

Peachy Parallel Assignments (EduHPC 2026)

Hamdan Alabsi
Bowie State University,
Bowie, MD, USA
halabsi@bowiestate.edu

Nathan Bean
Kansas State University,
Manhattan, KS, USA
nhbean@ksu.edu

Uma Maheswari Chinta
Texas Tech University,
Lubbock, TX, USA
uchinta@ttu.edu

Hoda El-Sayed
Bowie State University,
Bowie, MD, USA
helsayed@bowiestate.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

Lisa L. Lacher
University of Houston Clear Lake,
Clear Lake, TX, USA
lacher@uhcl.edu

Rand Obeidat
Bowie State University,
Bowie, MD, USA
robeidat@bowiestate.edu

Sriram Srinivasan
Bowie State University,
Bowie, MD, USA
sriram882009@gmail.com

Yuri Torres
Universidad de Valladolid,
Valladolid, Spain
yuri.torres@infor.uva.es

Richard Weiss
Evergreen State College,
Thurston County, WA, USA
weissr@evergreen.edu

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

Abstract—Peachy Parallel Assignments are tested and ready-to-adopt assignments to teach parallel and distributed computing concepts. They are selected competitively for being “cool and inspirational” so that students spend time working on them. After selection, they are presented at the Edu* series of workshops.

This article describes three assignments presented at the EduHPC 2026 workshop. The first is an unplugged assignment that introduces MapReduce concepts by having students count word occurrences in a popular children’s book. In the second assignment, students parallelize a power grid simulation using multiple parallel programming abstractions (OpenMP for shared memory systems, MPI for distributed memory systems, and CUDA/HIP/OpenCL for GPUs). For the third assignment, they perform distributed fine tuning of a Large Language Model (LLM), a popular but extremely resource-intensive application.

Index Terms—Peachy Parallel Assignments, parallel computing education, high-performance computing education, unplugged activities, active learning, MapReduce, Large Language Models, performance profiling, National Research Platform, hardware-software co-design, parallel and distributed computing

I. INTRODUCTION

Assignments are a crucial part of computer science courses. This is where students likely spend the majority of their time on a course and good assignments make this time rewarding, encouraging students to explore and master the material. Creating strong assignments, however, requires the instructor to invest time and effort. This work is also not guaranteed to pay off since assignments can fail for unforeseen reasons.

Peachy Parallel Assignments are designed to disseminate successful, high-quality assignments that teach concepts in parallel and distributed computing (PDC) while rewarding the instructors who create them. Submissions for Peachy assignments are solicited for the EduHPC and EduPar workshops and judged competitively based on the following criteria:

- Tested: The assignment has been tested with real students.

- Adoptable: The assignment includes materials so others can adopt it. Ideally, it covers widely-taught concepts using common languages/libraries.
- Cool and inspirational: Students will be motivated to work on the assignment. Ideally, it is something they are inspired to share with friends.

The accepted assignments are then published in the workshop proceedings (e.g. [1]–[3]) and posted on the Peachy Assignments webpage: <https://cdcenter.org/peachy-assignments/>.

This paper presents the three Peachy Parallel Assignments selected for EduHPC 2026. The first contribution is an Unplugged activity [4] that introduces students to the MapReduce [5] computing paradigm without requiring them to program. Because of its lack of prerequisites, it is even suitable for use in introductory classes. The second assignment asks students to model a power grid under varying load, inspired by the 2025 Spanish power grid blackout. The same problem is presented for parallelization on shared memory, distributed memory, and GPU systems, allowing students to compare the different approaches and their formalisms. The third assignment asks students to parallelize the distributed fine tuning of a Large Language Model (LLM), capitalizing on the popularity of LLMs. This assignment is designed to run on the National Research Platform [6] so that it can be used at institutions without the local hardware resources needed to run it.

II. GREEN EGGS AND MAPREDUCE

Our first “assignment” is an unplugged classroom activity that introduces fundamental concepts in parallel and distributed computing (PDC). In the activity, students simulate parallel processing by manually counting word frequencies in Dr. Seuss’s Green Eggs and Ham. The lesson begins with a serial implementation to establish a baseline and then switches to a data-parallel version with multiple teams processing parts of the text before combining results in a final reduction step.

The activity was implemented in undergraduate Scripting II (CS2) courses, providing an accessible introduction to core parallel and distributed computing concepts without requiring specialized hardware or software. Through the exercise, students explored task decomposition, MapReduce, load balancing, and result aggregation. The complete activity, including instructor materials and student worksheets, is available in the Project Materials Repository (<https://github.com/lllacher/Green-Eggs-and-MapReduce-Peachy/tree/main>).

A. Main Idea and Student Motivation

This assignment introduces parallel processing concepts without requiring programming, special software, or hardware. Students manually count word frequencies in Dr. Seuss's Green Eggs and Ham, whose limited vocabulary allows them to focus on task decomposition, parallel execution, and aggregation rather than implementation details. An Aggregator Team combines each groups' partial results, demonstrating the coordination and communication required in parallel systems. A survey of students showed that they found the activity engaging and helpful for understanding the underlying concepts.

B. Targeted Students and Context

Scripting II / CS2 and Data Structures/Algorithms undergraduates are the target audience. This assignment is a conceptual bridge to use before writing concurrent code. Parallel data processing is a fundamental problem in systems that require log analysis, text mining, and distributed computing. This assignment was tested in standard classrooms over a single 50–75 minute session; no prior PDC knowledge assumed.

C. Covered Concepts and Prerequisites

The core concepts covered include the following:

- **Data Parallelism & Decomposition:** splitting a file into static chunks.
- **MapReduce Workflow:** local “map” counts, centralized “reduce” merge.
- **Bottlenecks & Coordination Overhead:** synchronized delivery delays.

The only prerequisites are general problem-solving experience consistent with the Scripting II/CS2 level; no prior knowledge of OpenMP, MPI, or CUDA is assumed.

D. Implementation and Adoption

Materials needed: printed copy of Green Eggs and Ham book, word-frequency worksheets per team, and a whiteboard/poster for aggregation.

To prepare for this activity, print copies, divide the text into 4–6 chunks of roughly equal word counts, prepare worksheets for each team, and assign each team a “processor ID” or color.

For the baseline (serial) run, one student/small team tallies the entire text alone, times it, and records the results. Next, debrief: Was it tedious? Where were the bottlenecks? How does time scale with word count? Approximate speedup?

Then, divide the class into teams of 2–4 (depending on class size). Each team receives one chunk. Next, the teams do the task together, as shown in Figure 1:

- **Map Phase:** Teams tally word frequencies in their chunk; only base-word forms are counted (punctuation is ignored, e.g., “Sam” and “ham” are separate); optionally time the map step and note challenges.
- **Reduce Phase:** Teams send tallies to an Aggregator Team, which merges them (e.g., Sam: 5 + Sam: 6 = Sam: 11).

Afterwards, lead the class in discussion and reflection. Was parallel faster than serial? What made aggregation hard? Discuss how simultaneous arrivals slow aggregation, mimicking communication bottlenecks. Optionally discuss alternate aggregation schemes. How would this scale to 100 processors or 10 books? What real-world systems does this resemble?

We tested this in fall and spring semesters of Scripting II (CS2) courses. Student feedback was collected using the m-ASPECT survey [7]. During the initial implementation, having every student complete a full word tally of Green Eggs and Ham consumed more class time than anticipated, leaving insufficient time for the collaborative MapReduce phases. The following semester, the students were limited to tallying a subset of five words, which significantly reduced the time required while preserving the intended learning objectives.

Student feedback suggests that the activity was well received. Responses indicated high levels of engagement, sustained effort, enjoyment, and perceived learning, with students reporting that teamwork improved their understanding and increased their interest in the material. Reverse-coded survey items showed little evidence of disengagement, supporting the consistency of these findings. The primary area of concern was in the area of team dynamics. Although students generally felt that team members contributed equally, some students thought that a single student occasionally dominated discussions. Overall, the results indicate that the unplugged activity effectively engaged students while providing meaningful exposure to foundational parallel and distributed computing concepts.

A few sticking points tend to come up. Students often expect parallel execution to give a straightforward linear speedup and are surprised when it doesn't. Many also underestimate how much time the reduce phase takes up or mix up what the mappers are supposed to do versus the reducers. Coordinating the aggregation step trips people up too — it's easy for teams to lose track of who has submitted what.

A few things help here. Running the serial baseline first, before anyone touches parallelism, gives students something to compare against later. Color-coding mapper and reducer roles cuts down on confusion about who's doing what. It also helps to walk through the workflow visually rather than just describing it, and to build in some structure around how teams hand off their results during aggregation. Finally, taking a couple of minutes after each phase to ask what worked and what didn't go a long way toward catching misconceptions.

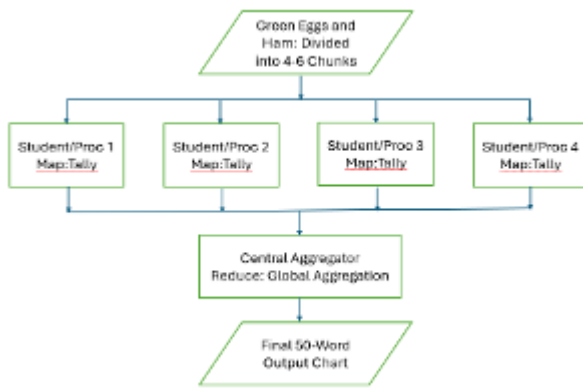


Fig. 1. Parallel map phase and then aggregate results

E. Strengths and Weaknesses

This activity has several strengths. It requires no hardware or compilation dependencies, making it immediately deployable in classrooms. Human-induced slowdowns effectively mimic real-world interconnect contention, simulating distributed-systems bottlenecks. A constrained 50-word vocabulary ensures that tallies remain easy to verify.

Unfortunately, manual counting can introduce arithmetic errors, and the activity teaches parallel computing concepts without exposing students to parallel programming constructs. A subsequent programming exercise can help connect the conceptual model to implementation.

F. Pedagogical Variations

We thought of several variations. To teach load imbalance, the instructor could give one team a chunk twice as large as the others. This exposes straggler effects on overall completion time. Alternatively, to illustrate fault tolerance, the instructor can declare a team’s “processor failed” mid-activity to prompt discussion of data redundancy and work re-allocation.

G. Relation to Prior Work

Although large-scale frameworks such as Hadoop and Spark are typically introduced in advanced courses, unplugged activities provide an accessible way to introduce parallel and distributed computing (PDC) concepts early in the curriculum. The Green Eggs and MapReduce activity adapts distributed processing to a hands-on activity so students can observe data decomposition, parallel execution, and result aggregation without the complexity of programming or distributed systems.

This assignment builds on prior unplugged PDC work, including card-sorting [8], acted-out cooperative computing [9], gridded-paper exercises [10], and map-reduce card activities [11], emphasizing conceptual understanding through physical interaction instead of code. Our evaluation in an IT-focused course extends prior evaluation work [12] and is a candidate for inclusion in resources such as PDCunplugged [13].

III. POWER GRID BLACKOUT

Our second assignment focuses on the parallelization and optimization of an application using different parallel programming models. The assignment has been successfully deployed in a third-year Parallel Computing course. Since 2018, a different problem has been proposed each academic year to illustrate the conceptual and technical differences among parallel programming models. This new assignment consists of a simulation of a power-grid blackout. The power demand fluctuates throughout the day. When random demand spikes occur, power plants begin to overheat and disconnect from the grid. The resulting power deficit must then be supplied by neighboring plants, potentially triggering a cascading failure that can lead to a complete blackout. The power grid can only recover once the demand decreases. The assignment addresses fundamental concepts such as race conditions, reductions, collective operations, and point-to-point communication. It also requires students to make key design decisions related to cache-aware programming and load balancing.

A. Idea and context

Learning parallel programming involves more than mastering the syntax of a particular API. Students must understand how the execution and memory models affect the decomposition of a problem, the management of data dependencies, and the techniques required to obtain scalable performance. This assignment addresses these aspects by asking students to parallelize the same sequential application using OpenMP for shared-memory systems, MPI for distributed-memory platforms, and CUDA, HIP, or OpenCL for GPU accelerators. This allows students to directly compare the abstractions, implementation decisions, and optimization opportunities associated with each programming model.

The application simulates the evolution of a simplified electrical power grid represented as a two-dimensional mesh of power-generation stations. Each station has a maximum generation capacity and must satisfy an electrical demand that changes over time. The baseline demand follows a daily cycle, with lower consumption during the night and higher demand during the day. Random demand spikes are also generated at particular locations and times.

When the demand assigned to a station exceeds its generation capacity, the station is temporarily disconnected to protect the system. Its unmet demand is then redistributed among adjacent stations that remain operational. This additional load may overload further stations, producing a cascading failure that can propagate through the grid and eventually lead to a widespread blackout. Stations can try to reconnect after a configurable cooldown period. The grid can progressively recover when the overall demand decreases.

The input parameters allow instructors to modify the grid dimensions, the spatial distribution of generation capacity and demand, the simulated time interval, the duration of each simulation step, the cooldown period, and the temporal and spatial characteristics of demand spikes. Consequently, the same application can generate scenarios with different degrees

of load imbalance, contention, and cascading behavior. The model is not intended to accurately reproduce a real electrical system; instead, it provides a compact and intuitive application containing dynamic data dependencies and irregular workloads that are suitable for studying parallelization and performance optimization. The starter sequential code is small enough to be quickly understood, around 1,000 code lines. It is clearly structured and commented, with the sections to be optimized and parallelized explicitly indicated. The project also contains a tool to generate an animation of the simulation to help students understand the program evolution and the output.

B. Prior work

Hands-on assignments are widely used in parallel and distributed computing education to connect parallel programming concepts with their implementation and experimental evaluation on real computing platforms [14]. Programming contests and online judging systems have also been explored as educational tools that provide rapid feedback, support repeated experimentation, and motivate students to progressively improve their solutions [15]–[17]. This assignment combines both approaches: students parallelize the same application using three different programming models and participate in an online optimization contest. Correctness is required for a solution to be accepted, while the ranking encourages students to explore alternative implementations and improve execution efficiency on a common computing platform.

The assignment idea arose after the 2025 Spanish power grid blackout, to attract the attention of students using a real-world problem. The simulation is inspired by a heterogeneous cellular-automata approach (see e.g. [18]). The assignments proposed in previous editions of the course [19] have proven to be effective in teaching different programming models and developing critical analysis skills to determine the most suitable programming model and solutions for a problem.

C. Concepts covered

The assignment addresses several parallel programming concepts introduced throughout the course. The computation of the power demand and the update of the station recovery timers happen independently for each station. Random demand spikes are generated using independent pseudo-random number generator (PRNG) sequences for each station, but the sequences are initialized with a master PRNG sequence so parallelization requires using the *skip-ahead* function provided by our custom linear congruential generator.

The redistribution of unsatisfied demand introduces neighbor-scatter communication in which multiple stations may concurrently update the same destination, leading to update races with parallelization. The reference code uses an auxiliary array of redistributed demand to decouple the redistribution and consumption phases, but leaves synchronization to be addressed during parallelization.

The simulation uses several global reduction operations to compute aggregate energy metrics, peak demand values, and a histogram of the distribution of generated power across

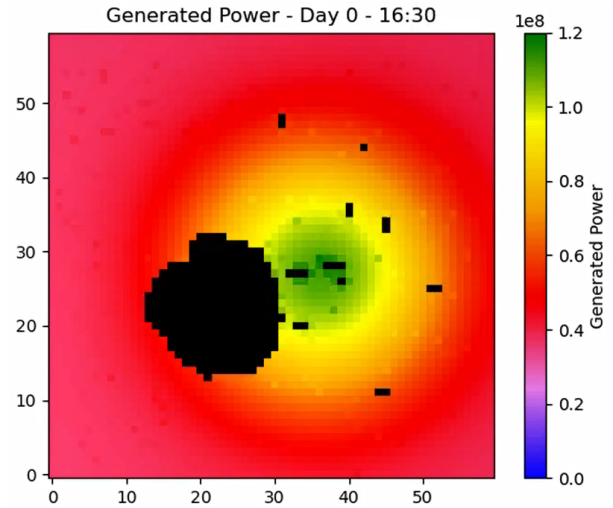


Fig. 2. Example of a the start of a cascading blackout in the single-circular-city scenario. Black colored stations are disconnected due to overload, while colors indicate the power generated by the remaining stations.

the grid. These computations introduce race conditions and chances to optimize memory accesses, synchronization, and reduction operations. For MPI, the power grid is decomposed across multiple processes. Demand redistribution requires halo exchanges between neighboring partitions, whereas the computation of global statistics relies on collective communication.

Finally, the irregular evolution of cascading failures may lead to workload imbalances, creating opportunities to evaluate load-balancing and communication optimizations. The problem is open ended for the students and allows them to explore further optimizations and different forms of implementation.

D. Possible variations

The simulation parameters allow the generation of diverse execution scenarios. The installed generation capacity can be distributed around a single circular large city or multiple urban centers, producing different spatial demand patterns. The baseline demand follows a daily cycle with lower nighttime consumption and higher daytime activity, while random demand spikes can be configured through their timing, duration, probability, and magnitude. Together with the station recovery time, these parameters determine whether cascading failures remain localized, propagate across the grid, or evolve into a complete blackout before the system eventually recovers. We provide a version of the code already prepared to be compiled with the tools of each programming model. Thus, it is direct to use the assignment in courses that covers only one of the programming models considered. It is also possible to use the starter code version in basic programming courses to explore optimizations only for sequential code.

E. Using the Assignment

a) *Teaching context:* This assignment assumes prior knowledge of operating systems, concurrency, and the C language. During the 2025–2026 academic year, 36 students were

enrolled in the course, of whom 30 completed the practical assignments, working either individually or in pairs. Students are provided with a reference sequential implementation, a handout, and a set of example configurations illustrating different execution scenarios. The parts of the code intended for modification are clearly identified, and the sequential version can generate an animation of the simulation.

Each programming model is introduced through a combination of lectures and laboratory sessions over a three-week period, using progressively more challenging exercises that illustrate the main concepts. The final exercises are directly related to the assignment, including the parallelization of random number generation. Afterwards, students are given 5 days to finish the assignment in that programming model. Finally, the exams include questions on the implemented solutions to evaluate their understanding of the parallelization strategies and their involvement in the development process.

b) Tools: The assignment requires an OpenMP-compatible C compiler, an MPI library, and the CUDA, HIP, or OpenCL toolkit. OpenMP and MPI solutions can be developed and tested on any multi-core system, whereas GPU solutions require compatible hardware. The best experience is obtained on a shared computing cluster, where all implementations are evaluated under identical conditions.

Each assignment period is organized as an optimization contest. An online judge system manages the submission queue, executes the proposed solutions on the cluster servers, and ranks them according to their correctness and performance.

c) Results: This academic year, there were 7,720 submissions to the online judge, an average of more than 257 per student. We conducted a post-course survey, which was completed by 43.3% of the students. On a Likert scale (1–5), the question “Are you satisfied with the overall experience of the course, activity types, evaluation method, etc.?” scored an average of 4.23 and a median of 4. The question “Do you think that the competitive part is adequate, and generates interest and a more active participation?” scored an average of 4.08 and a median of 4, while “Do you think that the weight in the grade of the practical work, exams, etc. is adequate?” scored an average of 4.00 and a median of 4. In personal interviews, students agree that the assignment illustrates the course’s main concepts and provides opportunities to delve into them. For example, some students’ CUDA codes ran more than 15 times faster than the baseline parallel version. They used very advanced optimization techniques and acknowledged help from AI tools to implement them. The submission execution times, contest results, and other statistics are available at <http://frontendv.infor.uva.es>.

IV. PARALLEL & EFFICIENT FINE-TUNING OF LLMs

Our third assignment has students profile the parallel fine tuning of a Large Language Model (LLM). Current PDC curricula often lack accessible, hands-on assignments that tackle these massive AI workloads, a gap we now fill. Using the National Research Platform (NRP) [6] to remove local hardware barriers, students empirically profile and compare

multiple parallelization strategies. Across the two classroom deployments, approximately 90% of enrolled students completed the assignment. This descriptive result supports the feasibility of the module in our setting; it does not establish a causal effect on learning or demonstrate equivalent outcomes across institutions.

We designed this module for senior undergraduate and graduate STEM students with working knowledge of Python, a deep learning training loop, and the GPU execution and memory concepts specified in the Prerequisites subsection. During the Spring and Summer 2026 semesters, we gave this assignment in a cross-listed “Special Topics” course focusing on Gen AI, LLMs, and hallucination mitigation with approximately 50 enrolled students, including graduates and undergraduates. Students engaged deeply with the assignment, and many adapted the assignment’s framework for their own research projects. The workflow uses the NRP [6] to provide access to containerized GPU resources without requiring institutions to purchase local accelerators. Instructors adopting the assignment should arrange platform access and confirm suitable GPU availability before scheduling experiments.

A. Motivation & Problem Description

Generative artificial intelligence (AI) has fundamentally transformed the technological landscape, with LLMs now driving innovation across education, academic research, and industry. Developers train these powerful models on massive datasets containing billions of parameters.

Processing massive datasets without parallelization takes too long. The size of an LLM can also exceed the memory capacity of a single consumer-grade GPU. To mitigate these issues, developers must efficiently partition models across multiple GPUs and use multiple techniques to parallelize the training process. This assignment provides students with direct experience fine tuning LLMs on massive datasets and teaches them how to navigate these resource constraints to execute distributed LLM training effectively. Students actively explore various data, tensor, pipeline, and parameter-efficient parallelization strategies, analyze the trade-offs of each approach, and apply these techniques to train a model on a large dataset.

In this assignment, students investigate each parallel method. Students initially execute a guided example to train a model using parallel processing architectures. They subsequently leverage their empirical findings to run the training process on a vastly larger dataset. Through this progression, students gain practical experience with performance profilers, diagnosing system bottlenecks, and finding the most efficient approach by analyzing trade-offs across speedup, power consumption, and memory footprint. As outlined in Fig. 3, this pedagogical workflow guides students through a rigorous evaluation of parallelization strategies.

Ultimately, students synthesize their results to deliver a data-driven efficiency verdict that weighs speedup, memory footprint, and power consumption. A well-supported verdict explains why a faster configuration may be less suitable when

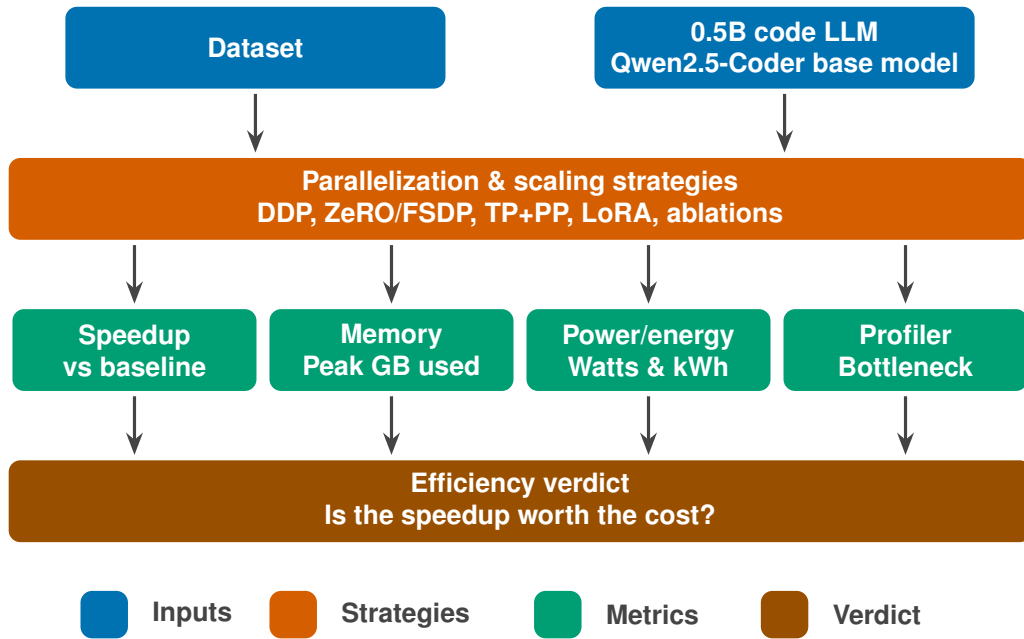


Fig. 3. Assignment overview highlighting the pedagogical workflow. Students begin with standard models and datasets, evaluate parallelization strategies across four performance metrics, and synthesize findings into a final efficiency verdict.

its memory demand or energy-to-solution is higher, and identifies the resource constraints under which the recommended strategy is appropriate. The assignment actively cultivates critical thinking and advanced problem solving skills.

To support this workflow, we selected the HPC-Instruct dataset [20], which trains LLMs to write efficient HPC code.

B. Concepts Covered

This assignment covers a comprehensive suite of PDC concepts anchored in the practical domain of LLM fine-tuning. Students actively explore multiple parallelization paradigms, including Data Parallelism (DDP), Fully Sharded Data Parallel (FSDP) [21], ZeRO optimizations [22], and Tensor and Pipeline Parallelism [23]. We also introduce Parameter-Efficient Fine-Tuning, specifically LoRA [24] and QLoRA [25], to illustrate critical memory-compute trade-offs. Beyond model execution, the module rigorously integrates hardware performance evaluation. Students calculate strong and weak scaling efficiency, measure throughput, and directly apply Amdahl’s Law to real AI workloads. Furthermore, the assignment emphasizes energy-aware computing. Students actively use profilers to conduct bottleneck analyses comparing compute versus communication overhead and quantify the total energy-to-solution (in kWh) for different parallelization strategies.

C. Prerequisites

We specify prerequisites as competencies since course titles and coverage differ across departments. Students should be able to (1) create a Python environment, install dependencies, and run a script from the command line; (2) explain the roles of a mini-batch, forward pass, loss, backpropagation, and optimizer update in a deep learning training loop; (3) describe

how GPU threads are organized into blocks, distinguish host memory from GPU device memory, and explain why model parameters, activations, and optimizer states can exhaust device memory; and (4) launch a distributed training example with a tool like `torchrun` and interpret basic execution logs.

The special topics course includes a dedicated GPU architecture module [26]. For adoption in other courses, we recommend a preparatory exercise in which students run a supplied single-GPU training script, inspect its memory use, and launch a supplied two-GPU example. Students who need additional preparation can complete guided Python, training-loop, and GPU-memory activities before attempting the full assignment. Instructors can use provided kernels and training scripts without requiring students to write custom CUDA kernels. Tensor and pipeline parallelism can be reserved for students with additional distributed computing preparation. These competencies, rather than a particular prerequisite course title, determine readiness for use in senior undergraduate and graduate PDC, Cloud Computing, or Applied Machine Learning courses.

D. Strengths

This assignment uniquely infuses PDC concepts into STEM curricula via the high-interest domain of LLM fine-tuning. A core novelty is the deep integration of empirical performance profiling; students deploy the PyTorch profiler and TensorBoard to visually diagnose bottlenecks and quantify compute vs communication overhead. Deployment on the NRP’s [6] Kubernetes-based GPU infrastructure reduces dependence on institution-owned accelerators and supports adoption by institutions with limited local hardware. While absolute performance metrics from containerized GPUs may differ from bare-metal runs, we explicitly discuss this nuance with students.

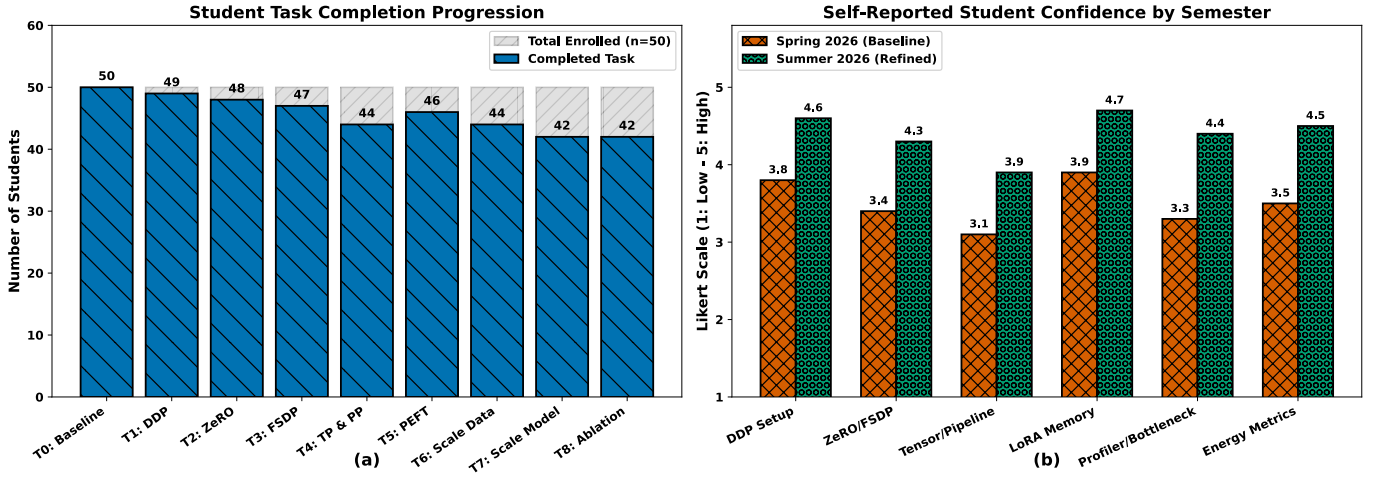


Fig. 4. Pedagogical outcomes across the Spring and Summer 2026 deployments. (Left) Student progression and task completion rates out of 50 enrolled students, demonstrating a $\sim 90\%$ overall completion rate. (Right) Self-reported student confidence on a 5-point Likert scale across the two deployments. Higher reported Summer confidence coincided with the introduction of additional scaffolding; this descriptive comparison does not establish statistical significance, cohort equivalence, or a causal effect of scaffolding.

E. Using the Assignment

We provide a comprehensive suite of scaffolding materials, job manifests, and source code via the supplementary repository. Instructors retain the flexibility to substitute alternative LLMs or datasets to suit their curricular needs. To support flexible adoption, instructors can deploy the assignment as a comprehensive multi-week project or partition it into standalone modules. For instance, introductory classes might assign only the data parallelism and LoRA tasks, while advanced seminars can independently assign tensor parallelism and the hyperparameter ablation studies. This modularity allows educators to precisely tailor the workload to their students' experience levels and available compute budgets.

To maximize adoption, we include ready-to-use job submission templates for both traditional SLURM clusters and Kubernetes-based environments. We tested the assignment scaffolding on the NRP [6]. Adopting instructors should check the supplied environments and resource requests against their available infrastructure before classroom use. Additionally, instructors can build on our framework to introduce advanced topics such as quantifying communication-compute overlap, conducting heterogeneous performance-per-watt comparisons, or implementing manual tensor-parallel layers.

F. Results

Student feedback and submitted work provided descriptive evidence of engagement with PDC concepts, LLM fine-tuning, and performance profiling. We deployed the assignment in two phases. First, students configured their environments and ran scaffolded, rapid-iteration parallelization experiments. Next, they scaled their fine-tuning to a larger dataset, which required up to three hours of compute time.

Early classroom use revealed that providing minimal starter code made students rely heavily on generative AI, causing prolonged debugging of precise library dependencies. In response,

we introduced additional scaffolding for baseline training to help students develop their own parallel scripts. The supplied NRP and SLURM deployment templates support different computing environments, although local configuration and resource availability still affect setup effort and scheduling.

Fig. 4(a) summarizes task completion across the Spring and Summer 2026 deployments, with an overall completion rate of $\sim 90\%$. Students finishing the assignment submitted technical work and performance analyses. The aggregate rate denotes completion in our setting; it does not include if completion differed between semesters or prove mastery of each concept.

Fig. 4(b) summarizes self-reported confidence. The Summer deployment included additional baseline scaffolding and showed higher reported confidence, including in bottleneck diagnosis and memory trade-offs. We interpret this pattern descriptively. The reported evaluation does not establish that the Spring and Summer student populations were comparable in prior preparation, graduate or undergraduate composition, or other characteristics. Differences in enrollment patterns, course pacing, and student availability could also affect the comparison. Consequently, we cannot attribute the observed confidence differences to scaffolding alone. Self-reported confidence also does not directly measure conceptual understanding or independent performance.

In future deployments, we will integrate National Artificial Intelligence Research Resource (NAIRR) Pilot resources and utilize bare-metal institutional clusters. This shift will enable students to empirically compare virtualized Kubernetes environments against native hardware architectures.

G. Limitations and Evaluation Improvements

The evaluation covers two offerings of one cross-listed special topics course with approximately 50 students. This limited setting constrains generalization to other institutions, course levels, and student populations. The reported aggregate

completion rate and confidence summaries do not provide a controlled estimate of learning gains. The evaluation as presented does not establish baseline equivalence or provide the cohort-specific enrollment, survey-response counts, and uncertainty estimates needed to assess the Spring-to-Summer comparison more fully. Selection differences between semesters therefore remain a threat to validity.

The assignment also has instructional and operational limitations. Its combination of distributed training, environment management, and performance profiling can impose a substantial workload on students with uneven preparation. Starter code can reduce setup friction, but completion of scaffolded tasks alone does not show that students can independently select or implement a parallelization strategy. NRP access reduces local hardware costs, while resource availability, environment configuration, and storage behavior can still affect reproducibility and the time required to complete experiments. The reported active-training measurements exclude PVC-related I/O delays and therefore should not be interpreted as complete end-to-end runtime or energy-to-solution measurements.

For future evaluations, we would report enrollment, completion, and survey-response counts for each offering, documenting prior preparation and student level, and assess submissions with a common rubric. Paired pre- and post-tests within each offering would measure changes in conceptual understanding and confidence, while an individual explanation or transfer task would assess students' independent reasoning. Comparisons of scaffolding approaches should use comparable conditions and account for baseline differences where feasible, with effect sizes and uncertainty reported alongside descriptive results. The rubric should explicitly assess final efficiency, including how students justify trade-offs among speedup, memory, power, and energy-to-solution. Future profiling should report both active-training performance and end-to-end costs, including initialization, data loading, and checkpointing, so students can evaluate the entire workflow.

H. Infrastructure and I/O Latency

While the NRP provides equitable, zero-cost access to GPU resources via its Kubernetes-based architecture, this infrastructure introduces data transfer challenges. In the NRP ecosystem, containers access storage through Persistent Volume Claims (PVCs), which typically provision network-attached storage.

Initially, we relied on PVCs for data storage to support robust checkpointing. By saving model states directly to the persistent network volume, students could seamlessly resume training after node failures or out-of-memory (OOM) errors. However, during the fine-tuning of LLMs, we observed significant I/O latency when the system loaded massive model checkpoints or saved distributed optimizer states. In our deployment, these network-storage transfers introduced bottlenecks that stalled the accelerators; the relative performance of network and local storage depends on the storage configuration and workload. Consequently, our initial traces revealed periods of low GPU utilization when the system was I/O bound.

To address this latency in future deployments, we plan to shift the heavy training workload to the node's local computing workspace. Students will execute active fine-tuning on local storage where available, with periodic checkpoints and final outputs copied to persistent storage. This approach aims to reduce training-time I/O overhead while retaining recovery options if node-local data is lost. To bypass the storage bottleneck during deployment, students will push their fine-tuned model weights directly to the Hugging Face Hub so they can quickly pull the model from the Hub during inference.

Meanwhile, to isolate algorithmic efficiency from infrastructural variance, we deliberately excluded the PVC-induced I/O bottleneck timings from our core performance profiling. We configured the performance profilers to capture traces strictly during the active training loop, focusing solely on GPU kernel execution and cross-device communication. This measurement scope helps examine compute and communication behavior under the observed storage variability. However, excluding I/O narrows the interpretation of the results: an active-training advantage may not translate into lower end-to-end runtime or energy use. Students should identify this measurement boundary in their efficiency verdict and avoid generalizing active-training measurements to the complete workflow.

I. Artifact Description

To facilitate immediate classroom adoption, we provide the complete source code, environment configurations, and job manifests in a ZIP archive. Furthermore, all student-facing instructions and deployment guides are hosted on a dedicated project website [27]. Together, these resources enable seamless adoption to diverse local or cloud computing environments.

ACKNOWLEDGMENT

Spelling and grammar improvements were made with Grammarly in Section III and Gemini (3.1 Pro with Extended Thinking enabled) in Section IV. For the LLM assignment, Claude Fable 5.1 and Sonnet 5 were used to brainstorm assignment concepts, draft initial assignment instructions, and provide initial structuring for the given code, which was then heavily modified by the authors to ensure full correctness.

The blackout assignment was developed in the GAMUVa group (<https://gamuva.infor.uva.es/>), and was partially supported by Vicerrectorado de Innovación Docente y Transformación Digital de la Universidad de Valladolid, Proyectos de Innovación Docente PID2526_108, the Spanish Government MCIN/AEI and FSE+, projects NATASHA (PID2022-142292NB-I00) and HECATE (PID2025-170366NB-I00), and Junta de Castilla y León - FEDER Grants, projects PROPHET-2 (VA226P20) and HUSAR (VA127P25). The NVIDIA Hardware Grant Program provided GPU devices.

This research on parallel LLM fine-tuning was supported by grant funding from the National Science Foundation (NSF) under the project "CAP: Expanding Use-Inspired AI Research and Instruction Across Campus," Award #2401658.

REFERENCES

- [1] M. Gerten, J. Iqrah, and J. Spacco, “Knoxcraft Terps: Parallel Minecraft for novices, a Peachy Assignment,” in *Proc. 16th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar)*, 2026.
- [2] C. Almeida, E. Shoop, D. García-Álvarez, A. Gonzalez-Escribano, D. Guerrero-Pantoja, C. Maloney, M. Pantoja, S. Rizzi, and D. Bunde, “Peachy parallel assignments (EduHPC 2025),” in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2025.
- [3] H. Bücker, J. Schoder, X. Suo, and D. Bunde, “Peachy parallel assignments (EduPar 2025),” in *Proc. 15th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar)*, 2025.
- [4] T. Bell, F. Rosamond, and N. Casey, *Computer science unplugged and related projects in math and computer science popularization*. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 398–456.
- [5] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *Proc. 6th Symp. Operating Systems Design and Implementation*, 2004.
- [6] D. Weitzel, A. Graves, S. Albin, H. Zhu, F. Würthwein, M. Tatineni, D. Mishin, E. Khoda, M. Sada, L. Smarr, T. DeFanti, and J. Graham, “The national research platform: Stretched, multi-tenant, scientific kubernetes cluster,” in *Proc. Practice and Experience in Advanced Research Computing (PEARC ’25)*, 2025.
- [7] N. Naibert, E. Shortlidge, and J. Barbera, “Modifying the ASPECT survey to support the validity of student perception data from different active learning environments,” *J. Microbiology & Biology Education*, vol. 22, no. 3, 2021.
- [8] S. Ghafoor, D. Brown, M. Rogers, and T. Hines, “Unplugged activities to introduce parallel computing in introductory programming classes: An experience report,” in *Proc. ITiCSE ’19*, 2019, p. 309.
- [9] G. Bachelis, D. James, B. Maxim, and Q. Stout, “Bringing algorithms to life: Cooperative computing activities using students as processors,” *School Science and Mathematics*, vol. 94, no. 4, pp. 176–186, 1994.
- [10] M. Smith, S. Srivastava, D. Bunde, A. Crockett, M. Gerten, P. Maher, J. Spacco, X. Suo, J. Wang, and M. Zhu, “A visual unplugged activity to introduce PDC,” in *Proc. 15th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar)*, 2025.
- [11] M. Smith and S. Srivastava, “Introducing PDC concepts through the use of flashcards and a card game,” in *Proc. 13th NSF/TCPP workshop on parallel and distributed computing education (EduPar)*, 2023.
- [12] S. Srivastava, M. Smith, A. Ghimire, and S. Gao, “Assessing the integration of PDC in early undergraduate CS curriculum using unplugged activities,” in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2019, pp. 17–24.
- [13] S. Matthews, “PDCunplugged: A free repository of unplugged parallel & distributed computing activities,” in *Proc. EduPar/IPDPSW*, 2020, pp. 284–291.
- [14] J. C. Adams, R. Brown, S. J. Matthews, and E. Shoop, “Teaching PDC in the time of COVID: Hands-on materials for remote learning,” in *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 2021, pp. 342–349.
- [15] V. Khera, O. Astrachan, and D. Kotz, “The internet programming contest,” in *Proc. 24th SIGCSE Tech. Symp. Computer Science Education*. ACM, 1993, pp. 48–52.
- [16] J. Fresno, A. Ortega-Arranz, H. Ortega-Arranz, A. Gonzalez-Escribano, and D. R. Llanos, *Gamification-Based E-Learning Strategies for Computer Programming Education*. IGI Global, 2017, ch. 6. Applying Gamification in a Parallel Programming Course, pp. 278–302.
- [17] S. Combéris and J. Wautelet, “Programming trainings and informatics teaching through online contests,” *Olympiads in Informatics*, vol. 8, pp. 21–34, 2014.
- [18] Q. Yu, Q. Wang, and N. Cao, “Blackouts mechanism simulation model of power system based on heterogeneous cellular automata,” *Electric Power*, vol. 52, pp. 79–89, 2019.
- [19] GAMUVa group, “Peachy parallel assignments,” 2018–2024, on <https://gamuva.infor.uva.es/peachy-assignments/>.
- [20] D. Nichols, A. Marathe, H. Menon, T. Gambelin, and A. Bhatele, “HPC-Coder: Modeling parallel programs using large language models,” in *Proc. 39th Intern. Conf. ISC High Performance*. IEEE, 2024, pp. 1–12. [Online]. Available: <https://pssg.cs.umd.edu/assets/papers/2024-05-hpc-llm-isc.pdf>
- [21] Y. Zhao, A. Gu, R. Vasan, R. Hou, A. Baines, C. Zhai *et al.*, “PyTorch FSDP: Experiences on scaling fully sharded data parallel,” *Proc. VLDB Endowment*, vol. 16, no. 12, pp. 3848–3860, 2023.
- [22] S. Rajbhandari, J. Rasley, O. Ruwase, and Y. He, “ZeRO: Memory optimizations toward training trillion parameter models,” in *Proc. Intern. Conf. High Performance Computing, Networking, Storage and Analysis (SC ’20)*. IEEE, 2020, pp. 1–16.
- [23] M. Shoenybi, M. A. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro, “Megatron-LM: Training multi-billion parameter language models using model parallelism,” *arXiv preprint arXiv:1909.08053*, 2019.
- [24] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “LoRA: Low-rank adaptation of large language models,” in *Prod. Intern. Conf. Learning Representations (ICLR)*, 2022.
- [25] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Efficient finetuning of quantized LLMs,” in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 36, 2023.
- [26] S. Srinivasan, H. Alabsi, R. Obeidat, N. Ponnala, and A. Zenebe, “GPU programming for AI workflow development on AWS SageMaker: An instructional approach,” in *Proc. SC ’25 Workshops of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC Workshops ’25. New York, NY, USA: Association for Computing Machinery, 2025, pp. 384–392. [Online]. Available: <https://doi.org/10.1145/3731599.3767381>
- [27] S. Srinivasan, H. El-Sayed, H. Alabsi, and R. Obeidat, “Special topics in LLM fine-tuning: Course assignment and artifacts,” 2026, supplementary Course Website. [Online]. Available: <https://sites.google.com/view/specialtopicsassignment/home>

Appendix: Artifact Description/Artifact Evaluation

Artifact Description (AD)

I. OVERVIEW OF CONTRIBUTIONS AND ARTIFACTS

A. Paper's Main Contributions

- C_1 An unplugged activity to count the frequencies of words in the book Green Eggs and Ham.
- C_2 A series of assignments to parallelize a power grid simulation using different programming paradigms (shared-memory, distributed-memory, and accelerators).
- C_3 An assignment to profile and optimize the fine tuning of an LLM.

B. Computational Artifacts

- A_1 <https://github.com/lllacher/Green-Eggs-and-MapReduce-Peachy>
- A_2 <https://gamuva.infor.uva.es/peachy-assignments/>
- A_3 <https://sites.google.com/view/specialtopicsassignment/home>

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1	
A_2	C_2	Figure 2
A_3	C_3	

II. ARTIFACT IDENTIFICATION

A. Computational Artifact A_1

Relation To Contributions

The artifact is the activity. The provided URL gives the contents of the book, a worksheet to use when counting word frequencies, and the correct solution. All documents are provided in markdown, pdf, and docx.

Expected Results

The Green Eggs and Ham activity helps students understand and gain an appreciation for Parallel and Distributing Computing Concepts, which include how parallelizing tasks can improve performance, but can also cause bottlenecks and increase coordination overhead. Because students use an unplugged activity with no specialized hardware or software, this activity is extremely accessible to all students. The survey results demonstrated that students not only enjoyed the activity, but also indicated that their knowledge of parallel and distributed computing increased. This substantiates the main contributions of this activity.

Expected Reproduction Time (in Minutes)

The estimated time required to reproduce the artifact is approximately 30–45 minutes. The Preparation step is done in advance and takes 5 minutes. The Baseline (Serial) step takes 5–10 minutes, depending on the number of discussion questions. The Team Assignment (Parallel) step takes 5 minutes. The Map Phase takes 5–10 minutes because some student teams are slower than others. The Reduce step takes 5 minutes. The Discussion and Reflection step takes 10–15 minutes, depending on the number of reflection questions the instructor chooses to discuss.

Artifact Setup (incl. Inputs)

Hardware: No hardware is needed for this activity. Some instructors may want to have copies of the printed book instead of the text files of its contents that we provide.

Software: There is no software necessary to perform the Green Eggs and Ham unplugged activity.

Datasets / Inputs: The only dataset is the text of Green Eggs and Ham, which is provided.

Installation and Deployment: To deploy the Green Eggs and Ham unplugged activity, download the materials from GitHub to print and collate.

Artifact Execution

The Green Eggs and Ham unplugged activity workflow consists of six steps: 1) printing copies of the text and worksheets, 2) have a student (or a few if you wish to average time results) tally the entire text alone and tally the results, 3) divide the class into teams of 2-4 and have them separate the text into equal chunks, 4) have the teams tally word frequencies for their chunk, 5) have teams send their tallies to an aggregator(s), 6) lastly, pose discussion questions to the class to reflect. All steps must be done in order as each step is dependent on the previous step.

Artifact Analysis (incl. Outputs)

N/A

B. Computational Artifact A_2

Relation To Contributions

The artifact is the activity. The provided URL gives a student handout for each version of the assignment, the starting sequential code, and some program arguments to be used as examples.

Expected Results

The application simulates the evolution of a simplified electrical power grid represented as a two-dimensional mesh of power-generation stations. When the demand assigned to a station exceeds its generation capacity, the station is temporarily disconnected to protect the system. Its unmet demand is then redistributed among adjacent stations that

remain operational. This additional load may overload further stations, producing a cascading failure that can propagate through the grid and eventually lead to a widespread blackout. The grid can progressively recover when the overall demand decreases. The model is not intended to reproduce a real electrical system accurately; instead, it provides a compact and intuitive application that expands the idea of an iterative loop stencil application with dynamic data dependencies and irregular workloads. It is suitable for studying parallelization and performance optimization in different programming models for shared memory, distributed memory, or accelerators.

Expected Reproduction Time (in Minutes)

The sequential code is ready for compilation and execution. Adjusting the Makefile to the local compiler, compiling, executing, and checking the results provided in the handout example takes less than five minutes.

Artifact Setup (incl. Inputs)

Hardware: A single node suffices to run the provided sequential code and the OpenMP and MPI versions. The node should have at least one GPU accelerator to run the CUDA/HIP/OpenCL version. Our students were provided with additional resources so they could achieve speedups and observe the effect of advanced optimizations.

The machine used for the OpenMP part is named *medusa*. It is a server with two Intel Xeon Silver 4208 CPU @2.10GHz, and 120 GB of RAM, with a total of 16 cores exploiting hyperthreading to provide 32 threads.

The machine used for the CUDA/HIP/OpenCL part is named *manticore*. It is a server with two Intel(R) Xeon(R) Platinum 8160 @2.10GHz CPUs, and 512 GB of RAM, with a total of 48 physical cores using hyperthreading to provide 96 threads. It is equipped with 2 NVIDIA's V100.

For the training of the MPI part, we used a combination of both machines. They are interconnected by a 40Gb Ethernet network fabric. During the contest, the submissions were restricted to using only *medusa*, to simplify the load balancing techniques in a homogeneous system.

Software: The cluster machines are managed by a Rocky 9 operating system. The compiler and toolchain software used are GCC v11.5, Mpich 4.2.1, and CUDA v12.9. The assignment runs in a standard Linux environment, but we used additional tools to run the assignment as a student contest. The online judge program used is named *Tablon*, and it was developed by the Trasgo research group at the University of Valladolid (<https://trasgo.infor.uva.es/tablon/>). It 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 23.11.6 release.

The script to generate video animations from the output data requires Python 3.x and the modules *numpy* and *matplotlib*.

Datasets / Inputs: The program's only inputs are the arguments on the command line. They allow instructors to modify the grid dimensions, the spatial distribution of generation capacity and demand, the simulated time interval, the

duration of each simulation step, the cooldown period, and the temporal and spatial characteristics of demand spikes. Consequently, it can generate scenarios with different degrees of load imbalance, contention, and cascading behavior.

Valid command lines for typical example scenarios are provided on the handout.

Installation and Deployment: The bulk of artifact installation is the installation of the software described above. This would not pose a significant burden since this software is fairly standard on clusters.

Artifact Execution

The provided package can be decompressed in any user directory. The package contains the code and a makefile prepared to compile the sequential executable and versions for OpenMP, MPI, and CUDA, switching on compiler optimizations. The makefile can be adjusted for the exact compilers available.

Artifact Analysis (incl. Outputs)

The program generates a result text line with several parameter values and statistical information of the simulation evolution and its final state. As the simulation evolves, minor differences at any step lead to different results in the output. Using the same program arguments, the user can compare the result line of the sequential code with the result line of a parallel version. The online judge compares the result line against previously computed results, in several scenarios, to check the correctness of the submitted program.

The Makefile also includes an option to compile the sequential code with full output data enabled. Then, the program writes to a file the state of the power grid on each iteration of the simulation. The package includes a Python script that reads an output file and generates a video with a visual animation of the power grid evolution for that run. This representation helps the students to understand the program behavior in different scenarios. Figure 2 in the paper presents a frame of a video generated with example arguments provided in the handout.

C. Computational Artifact A_3

Relation To Contributions

The artifact is the activity. The URL provides the complete source code, environment configurations, and job manifests in a ZIP archive. The site also provides student-facing instructions and deployment guides.

Expected Results

Reproducing this artifact yields a comparative performance and resource profile across nine parallelization, parameter-efficient, and scaling strategies for model fine-tuning. The evaluation measures wall-clock speedup, peak memory, power, total energy, and profiling bottlenecks across Distributed Data Parallel (DDP), Zero Redundancy Optimizer (ZeRO Stages 1–3), Fully Sharded Data Parallel (FSDP), Pipeline Parallelism, and Low-Rank Adaptation (LoRA / Quantized LoRA).

Reproducers should expect to observe core compute-memory-energy trade-offs: data parallelism improves throughput, state-sharding and parameter-efficient adapters reduce

peak memory footprints, and scaling sequence length leads to non-linear increases in memory and energy usage. These empirical outcomes substantiate the assignment’s primary claim: students directly discover and analyze system trade-offs using open-source profiling tools, reinforcing theoretical concepts through hands-on benchmarking.

Expected Reproduction Time (in Minutes)

Wall-clock estimates below are derived from the actual step counts, batch sizes, and GPU counts specified in each task’s script, calibrated against the paper’s own reported figure that scaling to the full dataset “required up to three hours of compute time”. All estimates assume a single A100-class GPU per device; NRP’s default commodity GPUs (L40/A10, per the Kubernetes manifest) may run 1.3-2x slower, since no `nodeSelector` pins a specific card. Profiled runs (`--profile`) add roughly 10-20% overhead from the `torch.profiler` trace capture and are not double-counted separately below.

TABLE I
ESTIMATED COMPUTE TIME PER TASK (UNPROFIED, 0.5B MODEL
UNLESS NOTED)

Task	Config	Est. time
T0	1 GPU, 200 steps	3–5 min
T1 DDP	2/4/8 GPU (3 runs)	6–12 min
T2 ZeRO	Stg 1–3 $\times$ 2/4/8 GPU (9 runs)	30–40 min
T3 FSDP	2/4/8 GPU (3 runs)	12–18 min
T4a PP	2/4 GPU (2 runs)	6–12 min
T4b TP	tp=1,2 (2 runs)	<2 min
micro		
T5	DDP 4 GPU + 1 GPU 4-bit	6–8 min
LoRA/QLoRA		
T6 Scale	Best strat., 1000+ steps	up to 3 h
data		
T7 Scale	DDP + best strat. (2 runs)	30–90 min
model		
T8	256/512/1024/2048 (4 runs)	20–25 min
MAX_LEN		
T8 2nd	4 values (4 runs)	15–20 min
param		

Summing these, Artifact Execution (compute only, all nine tasks, one full pass, unprofiled) totals approximately **5.5–7 hours**, dominated by Task 6’s large-scale run. Adding required profiling passes for every completed task adds roughly 10–20% on top of the already-profiled subset, bringing total GPU wall-clock to approximately **6–8.5 hours**.

Artifact Setup: 30–45 min, covering `venv` creation, pinned-dependency installation, and HPC-Instruct download/tokenization.

Artifact Execution: 6–8.5 hours as broken out above, for one complete pass through Tasks 0-8 including required per-task profiling.

Artifact Analysis: 45–90 min to run `merge_energy.py` per completed task, `aggregate_and_plot.py` once at the end, and `aggregate_profiler_results.py` to produce the cross-strategy bottleneck comparison.

Artifact Setup (incl. Inputs)

Hardware: We ran the assignment for our class on the National Research Platform (NRP/Nautilus), a free, Kubernetes-based, NSF-funded GPU cluster open to any InCommon-affiliated institution. A single pod is schedulable on up to 8 GPUs, which covers every task in the assignment without requiring true multi-node orchestration. The assignment is equally supported on an existing SLURM-scheduled HPC cluster with no changes to the training scripts, only to the job-submission mechanics; there, GPU type is whatever the institution’s existing allocation provides. All nine tasks require access to 2–8 GPUs across up to two nodes; Task 7’s larger-model comparison benefits from at least one GPU with 24GB or more of memory, or an out-of-memory result is itself expected and reportable.

Software: We used Python 3.11. Core environment: CUDA 12.2, GCC 12.2. Python packages, pinned exactly as specified in the assignment’s setup script: PyTorch 2.4.1 (cu121 wheel, installed from the PyTorch CUDA 12.1 index to match the driver’s maximum supported CUDA version), Hugging Face Transformers 4.44.0, Accelerate 0.33.0, DeepSpeed 0.14.4, PEFT 0.12.0, and the Hugging Face `datasets` and `huggingface_hub` libraries. `bitsandbytes` 0.43.3 is installed only when running QLoRA (Task 5, `--use_4bit`), in a separate virtual environment from plain LoRA/DDP runs, since an unconditional `bitsandbytes` install can break unrelated runs on some Triton versions. Profiling uses PyTorch’s built-in `torch.profiler` together with the open-source `torch-tb-profiler` TensorBoard plugin and TensorBoard itself. Aggregation and plotting scripts use `pandas` and `matplotlib`. Students will also need to submit and manage jobs. This requires them to use `sbatch` on a SLURM cluster; on NRP, it is either `kubect1` or NRP’s GPU-backed JupyterLab.

Datasets / Inputs: The project uses Hugging Face’s HPC-Instruct dataset (<https://huggingface.co/datasets/hpcgroup/hpc-instruct>), a real, MIT-licensed, publicly hosted instruction-response dataset containing 121,551 synthetically generated samples (via Gemini Pro, DBRX, Llama-3, and Mixtral) spanning four HPC/parallel-programming task types: code synthesis, parallelization, optimization, and translation. It is distributed as Parquet and loads directly via the `datasets` library with no manual staging. We fix a reproducible 98/2 train/validation split with `seed=42`. For the core scaling experiments in Tasks 0-5, we subsample a fixed number of examples (e.g. 8,000) with the same seed so every group’s numbers stay directly comparable; Task 6 removes or raises this subsample size, and the optional extensions in Section 20 use the full dataset.

Installation and Deployment: We provide the complete source code, environment configuration files, and job manifests in an attached ZIP archive, including a one-time NRP kube-config setup script, a ready-to-use Kubernetes Job manifest requesting up to 4 GPUs per pod with an embedded `pip` install step and background `nvidia-smi` power logger, and an equivalent SLURM `sbatch` template with the same

power-logging pattern. Students and instructors follow the deployment guide hosted on the project website [9], which walks through venv setup, dataset download and tokenization, and the launch command for each of the nine tasks.

Artifact Execution

The artifact workflow consists of nine dependent tasks plus three post-processing scripts. T_0 establishes the single-GPU baseline (200 steps, fixed sequence length 1024) that all speedup numbers are normalized against. T_1 through T_5 each apply one parallelization or memory-optimization strategy (DDP; ZeRO Stages 1–3; FSDP; naive pipeline parallelism plus a tensor-parallel micro-benchmark; and LoRA/QLoRA) to the same baseline model, dataset, and step count, each independently comparable to T_0 and to each other at matched GPU counts. T_6 re-runs the single best-performing strategy from T_1 – T_5 at a larger dataset size and step count to test whether short-run measurements predict real-scale behavior. T_7 re-runs two strategies (DDP and the most memory-efficient strategy) with a larger model to test whether the efficiency ranking holds as model size grows, treating an out-of-memory failure as a valid data point. T_8 sweeps sequence length (256/512/1024/2048) plus one additional hyperparameter of the student’s choice against the T_0 baseline script. Every task run individually calls `merge_energy.py` to fold its background power-draw CSV into its own results JSON; two aggregation scripts then run once at the end: `aggregate_profiler_results.py` reads every task’s profiler bottleneck JSON to build the cross-strategy compute/communication/data-loading breakdown, and `aggregate_and_plot.py` reads every task’s results JSON to compute speedup, parallel efficiency, and the final scaling/energy plots.

We fixed the base model at Qwen2.5-Coder-0.5B and `max_steps=200` for T_0 – T_5 to keep the comparison tractable on a single GPU while still exposing meaningful parallelization trade-offs; `MAX_LEN` is fixed at 1024 tokens for every task except T_8 , where it becomes the swept parameter. DDP, ZeRO, and FSDP strategies were run at 2, 4, and 8 GPUs where available, holding per-GPU batch size fixed for a strong-scaling measurement and holding global batch size fixed for a weak-scaling measurement. Energy is measured per run via a background `nvidia-smi` logger sampling power draw every 5 seconds, integrated via a Riemann sum in `merge_energy.py`; profiled runs (an added `--profile` flag, using `torch.profiler`’s `schedule(wait=5, warmup=5, active=10)`) are kept separate from timing/energy runs, since profiling overhead invalidates wall-clock measurements.

Artifact Analysis (incl. Outputs)

The artifact produces, per strategy and GPU count, a results JSON (wall-clock time, throughput, peak memory, average power, and total energy in kWh) and, for profiled runs, a TensorBoard-viewable trace plus a compute/communication/data-loading percentage breakdown with an automated bottleneck verdict. Aggregation produces

five required plots (speedup vs. GPU count with an ideal-linear reference line, parallel efficiency vs. GPU count, peak memory vs. GPU count, and total energy and average power vs. GPU count, each with one line per strategy), a stacked-bar cross-strategy profiler comparison, and a completed “is the speedup worth it” table weighing speedup against memory and energy cost for every strategy, matching the efficiency-verdict workflow.