to be used as test-beds for the students, and other test-beds used by the on-line judge during the contest.

The results of the contests for the four Peachy assignments in this series are publicly available at http://frontendv.infor.uva.es.