

Peachy Parallel Assignments (EduHPC 2025)

Clara Almeida
University of California, Santa Cruz
Santa Cruz, CA, USA
cjalmeid@ucsc.edu

Elizabeth Shoop
Macalester College
St. Paul, MN, USA
shoop@macalester.edu

Diego García-Álvarez
Universidad de Valladolid
Valladolid, Spain
diego.garcia.alvarez@uva.es

Arturo Gonzalez-Escribano
Universidad de Valladolid
Valladolid, Spain
arturo@infor.uva.es

David Guerrero-Pantoja
University of California, Santa Cruz
Santa Cruz, CA, USA
dguerr15@ucsc.edu

Cameron Maloney
California Polytechnic State Univ.
San Luis Obispo, CA, USA
cmalon04@calpoly.edu

Maria Pantoja
California Polytechnic State Univ.
San Luis Obispo, CA, USA
mpanto01@calpoly.edu

Silvio Rizzi
Argonne National Laboratory
Lemont, IL, USA
srizzi@anl.gov

David P. Bunde
Knox College
Galesburg, IL, USA
dbunde@knox.edu

Abstract

Peachy Parallel Assignments are high-quality assignments that require students to practice concepts in parallel and distributed computing. They are selected competitively and published in the Edu* workshops to provide instructors with inspiration and easy-to-adopt assignments. The assignments must have been successfully tested with real students, easy to adopt by other instructors in a variety of contexts, and “cool and inspirational” for students completing them.

This article presents three Peachy Parallel Assignments selected for presentation at EduHPC 2025. The first is simulation of the growth of “fairy rings”, a biologically-motivated variation of the Game of Life. The second assignment asks students to simulate flooding over uneven terrain and in the presence of active rainfall. The third assignment has them implement the softmax function in parallel, motivated by applications in deep learning.

CCS Concepts

• **Social and professional topics** → **Computing education**; • **Computing methodologies** → **Parallel algorithms**.

Keywords

Peachy Parallel Assignments, Parallel computing education, Parallel programming, Parallel structured grid pattern, Softmax, Inquiry-based learning, Parallel random number generator, OpenMP, MPI, GPGPU programming

ACM Reference Format:

Clara Almeida, Elizabeth Shoop, Diego García-Álvarez, Arturo Gonzalez-Escribano, David Guerrero-Pantoja, Cameron Maloney, Maria Pantoja, Silvio Rizzi, and David P. Bunde. 2025. Peachy Parallel Assignments (EduHPC 2025). In *Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC Workshops '25)*, November

16–21, 2025, St Louis, MO, USA. ACM, New York, NY, USA, 7 pages. <https://doi.org/10.1145/3731599.3767388>

1 Introduction

Assignments are an important part of computer science courses since they are often where students get the majority of their time practicing the course material. Motivating assignments encourage students to increase this time and potentially also improve the perception of the field as student excitement is shared with friends and family.

Designing assignments that have this affect is not easy, however; new assignments often require significant development time and there is a risk that seemingly attractive assignment ideas are less motivating when implemented.

Peachy Parallel Assignments are an effort to share the workload and reduce the risk of creating excellent assignments for topics in parallel and distributed computing (PDC) and scientific computing. Submissions of potential Peachy Assignments are solicited from the community for the EduHPC and EduPar workshops and then judged based on the following criteria:

- **Tested:** The assignment should have been tested with real students.
- **Adoptable:** The assignment should include materials so others can adopt it. Ideally, it should cover widely-taught concepts using common languages/libraries.
- **Cool and inspirational:** Students should be motivated to work on the assignment. Ideally, it should be something they are inspired to share the results with friends.

The best assignments are then published in the workshop proceedings (e.g. [4, 11]) and posted to the Peachy Parallel Assignments webpage (<https://tcpp.cs.gsu.edu/curriculum/?q=peachy>).

This paper presents three Peachy Parallel Assignments, which were selected for the EduHPC 2025 workshop. The first assignment is a simulation of fairy rings, rings of mushrooms that can form spontaneously as the descendants of a mushroom grow out in all directions from a starting point. This assignment models a natural phenomenon and is graphical to make it more interesting for students. It is also similar to the classic Game of Life problem, but with



This work is licensed under a Creative Commons Attribution 4.0 International License. *SC Workshops '25, St Louis, MO, USA*

© 2025 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-1871-7/2025/11

<https://doi.org/10.1145/3731599.3767388>

less material online, and has the added advantage of motivating a discussion of parallel random number generation. The second assignment simulates flooding, with rainfall spreading water over a terrain and then elevation differences causing it to redistribute as it seeks lower areas. Several recent severe floods unfortunately make this a topical problem and it also includes a graphical visualization. The third assignment asks students to approximate the reduction of the softmax of an array, an operation that used for deep learning. The assignment inherits its appeal to students from this connection to AI.

2 Fairy Rings

Our first assignment was designed for undergraduates in a dedicated PDC course with some experience using OpenMP. Simulations based on the structured grid parallel computational pattern are approachable by these students. Our first assignment uses a structured grid to model the growth of Mushroom Fairy Rings. The cells in the grid can be in one of several states, are updated over iterations representing time steps, and change values based on the probability of a state change occurring. This assignment thus goes beyond the simple game of life cellular automata, providing students with the challenge of properly using a random number generator inside parallel loops. Students parallelize a base sequential version using OpenMP, then conduct experiments to determine its strong and weak scalability. Scaffolding for success is provided in the form of additional freely available prerequisite class activities.

2.1 Motivation and Problem Description

One of the parallel computational patterns described in [2] and on the Our Pattern Language site is the structured grid for simulations [9], which includes the stencil pattern and the use of double buffering, where the initial 2D grid is updated in a new copy by applying rules to each cell that update its value based on the neighboring cells around it in the grid. Once all new values are computed, the new grid is copied back into the original grid and a new iteration begins (a technique known as double buffering).

These simulations are also referred to as cellular automata because the grid of cells are updated at each iteration in an automated fashion. The most famous of these is John Conway's Game of Life [8], where each cell is initialized to be dead or alive and each cell changes value to one of these 2 states based on the values of its 8 neighboring cells. When combined with Monte Carlo techniques that introduce the probability of a state change occurring, these structured grid computations become stochastic cellular automata, which have been used in many simulations of natural phenomena.

In our experience, undergraduate students find parallelizing this computation understandable, because it is easy to envision how computing each cell's new value is independent of the other cells when using the double buffering technique: each thread can write to one cell in the new grid and the old grid is read-only by the threads.

Inspired by an assignment described by Angela and George Shiflet [16, 711–719], we present a simulation of the growth of Mushroom Fairy Rings, a natural phenomenon shown in Figure 11a. Fungi are organisms that produce spores as their mechanism of reproduction: each spore can germinate into a new fungus. The

fungi rely on nutrients in the ground around them to grow. Each fungus sprouts small threads underground called mycelium, which grow outward in a circular pattern. Some fungi are invisible or not easily seen as they grow at the ends of the mycelium. The mushroom structure that we are most familiar with is formed by the organized growth of hyphae from the mycelium. The mushroom's hyphae eventually become spore bearing, causing the growth of more mushrooms in a circular pattern. This radial form of growth enables the mushrooms to grow into new areas searching for nutrients, leaving the inside of the circles without nutrients, or inert. Hyphae in the center of the forming circles decay and die out; those on the outside form irregular rings. Under the right conditions, these rings can continue to grow for long periods of time.

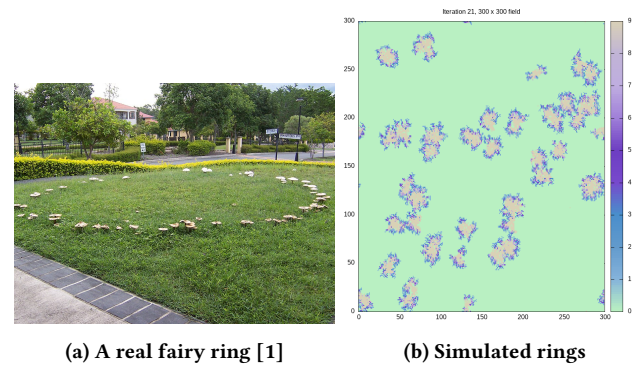


Figure 1: Fairy Rings: real and simulated in this assignment.

The ground is modeled as a 2D grid of cells that initially contain spores. Each cell is examined during several iterations that mimic the marching of time. The cells have values based on the state of the mushroom at the time of the iteration. These states correspond to the values shown in Table 2a (adapted from [16]).

The eight cells around a cell (the neighborhood, or stencil pattern) are examined to determine whether an EMPTY cell could become a SPORE because a neighboring cell is a spore. Cells going from SPORE to YOUNG to MATURING and so on through the states are determined by rules. The difference between this mushroom fairy ring model and the classic game of life is that the simulation needs probabilities for the chance of a cell to go from one state to another in some cases (see Figure 2).

The sequential version of the code can be run immediately by students so that they can see what the correct output should be. Figure 1b depicts an available graphical output of the program. The coloring of the cells is mapped to the states in Figure 2a. Drawing this output requires gnuplot on Linux machines; if that is not available, students can get text files as output. Either output helps students to debug as they add parallelization. There is an option to run an animation that displays the states at each iteration.

All options are described in the assignment, which is found as part of a GitHub repository of PDC assignments maintained by the CSinParallel project [5]. The code is decomposed into several directories and files, which are explained in the README, which constitutes the assignment itself.

Name	Value	Description
EMPTY	0	Empty ground
SPORE	1	Containing a spore
YOUNG	2	Young hyphae that haven't formed mushrooms yet
MATURING	3	Maturing hyphae; still haven't formed mushrooms
MUSHROOMS	4	Older hyphae with mushrooms
OLDER	5	Older hyphae with no mushrooms
DECAYING	6	Decaying hyphae w/o nutrients
DEAD1	7	Newly dead hyphae
DEAD2	8	Longer dead hyphae
INERT	9	Area where plants cannot grow

(a) Possible States

If EMPTY and one neighbor is a spore, the cell becomes a SPORE with probability `PROB_SPREAD`.
 If SPORE, it becomes YOUNG with probability `PROB_SPORE_TO_HYPHAE`.
 If YOUNG, it becomes MATURING.
 If MATURING, it becomes MUSHROOM with probability `PROB_MUSHROOM`, otherwise goes to OLDER.
 If MUSHROOM or OLDER, it becomes DECAYING.
 If DECAYING, it becomes DEAD1.
 If DEAD1, it becomes DEAD2.
 If DEAD2, it becomes INERT.
 Once INERT, it stays that way.

(b) Rules for Transition

Figure 2: States and Rules for Fairy Ring Simulation

2.2 Concepts covered

Given a reference sequential implementation, students complete OpenMP versions of the simulation of the growth of these rings. They practice writing OpenMP pragmas for nested for loops that iterate over the 2D grid and must also use a random number generator library to generate numbers between 0.0 and 1.0 for each cell at each iteration to determine the next cell value based on both the neighborhood around the cell and the probability of a state change occurring. Starting points are provided for two random number generators (RNG): the C++ random library and the `trng` parallel random number generation library [3]. The first of these is non-deterministic and the second is deterministic; we deliberately include both to support class discussion about issues arising with parallel random number generation.

The assignment contains a section about profiling code using `gprof` on a Linux machine. Because the code is structured into functions with a single purpose, students can examine profile results to decide which functions to work on parallelizing first. Parallelizing nested for loops on a flattened 2D grid is the primary learning activity, which is surprisingly tricky because of the need to generate a new random number for each cell inside the nested loop: some guidance is given for ensuring that each thread works on a column. An available prerequisite activity introduces this. After debugging and ensuring that their code is correct, students then

run experiments for strong and weak scalability (an additional prerequisite activity provides guidance with this). This application is quite scalable under certain problem sizes, which enables students to determine when the parallelized versions are most effective.

2.3 Using the assignment

This assignment has been used in a dedicated PDC course at the undergraduate level for several years in various forms. It started as a more open-ended project and has transformed into an assignment with guidance and a series of prerequisite activities that provide scaffolding for successful completion. Students work through these publicly available activities in class before getting this assignment:

- (1) OpenMP 'patternlet' examples in Chapter 2 of [19].
- (2) Parallel RNG using `trng` in Chapter 3 of [19].
- (3) Observing Scalability of Trapezoidal Integration [17].
- (4) Parallel Game of Life with OpenMP and Randomness [18].

Aside from the inclusion of the `trng` library, the concepts in these types of activities are typically taught in a PDC course when introducing OpenMP. We prefer an emphasis on parallel patterns (a theme in the freely available PDC textbook [19]). Activity 3 above also uses reading about scalability from another freely available book aimed at beginning PDC students [6]. The `trng` library requires installation. We use a departmental server for student work so that they have access to a machine that has enough cores to run the experiments and this assignment and see scalability. Instructors can eliminate `trng`, however, and it is still a good assignment using the C++ random library. Emphasis on detailed scalability experimentation could also be decreased. An OpenACC version has also showed scalability on GPUs. This is suggested in the assignment and could be another choice for instructors.

3 Flood Simulation

Next, we present an assignment successfully implemented in a third-year Parallel Computing course of a Computer Engineering degree program. The assignment is part of a series, with a different problem posed each academic year since 2017/2018 to illustrate the conceptual and technical differences of using different parallel programming models. The problem chosen for this year implements a flood simulation. Cloud fronts move across a scenario, dropping water. The simulation computes the flow of water from the highest ground to the lowest ground, leaking out at the scenario boundaries or accumulating in sinks and dams to form pools and lakes. The assignment addresses foundational concepts, such as race conditions, reductions, collective operations, and point-to-point communications. It also offers critical choices related to cache-aware programming or using atomic operations vs. more memory accesses with ancillary structures. The supporting materials for previous assignments in this series are available at <https://gamuva.infor.uva.es/peachy-assignments/>.

3.1 Idea and context

Idea. The approach to parallelize a program is influenced by the programming model employed. Students should learn to use and practice different models to understand the differences between them and acquire the required skills to fully exploit modern heterogeneous platforms. The Computer Engineering degree at

the University of Valladolid offers an elective course in parallel programming in the third year. The course covers fundamental concepts of three programming models: OpenMP for shared memory, MPI for distributed memory, and CUDA/HIP/OpenCL for GPU programming.

The assignments proposed in previous editions of the course [7] have proven to be effective in teaching these programming models and understanding the portability of approaches and techniques. Students who participate in these assignments gain a deeper understanding of similarities and conceptual shifts between models, developing critical analysis skills to determine the most suitable programming model and solutions for each type of problem.

The assignment problem. Floods can be modeled by simulating the accumulation of rainwater and its flow across a discretized surface. The height of the ground for each grid point can be represented with a value in a matrix. The assignment provides four predetermined ground scenarios generated with a function: a mountain terrain, a wide valley with hills, and two versions of the same valley with dams of different positions and heights. The program can be easily adapted to read real terrain elevation data from a file. However, the read time for big scenarios with enough work load to properly exploit and show the intended parallelism effects can be huge. Thus, this is not the default option. We include a model to parametrize the generation of random collections of clouds that form fronts of a given size and depth. Each cloud is represented with a radius, direction, velocity, rain intensity, etc. Individual clouds with fixed parameters can also be added. These data are provided as program arguments. Thus, students can easily explore the effects of different scenarios, and the teacher can select specific configurations for testing solutions. On each simulation step, the clouds advance across the scenario, dropping rain and increasing the water level of the grid points below the cloud. This is followed by a redistribution of water. For each matrix position, the amount of water that flows to the four neighboring positions is computed proportionally to the relative differences of their height and water level. The positions in the boundaries spill part of the water out of the matrix. In a second stage, each position is updated with the water coming from neighbors. This process is repeated for each iteration. This simulates the movement of water towards the lower areas of the terrain until the system reaches equilibrium or a maximum number of iterations has been computed. Figure 3 illustrates an example of the simulation results. Simple metrics obtained during the simulation can be used to verify the results, such as the number of iterations executed, the total amount of rainfall, the final amount of water in the matrix, and the iteration that presented the maximum flow of water across two matrix positions, etc.

3.2 Concepts covered

Concepts. This assignment addresses different parallel programming concepts introduced during the course. The program presents multiple parts that can be parallelized, facing different problems. The ground heights calculation is embarrassingly parallel. The generation of clouds uses a linear-congruent pseudo-random number generator that can be used in parallel with a provided skip-ahead function [12]. Parallelizing the rest of the loops requires identifying and solving read, write, and update race conditions. Critical regions,

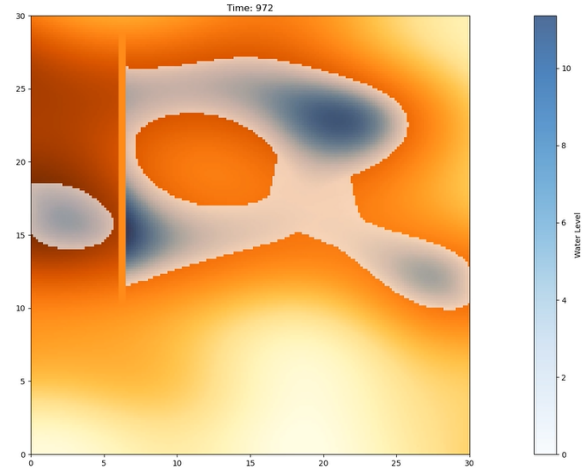


Figure 3: Example of the simulation result after a stormy cloud front has dropped a large amount of water in a scenario with a dam.

atomic operations, or reduction transformations are needed. The water redistribution presents a pattern that is the reverse of a stencil operation. Data that is computed on each position is used to update neighbors. In the provided code, these data dependencies are avoided using a two-stage process. In the first one, an ancillary structure temporarily stores the spillage of water to each neighbor. In the second stage, each cell reads from the ancillary structure the water received from different neighbors. The accesses to this structure can be optimized by reordering the layout to better exploit caches, or coalescence in the case of GPUs. Moreover, there is an interesting performance tradeoff between the use of this structure and its elimination to directly use atomic operations to update neighbors. Loop or kernel fusion can be used in the last steps, and code extraction can be used to simplify a reduction to obtain the iteration with the maximum water flow across matrix positions. Using MPI, the terrain mesh must be decomposed and mapped to several processes, requiring a two-stage neighbor communication pattern across partition boundaries. Load-balancing techniques can be used in specific parts to enhance performance.

Variants. The input parameters of the program can be used to generate different scenarios with different probabilities of race conditions, load imbalance situations, and computation to communication ratios. This is achieved by changing the height map, generating fronts with more or fewer clouds, and different cloud radii to force them to overlap during the rainfall step, etc.

3.3 Using the Assignment

Teaching context. This assignment assumes that students have been engaged in previous courses that include fundamentals of operating systems, concurrency, and the C programming language. This academic year, the total number of students enrolled in our course was 45. Forty-two students participated in all of the scheduled practical sessions, working in groups of two. Students are introduced to the flood simulation problem, and are provided with

a basic sequential code, a handout, and examples of parameters to generate small illustrative scenarios. The parts and functions of the program that can be modified are clearly indicated in the sequential code. The sequential version can also generate an output to create an animation of the simulation process.

Each programming model is presented through a combination of lecture and laboratory sessions during three weeks, solving problems that gradually introduce and exemplify the main concepts. The last problems are directly related to the assignment, for example, showing how to parallelize the random number generation. Afterwards, each group has an additional week to solve the flood simulation assignment with the corresponding parallel programming model. Finally, there is an exam with questions related to the assignment solutions obtained, to assess their understanding and degree of involvement in the development of the solution.

Tools. The only software required is an OpenMP-compatible C compiler, an MPI library, and the CUDA, HIP or OpenCL toolkit. Solutions based on OpenMP or MPI can be tested on a multi-core computer, while the third model requires a CUDA, HIP or OpenCL compatible graphics card. The best experience, however, is obtained by using a shared cluster, which allows students to fairly compare their performance improvement results. The week dedicated to work in the assignment is organized as a contest. We use an online judge system to manage the queues that send the students' proposed solutions to our cluster servers (see the appendix for more information on this topic), and to evaluate and classify the solutions according to their correctness and efficiency.

Results. This academic year, due to organizational and personal problems, it was not possible to organize a contest for the CUDA part. There were a total of 3,499 submissions during the OpenMP and MPI contests, with an average of more than 166 submissions per pair of students. This is similar to previous years, as most requests were typically generated during the CUDA contest. Many students do not have a high-end GPU on their laptops or desktop computers for local tests, and the CUDA contests usually present a higher number of executions with errors due to small bugs that do not crash the execution, and are more difficult to find. In personal interviews at the end of the course, the interviewed students agreed that the assignment exemplifies the main concepts of the course and provides clear opportunities to delve into the subject. For example, some students optimized their MPI codes to execute more than 40% faster than the baseline parallel version. They achieved optimizations not only in the code inside the sequential processes, but in the basic neighbor communication schemes.

4 Scalable Softmax for Efficient Attention

As large-scale deep learning models become integral to scientific discovery and engineering applications, it is increasingly important to teach students how to implement them efficiently and at scale. This section presents a coding assignment that focuses on optimizing the Softmax function, a central component of many deep learning models, including attention mechanisms in transformer models. The assignment is designed for an undergraduate level Distributed Computing course (CPE 469, 10-week quarter system), and

tailored to students with little or no prior experience in machine learning.

This assignment is one of seven designed to reinforce the foundational concepts of parallel programming. It was developed as part of an inquiry-based learning approach [15], [21], encouraging students to actively investigate, experiment, and discover solutions to real-world challenges. The assignment introduces essential deep learning concepts, then guides students through identifying independent tasks within the Softmax computation so they can implement parallel solutions using OpenMP and CUDA.

By integrating modern AI workloads into an HPC curriculum, this work equips students with both the conceptual understanding and practical experience needed to build scalable solutions in scientific computing.

4.1 Motivation

Inquiry-Based Learning (IBL) is widely recognized as an effective pedagogical approach, particularly in STEM education, when thoughtfully implemented. It enhances students' understanding, critical thinking, and engagement. However, it also presents challenges since IBL requires substantial preparation, scaffolding, and time, and students may struggle without sufficient guidance especially in the early stages. IBL is a form of active learning that starts by posing questions, problems or scenarios. It contrasts with traditional education, which generally relies on the teacher presenting facts and their knowledge about the subject.

To address these issues, we designed an assignment that is both highly relevant and accessible. It centers on a topic students are eager to learn about, machine learning, yet it does not require prior knowledge of deep learning models, making it suitable for undergraduates. The assignment starts with a manageable implementation that gradually increases in complexity, providing a smooth learning curve, asking students at every step how they will improve the performance. Along the way, it introduces foundational GPU concepts such as shared vs. local memory and the importance of memory optimization through practical, low-level programming.

Main Idea. In this assignment, students explore the evolution of the Softmax function from its traditional formulation to high performance variants of distributed Softmax (which will allow the introduction of FlashAttention if instructors desire). The goal is to bridge theoretical understanding with practical implementation while introducing students to real-world challenges in high-performance computing (HPC) for machine learning.

Concepts Covered. Students tackle key computational challenges including, memory bandwidth limitations, numerical stability, and inter-node communication overhead. To address these, students apply core HPC techniques such as Parallel reduction, Memory coalescing, Task decomposition, and Kernel fusion. The assignment is structured around progressive implementation and GPU performance analysis, helping students reason about hardware-aware optimization strategies.

Target Audience. Sophomore-level undergraduate students in Computer Science, enrolled in an introductory course on Distributed and Parallel Programming.

```

8 // Numerically stable softmax
9 void softmax(float* input, float* output, int n) {
10 // Step 1: Find maximum for numerical stability
11 float max_val = -FLT_MAX;
12 for (int i = 0; i < n; i++) {
13     if (input[i] > max_val) {
14         max_val = input[i];
15     }
16 }
17 // Step 2: Compute exp(x - max) and sum
18 float sum = 0.0f;
19 for (int i = 0; i < n; i++) {
20     output[i] = expf(input[i] - max_val);
21     sum += output[i];
22 }
23 // Step 3: Normalize
24 for (int i = 0; i < n; i++) {
25     output[i] /= sum;
26 }
27 }
28 }
29 }

```

Figure 4: Safe SoftMax Calculation 3-Pass (No Optimization)

Prerequisites. Prior experience with system-level programming in C/C++. Also recommended is familiarity with OS-level threading concepts (e.g., from an Operating Systems course).

Strengths. The Softmax function is conceptually simple and widely known, requiring minimal scaffolding to introduce. The topic is highly relevant and modern, sparking student interest and motivation. The assignment provides a hands-on, approachable way to introduce GPU optimization concepts without requiring prior knowledge of deep learning.

Observed Impact. The assignment was well received—students found the topic engaging and accessible, and appreciated its connection to cutting-edge developments in AI and HPC.

4.2 Method

The Softmax function takes a vector of real numbers and transforms it into a probability distribution

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}} \quad (1)$$

Where x is the input vector and x_i is the i th component. Equation (1) puts strain on the maximum range our floating point values can hold. The maximum value a 16-bit floating point can hold is 65,504, so if x is greater than 11, this will cause an overflow and wreck our calculation. So we implement the softmax as follows.

$$\text{softmax}(x_i) = \frac{e^{x_i - m}}{\sum_{j=1}^n e^{x_j - m}} \quad (2)$$

Where m is the maximum value. Calculating the maximum and the sum of elements in an array for the softmax function typically requires three separate passes, each with its own loop, as illustrated in Figure 4. This sequence is challenging to parallelize efficiently due to the data dependencies between these passes.

The optimization approach proposed in [20] and [10] involves dividing the input vector into smaller partitions (tiles) that can be processed in parallel. By computing the maximum and partial sums within each tile and updating global values as needed (illustrated in Figure 5), this method enables two key parallelization techniques: loop fusion and tiling. Loop fusion reduces overhead by combining multiple passes into a single loop, while tiling improves memory locality and load balancing across threads.

```

10 void softmax_parallel(float* input, float* output, int n) {
11     float max_val = -FLT_MAX;
12     float d = 0.0f;
13     #pragma omp parallel for reduction(max:max_val) reduction(+:d)
14     for (int i = 0; i < n; i++) {
15         if (input[i] > max_val) {
16             d = d + expf(max_val - input[i]) + 1.0;
17             max_val = input[i];
18         }
19         // Add the contribution of this element to the sum
20         d += expf(input[i] - max_val);
21     }
22 }
23 //Parallel normalization
24 #pragma omp parallel for
25 for (int i = 0; i < n; i++) {
26     output[i] = expf(input[i] - max_val) / d;
27 }
28 }
29 }
30 }

```

Figure 5: Safe SoftMax Calculation 2-Pass Fused (Optimized)

4.3 Using the assignment

Materials. The following links provide a complete set of instructional materials for this assignment. This includes:

- A full slide deck [14] explaining the context of the problem, computational challenges, and desired optimizations.
- A hands-on tutorial delivered through a Google Colab notebook [13], which includes a working solution with starting code that allows students to run and modify code without the need to install compilers or set up a development environment; only a Google account is required.

Instructors have the flexibility to decide how much of the code to provide upfront, making it easy to adapt the assignment for different levels of scaffolding or instructional goals.

Extensions and Variations. This assignment can be extended to include vector dot products and optimized in ways similar to FlashAttention techniques, as discussed in recent research such as Flash-D [20]. This opens the door for further exploration of memory-efficient attention mechanisms and fused kernel operations.

Students can learn from different versions of the acceleration: We provide versions in basic C, OpenMP, Cuda (Nvidia GPU), and SYCL (multivendor accelerator). Students start with C on a single CPU core to provide a performance baseline and observe the time of each step. Adding OpenMP pragmas parallelizes the same code across all cores in a multicore processor with minimal changes, to demonstrate shared-memory scaling. The CUDA version targets NVIDIA GPUs, using threads and on-chip shared memory. The SYCL implementation lets the same code run on GPUs, CPUs, or other accelerators.

Acknowledgments

E. Shoop thanks the students of the Macalester Parallel and Distributed Computing course who took on the fairy ring assignment and enabled her to improve it over several years. D. Garcia-Álvarez and A. Gonzalez-Escribano developed the flood simulation assignment in the context of the GAMUVa group (<https://gamuva.infor.uva.es/>), and it has been partially supported by Vicerrectorado de Innovación Docente y Transformación Digital de la Universidad de Valladolid, Proyectos de Innovación Docente PID2425_91, and by the NVIDIA Hardware Grant Program providing GPU devices used during the assignments. Development of the softmax assignment by C. Almeida, D. Guerrero-Pantoja, C. Maloney, M. Pantoja, and S. Rizzi was partially supported by the U.S. Department of

Energy Office of Science and the National Nuclear Security Administration and by Argonne National Laboratory. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

- [1] Anonymous. 2008. *Fairy Circle on a suburban lawn. Forest Lake, south of Brisbane in Queensland*. <https://commons.wikimedia.org/w/index.php?curid=3352618>
- [2] Krste Asanovic, Ras Bodik, Bryan Catanzaro, Joseph Gebis, Parry Husbands, Kurt Keutzer, David Patterson, William Plishker, John Shalf, and Samuel Webb Williams. 2006. *The landscape of parallel computing research: A view from Berkeley*. <https://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.pdf>
- [3] Heiko Bauke and Stephan Mertens. 2007. *Tina's Random Number Generator Library*. <https://www.numbercrunch.de/trng/>
- [4] H.M. Bucker, J. Schoder, X. Suo, and D.P. Bunde. 2025. Peachy Parallel Assignments (EduPar 2025). In *Proc. 15th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar)*.
- [5] CSinParallel. 2023. *Repository of PDC Assignments*. <https://github.com/csinparallel/pdc-assignments>
- [6] CSinParallel project members. 2022. *Parallel Computing for Beginners*. <https://www.learnpdc.org/PDCBeginners2e/>
- [7] GAMUVa group. 2018-2023. *Peachy Parallel Assignments*. on <https://gamuva.infor.uva.es/peachy-assignments/>.
- [8] M Gardner. 1970. The Fantastic Combinations of John Conway's New Solitaire Game Life. *Scientific American* 223, 4 (1970), 120–123.
- [9] Katya Gonina, Hovig Bayandorian Shepherds, and Erich Strohmaier. 2025. *OPL Structured Grids*. https://patterns.eecs.berkeley.edu/?page_id=498
- [10] Matthew Gunton. 2025. Online Softmax to Flash Attention — and Why it Matters. <https://medium.com/data-science-collective/online-softmax-to-flash-attention-and-why-it-matters-9d676e7c50a8>.
- [11] A. Lazar, E. Scheelk, E. Shoop, and D.P. Bunde. 2024. Peachy parallel assignments (EduPar 2024). In *Proc. 14th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*.
- [12] Melissa E. O'Neill. 2014. *PCG: A Family of Simple Fast Space-Efficient Statistically Good Algorithms for Random Number Generation*. Technical Report HMC-CS-2014-0905. Harvey Mudd College, Claremont, CA.
- [13] M. Pantoja, C. Almeida, D. Guerrero-Pantoja, C. Maloney, and S. Rizzi. 2025. Distributed Softmax Implementation. (2025). <https://colab.research.google.com/drive/1sbsBli2mdWcZ9dRAekJI-p9I42AKowkq>.
- [14] M. Pantoja, C. Almeida, D. Guerrero-Pantoja, C. Maloney, and S. Rizzi. 2025. Scalable Softmax for Efficient Attention: Parallel and Distributed Strategies. (2025). <https://docs.google.com/presentation/d/1r5lnqy-o7pBPpSiYsmXBq3tEDrpFFx-pEXyv9q9tx0w/>.
- [15] Margus Pedaste, Mario Mäeots, Leo A Siiman, Ton De Jong, Sanne A N Van Riesen, Ellen T Kamp, Constantinos C Manoli, Zacharias C Zacharia, and Eleni Tsourlidaki. 2015. Phases of inquiry-based learning: Definitions and the inquiry cycle. *Educational Research Review* 14 (2015), 47–61. doi:10.1016/j.edurev.2015.02.003
- [16] Angela B Shiflet and George W Shiflet. 2014. *Introduction to computational science: modeling and simulation for the sciences, 2e*. Princeton University Press, 41 William Street, Princeton, New Jersey 08540.
- [17] Elizabeth Shoop. 2021. *Observing Scalability of the Trapezoidal Integration Example*. <https://github.com/csinparallel/CSinParallel/tree/main/Exemplars/TrapezoidIntegrationScaling>
- [18] Elizabeth Shoop. 2021. *Parallelizing the Game of Life with OpenMP and Randomness*. <https://github.com/csinparallel/CSinParallel/tree/main/Exemplars/StochasticGameOfLife>
- [19] Elizabeth Shoop and CSinParallel project members. 2022. *Intermediate Parallel and Distributed Computing*. <https://www.learnpdc.org/IntermediatePDC/>
- [20] Xiaoxia Wu, Chenxin Zhang, Ziheng Chen, Tianjun Zhang, Qiaoling Zhang, Wei Lin, and Ang Li. 2025. FLASH-D: FlashAttention with Hidden Softmax Division. *arXiv preprint arXiv:2505.14201* (2025). <https://arxiv.org/abs/2505.14201>
- [21] Xiaodan Zhang, Tao Lin, Ying Zhan, Zhihong Ren, and Cheng Wang. 2021. The impact of inquiry-based learning on students' STEM self-efficacy: A meta-analysis. *International Journal of STEM Education* 8, 1 (2021), 1–15. doi:10.1186/s40594-021-00279-1