Peachy Parallel Assignments (EduPar 2022)

H. Martin Bücker*, Henri Casanova†, Rafael Ferreira da Silva‡,
Alice Lasserre§, Derrick Luyen†, Raymond Namyst§,
Johannes Schoder*, Pierre-André Wacrenier§, David P. Bunde¶

*Institute for Computer Science, Friedrich Schiller University Jena, Jena, Germany
{martin.buecker, johannes.schoder}@uni-jena.de

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

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

§Computer Science Department, University of Bordeaux, Inria Bordeaux Sud-Ouest, Talence, France
alice.lasserre@etu.u-bordeaux.fr, {raymond.namyst,pierre-andre.wacrenier}@u-bordeaux.fr

¶Computer Science Department, Knox College, Galesburg, IL, USA
dbunde@knox.edu

***Abstract*—The presentation of Peachy Parallel Assignments in several workshops on parallel and distributed computing education aims to promote the reuse of high-quality assignments, both saving precious faculty time and improving the quality of course assignments. Presented assignments are selected competitively— they must have been successfully used in a real classroom, be easy for other instructors to adopt, and be "cool and inspirational" to encourage students to spend time on them and talk about them with others. Winning assignments are also archived on the Peachy Parallel Assignments website.**

In this installment of Peachy Parallel Assignments, we present three new assignments. The first assignment is to simulate an Abelian Sandpile, with grains of sand moving from tall piles to shorter ones. This is a discrete simulation that creates colorful and intricate images. The second assignment is a Big Data problem in which students use the MapReduce paradigm to recreate "Warming Stripes", a visualization of climate data that highlights climate change. The third assignment introduces climate-oriented optimization by asking students to schedule distributed workflows to minimize their carbon footprint.

***Index Terms*—Peachy Parallel Assignments, Parallel computing education, High-Performance Computing education, Parallel programming, Curriculum Development, Abelian Sandpile, Parallel Simulation, MapReduce, Big Data, Warming Stripes, Distributed Workflow Scheduling, Carbon Footprint**

This work is partially funded by NSF contract #1923621. The research used the Ara cluster at Friedrich Schiller University Jena, which is supported by DFG grants INST 275/334–1 FUGG and INST 275/363–1 FUGG, and resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

I. Introduction

One aspect of teaching is to assign the homework and laboratory exercises that students use to learn the material and demonstrate their mastery of it. Creating high-quality assignments can be time-consuming, requiring both a good idea and developing the necessary materials (assignment description, given code, etc). It can also be risky since even an assignment that seems good initially may have hidden prerequisite knowledge or simply be harder than the instructor intended.

Peachy Parallel Assignments aim to address this challenge by encouraging the reuse of high-quality assignments. They are selected competitively based on the following criteria:

- Tested: All assignments must have been successfully used with real students
- Adoptable: The assignments must be useful to other instructors, with clear descriptions and the resources needed for adoption by others (handouts, given code, references for more information, etc). Ideally, they focus on core PDC topics using widely-used languages and toolsets, with suggested customizations that can make them suitable for students at a variety of levels.
- Cool and inspirational: The assignments must motivate students through the artifacts they create (e.g. images) or the concepts taught. Ideally, students should want to talk about the assignment with friends and show it off to others.

Assignments selected as Peachy Parallel Assignments

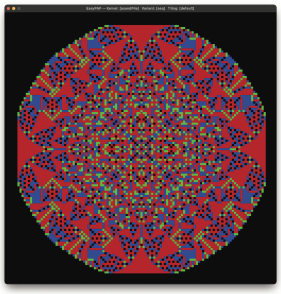
join a series published at EduPar and EduHPC workshops, e.g. [?], [?], [?], [?]. The assignments are also archived at the Peachy Parallel Assignments webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy), along with all the materials needed to adopt them. The assignments are meant to be used as-is or adapted to fit the context of the reader's class. They can also serve as inspiration for the reader's own assignment creation.

This paper describes three new Peachy Parallel Assignments selected for presentation at EduPar 2022. Section **??** describes an assignment to simulate an Abelian Sandpile, with grains of sand moving from tall piles to shorter ones. The assignment creates colorful and intricate images. Section **??** describes an assignment in which students use the MapReduce paradigm to recreate "Warming Stripes", a visualization of climate data that highlights climate change. It is topical in both subject matter (climate change) and technique (Big Data) in addition to generating an powerful visualization. Section **??** also describes an assignment about climate change, this time focusing on mitigation. It asks students to schedule a distributed workflow between systems with varying carbon efficiency to minimize the workflow's carbon footprint. This assignment has the additional advantage of being entirely browser-based, eliminating software installation and maintenance issues.

II. Abelian Sandpile Simulation

The Abelian Sandpile assignment was originally proposed in the context of the *Parallel Programming* course of the CS Masters curriculum at the University of Bordeaux. In this course, students are introduced to high performance computing by studying advanced topics related to multicore, GPU and cluster programming. They progressively get familiar with various algorithmic techniques such as cache-conscious algorithms and scheduling strategies for regular and irregular computations. Student evaluation is based on a capstone assignment in which they progressively parallelize a 2D stencil code using OpenMP, OpenCL, and MPI. The Bak-Tang-Wiesenfeld *Abelian Sandpile Model* [**?**] is a good candidate for such a "*putting it all together*" activity because it starts with simple sequential code, offers numerous optimization opportunities, and can be easily ported to a GPU or distributed over a cluster of machines. Moreover, the visualization of the simulation leads to attractive fractal animations (see Fig. **??**).

During the lab session, we use the EasyPAP [**?**] framework, an easy-to-use C programming environment designed to help students to learn HPC. EasyPAP's main philosophy is to let students focus on computation kernels while hiding most of the implementation details related to program initialization, code instrumentation, and interactive display. Moreover, EasyPAP features performance graph plot tools, real-time monitoring facilities, and off-line trace exploration utilities that provide new ways of visualizing tasks together with their associated data. EasyPAP relies on the SDL library [**?**]. It is available on both Linux and Mac OS X systems and can be downloaded from [**?**].



(a) Starting with 25.000 grains in a center cell.

(b) Starting with 4 grains in each cell.

Figure 1: Two stable configurations reached over 128×128 sand-piles. Black pixels correspond to cells with 0 grains, green to 1, blue to 2, and red to 3.

A. The abelian sandpile model

In the Abelian Sandpile Model [**?**], the sandpile is a $N \times M$ 4-connected cellular automaton such that the border cells are connected to a special cell called *sink*. The state of a cell is an integer corresponding to the number of sand grains it contains. By definition, a regular cell is said to be *stable* whenever it contains 3 or fewer grains and *unstable* otherwise. The rule of the automaton is the following: except for the sink, an unstable cell gives its surplus grains to its 4 neighbors by distributing them equitably. For example, if a cell contains 11 grains, then it will give 2 to each neighbor and keep the remaining 3 grains. Starting from an initial unstable configuration, the simulation of a sandpile consists of iteratively applying the automaton rule until a stable configuration is reached (i.e. all cells are stable). Dhar [**?**] proved that a unique final stable configuration is reached regardless of the computation order, allowing great flexibility with respect to parallelization.

B. The assignment

The goal of the assignment is to efficiently compute the final stable configuration. Student are provided with two codes (Fig. **??**): a synchronous variant where all cells are treated simultaneously using an auxiliary array, and an asynchronous variant where sand slides are computed "in place" and impact their neighboring cells immediately.

The organization of this project is the following: the project is carried out by pairs of students and is divided into four assignments. For each assignment, students have to provide their source code, as well as a scientific report in which they justify their approach with the help of performance plots and execution traces.

```
inline int sync_compute_new_state (int y, int x)
{
  next_sandpile(y,x) =  sandpile (y, x) % 4
                      + sandpile (y, x - 1) / 4
                      + sandpile (y, x + 1) / 4
                      + sandpile (y - 1, x) / 4
                      + sandpile (y + 1, x) / 4;

  return next_sandpile(y,x) !=  sandpile (y, x);
}

inline int async_compute_new_state (int y, int x)
{
  if (sandpile (y, x) < 4) return 0;
  unsigned long int div4 = sandpile (y, x) / 4;
  sandpile (y, x - 1) += div4;
  sandpile (y, x + 1) += div4;
  sandpile (y - 1, x) += div4;
  sandpile (y + 1, x) += div4;
  sandpile (y, x) %= 4;
  return 1;
}
```

Figure 2: Synchronous and asynchronous variants of kernel `sandpile`.

The first assignment is centered around the basic OpenMP parallelization of the 2D stencil. Students are expected to implement some simple optimizations to avoid false sharing and unnecessary synchronizations. They are also asked to experimentally determine the most suitable OpenMP loop scheduling policy.

During the second assignment, students progressively explore the benefits of tiling with respect to cache reuse and observe the influence of code simplification on compiler auto-vectorization. Since every pixel is read multiple times in each iteration, students are invited to implement a tiled parallel version to maximize cache reuse. In addition, they have to develop a lazy evaluation algorithm that avoids computing tiles whose neighborhood was in a steady state at the previous iteration. Once they have an effective OpenMP lazy variant, students can look at the EASYPAP tiling window to make sure that "areas where nothing changes" are not computed (Fig. **??**). To address the challenge of mitigating the load imbalance introduced by sparse configurations, students have to experiment with various scheduling policies and various tile sizes (Fig. **??**).

The third assignment focuses on SIMD vectorization and GPU programming. Outer tiles need special attention, because they contain border cells which should not be computed (sink). This poses alignment issues when trying to efficiently vectorize the code. By observing that such issues only happen with outer tiles, students are invited to implement a separate variant for inner tiles to enable aggressive compiler optimizations.

The last assignment is devoted to combining two different programming paradigms to address hybrid or heterogeneous architectures (e.g., MPI + OpenMP, OpenMP + OpenCL). It implements the well-known *Ghost Cell Pattern* [**?**]: in every iteration, each pair of neighboring processes exchange a copy of their borders. However, the

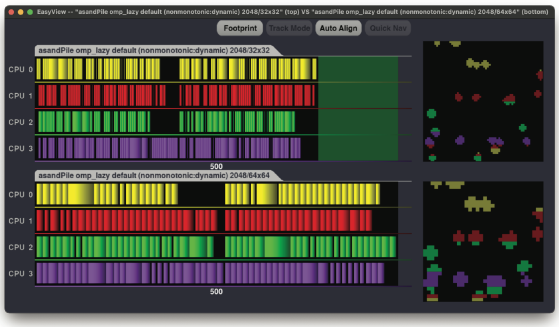


Figure 3: Comparison of two execution traces of the `asandPile` kernel over a 2048×2048 sparse configuration. The traces display tasks executed during the same 500th iteration performed by a lazy OpenMP variant. The top trace features 32×32 tiles, against 64×64 tiles for the bottom one.

communication overheads are such that students have to develop a solution that trades redundant computation for less-frequent communication. Finally, the CPU+GPU assignment is an opportunity for students to implement dynamic load balancing strategies between CPUs and GPUs.

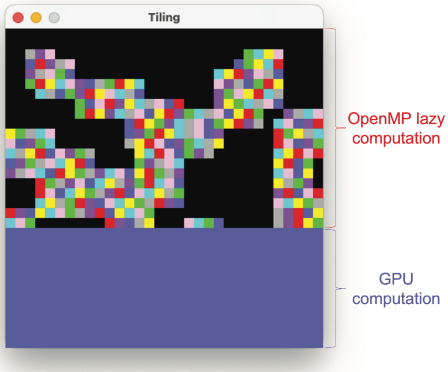


Figure 4: Distribution of tiles during the execution of a hybrid OpenMP-OpenCL variant. On the CPU side, the color of a tile indicates the target core. Black areas represent stable tiles.

C. Comments

The rationale behind the decomposition of this project into four assignments is to provide students with frequent feedback about their scientific writing skills and their performance evaluation methodology. However, various other strategies are possible. For instance, the first three assignments can be merged together and may even be used in the context of undergraduate courses. Other alternatives are reducing the range of programming paradigms covered, or narrowing down the project to the (a)synchronous variant only. Indeed, both asynchronous and synchronous variants cover a

wide range of difficulties and have their own pedagogical advantages.

The synchronous variant is a mere illustration of grid-based computational simulations. It can be easily parallelized, vectorized, and implemented on GPUs. Student can thus focus on performance analysis and parameter tuning.

The asynchronous variant is more difficult to parallelize because of potential race conditions. Therefore, student must either use synchronization primitives or work around the problem and use multi-wave task scheduling policies. Nevertheless, this variant allows plenty of algorithmic and code optimizations.

D. Feedback

This project was given in 2020 and we were very satisfied with the results. Most students were very involved, and using EASYPAP has increased their productivity and motivation (Fig.**??**), mostly because interactive display and monitoring were helpful, and because they found that the learning curve was gentle.

For the first report, we found that half of the students had submitted at least one buggy version. Thanks to the detailed feedback they got, they greatly improved the quality of their contributions. Some students had clearly gone beyond our expectations: some had designed a lazy GPU implementation and others had implemented a smart dynamic algorithm to load balance between CPUs and GPUs. In terms of their scientific writing and performance evaluation methodology, we found that students became more rigorous from their second report onwards.

At the end of the project, we conducted a survey (Fig.**??**) to figure out how students evaluated the pedagogical impact of EASYPAP. Open-ended comments from students were positive and included the following: *"It is very easy to add and try a new code variant. We just add a few lines of code, we compile and it is ready for command line testing. It makes the platform really nice to use. We really focus on the code, it's great!"; "EasyPaP is a tool that makes the difference by allowing us to get straight to the point: parallelism!".*

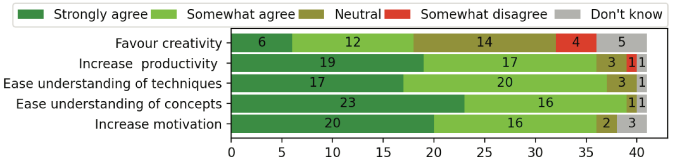


Figure 5: Summary of a survey about EASYPAP.

III. WARMING STRIPES: A COOL START TO BIG DATA PROCESSING

Processing massive amounts of data is increasingly becoming an important topic in the development of modern computing curricula. Therefore, our next assignment introduces the MapReduce programming paradigm. The assignment illustrates aspects of distributed computing and, at the same time, it is designed around the analysis of climate data, motivating students to explore one of the defining crises of our times.

An important competence for today's graduates in a wide range of scientific disciplines is the capability to handle compute- and data-intensive tasks analyzing large-scale data sets. Though somewhat dated, Apache Hadoop and the MapReduce programming model [**?**] still provide the methodological basis of many contemporary big data frameworks. In this section, we introduce a beginner's programming assignment that teaches students the basics of MapReduce and guides them through the essential phases of a typical data science project while simultaneously aiming to raise awareness of the climate crisis.

The task of the programming assignment is to rebuild the warming stripes, made popular by British climatologist Ed Hawkins. The original warming stripes show clearly and vividly the trend in global average temperatures over the past decades. In the programming assignment, we calculate the annual average temperatures for a time span in a specific geographical region and yields the single, stunning image given in Fig. **??**. This image visualizes the long-term rise in annual average temperature for Germany in the period from 1881 to 2019. The annual temperature ranges from a low around 7 °C to a high around 10 °C. The range of temperature values used in the colorbar are manually specified by first computing the average temperature of the whole time span and then adding and subtracting 1.5 °C to set the maximum and minimum temperature values, respectively.

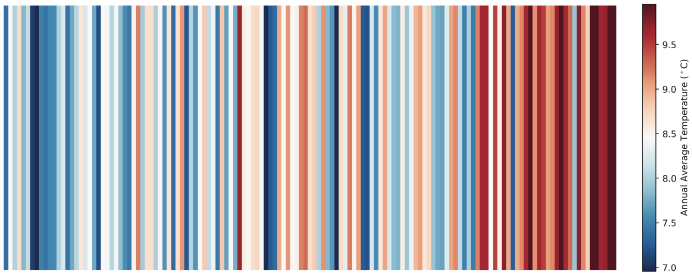


Figure 6: Annual average temperature rise for Germany ranging from 1881 (left) to 2019 (right); inspired by https://showyourstripes.info.

A. Beginner's Programming Assignment

The assignment was originally conceived in 2020 as part of the "Public Climate School." This nationwide series of annual week-long events is initiated by "Students for Future" and motivates lecturers at German universities to integrate novel educational elements into their courses in an attempt to increase awareness of the progressive heating of the Earth. The assignment is

available in a public git repository [**?**]. The repository contains a Jupyter notebook that specifies the assignment in the form of a template to be completed by the students to obtain the solution.

1) Intended Audience and Necessary Prerequisites: The assignment is designed within a course on big data processing at Friedrich Schiller University Jena. This course is relevant not only for different computer science degree programs but also for students enrolled in the master's program "Computational and Data Science". The latter is an interdisciplinary degree program that is open to students with a bachelor's degree in a wide variety of scientific areas, including computer science and mathematics as well as natural and engineering sciences. In general, the overall course is designed to be accessible to students with diverse scientific backgrounds and little programming experience. However, for this specific assignment, moderate programming skills in Python, a basic understanding of MapReduce, and experiences in using the Apache Hadoop Streaming API are required.

2) Problem Formulation and Main Concepts: As a first step, the assignment asks the students to download temperature data from Germany's National Meteorological Service, *Deutscher Wetterdienst* (DWD). The Federal Republic of Germany comprises 16 constituent states. The data consist of monthly average temperature values of different states over a time span starting 1881. These values are distributed across 12 input files storing the average temperature of one month for every year (row) in every state (column). The task is to compute the annual average temperatures in Germany for all years. Since the focus of this assignment is on the MapReduce programming paradigm rather than on general-purpose programming, the Jupyter notebook of the assignment provides software that the students will use to visualize the results.

This course assignment covers several main concepts. While the task is simple in a general-purpose programming language, it is more difficult in the MapReduce programming paradigm. The reason is that MapReduce forces the programmer to employ a three-phase approach, starting with the map phase, followed by the group-by-keys phase, ending with the reduce phase. For beginners, it is difficult to reformulate a given problem under the severe constraints of this three-step approach. The concept of finding a suitable reformulation of a given problem in terms of mappers and reducers is shown via a mapper whose key-value pairs at the output represent a year as the key and temperatures averaged over all states as the value. The group-by-keys phase then rearranges all values at the output of the map phase into different groups at the input of the reduce phase. Each group is associated with a key which corresponds to a particular year in this MapReduce program. For each year, a reducer then averages over all months.

3) Data Science Workflows: Without considering details, a typical workflow of a data science project consists of the following four phases: (1) data acquisition, (2) data pre-processing, (3) computations to analyze data, and (4) result validation. This assignment uses averaging temperatures over time as a simple example and guides students through all of these stages. In particular, it shows that the last step is essential, as the data set provided by the DWD may be incomplete. For instance, when students downloaded the data from DWD in late 2020, the temperatures of the last few months of that year were missing. So, if they don't take steps to remedy the situation with missing data for winter 2020, the average temperature of this year will be too high. From this example, the students are encouraged to critically evaluate the quality of the data set.

4) Software Engineering and Reusability: In the actual assignment, we use data sets that are small. The reason is that this introductory example is designed to start learning MapReduce. That is, we consider this assignment similar to a "Hello World!" program that can first be executed on the student's local machine and that intentionally postpones the additional challenges of accessing a departmental compute cluster. However, the cluster is used to execute the final implementation of this assignment not only for small data sets but optionally also for larger data sets to be downloaded by the students from various different sources. It is also used in all later programming assignments for the course (not detailed in this manuscript).

Climate data sets can grow very fast in size, for example by increasing the number of weather stations and/or the time resolution. Also, different shapes of input data are possible. For instance, there could be different input data files associated with individual weather stations rather than with months. We ask the students to design a MapReduce implementation that is relatively invariant to different data formats. In particular, the mapper should be capable of averaging any kind of data and not be restricted to the computation of average temperatures. Therefore, it should include a data-pre-processing stage that reorders and rearranges the input, enabling it to process different data in the same way. The aim is to encourage students to pay attention to good software engineering practices.

5) Hardware Infrastructure: Besides using a local machine, it is recommended to run the assignment on a Hadoop cluster. In our course, we use the Hadoop partition of the Ara cluster of Friedrich Schiller University Jena, consisting of 16 compute nodes, each equipped with 192 GB RAM and two Intel Xeon Gold 6140s, each with 18 cores and 2.3 GHz.

B. Classroom Evaluation

The assignment has been tested in a big data course for two subsequent years. A subset of eight participants of

the course in winter 2021/2022 took part in an optional survey that collected student feedback. The survey consisted of three multiple-choice questions each with five possible answers which are given below in parentheses with an italic font. The remaining questions left room for students to write freely. Although the sample size is small, we believe the survey results are interesting. Specifically, we found the following:

- Six students thought that the prerequisites taught in class were *sufficient* for the assignment, while two thought their knowledge was *absolutely sufficient*. Choices: (*absolutely sufficient*, *sufficient*, *neutral*, *insufficient*, *absolutely insufficient*).
- Seven students found the assignment to be *reasonable* and one student thought it was *difficult*. Choices: (*too difficult*, *difficult*, *reasonable*, *easy*, *too easy*).
- We asked the students whether and why the assignment increased or decreased their interest in MapReduce. Seven participants answered that it increased their interest in MapReduce. Answers mentioned an up-to-date problem, a practicably relevant exercise, and a rapid and simple approach.
- Seven participants remarked that the assignment helped in understanding the typical steps of a data science project, from data acquisition to the evaluation of the results.
- Four students found that the assignment helped them to solve more complex assignments that followed later in the course. Some students mentioned that it also prepared them to start their homework projects which were mandatory for the admission to the final exam.
- Seven participants found the assignment to be *mostly cool* and one person *very cool*, mainly because of the real practical data and up-to-date topic. Choices: (*very cool*, *mostly cool*, *okay*, *mostly boring*, *very boring*).
- Seven students did not think that the assignment changed their awareness of the climate crisis, mostly because their level of awareness was already high. Two students noted that it was interesting to reproduce the warming stripes.

IV. Performance and Carbon Footprint of Distributed Workflow Executions

Our third assignment exposes students to the notion that, in addition to performance, a pressing concern for the execution of distributed applications is their carbon footprint. Students are presented with a scientific workflow application and are asked to reason about and experiment with different execution scenarios with the reduction of the carbon footprint as an objective. This assignment requires no programming and active learning is achieved via interactive, in-the-browser simulation experiments. No hardware or software is required besides a web browser. This assignment is hosted on the EduWRENCH site, which hosts several pedagogic modules that cover all prerequisite material if needed [**?**]. Although intended for advanced undergraduate students, to date, this assignment has been used in one offering of a graduate-level HPC course. Feedback gathered from that one course was very positive. It was also used to improve the pedagogic content and its presentation.

A. Overview and Prerequisites

The premise of this assignment is that students work for an organization that needs to repeatedly execute an astronomy scientific workflow (738 tasks with a 7.5GB total data footprint). The organization has access (i) to a local cluster where nodes are powered by a non-green energy source and can be turned off or down-clocked to improve power efficiency; and (ii) to a few virtual machine instances running on a remote cloud whose physical nodes are powered by a green energy source. Students are tasked with reasoning about the workflow execution and optimizing it for various objectives and using various methods, including resource provisioning, resource configuration, and task scheduling decisions. Students do this interactively through their browser. The simulator is hosted on a back-end server. It is implemented using the WRENCH (https://wrench-project.org) and SimGrid (https://simgrid.org) simulation frameworks [**?**].

The prerequisites for this assignment include basic knowledge of parallel computing concepts (multi-core and multi-node parallelism, speedup, efficiency), of distributed computing concepts (network data transfers, network data proximity), and of workflows (task data dependencies, width). In courses focused on parallel and distributed computing, it may be that students already have covered these prerequisites. Regardless, this assignment is hosted on the EduWRENCH site (https://eduwrench.org), which hosts several pedagogic modules that cover all the above concepts and more. These modules all include learning objectives, pedagogic narratives, interactive simulation applications, practice questions, as well as open-ended questions. It should thus be straightforward for an instructor to point students to particular prerequisite content that they must cover before starting on the assignment.

B. Assignment Description

The assignment is available at https://eduwrench.org/pedagogic_modules/workflow_co2/ as a single page with two tabs, each with learning objectives, a narrative, an interactive simulation application, and a set of questions. **No software download/installation is required, and any teacher/student at any institution can go through the assignment today via any Web browser.**

Tab #1 – The first tab introduces the workflow of interest, which is an instance of the Montage astronomy

application. This workflow is to be executed on a 64-node cluster powered by a power plant that generates 291 gCO2e (gram CO2 equivalent) per kWh. The cluster nodes can be configured to operate in one of seven power states (p-states), each corresponding to a different trade-off between compute speed and power consumption. Furthermore, a number of nodes can be powered off. The assignment makes the simplifying assumption that all powered on nodes operate in the same p-state (i.e., the cluster is homogeneous). Students are provided with an in-the-browser simulation application in which they can pick the number of nodes that are powered on and their p-state, simulate the workflow execution (which takes a few seconds), and see the simulation output as execution time, power consumed, and gCO2e generated.

Students are then asked to answer three questions. The first question establishes a baseline for execution time, parallel speedup, and parallel efficiency when powering on all nodes in their highest p-state (i.e., aiming for highest performance). The second question states that, in fact, it is only necessary to execute the workflow in under 3 minutes. Students are then asked to evaluate two (for now mutually exclusive) options for minimizing CO2 emission given this execution time bound: power off some nodes or downclock all nodes. They are asked to perform a binary search to identify the minimum number of nodes to power on and the minimum p-state to use, and then report to their hypothetical boss on the merit of each option. Finally, the third question presents students with a heuristic designed by their hypothetical boss. This heuristic combines both power management techniques (powering off and downclocking) and students are asked to evaluate how well this heuristic works. It turns out that it leads to lower CO2 emission than both previously evaluated options, showing that combining power management techniques can be useful.

Tab #2 – In the second tab, students are told that their organization has purchased 16 virtual machine instances on a remote, *green*, cloud. As a result, the organization now only powers on 12 nodes of the local cluster, all operating at the lowest possible p-state. The remote cloud is accessible via a network link with limited bandwidth. The key issue now is to decide whether a task should be executed on the local cluster or on the remote cloud. Note that the remote cloud has storage, so the output of a task executed on the cloud is available locally to a subsequent child task that also executes on the cloud. In other words, there is possibility of data locality. Like the previous tab, this tab also includes an interactive simulation application. Students can use it to see the effect, in terms of performance and CO2 emission, of running some fraction of tasks in particular workflow levels on the remote cloud.

Students are then asked to answer five questions. The first question establishes baselines for “all on the local cluster" and “all on the cloud" executions. The second question asks students to reason about, and then compare using simulation, three options for executing the first two levels of the workflow. Subsequent questions guide students toward coming up with configurations that execute fractions of some workflow levels on the cloud, engaging in a “treasure hunt” for the configuration that minimizes CO2 emission. In particular, for the last question students are free to experiment with whatever scheme they can come up with. During in-class sessions in which students went through this assignment, the instructor observed students actively experimenting and trying to beat the footprint achieved by other students, denoting a reasonably high level of student engagement in the assignment. In the future, we will run our simulator to exhaustively evaluate all possible options so as to compute the actual optimal CO2 emission for this (NP-complete) problem and state its value in the assignment, so that students know how far their solution is from the optimal.

C. Strengths and Weaknesses

The main weakness of this assignment is that it is only in simulation: although it allows for active learning via interactive experiments, these experiments are not “real”. This is a deliberate design choice of this assignment and in fact of the EduWRENCH project as a whole: programming is not required so that assignments can be easily integrated in early courses and/or in non-computer-science curricula. But, as a result, some of the “excitement” is lost. Note, however, that student feedback collected for this and other EduWRENCH assignments does not indicate that students deem this a big impediment to their learning experience.

A strength of this assignment is that it should be extremely easy to integrate into existing courses. Although it has prerequisites, these prerequisites are covered in other modules available on the same Web site (which also includes a handy glossary of terms). The key strength of the assignment is that it teaches learning objectives that would be extremely difficult to achieve without simulation. In most institutions, it would be close to impossible to provide students with the necessary hardware and software environments for running meaningful experiments. Even if provided, students would have to learn myriads of technical details and skills for using these environments. This would preclude achieving the learning objectives hands-on in most courses, and in particular in undergraduate courses.

D. Previous Uses

This assignment was used in a graduate HPC course (ICS 632) at the University of Hawai‘i at Mānoa in Fall 2021. 11 students completed a self-assessment questionnaire, and provided feedback on the pedagogic mate-

Table I: Student feedback ($n = 11$).

Question	Choices	#Answers
How easy / difficult is the assignment?	very easy	1
	somewhat easy	6
	neither easy nor difficult	4
	somewhat difficult	-
	very difficult	-
How useful is the assignment?	very useful	5
	useful	3
	somewhat useful	3
	of little use	-
	not useful	-
To what extent did the assignment help you learn new things?	to a great extent	5
	to a moderate extent	4
	to some extent	2
	to a small extent	-
	not at all	-
Are you interested in learning more about this topic?	yes	10
	no	1
How useful is simulation in this assignment?	very useful	6
	useful	3
	somewhat useful	3
	of little use	-
	not useful	-
How valuable is the overall learning experience in the module?	very much	7
	quite a bit	3
	somewhat	1
	a little	-
	not at all	-

rial (9 were Computer Science graduate students and 2 Computer Science undergraduate students). All student feedback is summarized in Table **??**. These self-assessment results are a good indication that the assignment accomplishes its objectives. Assessment results showed that almost all students in the course scored more than 90% on this assignment, which is expected in a graduate course. Open-ended comments entered by students in the self-assessment questionnaires were very positive and include comments such as "The writing and explanations are great, being able to do a lot of trial and error is fun" and "The visual aspect of the simulation really helped me understand what was happening". Finally, students have provided constructive feedback on the assignment, all of which has been taken into account for improving the pedagogic content and its presentation.

REFERENCES

[1] "SDL: Simple directmedia layer," (Visited on 2020-03-09). [Online]. Available: https://www.libsdl.org

[2] M. Agung, M. Amrizal, S. Bogaerts, R. Egawa, D. Ellsworth, J. Fernandez-Fabeiro, A. Gonzalez-Escribano, S. Kundu, A. Lazar, A. Malony, H. Takizawa, and D. Bunde, "Peachy parallel assignments (EduHPC 2019)," in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2019.

[3] P. Bak, C. Tang, and K. Wiesenfeld, "Self-organized criticality," *Physical Review A*, vol. 38, pp. 364–374, 07 1988.

[4] H. Casanova, R. F. da Silva, A. Gonzalez-Escribano, W. Koch, and Y. Torres, "Peachy parallel assignments (EduHPC 2020)," in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2020.

[5] H. Casanova, R. F. da Silva, A. Gonzalez-Escribano, H. Li, Y. Torres, and D. Bunde, "Peachy parallel assignments (EduHPC 2021)," in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2021.

[6] H. Casanova, R. Ferreira da Silva, R. Tanaka, S. Pandey, G. Jethwani, W. Koch, S. Albrecht, J. Oeth, and F. Suter, "Developing accurate and scalable simulators of production workflow management systems with wrench," *Future Generation Computer Systems*, vol. 112, pp. 162–175, 2020.

[7] H. Casanova, R. Tanaka, W. Koch, and R. Ferreira da Silva, "Teaching parallel and distributed computing concepts in simulation with wrench," *Journal of Parallel and Distributed Computing*, vol. 156, pp. 53–63, 2021.

[8] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: https://doi.org/10.1145/1327452.1327492

[9] D. Dhar, "Self-organized critical state of sandpile automaton models," *Phys. Rev. Lett.*, vol. 64, pp. 1613–1616, Apr 1990.

[10] F. B. Kjolstad and M. Snir, "Ghost cell pattern," in *Proceedings of the 2010 Workshop on Parallel Programming Patterns*, ser. ParaPLoP '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1953611.1953615

[11] A. Lasserre, R. Namyst, and P. Wacrenier, "Easypap: A framework for learning parallel programming," *J. Parallel Distributed Comput.*, vol. 158, pp. 94–114, 2021.

[12] R. Namyst and P.-A. Wacrenier. (2018) The EASYPAP web site. (Visited on 2020-03-09). [Online]. Available: https://gforgeron.gitlab.io/easypap/

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

[14] J. Schoder and H. M. Bücker, "Peachy parallel assignment: Warming stripes with MapReduce," Git Repository, 2022. [On-

line]. Available: https://git.uni-jena.de/big_data_assignments/warming_stripes.git