# Peachy Parallel Assignments (EduHPC 2020)

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

*University of Hawaii
Honolulu, HI, USA
{henric,kochwill}@hawaii.edu

†University of Southern California
Marina Del Rey, CA, USA
rafsilva@isi.edu

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

§Knox College
Galesburg, IL, USA
dbunde@knox.edu

***Abstract*—Peachy Parallel Assignments are high-quality assignments for teaching parallel and distributed computing. They are selected competitively for presentation at the Edu* workshops. All of the assignments have been successfully used in class and they are selected based on the their ease of adoption by other instructors and for being cool and inspirational to students. This paper presents a paper-and-pencil assignment asking students to analyze the performance of different system configurations and an assignment in which students parallelize a simulation of the evolution of simple living organisms.**

***Index Terms*—Peachy Parallel Assignments, Parallel computing education, High-Performance Computing education, Parallel programming, Curriculum Development, Performance analysis, Parallel simulation, OpenMP, MPI, GPGPU**

## I. Introduction

Class assignments are an important part of teaching parallel and distributed computing and high-performance computing. Students spend a lot of time on the assignments, which provide the practice students need to master concepts taught in lecture and reading. Good assignments also provide context for the material taught, demonstrating its applicability to important real-world problems and, ideally, promoting student interest in their field. It is not easy for instructors to create such assignments as it requires both creativity and time. There is also risk involved because not every seemingly-great assignment idea works well in practice.

To help educators save time and improve the quality of their assignments, a Peachy Parallel Assignment track was added to the Edu* series of workshops on Parallel and Distributed Computing Education. The assignments are presented at the workshops [1]–[3] and also collected on a webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy) to make them easy for others to find and adopt.

Peachy Parallel Assignments are selected via a competitive process. All of them must have been successfully used in class. Then 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 that students create. This will encourage students to spend time on the assignment and ideally tell others about it.

This effort is inspired by the SIGCSE conference's Nifty Assignment sessions, which focus on assignments for introductory computing courses. (See http://nifty.stanford.edu for more details.)

In this paper, we present the following Peachy Parallel Assignments:

- A paper-and-pencil assignment asking students to analyze the performance of different system configurations, and
- The third assignment of a series dedicated to familiarizing students with OpenMP, MPI, and CUDA/OpenCL, and the differences between them. This third one proposes the parallelization of a simulation of the evolution of simple living organisms.

The Peachy Parallel Assignments webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy) has the materials needed to adopt each of these assignments. It also lists the Peachy Parallel Assignments presented at previous workshops. Please come and browse these great assignment ideas. Then, consider submitting your own assignments to our next competition.

## II. Reasoning about Multi-core Application Performance (Casanova, Ferreira da Silva, Koch)

In the first assignment, students estimate the performance of a parallel application on a multi-core machine, and then evaluate various hardware/software upgrades. *This assignment*

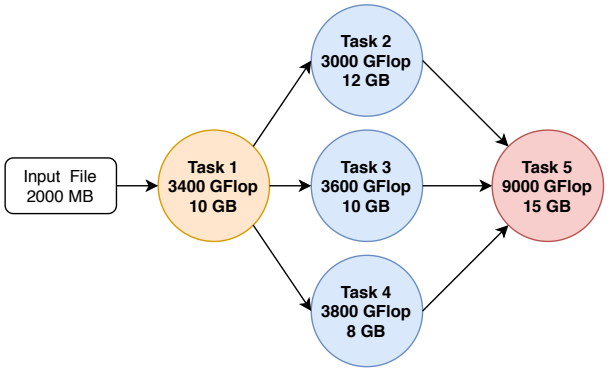


Fig. 1. A DAG for a program that implements bioinformatics computations on a large database of DNA sequences. Task 1 applies some simple cleanup process to the sequences. Then, three tasks (Tasks 2-4) need to be executed to compute different similarity metrics between the sequences in the database. Once all these metrics are obtained, a complicated machine learning classification process is applied to the metrics (Task 5).

*requires no programming*, and can thus be used in early courses. It could also be used in later courses to make sure that students can reason about parallel performance before embarking onto more involved homework and/or programming assignments. The learning objectives for this assignment are that students will

- Be able to estimate the execution time of a parallel program on a multi-core machine;
- Be able to reason about task- and data-parallelism;
- Be able to quantify how load-balancing and idle time impact performance;
- Be able to reason about task dependencies; and
- Be able to reason about how RAM constraints limit parallelism.

Students are presented with an application structured as a 5-task Directed Acyclic Graph (Fig. 1). Each task has an amount of work to perform (in Gflop) and a RAM footprint (in GB). The first task also reads in an input file. Three of the five tasks are independent. In a first question, student are asked to estimate the execution time of the application on a particular setup: a 2-core virtual machine with given memory capacity and disk bandwidth. This setup does not make it possible to execute the three independent tasks in parallel.

In a second question, students are given the choice of four possible upgrades to the virtual machine configuration: (*i*) increasing the disk bandwidth; (*ii*) adding one more core and increasing RAM; (*iii*) increasing core speed; and (*iv*) changing the implementation of one of the tasks to expose data-parallelism (with a remaining sequential component). The objective is to execute the application in under one minute, but only one of the above upgrades can be purchased. For each upgrade, students must determine whether the objective is achieved or not.

The assignment is available on-line at: https://eduwrench.org/pedagogic_modules/pdcc/multi_core_computing/. The assignment itself is on the last tab ("Capstone") of that page, while the preceding tabs target prerequisites, as explained hereafter.

### A. Prerequisites

This assignment requires no programming whatsoever, but does require that students have learned the concepts behind parallelism, load balancing, I/O overheads, RAM capacity, task dependencies, and data parallelism. This is why the assignment is part of an on-line pedagogic module on multi-core computing. Specifically, it is the 6th and last tab in a Web page where the previous 5 tabs are: Parallelism on Multicore machines; Load Imbalance; I/O and RAM; Task Dependencies; and Data Parallelism. Each of these 5 tabs consists of not only a pedagogic narrative, but also of hands-on activities including simulation experiments (that students can run in the browser, i.e. no software/hardware installation or configuration is required) [4] and sets of practice and open questions. Therefore, although the assignment has the above listed concepts as prerequisites, all these concepts are covered in the same Web page and available to students if needed.

Other prerequisites are that students understand that a program can consist of a set of "tasks", where each task needs to perform I/O and compute operations, and needs to occupy some space in RAM. Therefore, students should understand notions of RAM capacity, I/O speed, and CPU compute speed. These prerequisites are actually targeted by another pedagogic module ("Single-Core Computing") on the same Web site (see https://eduwrench.org/modules/). The only prerequisite for that module, is that students understand the concepts of programs running on a computer.

Finally, some of the pedagogic narrative in the above makes reference to Computer Architecture and Operating Systems concepts, each time explaining these concepts in a simple, and self-contained manner (only referencing classic textbooks in case students want to know more details). It should thus be possible for university freshmen to go through the above two modules, culminating in the Peachy assignment, within their first year of Computer Science education.

### B. Weaknesses, Strengths, and Previous Uses

**Weaknesses –** The main weakness of this assignment is that it is "pencil-and-paper" and thus does not provide students with hands-on experience or "excitement" about doing something real. This is a design choice for the pedagogic module that comprises this assignment, which targets students with no programming experience so as to allow for easy integration in early courses. Note that in the prerequisite sections on the same Web page, students do have opportunities for hands-on experience by running simulation experiments in the browser.
**Strengths –** The assignment should be extremely easy to integrate into existing courses, either as a first introduction to performance issues for parallel computing, or as a complement to other assignments that may entail programming. Although the assignment has prerequisites, these prerequisites are covered in the same Web page in a compelling manner. It should be possible for an instructor to simply tell students "go through

the module, make sure you do the practice questions, and turn in your answers to the assignment (which is the last section of the module)". In addition, the prerequisite sections also include open questions (i.e., whose answers are not provided), which an instructor may add as a preamble to the assignment.

**Previous Uses –** This assignment (in fact the entire pedagogic module that it is a part of) has been used successfully in two offerings of a 300-level undergraduate Computer Science course at the University of Hawai'i at Mānoa [5]. It is currently being used in a 400-level course in the Fall 2020 semester. The 300-level course is an Operating Systems (OS) Principles course, in which students learn what OSs do and how they do it through both non-programming and programming assignments. This assignment, which is part of a Parallel Computing module toward the end of the course, then turns students' attention to application performance issues given underlying hardware (and the OS managing it). The 400-level course is a Concurrent and High Performance Programming course in which students develop a lot of multi-threaded code for interactivity and performance goals. While a lot of the course focuses on programming, it is important that students also reason about performance at a more general/abstract level. This is where the module is being used, pointing out performance goals/issues that students then implement in code for a subsequent programming assignment. The authors have not yet been able to use the assignment in lower-division courses, but other instructors who do teach those courses are being contacted.

This assignment has also been used to train individual students before they participate in parallel computing and high performance computing research projects at the University of Southern California.

**Evaluation Results –** Quantitative and qualitative data was collected about the student experience in the above university courses. Information collected included: answers to pre- and post-knowledge tests, questionnaires about overall experience and perceived self-learning, logs of student use of the Web page (and in particular of their use of the simulators in the tabs that precede the assignment), and assessment of learning via exam questions. Each semester, the assignment was significantly evolved (in terms of presentation, structure, and content) based on student feedback. Student feedback was overall very positive.

In the Fall '19 semester (55 students), the last time the assignment was used, 89% of the students were engaged in the material as seen in their active use of the simulation activities (i.e., the active learning opportunities leading up to the assignment). Results on the final exam questions for the assignment's learning objectives showed that most students have achieved these objectives.

A key question is whether students enjoyed the overall experience, which we can determine based on questionnaire answers. Although 45% of students found the assignment "somewhat difficult" or "very difficult", 100% of them found it "useful" or "somewhat useful". 87% of the students answered "yes" to the question "Are you interested in learning more about Parallel and Distributed Computing?". In university course evaluations, student comments included "I liked the simulation. It was a nice addition to visually see as well as check my work", "Getting exposure to this type of content as it wasn't really brought up anywhere else", and "Love the visuals". Note that some of the above questions were more about the overall pedagogic approach than about the details of this particular assignment, and that feedback changed (for the better) with each version of this assignment. In fact, the assignment as it is available on-line has been modified significantly since the last time it was used in the classroom.

## III. Simulation of Life Evolution (Gonzalez-Escribano, Torres)

The second assignment was used in a Parallel Computing course to teach the approaches to 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 life evolution, where protozoa learn to survive getting food in a competitive environment. The program is designed to be simple, easy to understand by students, and to include specific parallelization and optimization opportunities. Although there is a quite direct parallel solution in the three programming models, the program has plenty of opportunities for further improvements. It extends the ideas of two previously presented assignments, in order to use different synchronization and communication structures, load balancing strategies, and code optimizations. This assignment has been successfully used in parallel programming contests during a real course using the performance obtained by the students' code as a measure of success.

### 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 a course of Parallel Programming that introduces the basics of OpenMP, MPI, and CUDA or OpenCL. Two 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 [6]. Although this kind of assignment can be used to teach a single programming model, the series of them can also show which concepts and ideas can be reused across different models, and which can not, 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.

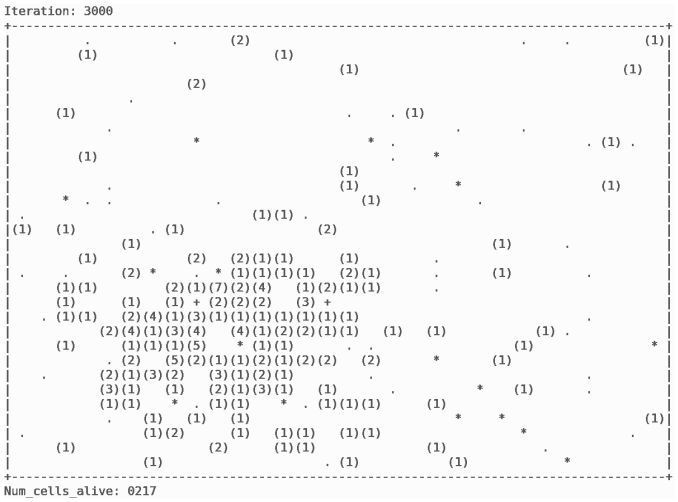


Fig. 2. Graphical representation of the culture space at a given step, provided as output by the evolution simulation program. The numbers in brackets represent the number of protozoa at the different points of the 2D surface. Symbols represent the amount of remaining food in other positions.

This new assignment targets different synchronization and communication structures and different heterogeneous programming techniques in GPUs. Some examples include variable size communications, clear opportunities for CPU and GPU computation overlapping, and the use of fixed-point arithmetic to solve concurrency problems. 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 all the three programming models considered.

The idea is based on a simulation of the evolution of bugs feeding on bacteria in the muddy bottom of a stagnant pool of water [7]. The culture space is described by a 2-dimensional matrix where food values are stored. Some new food is randomly spread during each iteration, either evenly or with a higher concentration is a specific zone. Some protozoa are randomly located at the beginning. The organisms are described by genes indicating the probability of advancing straight-forward or turning during their search for food. The food found by an organism is accumulated, and spent each iteration to keep it alive and moving. When an organism reaches maturity (a given number of simulation steps alive), if they have accumulated enough food, they may split in two new organisms. However, random mutations occur, and the genes of the descendants slightly change, modifying their moving behaviour.

During the simulation, we observe how the population grows and decreases, how the organisms survive or starve. After enough iterations, the genes of the survivors present different features depending on the scenario parameters. The simulation results are determined by random seeds provided as arguments. Thus, they are reproducible. The input parameters can be chosen to generate specific situations with different growing and shrinking population sizes, different load distributions across the culture, different ratios of concurrency problems when protozoa collide in the same cell to get food, etc.

### *B. Using the assignment*

As in the previous assignments of this series, the provided material includes a sequential code in C language, a test-bed of input parameters, 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 [6], [8]. The judge configuration is done by simply providing tuples of input arguments and expected output results, representing the scenarios chosen by the teacher. The tool executes the programs in a real parallel system and measures the total performance to rank the students. The sections of the sequential code that should be parallelized and optimized by the students are clearly marked, skipping arguments processing, scenario initialization, OpenMP/MPI/CUDA setup, time measuring and results output. Thus, the original codes can be directly compiled and run by the students, or submitted to the judge tool even before starting to parallelize them.

The assignment was used in an elective Parallel Programming course in the third year of Computer Engineering degree at the University of Valladolid (Spain). The students 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 70 students enrolled. The degree of participation was high, with more than 12,000 requests for program execution on 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. In a Likert scale from 1 to 5, the mode value is 5, and the average is approximately 4. They think that this practical work illustrates the main concepts of the course and provides opportunities to deepen in the subject. For example, some students optimized their codes to obtain a CUDA program up to 10 times faster than the best code of other students that passed the minimum level.

The results also indicate that this third Peachy Parallel Assignment in the series requires up to 50% more working time than the previous ones [1], [3], mainly due to the debugging of dynamic-memory management errors.

### *C. Concepts covered*

The stages of each simulation step of the program are: (1) Randomly spreading new food on the culture matrix; (2) Moving the organisms according to their genes; (3) Harvesting food from the culture; (4) Eliminating dead organisms from the list; and (5) Using reductions to collect statistical data. The output of the program is a set of statistical data such as the maximum number of organisms alive in any iteration, the age of the oldest organism observed, the maximum level of food accumulated in a cell of the culture, etc. If desired, the

program can also write a text-mode graphical representation of the culture space after each simulation step. This can be used to visualize the evolution of the simulation (see Fig. 2) and the genetic code of the protozoa at the end.

The basic concepts covered when using the OpenMP version of the assignment are parallelization of loops, reductions, atomic operations and scheduling. In the MPI version, the students work with array partitions, variable size communications, reductions, asynchronous operations, communicators and load balance. For GPU programming, the main ideas are embarrassing parallel kernels, thread-block geometries and sizes, non-trivial atomic operations, simple reductions, minimizing communication operations, and overlapping of kernels and host computing. Taylor approximations are used in the sequential program to compute trigonometric functions in a form that is portable to GPUs. The program also shows how to use fixed-point arithmetic to avoid precision and concurrency problems in all models.

Several advanced optimizations can also be discovered and applied. Examples include better dynamic memory management, code reorderings to allow easier parallelization or better operation overlapping, clever load-balancing techniques adapted to the scenario features, the use of proper loop scheduling clauses in OpenMP, thoughtful 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 interactions of particles or agents with an environment represented with a grid. The different parts of the code inside the simulation iteration present different problems that can be solved incrementally. The students can use similar parallelization approaches and optimization techniques in many other programs and situations.

The assignment can easily be adapted and modified by the teacher to include new simulation scenarios, different food spreading policies, or different protozoa features, such as movement, replication or gene mutation functions. Different data structures can be used to store the protozoa lists, resulting in different algorithms and techniques to manage them. More sophisticated protozoa that can explore neighbor cells and react to the information obtained can be devised for a more complex and realistic simulation with a different communication structure. Finally, better graphical and online interfaces can be devised to enrich the learning experience.

## References


[1] 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 *IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC 2019)*. Denver (CO), USA: IEEE, 2019.

[2] 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.

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

[4] 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.

[5] R. Tanaka, R. Ferreira da Silva, and H. Casanova, "Teaching Parallel and Distributed Computing Concepts in Simulation with WRENCH," in *Workshop on Education for High-Performance Computing (EduHPC)*, 2019.

[6] A. Gonzalez-Escribano, V. Lara-Mongil, E. Rodriguez-Gutiez, and Y. Torres, "Toward improving collaborative behaviour during competitive programming assignments," in *IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC 2019)*. Denver (CO), USA: IEEE, 2019.

[7] A. K. Dewdney, "Simulated evolution: wherein bugs learn to hunt bacteria," *Scientific American*, vol. 260, no. 5, pp. 138–141, May 1989.

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


## Appendix: Reproducibility

### A. Reasoning about Multi-core Application Performance

This assignment and the pedagogic module that it is a part of has been used successfully in two offerings of a 300-level undergraduate Operating Systems (OS) Principles course at the University of Hawai'i at Mānoa. It is currently (Fall 2020) being used in a 400-level course on Concurrent and High Performance Programming. It has also been used to train individual students before they participate in parallel computing and high performance computing research projects at the University of Southern California.

The entire module, including supporting simulations, runs within a webpage. Thus, instructors adopting this assignment do not need to worry about provisioning hardware or installing software.

### B. Simulation of Life Evolution

The 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 *heracles*, a server with four AMD Opteron 6376 @ 2.3Ghz CPUs, having a total of 64 cores and 128 GB of RAM.

The machine used in the CUDA/OpenCL contests is *hydra*, 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 an 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 and the input parameters of the test-beds for the students. Other test-beds used by the on-line judge during the contest are also provided.

The results of the contests are publicly available until the start of the next semester at http://frontendv.infor.uva.es.