

A Portuguese-Language Ecosystem for HPC Education: Design, Formative Development, and Evidence of Adoption

Gabriel P. Silva
Instituto de Computação
Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brazil
gabriel@ic.ufjr.br

Calebe P. Bianchini
CESAR Innovation Center
Recife, PE, Brazil
cpb@cesar.org.br
Mackenzie Presbyterian University
São Paulo, SP, Brazil
calebe.bianchini@mackenzie.br

Evaldo B. Costa
CESAR Innovation Center
Recife, PE, Brazil
ebc@cesar.org.br

Abstract—This paper presents a Portuguese-language educational ecosystem designed to support the progressive teaching of high-performance computing (HPC). Developed through an independent, author-led initiative, the ecosystem combines two textbooks with reproducible C programs using MPI, OpenMP, and OpenACC, exercises, instructor slides, Colab-compatible laboratory notebooks, interactive learning tools, and training and outreach activities. Its development followed an incremental process in which materials initially used in courses, bootcamps, and programming competitions were reorganized, tested on real systems, and refined through observations and feedback from actual educational contexts. Evidence of sustained use includes activities conducted between 2019 and 2026 that reached approximately 500 participants, as well as the adoption of the first textbook in 14 undergraduate courses across 13 Brazilian institutions, representing a potential annual reach of approximately 700 students. A preliminary analytical evaluation examines the ecosystem in terms of conceptual alignment, progressive complexity, reproducibility, adoptability, and authentic use in educational and training settings. The results indicate that its components form a coherent and reproducible educational pathway covering distributed-memory, shared-memory, and accelerator programming. Although the available evidence supports claims of adoption and sustained use, controlled studies are still required to determine the ecosystem’s effects on student learning.

Index Terms—High-Performance Computing, Parallel Programming, HPC Education, Open Educational Resources, Interactive Visualization, MPI, OpenMP, OpenACC, Hybrid Programming

I. INTRODUCTION

Modern scientific and technological progress increasingly relies on the capacity to analyze extensive data sets and execute intricate simulations. Applications including climate models, molecular dynamics, genomics, and artificial intelligence

The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources on the Santos Dumont (SDumont) supercomputer (<http://sdumont.lncc.br/>), used in the technical validation of selected examples reported in this paper. Calebe Bianchini acknowledges support from (i) MackCloud (<https://mackcloud.mackenzie.br/>); (ii) the SPRACE project – Grant No. 2018/25225-9, São Paulo Research Foundation (FAPESP); and (iii) the Mackenzie Research and Innovation Fund (MackPesquisa) – Project No. 251005.

require computational capacity exceeding that provided by a single processor, rendering parallel and distributed programming a crucial competency across multiple disciplines [1].

Despite the availability of robust HPC infrastructure in Brazil through institutions such as LNCC and CENAPAD, as well as several universities and industries, its effective use remains limited by a shortage of qualified personnel. This skills gap affects critical areas such as energy, health, and climate research [2].

The initiative was not established as a formal institutional program and did not receive continuous, dedicated funding for its development. It nevertheless benefited from indirect and activity-specific support, including access to computing infrastructure and travel or accommodation provided for individual training events.

Emerging from educational and professional training activities, the initiative gradually evolved into an integrated ecosystem rather than a collection of independent publications. Its development was motivated by the limited availability of structured, locally produced Portuguese-language materials combining conceptual explanations, executable examples, exercises, and instructor resources for HPC education.

Within this ecosystem, the textbooks provide the curricular sequence and conceptual foundation; programs, slides, exercises, and repositories support reproducible practice; interactive resources make parallel execution behavior visible; and short courses, bootcamps, competitions, and programming marathons connect the materials with actual educational and training contexts.

Accordingly, the purpose of this paper is to examine how this Portuguese-language HPC education ecosystem was designed and refined, how its components establish a progressive pathway across distributed-memory, shared-memory, accelerator, and hybrid programming, and what available evidence supports its adoption, reuse, technical reproducibility, and sustained use. The paper does not claim demonstrated learning gains. Instead, it distinguishes technical validation, formative

refinement, adoption evidence, and the empirical learning evaluation that remains to be conducted.

The contribution is therefore the ecosystem model and its articulation: two progressive textbooks, reproducible programming resources, interactive learning resources, and sustained training activities developed through a common formative process. The analysis also documents adoption in Brazilian higher education and applies an explicit framework to assess conceptual alignment, progressive complexity, reproducibility, adoptability, authentic use, and limitations.

II. RELATED WORK

The literature has increasingly focused on HPC education, while diagnostic assessments indicate that its integration into curricula continues to face obstacles such as conceptual complexity, limited infrastructure, insufficient educator training, and the lack of suitable instructional materials [1], [3]. This review focuses on publications from 2021 to 2025, particularly those appearing in workshops and journals identified by the NSF/IEEE-TCPP Curriculum Initiative¹, although relevant earlier studies also exist. The literature has expanded from reports of individual courses to reusable assignments, laboratories, support tools, curricular proposals, and training for users and educators.

A first category concerns courses and professional training. Frachtenberg [4] discusses an undergraduate course on heterogeneous architectures, emphasizing engagement with distinct execution models and platforms. Adams [5] examines OpenMPI with Java for teaching parallel and distributed computing concepts, whereas Islam and Phelps [6] present a hands-on methodology for training HPC software architects. Siegmann et al. [7] address the preparation of users for advanced computing technologies. These works demonstrate the value of authentic practice, but each is bounded by a particular course, module, or training program.

A second category is reusable assignments. The *Peachy Parallel Assignments* series [8]–[12] represents a sustained effort to disseminate complete instructional packages. These packages include problem statements, sequential code, solution frameworks, test inputs, and implementations using OpenMP, MPI, CUDA, OpenCL, or other programming models. They treat instructional material as a reusable educational product rather than only as a report of a local experience. Their scope, however, remains centered on individual assignments rather than a complete progression from foundational concepts to hybrid programming and accelerators.

A third category comprises visualization, testing, and support tools. Ballard and Parsons [13] introduce a graphical library for visualizing parallel dynamic-programming algorithms. Dewan et al. [14] investigate automated testing for concurrent programs whose behavior may depend on non-deterministic interactions. Coulter et al. [15] present ICE 2.0 as an interactive computing environment for education and

research. Guitart [16] uses scaffolding in inquiry-based laboratories for distributed systems, while Netterville et al. [17] provide a visual treatment of MPI all-to-all communication. These contributions make specific behaviors or activities more accessible, but they do not connect them to a broader sequence of textbooks, exercises, and sustained training activities.

A fourth category concerns curricula and faculty development. Bunde et al. [18] report the adoption of heterogeneous-computing modules through a summer faculty workshop. Brown et al. [19] examine faculty development workshops for integrating parallel and distributed computing into undergraduate courses. Neuwirth [20] describes the simulation of an HPC conference to introduce scientific communication, peer review, and result presentation. At the curricular level, Läufer and Thiruvathukal [21] discuss oneAPI in terms of performance, portability, and productivity, Perez [22] proposes an independent study for undergraduates, and Bouvry et al. [2] outline the European Master for HPC curriculum. Krishnasamy et al. [23] complement this perspective by examining student challenges in EUMaster4HPC.

Traditional textbooks remain essential references. *Using MPI* [24], *Using OpenMP* [25], *Parallel Programming in C with MPI and OpenMP* [26], and *OpenACC for Programmers* [27] provide authoritative treatments of their respective programming models. Within the reviewed set, however, these references do not provide an integrated Portuguese-language pathway combining textbooks, executable examples, exercises, visualization, and instructor resources for HPC education.

Table I summarizes the classification of the reviewed work. The main difference of the proposed ecosystem is not its geographic origin in itself, but the integration of resources that are commonly reported separately: textbooks, executable examples, instructor support, visualization, training, and evidence of adoption.

Taken together, the reviewed literature shows that HPC education has been advanced through several complementary forms of intervention. Courses and professional-training programs provide authentic contexts, assignment collections make individual activities reusable, visualization and testing tools make selected behaviors more accessible, and curricular or faculty initiatives support institutional adoption. These contributions establish the importance of each component, but they are usually presented as distinct educational products or local experiences.

The ecosystem examined here instead connects a curricular sequence, executable examples, instructor resources, interactive exploration, and recurring training activities within one progressive pathway. This distinction is important because the contribution is not a replacement for the cited courses, assignments, tools, curricula, or textbooks. It is the sustained articulation of these types of resources in Portuguese and the examination of how that articulation supports development, reuse, and adoption across educational contexts.

¹<https://tcpp.cs.gsu.edu/curriculum/?q=node/21242>

TABLE I
CLASSIFICATION OF RELATED WORK

Work	Region	Class	Work	Region	Class
Adams [5]	North America	Course	Almeida et al. [12]	North America/EU	Assignment
Ballard and Parsons [13]	North America	Tool	Bouvry et al. [2]	EU	Curriculum
Brown et al. [19]	North America	Faculty dev.	Bücker et al. [10]	North America/EU	Assignment
Bunde et al. [18]	North America	Faculty dev.	Carratalá-Sáez et al. [9]	North America/EU	Assignment
Casanova et al. [8]	North America/EU	Assignment	Coulter et al. [15]	North America	Tool
Dewan et al. [14]	North America	Testing	Frachtenberg [4]	North America	Course
Gonzalez-Escribano et al. [11]	EU	Assignment	Guitart [16]	EU	Laboratory
Islam and Phelps [6]	North America	Training	Krishnasamy et al. [23]	EU	Challenges
Läufer and Thiruvathukal [21]	North America	Tool	Netterville et al. [17]	North America	Tool
Neuwirth [20]	EU	Activity	Perez [22]	North America	Curriculum
Siegmann et al. [7]	North America	Training			

III. ECOSYSTEM DESIGN AND TARGET AUDIENCE

The design follows four principles: (i) progressive complexity, from introductory execution models to hybrid and heterogeneous programming; (ii) alignment among conceptual explanations, executable code, and observable program behavior; (iii) reproducibility with widely available tools and minimal deployment requirements; and (iv) modular reuse, allowing instructors to adopt either complete sequences or individual programs, exercises, slides, and learning objects. For the supporting resources distributed under open licenses, access, adaptation, and modular organization are consistent with the principles of Open Educational Resources and Learning Objects [28]–[30].

The primary audience is undergraduate students in computer science, computer engineering, software engineering, information systems, artificial intelligence, and related computing programs. The introductory progression, choice of examples, and supporting instructions were designed primarily around their backgrounds and curricular needs. Graduate students and professionals who need to learn, review, or extend parallel-programming skills constitute a secondary audience and may use the same materials selectively or as a basis for more advanced study.

Expected prerequisites include imperative programming and foundational knowledge of computer architecture, operating systems, and algorithm analysis. Familiarity with Linux and command-line tools is useful but not mandatory, because the first volume provides guidance on environment configuration, compilation, and the use of MPI, OpenMP, and OpenACC tools. Static web resources and freely available toolchains further reduce deployment requirements across institutions with different levels of HPC infrastructure.

IV. DEVELOPMENT AND EVALUATION METHODOLOGY

The ecosystem was developed through an incremental cycle of design, use in actual educational settings, technical verification, and revision. Content first used in short courses, classes, bootcamps, and previously reported parallel-programming marathons [31] was reorganized into textbook chapters and supporting resources. This adaptation involved improving explanations, ordering examples according to conceptual dependencies, identifying common errors, and adding

exercises. Observations and discussions with students, instructors, and professionals were used to refine topics that repeatedly required additional guidance, including process communication, variable scope, synchronization, collective operations, host–device data movement, and performance metrics.

Operationally, the formative cycle comprised five recurring stages: (i) identification of a concept or recurring difficulty; (ii) preparation or revision of explanations, examples, exercises, or visual resources; (iii) use of the material in a class, short course, bootcamp, or programming activity; (iv) technical verification and observation of difficulties encountered during use; and (v) revision and reintegration of the material into the textbooks, repositories, or supporting resources. The cycle was iterative rather than linear, because revised material could be reused in later activities and refined again.

The programming-model sequence covers complementary execution models: MPI for distributed-memory message passing, OpenMP for shared-memory parallelism, and OpenACC for accelerator programming. The second volume extends this foundation to OpenMP tasks, OpenMP offloading, OpenACC with multiple GPUs, and hybrid combinations.

C was adopted as the common implementation language because it is more familiar than Fortran to the primary audience in the Brazilian undergraduate computing programs targeted by the initiative. Its use permits students to build on prior imperative-programming experience while retaining broad support across MPI, OpenMP, and OpenACC toolchains. Fortran remains important in scientific computing and HPC, but introducing two implementation languages would add syntactic and maintenance overhead without directly advancing the initial parallel-programming objectives of the material.

Reproducibility was treated as a methodological requirement throughout this cycle. Complete examples were prepared so that readers could compile, execute, modify, and reuse them independently. The material therefore includes practical remarks on compiler selection, environment variables, process allocation, thread configuration, accelerator use, and interpretation of results. Whenever possible, verification used freely available tools, including GCC and the NVIDIA HPC SDK. Supporting repositories, exercise sets, and slides were maintained alongside the chapters so that corrections and revisions could be propagated beyond the printed volumes and so instructors could adapt the same content to semester

courses, short courses, bootcamps, and self-study.

Here, authentic use denotes documented use in actual educational or training activities rather than study-specific deployments.

The evidence used in this paper is divided into four categories. *Technical validation* concerns the compilation and execution of examples on real systems. *Formative refinement* concerns changes made from observations and feedback in courses and training activities. *Adoption evidence* concerns the use of the first volume in undergraduate courses and the sustained use of the broader material in training and outreach. *Learning evaluation* would require evidence that students improved their understanding; no controlled empirical study of this kind has yet been conducted.

The available evidence is not uniform across the ecosystem components. Course adoption evidence concerns only the first textbook. The second volume, only recently published, contributes to the designed conceptual progression but does not yet have equivalent adoption evidence, since some of its instructor-support material remains under preparation. The interactive learning resources have undergone technical verification and analytical examination, but they have not yet been empirically evaluated with students. The training and outreach activities provide evidence of authentic and sustained use, but not standardized evidence of learning outcomes.

Adoption data were collected between 2022 and 2026 from publicly available course syllabi and direct confirmation from some instructors. An adoption was recorded when the first volume was listed as required or recommended material in an undergraduate course. Training evidence was compiled from the activities conducted or supported by the authors between 2019 and 2026. The analytical evaluation then examined the criteria in Table II across the textbooks, source code, supporting material, learning objects, and documented uses.

TABLE II
ANALYTICAL EVALUATION FRAMEWORK

Criterion	Evidence examined
Conceptual alignment	Learning objectives, chapters, code, and visualizations
Progressive complexity	Transition from introductory models to hybrid programming and from integration to matrix–vector multiplication
Reproducibility	Compilers, source code, instructions, dependencies, and tested execution
Adoptability	Prerequisites, deployment requirements, language, and instructor material
Authentic use	Courses, short courses, bootcamps, competitions, and institutions
Limitations	Missing controlled assessment, simplified execution models, and uneven evidence across components

V. ECOSYSTEM COMPONENTS

A. Textbooks and Supporting Materials

The first volume [32] establishes the introductory pathway, covering fundamental concepts, performance analysis, MPI, OpenMP, OpenACC, and runtime configuration. The second [33] addresses parallel architectures, OpenMP tasks and offloading, advanced OpenACC, multiple GPUs, and hybrid MPI+OpenMP and MPI+OpenACC programming. Both

volumes combine explanation with complete programs, case studies, and exercises; slides are available for the first volume and are under preparation for the second. Their covers are shown in Fig. 1.



Fig. 1. The two textbook volumes that form the conceptual backbone of the ecosystem.

In the first volume, students progress from processes, threads, synchronization, decomposition, load balancing, speedup, and scalability to programming with message passing, shared-memory directives, and accelerator directives. The MPI chapters cover point-to-point and collective communication; the OpenMP material introduces work sharing, data environments, and synchronization; and OpenACC addresses host–device data movement and its **gang**, **worker**, and **vector** hierarchy.

The second volume assumes this foundation and concentrates on irregular task parallelism, accelerator management, asynchronous execution, multi-GPU programming, and hybrid models. Its examples discuss dependencies, affinity, communication overhead, portability, and performance on hierarchical and heterogeneous architectures. Table III summarizes how the volumes and the other resources contribute to the ecosystem.

B. Jupyter notebooks

The supporting material also includes six Colab-compatible Jupyter notebooks organized for laboratory activities. Three notebooks provide a progressive introduction to MPI, covering environment preparation, basic operations, point-to-point communication, collective operations, execution-time measurement, and case studies involving numerical integration, prime counting, and matrix–vector multiplication. Two additional notebooks address OpenMP tasks, progressing from task creation, synchronization, data-sharing attributes, and granularity control to dependencies, `taskgroup`, `taskyield`, and `taskloop`. A sixth notebook introduces hybrid programming with MPI and OpenMP, including thread-based execution and accelerator offloading.

The notebooks combine explanatory text, executable C programs, compilation and execution commands, guided experiments, questions, and code-completion activities. By allowing students to configure the environment and compile and execute the examples within the same interface, they

TABLE III
MAIN COMPONENTS AND THEIR ROLES IN THE ECOSYSTEM

Component	Topics	Educational role	Supporting resources
Volume 1	Concepts, MPI, OpenMP, OpenACC, runtime systems	Introductory pathway across distributed memory, shared memory, and accelerators	Code, exercises, slides, case studies
Volume 2	Architectures, tasks, offloading, advanced OpenACC, multiple GPUs, hybrid programming	Advanced pathway for hierarchical and heterogeneous architectures	Code, exercises, case studies; slides under preparation
Interactive learning resources	MPI decomposition and communication; OpenMP scheduling; OpenMP/OpenACC loop transformations	Visualization, prediction, experimentation, and correctness analysis	Static web tools, explanatory text, and parallel programs
Training and outreach	Applied MPI, OpenMP, OpenACC, offloading, and hybrid programming	Guided practice, community engagement, and workforce development	Short courses, bootcamps, competitions, and programming marathons
Laboratory notebooks	MPI fundamentals and collectives; OpenMP tasks; hybrid programming and accelerator offloading	Guided experimentation, code modification, and analysis of execution behavior	Six Colab-compatible Jupyter notebooks with executable C programs, questions, and laboratory activities

reduce laboratory setup requirements and support a transition from reading and visualization to direct experimentation. They can be used as complete laboratory sequences or selected individually according to the course content, available time, and computational resources.

C. Interactive Learning Resources

The ecosystem currently includes six web-based interactive learning resources. As open resources, they prioritize access, adaptation, reuse, and redistribution; as learning objects, they combine objectives, explanatory content, executable code, interactive activities, and assessment opportunities [28]–[30]. Three address introductory MPI concepts through numerical integration, prime counting, and matrix–vector multiplication. The remaining tools address OpenMP loop scheduling and the effects of the `collapse` and `tile` constructs in OpenMP and OpenACC. All resources are implemented as static web applications, require no specialized installation, and connect programming constructs with visual representations of work distribution, local computation, communication, and potential correctness problems.

1) *MPI learning objects*: The three MPI learning objects provide a progressive introduction to distributed-memory execution. Numerical integration illustrates cyclic iteration distribution, local accumulation, and reduction. Learners can change the number of processes and integration steps, observe the work assigned to each *rank*, and relate the partial sums displayed by the interface to the final result produced by `MPI_Reduce`.

The prime-counting object extends this model to an irregular workload. Odd candidates are distributed cyclically, each process accumulates a local count, and the counts are combined through reduction. Because the cost of testing primality varies among candidates, the example supports discussion of load balance and comparison between cyclic and contiguous distributions. The matrix–vector multiplication object subsequently introduces explicit data movement through `MPI_Bcast`, `MPI_Scatter`, and `MPI_Gather`, relating each collective operation to changes in the data available to the processes.

Taken together, these objects progress from regular decomposition and reduction to irregular computational cost and

collective communication. They can be used for classroom demonstrations, guided laboratory activities, or independent exploration. Students can predict an execution pattern, inspect it in the interface, identify the corresponding operations in the C/MPI program, and compare the visualization with the program output. The interfaces and explanatory materials for these objects are available in Portuguese, Spanish, and English.

2) *OpenMP loop-scheduling simulator*: The OpenMP loop-scheduling simulator visualizes how iterations of a parallel loop are assigned to threads. Learners can configure the number of iterations and threads, select `static`, `dynamic`, `guided`, or implementation-defined scheduling, and optionally specify a chunk size. The interface displays both an iteration map and the iterations assigned to each thread, allowing direct comparison of contiguous and cyclic static distributions and of the behavior produced by different chunk sizes.

For dynamic and guided scheduling, repeated simulations may produce different assignments because the next block is associated with the next available thread. This variability helps distinguish deterministic static mappings from runtime-dependent scheduling decisions. The tool also reports the number of iterations assigned to each thread and supports stepwise execution, providing a basis for discussing load distribution, scheduling overhead, and the fact that the OpenMP specification does not prescribe a unique thread assignment for all scheduling configurations. The simulation is therefore a conceptual model rather than a prediction of the exact behavior of a particular OpenMP runtime.

3) *Visualizing collapse and tile*: Two additional resources use matrix multiplication to illustrate the effects of the `collapse` and `tile` constructs in OpenMP and OpenACC. In the OpenMP version, learners can vary the matrix dimensions, number of threads, collapse depth, and tile dimensions. The interface colors the elements of the result matrix according to the responsible thread and shows how the iterations of the inner `k` loop contribute to a selected element. The OpenACC version provides an analogous representation based on gangs, allowing learners to examine how collapsed iteration spaces or tiles are distributed across accelerator execution units.

Beyond illustrating work distribution, the tools emphasize correctness. Collapsing or tiling the `i` and `j` loops preserves

the assignment of an entire result element to one thread or gang in the visualized model. Including the reduction loop k , however, may assign contributions to the same element of the result matrix to different execution units. The interfaces identify this situation and explain why the naive update using $+=$ would require a reduction, atomic operation, or algorithmic reorganization. Consequently, the resources connect loop transformation and locality not only to parallelism and performance, but also to data dependence and race conditions.

4) *Role and limitations:* The six resources complement the textbooks by making otherwise implicit execution behavior observable. The MPI objects emphasize processes, local results, and communication; the OpenMP scheduling simulator focuses on iteration assignment to threads; and the OpenMP/OpenACC tools relate loop transformations to work distribution, locality, and correctness. Their simplified execution models are intended to support conceptual understanding and prediction activities rather than reproduce all implementation-specific decisions of production runtimes. The OpenMP and OpenACC tools are currently available in Portuguese, and their translation and empirical classroom evaluation remain part of future work.

D. Training and Outreach Activities

Short courses, bootcamps, competitions, and programming marathons provide the contexts in which concepts and examples are applied under guided or realistic conditions. Before and after publication of the first volume, these activities supported the selection and sequencing of topics, exposed recurring learner difficulties, and connected the material to different audiences. Their sustained use also provides evidence that the ecosystem is not limited to the two books or to courses taught in a single institution.

VI. ADOPTION AND EVALUATION RESULTS

The adoption dataset comprises 14 undergraduate courses offered by 13 Brazilian institutions, shown in Table V. The institutions include federal institutes, public universities, and a private university. Based on official records maintained by the Brazilian Ministry of Education, the courses have a potential combined annual reach of approximately 700 students. This is adoption evidence rather than a measure of actual use by every enrolled student or of learning outcomes.

Between 2019 and 2026, the activities listed in Table IV collectively reached approximately 500 participants. These activities included recurring short courses offered at regional schools and the Santos Dumont Supercomputing School, bootcamps, and several editions of the Marathon of Parallel Programming. The audiences varied by event: primarily undergraduate students at ERAD events; graduate students and researchers at SSCAD and SBAC-PAD; graduate students, researchers, and professionals at the Santos Dumont School; and undergraduate and graduate students in the marathons. The table reports only information documented in the project records and makes no claims regarding completion rates or participant-level learning outcomes.

The analytical evaluation found alignment among the concepts introduced in the books, the programs that implement them, and the interactive representations of selected execution behavior. In the MPI sequence, numerical integration, prime counting, and matrix–vector multiplication progress from regular decomposition and reduction to irregular cost and collective data movement. The OpenMP and OpenACC tools extend the same correspondence to loop scheduling, transformations, data dependencies, and potential race conditions.

Progressive complexity also appears across the two volumes: introductory process, thread, synchronization, and directive concepts lead to OpenMP tasks and offloading, OpenACC with multiple GPUs, and hybrid models. Complete C programs and instructions support independent execution, while static web tools require no specialized visualization environment. Portuguese-language explanations, exercises, instructor material, and modular resources allow either complete sequences or individual components to be adopted.

Authentic use is evidenced by adoption at 13 institutions and by recurring activities between 2019 and 2026, ranging from introductory MPI to offloading, multiple GPUs, and hybrid programming. These findings support claims of internal coherence, technical reproducibility, adoptability, and sustained use; they do not establish pedagogical effectiveness or learning gains.

VII. DISCUSSION, LIMITATIONS, AND LESSONS LEARNED

The principal lesson is that accessibility results from the articulation of language, reproducibility, and modularity rather than from translation alone. Portuguese exposition lowers a linguistic barrier; complete programs and static web tools reduce deployment barriers; and modular resources let instructors select material appropriate to their courses. The advanced progression to tasks, accelerators, multiple GPUs, and hybrid models shows that broader access need not restrict the material to introductory topics.

Formative development must nevertheless be distinguished from educational effectiveness. Execution on real systems establishes technical validity; observations and feedback support refinement; recurring activities demonstrate continued relevance; and course adoption documents institutional use. None of these measures, alone or collectively, demonstrates improved learning. This distinction defines the need for the controlled evaluation proposed as future work.

Outreach served both dissemination and refinement. Short courses and bootcamps provided guided encounters with selected topics, whereas marathons and competitions emphasized application under constraints. Their recurrence allowed examples and explanations to be revisited before and after publication of the books, connecting resource development to authentic use.

The evidence also has important limitations. Adoption was identified from syllabi and confirmations from some instructors, but the dataset does not report how each instructor used the material, the number of students who accessed each resource, or whether the book was required, recommended, or

TABLE IV
ACTIVITIES SUPPORTING FORMATIVE DEVELOPMENT AND AUTHENTIC USE

Years	Type	Event	Theme	Hours/event	Instructor ^a
2019	Short course	Regional School on HPC/Rio de Janeiro (ERAD-RJ)	MPI programming	2	a
2019	Short course	Workshop on HPC Systems (WSCAD)	OpenACC programming	2	c
2019–2021	Short course	Regional School on HPC/Rio de Janeiro (ERAD-RJ)	OpenACC programming	2	c
2020–2022	Short course	Regional School on HPC/Southern Brazil (ERAD-RS)	OpenACC programming	2	c
2021	Short course	Regional School on HPC/Rio de Janeiro (ERAD-RJ)	OpenACC for beginners	2	b
2022–2026	Short course	Santos Dumont Supercomputer School	MPI programming	4	a
2022–2025	Marathon	IEEE/SBC Int. Symp. on Computer Architecture and High Performance Computing (SBAC-PAD)	Marathon of Parallel Programming	32	b
2023–2025	Marathon	Regional School on HPC/Southern Brazil (ERAD-RS)	Marathon of Parallel Programming	24	b
2024	Short course	Symposium on HPC Systems (SSCAD)	MPI + OpenMP offloading	2	c
2024	Short course	Regional School on HPC/Southern Brazil (ERAD-RS)	OpenMP tasks	2	b
2024	Marathon	Regional School on HPC/São Paulo (ERAD-SP)	Marathon of Parallel Programming	32	b
2024–2026	Bootcamp	Santos Dumont Supercomputer School	Applied parallel programming	8	b
2025	Short course	Regional School on HPC/Southern Brazil (ERAD-RS)	Multiple GPUs with OpenACC	2	b
2025	Marathon	Regional School on HPC/Northeastern Brazil (ERAD-NE)	Marathon of Parallel Programming	24	b
2025–2026	Short course	Santos Dumont Supercomputer School	Hybrid MPI+OpenMP programming	4	a

^a a - Gabriel P. Silva; b - Calebe P. Bianchini; c - Evaldo B. Costa.

TABLE V
UNDERGRADUATE COURSES USING THE FIRST VOLUME

Institution	No.	Course(s)	Institution	No.	Course(s)
IFAM	1	Computer Science	IFCE	1	Systems Analysis and Development
IMPA Tech	1	Mathematics of Technology and Innovation	UFABC	1	Computer Science
UFAM	1	Software Engineering	UFERSA	1	Computer Engineering
UFMS	1	Artificial Intelligence	UFRJ	1	Computer Science
UFSB	1	Computer and Information Engineering	UFT	1	Computer Science
UNICENTRO	1	Computer Science	Mackenzie Presbyterian University	2	Computer Science; Computer Engineering
FGV	1	Systems Analysis and Development			

supplementary in every individual implementation beyond the adoption criterion. The activity record provides dates, types, themes, duration, and an aggregate estimate of participants, but not standardized feedback, completion rates, or participant profiles.

The interactive resources are conceptual visualizations rather than performance models. They simplify runtime decisions, timing, communication, contention, scalability, and implementation details and should therefore serve as bridges to source-code execution and subsequent experiments, not as faithful runtime representations [34]. Interaction logging and controlled studies have not yet been conducted; consequently, the evaluation remains analytical and based on documented use rather than comparison groups or pre/post learning measures.

Usage data are not yet sufficient for a meaningful analysis because the standard GitHub traffic information available to repository maintainers covers only a rolling 14-day period and therefore does not provide a reliable historical record of views, clones, or downloads. For this reason, the present study does not make quantitative claims about the cumulative use of these resources.

VIII. CONCLUSION

This paper has presented two textbooks and their complementary resources as a Portuguese-language HPC education ecosystem whose scientific contribution lies in its design, formative development, progressive articulation, adoption, and preliminary evaluation. The ecosystem connects foundational and advanced textbooks, reproducible programs, instructor and

learner materials, interactive resources for MPI, OpenMP, and OpenACC, and sustained training and outreach.

The available evidence documents technical validation, formative refinement, authentic use, and adoption in 14 undergraduate courses at 13 Brazilian institutions, with a potential annual reach of approximately 700 students. It also records activities from 2019 to 2026 that reached approximately 500 participants. The analytical framework indicates conceptual alignment, progressive complexity, reproducibility, and adoptability, while the limitations prevent claims of demonstrated learning gains.

Future work will combine pre- and post-activity questions, program-understanding tasks, usability questionnaires, instructor feedback, and interaction logs to distinguish usability, engagement, conceptual understanding, and programming performance. It will also extend the visualizations, expand and consolidate the existing collection of Jupyter and Google Colab laboratory notebooks, and revise the examples as programming standards and heterogeneous architectures evolve.

Future releases will also incorporate privacy-preserving usage measurement and versioned archival records, when appropriate, to support longitudinal analysis of access and reuse. These indicators will be interpreted separately from evidence of learning and formal course adoption.

ACKNOWLEDGMENTS

The authors used AI tools to assist in restructuring and condensing the manuscript and to generate preliminary English-language formulations for selected passages, particularly in the

Abstract, table formatting, the Conclusion, and the Artifact Description appendix. The tools also assisted with LaTeX-related phrasing and consistency checks. They were not used to collect or fabricate data, perform experiments, calculate adoption metrics, or independently conduct the evaluation. The paper’s bibliographic references were independently selected and verified by the authors and were not generated by the tools. All AI-assisted content, factual claims, citations, and conclusions were critically reviewed, revised, and verified by the authors, who take full responsibility for the final content.

REFERENCES

- [1] R. K. Raj, C. J. Romanowski, J. Impagliazzo *et al.*, “High performance computing education: Current challenges and future directions,” in *Proc. ITiCSE Working Group Reports*, 2020, pp. 51–74.
- [2] P. Bouvry, M. Brorsson, R. Canal *et al.*, “The european master for HPC curriculum,” *J. Parallel Distrib. Comput.*, vol. 201, p. 105081, 2025.
- [3] T. de Jesus Oliveira Duraes, P. Sergio Lopes de Souza, G. Martins, D. Jose Conte, N. Garcia Bachiega, and S. Mazzini Bruschi, “Research on parallel computing teaching: state of the art and future directions,” in *2020 IEEE Frontiers in Education Conference (FIE)*, 2020, pp. 1–9.
- [4] E. Frachtenberg, “Experience and practice teaching an undergraduate course on diverse heterogeneous architectures,” in *Proc. EduHPC*, 2021, pp. 1–8.
- [5] J. C. Adams, “Teaching parallel and distributed computing concepts using OpenMPI and Java,” in *Proc. IEEE HiPC Workshops*, 2021, pp. 4–11.
- [6] T. Islam and G. Phelps, “HPC@SCALE: A hands-on approach for training next-gen HPC software architects,” in *Proc. IEEE HiPC Workshops*, 2021, pp. 29–34.
- [7] M. Siegmann, A. C. Calder, S. Feldman, and R. J. Harrison, “Educating HPC users in the use of advanced computing technology,” in *Proc. EduHPC*, 2021, pp. 16–23.
- [8] H. Casanova, R. Ferreira da Silva, A. Gonzalez-Escribano *et al.*, “Peachy parallel assignments (EduHPC 2021),” in *Proc. EduHPC*, 2021, pp. 51–57.
- [9] R. Carratalá-Sáez, A. Gonzalez-Escribano, D. P. Bunde *et al.*, “Peachy parallel assignments (EduHPC 2022),” in *Proc. EduHPC*, 2022, pp. 50–56.
- [10] M. Bucker, A. Cameron, R. S. Canon *et al.*, “Peachy parallel assignments (EduHPC 2023),” in *Proc. SC Workshops*, 2023, pp. 366–373.
- [11] A. Gonzalez-Escribano, D. García-Álvarez, and P. Cámara, “DNA sequence alignment: An assignment for OpenMP, MPI, CUDA, and OpenCL,” in *Proc. EduHPC*, 2024.
- [12] C. Almeida, E. Shoop, D. García-Álvarez *et al.*, “Peachy parallel assignments (EduHPC 2025),” in *Proc. SC Workshops*, 2025, pp. 409–415.
- [13] G. Ballard and R. Parsons, “Visualizing parallel dynamic programming using the thread safe graphics library,” in *Proc. EduHPC*, 2021, pp. 24–31.
- [14] P. Dewan, H. Dinh, P. V. Hai *et al.*, “Automating testing of visual observed concurrency,” in *Proc. EduHPC*, 2021, pp. 32–42.
- [15] J. E. Coulter, M. D. Weiner, A. Jezghani *et al.*, “ICE 2.0: Interactive computing environment for education and research,” in *Proc. SC Workshops*, 2023, pp. 591–597.
- [16] J. Guitart, “Embedded scaffolding for teaching and assessing inquiry-based hands-on laboratory on distributed systems,” *J. Parallel Distrib. Comput.*, vol. 201, p. 105082, 2025.
- [17] N. Netterville, K. Fan, S. Kumar, and T. Gilray, “A visual guide to MPI all-to-all,” in *Proc. IEEE HiPC Workshops*, 2022, pp. 20–27.
- [18] D. P. Bunde, D. F. Shell, L.-K. Soh *et al.*, “Adopting heterogeneous computing modules: Experiences from a ToUCH summer workshop,” in *Proc. EduHPC*, 2022, pp. 18–25.
- [19] R. Brown, X. Zhang, J. Yang *et al.*, “Faculty development workshops for integrating parallel and distributed computing in undergraduate CS courses,” in *Proc. SC Workshops*, 2023, pp. 314–323.
- [20] S. Neuwirth, “Bridging the gap between education and research: A retrospective on simulating an HPC conference,” in *Proc. EduHPC*, 2022, pp. 34–41.
- [21] K. Läufer and G. K. Thiruvathukal, “UnoAPI: Balancing performance, portability, and productivity (P3) in HPC education,” in *Proc. EduHPC*, 2022, pp. 1–10.
- [22] S. V. Perez, “Designing an independent study to create HPC learning experiences for undergraduates,” in *Proc. IEEE HiPC Workshops*, 2022, pp. 6–11.
- [23] E. Krishnasamy, D. Pleiter, and P. Bouvry, “Student challenges in scientific computing: Insights from EUMaster4HPC organization and execution,” in *Proc. IEEE HiPC Workshops*, 2025, pp. 36–42.
- [24] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message-Passing Interface*, 3rd ed. MIT Press, 2014.
- [25] B. Chapman, G. Jost, and R. van der Pas, *Using OpenMP: Portable Shared Memory Parallel Programming*. MIT Press, 2007.
- [26] M. J. Quinn, *Parallel Programming in C with MPI and OpenMP*. McGraw-Hill, 2003.
- [27] S. Chandrasekaran and G. Juckeland, *OpenACC for Programmers: Concepts and Strategies*. Addison-Wesley, 2017.
- [28] UNESCO, “Recommendation on open educational resources (oer),” 2019. [Online]. Available: <https://www.unesco.org/en/legal-affairs/recommendation-open-educational-resources-oer>
- [29] D. A. Wiley, “Connecting learning objects to instructional design theory: A definition, a metaphor, and a taxonomy,” in *The Instructional Use of Learning Objects*, D. A. Wiley, Ed. Bloomington, IN: Agency for Instructional Technology and Association for Educational Communications and Technology, 2002.
- [30] IEEE, “IEEE Standard for Learning Object Metadata,” IEEE Std. 1484.12.1-2020, Nov. 2020, [Online]. Available: https://standards.ieee.org/standard/1484_12_1-2020.html.
- [31] L. Marzulo, C. Bianchini, L. Santiago, V. Ferreira, B. Goldstein, and F. França, “Teaching high performance computing through parallel programming marathons,” in *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2019, pp. 296–303.
- [32] G. P. Silva, C. P. Bianchini, and E. B. Costa, *Programação Paralela e Distribuída com MPI, OpenMP e OpenACC para Computação de Alto Desempenho*. Casa do Código, 2022, <https://www.casadocodigo.com.br/products/livro-programacao-paralela>.
- [33] —, *Programação Paralela para Arquiteturas Híbridas: Com OpenMP, OpenACC e MPI*. Casa do Código, 2026, <https://www.casadocodigo.com.br/products/livro-paralela-arquiteturas-hibridas>.
- [34] V. Eijkhout, “Teaching MPI from mental models,” in *Proc. EduHPC*, 2016, pp. 14–18.

Appendix: Artifact Description

Artifact Description (AD)

I. OVERVIEW OF CONTRIBUTIONS AND ARTIFACTS

A. Paper's Main Contributions

- C_1 Design and formative development of an integrated, Portuguese-language ecosystem that provides a progressive pathway from introductory parallel-programming concepts to distributed-memory, shared-memory, accelerator, and hybrid programming.
- C_2 Evidence of adoption and sustained use obtained from courses, training events, and dissemination activities conducted between 2019 and 2026.
- C_3 Analytical evaluation of the ecosystem with respect to conceptual alignment, progressive complexity, technical reproducibility, adoptability, authentic use, and current limitations.

B. Computational Artifacts

The two artifacts have complementary roles. The project portal presents the two books as a single educational pathway and organizes direct access to their slides, source-code repositories, and interactive learning resources. The development repositories contain the programs, notebooks, exercises, slides, and implementation files that can be downloaded, executed, modified, and reused.

- A_1 **Project portal:** <https://programacao-paralela-e-distribuida.github.io/>.
- A_2 **Development repositories and archived release:** <https://github.com/Programacao-Paralela-e-Distribuida>; archived version associated with the paper: <https://doi.org/10.5281/zenodo.21711214>.

Artifact ID	Contributions supported	Related paper elements
A_1	C_1, C_3	Sections III, V-A and V-C: the two-volume curricular pathway, the association of slides and repositories with each volume, and the six interactive learning resources.
A_2	C_1, C_3	Sections III, V-A and V-B: reproducible C programs, exercises, slides, six Colab-compatible notebooks, and the progression from MPI and OpenMP to tasks, offloading, and hybrid programming.

Neither artifact is used to recalculate the adoption indicators reported under C_2 . The records summarized in the paper document adoption in 14 courses at 13 institutions, approximately 500 participants in training and outreach activities, and approximately 520 copies of the two volumes sold. Those

figures come from course and activity records and publisher sales totals, not from the portal or repository contents; no participant-level records are distributed with the artifacts.

II. ARTIFACT IDENTIFICATION

A. Computational Artifact A_1

Relation To Contributions

Artifact A_1 is the public entry point to the ecosystem. Under the heading *Programação Paralela e Distribuída—Livros e Recursos Educacionais*, it presents the books as two complementary stages of one learning trajectory. The first volume introduces fundamental concepts, performance evaluation, MPI, OpenMP, and OpenACC. The second advances to parallel architectures, OpenMP tasks and offloading, advanced OpenACC, multiple GPUs, and hybrid MPI+OpenMP and MPI+OpenACC programming. The same page then organizes the slides, repositories, and visualization tools associated with this trajectory. This supports C_1 and the adoptability dimension of C_3 without directing users to separate portals for the two volumes.

Expected Results

A reviewer should be able to use the portal to reconstruct the educational progression described in Sections III, V-A and V-B. The opening presentation identifies the first volume as the entry point and the second as its advanced continuation. The *Slides* section separates introductory MPI, OpenMP, and OpenACC materials from OpenMP-tasks and hybrid-programming materials. The *Repositórios* section identifies MPI with the first volume, OpenMP and OpenACC with both volumes, MPI+X with the second, and sequential programs with both. The reviewer should also be able to open the six browser-based learning resources described in Section V-C.

The three MPI resources move from regular cyclic decomposition and reduction in numerical integration, to an irregular prime-counting workload, and then to explicit data movement in matrix–vector multiplication. The OpenMP resource compares loop-scheduling policies, while the OpenMP and OpenACC resources show how `collapse` and `tile` change the mapping of a matrix iteration space.

The expected result is not a benchmark value. It is the correspondence between the concepts named in the portal, the controls exposed by each learning object, and the changes displayed in process/thread assignment, communication, loop transformation, data movement, or potential data-dependence problems.

Expected Reproduction Time (in Minutes)

Execution times may vary depending on the characteristics of the computing environment in which the artifacts are executed; the values below are approximate estimates. Accessing the portal takes less than one minute and requires no local setup. A representative traversal and interaction with

the six learning resources takes approximately 30–45 minutes. Comparing the observed progression with Sections III, V-A and V-B of the paper requires an additional 15–20 minutes.

Artifact Setup (incl. Inputs)

Hardware: Any computer capable of running a current graphical Web browser is sufficient. No cluster or GPU is required because the portal and its interactive learning objects are static Web resources that illustrate educational models of the parallel behavior.

Software: A current standards-compliant browser with JavaScript enabled is required. The portal was designed as a static website and does not require a local Web framework, compiler, account, or database. The source distributed through the project is licensed under the GNU General Public License v3.0.

Datasets / Inputs: No external dataset is used. Inputs are supplied through the controls of each learning object. Depending on the selected resource, they include the number of processes or threads, number of iterations or integration steps, scheduling policy, chunk size, matrix dimensions, `collapse` depth, and tile sizes.

Installation and Deployment: Open the A_1 URL in a browser. A local copy is optional; if one is used, the static files may be opened directly or served with any local static Web server.

Artifact Execution

The portal workflow uses the following paper-specific tasks:

- T_1^P **Identify the roles of the two books.** Read the opening *Uma trajetória de aprendizagem em dois volumes* and the two book panels. Confirm that the first volume is presented as the introductory pathway through fundamentals, performance, MPI, OpenMP, and OpenACC, while the second extends that pathway to tasks, offloading, multiple GPUs, and hybrid programming.
- T_2^P **Inspect the mapping of supporting resources.** In *Slides*, distinguish the material associated with Book 1 from the OpenMP-tasks and hybrid-programming material associated with Book 2. In *Repositórios*, verify the stated association of the MPI, OpenMP, OpenACC, MPI+X, and SERIAL repositories with one or both books.
- T_3^P **Inspect the MPI progression.** Open the numerical integration, prime-counting, and matrix–vector multiplication resources. Vary the process count and problem parameters, advance the simulations, and identify cyclic work distribution, local results and reduction, and the `MPI_Bcast`, `MPI_Scatter`, and `MPI_Gather` stages.
- T_4^P **Compare OpenMP schedules.** In the loop-scheduling resource, keep the iteration and thread counts fixed while comparing static, dynamic, and guided schedules, with and without an explicit chunk size.
- T_5^P **Explore loop transformations.** In the OpenMP and OpenACC matrix resources, change the `collapse` depth and tile sizes. Observe the mapping to threads or gangs

and the warning that appears when a transformation distributes contributions to the same result element across execution units.

T_6^P **Relate the observations to the paper.** Compare the two-volume and supporting-resource organization in T_1^P – T_2^P with Sections III, V-A and V-B, and compare T_3^P – T_5^P with the progression and limitations described in Section V-C.

The dependencies are

$$T_1^P \rightarrow T_2^P \rightarrow \{T_3^P, T_4^P, T_5^P\} \rightarrow T_6^P.$$

Each configuration needs to be run only once. Repetition is useful only when discussing the runtime-dependent assignments represented by dynamic or guided OpenMP scheduling; the paper makes no statistical performance claim from these simulations.

Artifact Analysis (incl. Outputs)

For T_1^P , the reviewer should find both books presented with equal visibility as the introductory and advanced stages of a common pathway. For T_2^P , the slide and repository groupings should make explicit which resources support Book 1, Book 2, or both. For T_3^P – T_5^P , changing the controls should update the visual distribution and explanatory output without requiring compilation. The MPI examples should distinguish local work from collective communication; the scheduling example should distinguish fixed static assignments from runtime-dependent schedules; and the loop tools should connect `collapse` and `tile` to work distribution and correctness.

These checks establish that the portal exposes the organization and interactive resources analyzed in the paper. They do not measure whether a student learned more after using them. Such a claim would require learner responses, pre/post measures, or comparison data, none of which were collected for this study.

B. Computational Artifact A_2

Relation To Contributions

Artifact A_2 contains the materials used to turn the progression presented by the portal and textbooks into practical activities. Its live organization separates MPI examples, OpenMP examples used by both books, OpenACC examples used by both books, hybrid MPI+X examples for the second volume, and sequential reference programs. It also includes compilation and execution instructions, exercises and case studies, slides, six Jupyter/Google Colab notebooks, and the source of the interactive resources. The live repositories record continued development, while the Zenodo DOI identifies the fixed version associated with the paper. This artifact supports C_1 and the technical reproducibility and reuse dimensions of C_3 .

Expected Results

The expected outcome is successful compilation and functional execution of a representative path through the materials. An introductory path covers MPI execution and collectives followed by OpenMP work sharing. The advanced

path adds OpenMP tasks, accelerator offloading, and a hybrid MPI+OpenMP or MPI+OpenACC example. The six notebooks provide a guided version of this progression: three cover MPI fundamentals and collectives, two cover OpenMP tasks, and one introduces hybrid programming and accelerator offloading.

Results should remain functionally correct when the process or thread count is changed, although ordering and timing may differ. Accelerator examples require a compatible GPU only when device execution is to be reproduced.

Expected Reproduction Time (in Minutes)

Downloading the archived release and checking the CPU toolchain takes 15–30 minutes on a prepared Linux system. Compiling and running the representative CPU sequence takes 45–60 minutes. Executing selected notebook activities takes 45–90 minutes. The optional accelerator path may add 30–60 minutes if the NVIDIA HPC SDK and a compatible GPU are already available. Output inspection takes approximately 20–30 minutes. Again, these are rough estimates.

Artifact Setup (incl. Inputs)

Hardware: The MPI and OpenMP programs require a conventional multicore computer running a POSIX-compatible system. MPI programs may be executed with multiple processes on one machine; a cluster is optional. OpenACC and accelerator-offloading programs require a compatible NVIDIA GPU for device execution. Google Colab may be used for the notebooks, subject to the resources made available by the selected runtime.

Software: The archived release was validated on Linux with the following toolchain:

- GCC 13.1 with OpenMP support, <https://gcc.gnu.org/>;
- Open MPI 4.1.2, <https://www.open-mpi.org/>;
- NVIDIA HPC SDK 26.5 for the OpenACC/offloading examples, <https://developer.nvidia.com/hpc-sdk/releases>;
- Jupyter or Google Colab for the notebooks; and
- a current standards-compliant browser for Web resources stored in the repositories.

Artifact A_2 is distributed under the GNU General Public License v3.0. The version-specific archive is identified by the DOI listed above.

Datasets / Inputs: No research dataset is required. The programs use inputs included in the examples, command-line parameters, or constants documented with the source. The notebooks create their source files and small teaching inputs within their cells. Parameters to vary include process and thread counts, problem size, scheduling policy, chunk size, task granularity, and the selected CPU or GPU execution path.

Installation and Deployment: Download and unpack the version-specific archive from the DOI. Begin with its top-level README, then follow the README or notebook instructions for each selected component. CPU examples are compiled with the documented `mpicc` or `gcc -fopenmp` commands. Notebooks are opened in Google Colab or Jupyter and executed from the environment-preparation cells onward.

OpenACC and offloading examples use the NVIDIA HPC SDK commands stated with those programs. The live GitHub organization may be inspected for later corrections, but reproduction of the paper should use the archived version.

Artifact Execution

The repository workflow uses the following paper-specific tasks:

T_1^R **Fix the evaluated version and prepare the tools.**

Download the Zenodo release, record its version, and verify the C, MPI, and OpenMP compiler commands documented in the top-level README.

T_2^R **Run the distributed-memory stage.**

Compile one MPI case study, such as numerical integration or matrix–vector multiplication, and run it with 1, 2, and 4 processes. Confirm that the global numerical result is preserved while work and communication are distributed among ranks.

T_3^R **Run the shared-memory stage.**

Compile one OpenMP loop example and run it with 1, 2, and 4 threads. Keep the input fixed and confirm that the final result remains unchanged. When the example permits it, vary the schedule or chunk size to inspect work distribution.

T_4^R **Execute the guided notebook sequence.**

Run at least one MPI notebook, one OpenMP-tasks notebook, and the hybrid notebook from top to bottom. Complete the prediction, code-modification, or interpretation steps included in the selected notebooks.

T_5^R **Exercise the advanced path.**

If a compatible GPU is available, compile and run one OpenACC or OpenMP-offloading example and compare its functional result with the CPU version. Otherwise, inspect the source and the documented compilation path; this task is optional for CPU-only review.

T_6^R **Relate execution to the progression.**

Check that the selected examples move from MPI and OpenMP fundamentals to tasks, offloading, or hybrid execution, as described in Sections V-A and V-B.

The dependencies are

$$T_1^R \rightarrow \{T_2^R, T_3^R, T_4^R, T_5^R\} \rightarrow T_6^R.$$

Only one functional run is required for each configuration. The paper does not derive comparative performance results from this artifact, so repeated timing runs, confidence intervals, and statistical tests are outside this validation workflow.

Artifact Analysis (incl. Outputs)

For T_2^R and T_3^R , the same problem input should produce the same final functional result across the selected process or thread counts; message ordering, thread assignment, and elapsed time may vary. For T_4^R , the selected notebooks should prepare the environment, generate or expose the C source, compile it, run it, and present the questions or modifications that turn the example into a laboratory activity. For T_5^R , the CPU and device paths should agree within the numerical tolerance documented with the example.

This workflow verifies that the executable materials supporting the curricular sequence are present and reusable. The reported adoption figures cannot be reconstructed from GitHub traffic: the Web tools and notebooks were released too recently to provide a meaningful longitudinal series, and the standard traffic view available to repository maintainers retains only a rolling 14-day window. For this reason, the paper uses course, activity, and publisher records for adoption and dissemination and makes no claim that repository accesses demonstrate either formal adoption or learning outcomes.