HPC@SCALE: A Hands-on Approach for Training Next-Gen HPC Software Architects

Tanzima Z. Islam
Computer Science
tanzima@txstate.edu
Texas State University

Chase Phelps
Computer Science
chaseleif@txstate.edu
Texas State University

***Abstract*—High Performance Computing (HPC) systems enable multi-scale simulations to gain meaningful insights into otherwise experimentally intractable phenomena such as climate change and destabilizing drug-protein interactions to cure cancer. High levels of parallelism and heterogeneous architectures in these systems offer unprecedented computational capability at the cost of complexity and dynamism. Recent efforts in workforce development have focused on preparing students with the background to write a program for these complex heterogeneous platforms successfully [1]. However, computing is only one of the three tasks an HPC application performs; the other two are communication and I/O. Since network bandwidth is not scaling proportionately with computational capabilities, moving the large volume of data generated by these applications through the network slows down scientific progress. A high-level data-driven analysis shows that most existing curricula do not prepare students to consider design choices to scale parallel I/O, which is a crucial building component of an end-to-end system. At its core, the problem of scaling data-intensive applications is common in both high-performance, high-throughput, and Cloud computing environments, so any training in that regard will have a broad impact. To fill this gap, we have designed a new course called HPC@SCALE to train students at Texas State University in building scalable end-to-end system software focusing on minimizing parallel I/O.**

***Index Terms*—HPC, parallel I/O, curriculum development, hands-on, checkpointing system, parallel file system.**

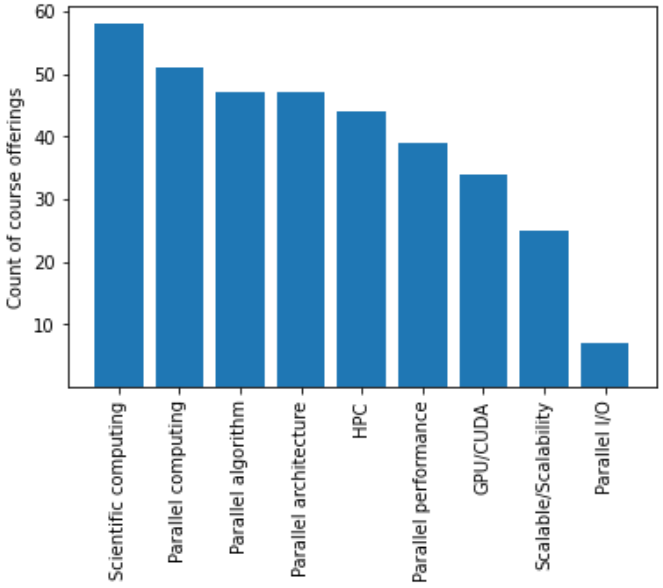


Fig. 1: Analysis of keywords from the course catalogs of 80 universities from the list of top 100 universities according to the EduRank website [2].

I. INTRODUCTION

Large-scale distributed systems enable high-performance and high-throughput applications to either execute many parameter sweeps or an application with large memory needs to finish fast. Typical large-scale applications include three phases of operations—computation, communication, and I/O. The strength of parallel programming is that it breaks up a large problem into smaller parts and leverages separate processes on distributed clusters to tackle smaller chunks of the problem. While some parts of the problem can be solved without any communication among the processes, periodically, they need to communicate and exchange information among themselves (communication and I/O). Although traditionally the term "I/O" means input-output to the disk, in HPC, the I/O phase can also include transferring data through the network to a remote memory or disk, which involves network communication. This paper refers to HPC I/O as "parallel I/O".

While Parallel Computing or Heterogeneous Programming courses prepare students to utilize the on-node resources, they do not focus on how to handle the challenges of data transfer. For example, we analyzed the publicly available course catalog descriptions of 80 universities from the top 100 list made available on EduRank [2]. To automate the process, we developed a software for pulling and parsing the course catalogs of these top universities, and searching for the following keywords most commonly associated with HPC: `scientific computing`, `parallel computing`, `parallel algorithm`, `parallel architecture`, `HPC`, `parallel performance`, `GPU or CUDA`, `scalable or scalability`, and `parallel I/O`. The X-axis in Figure 1 shows the keywords, and the Y-axis shows the number of universities having at least one course containing the corresponding keyword. If the same keyword appears across multiple courses in the same university's catalog, we consider them one appearance. This coarse-grained data-driven analysis draws attention to the disparity between the topics covered across top universities where 51 of them cover parallel

computing compared to only 7 teaching parallel I/O.

Excellent training resources exist as part of supercomputing centers such as XSEDE [3] and Texas Advanced Computing Center (TACC) [4]. However, these resources remain unknown to the students, or navigating them for self-training can be overwhelming. This gap in curriculum leaves students underprepared to be system architects, especially in the HPC domain, where the performance of an application is an aggregated metric combining that of computation, communication, and I/O phases. Failure to design these systems with all phases in mind will leave an application adopting parallel I/O patterns that are easier to implement but detrimental to their performance and scalability.

To fill this gap, we have designed a new course at Texas State University that takes a hands-on approach in teaching graduate (M.S. and Ph.D.) students the principles of designing scalable end-to-end systems. We also discuss our plan to adopt a subset of the modules for a future undergraduate-level offering of this course.

II. Design Principles

This course teaches the fundamental principles of building a scalable HPC system. Using a message-passing programming model, it covers the parallel I/O patterns and their characteristics and uses two large-scale systems– checkpointing and parallel file system (PFS), as use case scenarios to demonstrate how these principles are applied in practice. Additionally, students learn to measure and quantify performance trade-offs of their design choices and use the principles to implement their scalable software substrate.

We designed this course to be both remote- and face-to-face instruction model-friendly. The pilot was offered remotely in spring 2021, and student evaluations indicate that the format has successfully maximized their learning in this class. Figure 2a maps the activities designed in the HPC@SCALE course according to Bloom's taxonomoy [5]. The lectures teach concepts about shared-memory, distributed-memory programming, parallel I/O, hierarchical data format (HDF5), and libraries for scalable I/O. Each lesson follows up with a hands-on lab to implement the abstract concepts. The course also includes two comparatively larger assignments built on the labs with more complex computational logic. Finally, in the semester-long project, students propose a problem, formulate it as a system development project with parallel I/O, compare at least two different design choices using the performance metrics discussed in lectures.

III. Organization and Content

A. Context

This course is primarily intended for graduate students, although in this paper, we discuss the potential adoption of the modules for future offering to senior undergraduate students as an elective. The course is divided into six modules, with two weeks per module. Five of these modules are applicable for higher-division undergraduate students. The format of this course is a lecture followed by an in-class lab (seven), programming assignments (two), and a semester-long project. To adopt materials from this course into an undergraduate course, we need to incorporate a comprehensive final exam as per the undergraduate curricula requirement.

B. Topics

The HPC@SCALE course covers the following list of topics.

1) Concurrency and Parallelism
2) Parallel Communication
3) Parallel I/O
4) Performance Measurement and Analysis
5) Use Case Scenarios: Checkpointing System and Parallel File System
6) Advanced topics

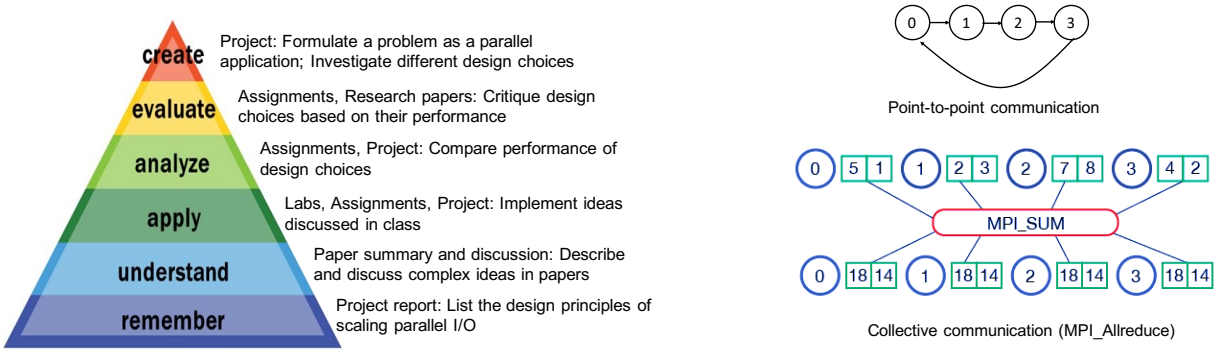
C. Learning Outcomes

Having completed this course, students should be able to:

- Describe the difference between a shared-memory and a distributed-memory programming model. List the design principles of scaling parallel I/O.
- Apply the theory behind scalable parallel communication and implement I/O patterns through hands-on labs, assignments, and a course-long project.
- Critique the design choices of parallel systems based on their performance measured using software tools.
- Design and develop a scalable and resilient parallel system.
- Describe and discuss complex ideas presented in research papers on relevant topics and communicate through research presentation and report.

D. Details

1) Module I: Concurrency and Parallelism: This module includes an introduction to shared- and distributed-memory parallel programming; presents examples of common parallel and heterogeneous architectures; introduces the three main phases of applications—computation, communication, and I/O. Students familiar with the organization of a computing system and inter-process communication are presented with the new concept of remote memory access through high-level programming constructs such as message passing interface (MPI). Additionally, to ensure that students without prior parallel computing experience can ease into the course, this module teaches parallel computing basics, terminologies, and parallel system architecture typical in HPC (CPUs and GPUs). This module runs for two weeks (three hours a week); each class delivers 1.5 hours of lecture followed by 1.5 hours of the hands-on lab to implement the theory covered. For shared-memory parallel programming, this module uses the OpenMP library and leverages training materials from Lawrence Livermore National Laboratory [6]. This section introduces distributed-memory parallel programming using MPI in this module and only covers the point-to-point communication aspect. This module primarily focuses on the compute phase of applications and thus may contain contents covered in many Parallel and HPC-related courses throughout the nation.



(a) (b)

Fig. 2: (a) Mapping the activities of the HPC@SCALE course according to Bloom's taxonomy [5]. (b) Two example parallel communication patterns implemented during the labs.

Lab: Each class ends with a 1.5-hour lab (total of two labs in Module I). The first lab provides instructions for (a) accessing a remote supercomputer, (b) submitting jobs through a job scheduler such as SLURM, and (c) implementing a simple program using OpenMP constructs. The second lab provides instructions to write a distributed-memory application using the point-to-point communication constructions in the MPI programming model to accomplish the top communication pattern illustrated in Figure 2b. Each rank sends a random number to the process with the id one greater than its own and receives that from the process with id one less than its own. Process 0 receives from the process with id $p-1$ (p is the total number of processes); process with id $p-1$ sends to process 0. Students face a common challenge during this lab: their programs hang when their calculation of the source or destination process id is incorrect. Through working out in a supervised setting, students first-hand learn to identify a mismatch in a parallel communication scenario, which prepares them to be cautious for the rest of the semester.

Programming assignment 1: In this assignment, students experiment with a tool `hwloc` [7]; learn to describe a computing system architecture for experimental purposes; parallelize a given sequential algorithm using a shared-memory programming model such as OpenMP. Currently, this course focuses on CPU architectures to reduce the learning curve for students; however, with OpenMP 5.0 and task offloading capability of the programming models, future revisions of this course can introduce students to leverage heterogeneous architectures without additional efforts. This module emphasizes validating output since parallelization can introduce unwanted errors in the computation. Specifically, students are asked to parallelize using OpenMP constructs (e.g., `pragma omp parallel for [static | dynamic(chunk_size)]`) to parallelize the computation of PI. In addition to writing code and comparing the results of the sequential and parallel implementations, students also learn to report the overhead of applications using the time function. Students also experiment with different loop scheduling constructs, chunk sizes, and thread affinity parameters (e.g., binding threads to `[cores | sockets]` in one of the following orders: `[compact | scatter]`. Students find first-hand that the choice of configuration parameters can have a detrimental effect on performance. This assignment introduces students to the systematic process of performance evaluation, analysis, and reporting.

2) Module II: Parallel Communication: This module teaches students about the other two phases of parallel applications—communication and I/O. I/O typically implies transferring data to and from remote memory or parallel file system via the network in an HPC environment. Hence, in this context, parallel I/O involves parallel communication as well as disk access. This module teaches about scalable communication patterns that students implement during labs. They also learn to use libraries that provide APIs implementing these patterns under the hood. Students implement both point-to-point and collective communication patterns, measure their performance, and compare. Specific topics covered in this module include types of parallelism, granularity, parallel execution models (e.g., producer-consumer, work pool, data-parallel, task-parallel), collective communication, and non-blocking communication.

Lab: This module contains two labs where students implement both blocking and non-blocking communication using various MPI functions such as MPI_gather, MPI_scatter, MPI_Allgather, MPI_Allscatter, MPI_Allreduce (the bottom image in Figure 2b) and MPI_Isend and MPI_Irecv. In these labs, students implement bare-bone programs, e.g., processes exchange their rank information across the network. These labs provide formative assessment to students and connect the theoretical concepts discussed in class to their implementations. These labs also provide the templates for their programming assignment in this module.

Programming assignment 2: In this assignment, students implement two commonly used execution models—producer-consumer and work pool—using collective and non-blocking communication constructs. Students also conduct experiments with increasing processes and run the two execution models for the same time. Then, students visualize the comparative

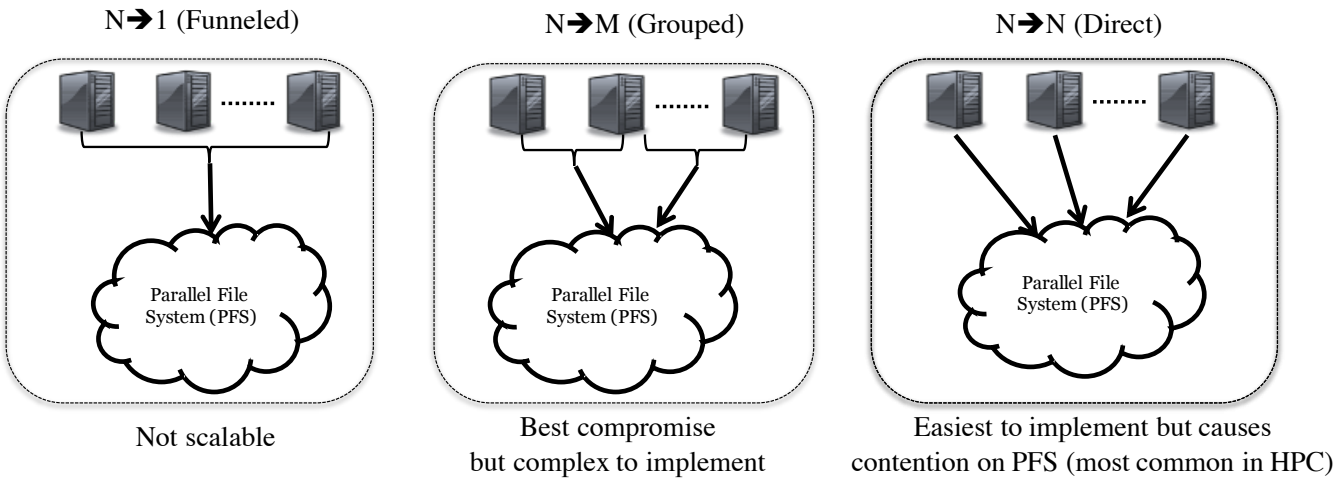


Fig. 3: Parallel I/O patterns common in HPC applications. Students implement the parallel I/O pattern $N \rightarrow M$ during the hands-on activities of the course.

performance of the two execution models based on their throughput and write a report explaining the bottleneck of the algorithms and how that manifests in throughput.

E. Module III: Parallel I/O

This module teaches students about scalable parallel I/O patterns (shown in Figure 3), namely $N \rightarrow 1$ (funneled), $N \rightarrow M$ (grouped), and $N \rightarrow N$ (direct). Students learn that while direct I/O from an individual process to the PFS is the easiest pattern to implement for an application scientist, the grouped communication pattern is the best one for scalable systems. The grouped I/O pattern is used by scalable systems such as large-scale checkpoint-restart library [8] for HPC environments. Students also learn about hierarchical data format and libraries such as HDF5 to annotate application output and achieve portable data management.

Lab: This module includes two labs where students implement the $N \rightarrow M$ parallel I/O pattern using MPI point-to-point and collective communication constructs, implement collaborative file writing to a shared file using MPI-IO [9], and implement a parallel application that collectively reads and writes files in HDF5 format.

F. Module IV: Performance Measurement and Analysis

In this module (four papers in two weeks), students learn about performance indicators, metrics, performance measurement tools and read related papers about different performance analytics methods. Specifically, students learn to use both source-annotation-based tools such as Perf-Dump [10] and dynamic library-based measurement tools such as Tau [11] and Caliper [12]. For communication performance, students use tools such as mpip [13], and for I/O performance, students use the Darshan [14] tool. Graduate students also read selected research papers to learn about different analysis methods ranging from analytical to data-driven and list the common performance-related research questions that emerge from those papers. Undergraduate students can use a statistical approach to reporting performance numbers such as box-and-whisker plots and quantifying distribution.

Lab: In the hands-on lab, students choose one of the performance measurement tools on their previously implemented assignments to measure and analyze their performance. Specifically, students collect hardware performance counters and execution time and analyze the usage characteristics of different levels of cache, memory, and CPU resources.

G. Module V: Use Case Scenarios

In this module (four papers in two weeks), students study research papers on use case scenarios that tackle communication and I/O-related design issues to scale. These use case scenarios range from user-level applications (e.g., deep learning) to system software such as checkpointing and parallel file systems.

1) Application: Deep Learning: Deep learning applications running on HPC platforms are I/O intensive. Students learn about various strategies used in state-of-the-art research, such as data aggregation and compression, staggered training, leveraging specialized hardware such as burst buffers to reduce the cost of data movement. Additionally, with the emergence of deep learning applications in HPC, students learn about using data-parallel and model-parallel methods to accelerate training large deep learning networks.

2) Middleware: Checkpointing Systems for Resilience: Students implement the bare substrate of a scalable checkpointing system in Module III. A similar system serves as the basis of an R&D 100 award-winning checkpoint-restart library SCR [15]. In this module, students primarily read research papers on different strategies to scale the checkpointing mechanism and discuss techniques to reduce the volume of communication and data through aggregation and compression.

3) System Software: Parallel File System (PFS): Students learn about the design of a production parallel file system, its scalability challenges and read papers to learn about the innovations that have taken place over the years to combat some of these issues. Students are asked to map these enhancements to the design principles discussed in the earlier modules. A key takeaway message of the use case scenarios is that performance is a trade-off space, and design choices may vary depending on the metric to optimize.

H. Module VI: Advanced Topics

In this module (typically one week), the instructor presents several active research topics in HPC, open questions, challenges of the existing solutions in the context of system development. For example, during the pilot offering of this course, students learned about the importance of dynamic resource management and job scheduling in HPC, the role of machine learning in that research space, and their performance and scalability challenges.

I. Course Project

In addition to the labs and assignments, this graduate course requires students to choose a problem either from their research area or from a provided list of problems to (1) parallelize an application using the hybrid programming model (MPI+OpenMP), (2) select a domain decomposition method (data-parallel, task-parallel), (3) implement shared parallel I/O, design metadata, and generate data in HDF5 format, (4) design performance evaluation experiments by identifying configuration parameters to tune (e.g., number of threads, number of processes, thread binding) and select performance metrics to report (e.g., execution time, device utilization, number of operations per cache miss, resource stall per byte moved). Finally, the students present their projects to peers and receive feedback to improve their research communication from both the instructor and peers. The following five questions guide student presentations:

- What is the problem?
- Why is it important to solve?
- What have others done, and where is the gap?
- What is the proposed solution?
- How does the proposed solution fill the identified gap(s)?

Students implement the scalable parallel I/O patterns using the MPI-IO or HDF5 libraries, non-blocking communication, use performance measurement tools to collect performance counters for different phases of an application, characterize performance and scalability using discussed metrics, compare performance and accuracy with a sequentially implemented version of the same code, and write a final report. The projects are evaluated based on their functionalities (whether the code worked, if the results match with the sequential implementation, if the characteristics of the performance or scalability metrics make sense as an application scales). Students often recreate similar experiments as presented in published papers.

IV. Computing Resources for Training

We applied for educational allocation at TACC on the following systems–Stampede (Intel Xeon Phi), Lonestar (CPU), and Maverick (GPU) supercomputers. The process was relatively straightforward. For an allocation, we wrote a proposal explaining the need for such training, the syllabus, the format of the course, how many students will participate and provided an estimate of the node hours. TACC is an HPC facility near the University of Texas at Austin and is committed to providing access to national computing resources for research and workforce development efforts.

V. Evaluation

We implemented a pilot version of this course at Texas State University in Spring 2021 for both Masters and Ph.D. students. This class had six enrolled students. The student evaluation form included questions such as (1) course organization, (2) relevance of course material to the learning objectives, (3) challenge level and if it was productive for maximizing learning, and (4) overall satisfaction with the course. Students answered each question on a scale of 0-5 (e.g., strongly disagree, disagree, neutral, agree, and strongly agree). On all accounts, the students chose "Strongly Agree", giving the course an average of 5.0 out of 5.0 in all regards. Due to the pandemic, this course was taught remotely on Zoom. The organization and materials of this course fit well with the remote learning environment since students can easily follow the lecture part of the course and implement the labs independently while asking questions via Zoom. The same design will work in a face-to-face environment if scheduled in a computing lab. There are several student labs in the CS department; hence, teaching resources will not be an issue when the course is taught in person.

VI. Conclusion and Future Work

This paper presents a curriculum development effort for training graduate students to develop scalable applications for HPC environments. The course covers parallel programming and I/O-relate concepts and complements the lectures with hands-on labs and assignments. Conducting experiments on production supercomputers reinforce their understanding of the conceptual materials. A preliminary evaluation is promising in terms of student learning outcomes. In the future, we plan to calibrate the content to offer it to senior undergraduate students as an elective course.

References

[1] A. Qasem, "A gentle introduction to heterogeneous computing for cs1 students," in *2019 IEEE/ACM Workshop on Education for High-Performance Computing, EduHPC@ SC 2019*, 2019.

[2] EduRank, "100 Best Computer Science schools in the United States," https://edurank.org/cs/us/.

[3] J. Towns, T. Cockerill, M. Dahan, I. Foster, K. Gaither, A. Grimshaw, V. Hazlewood, S. Lathrop, D. Lifka, G. D. Peterson, R. Roskies, J. R. Scott, and N. Wilkins-Diehr, "Xsede: Accelerating scientific discovery," *Computing in Science Engineering*, vol. 16, no. 5, pp. 62–74, 2014.

[4] T. A. C. C. (TACC). [Online]. Available: https://www.tacc.utexas.edu

[5] P. Armstrong, "Bloom's taxonomy," *Vanderbilt University Center for Teaching*, 2016.

[6] L. L. N. Laboratory, "OpenMP Tutorials," https://hpc.llnl.gov/tuts/openMP/.
[7] F. Broquedis, J. Clet-Ortega, S. Moreaud, N. Furmento, B. Goglin, G. Mercier, S. Thibault, and R. Namyst, "hwloc: A generic framework for managing hardware affinities in hpc applications," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 180–186.
[8] T. Z. Islam, K. Mohror, S. Bagchi, A. Moody, B. R. De Supinski, and R. Eigenmann, "Mcrengine: a scalable checkpointing system using data-aware aggregation and compression," *Scientific Programming*, vol. 21, no. 3-4, pp. 149–163, 2013.
[9] R. Thakur, E. Lusk, and W. Gropp, "Users guide for romio: A high-performance, portable mpi-io implementation," Argonne National Lab., IL (United States), Tech. Rep., 1997.
[10] T. Gamblin, "perf-dump," Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), Tech. Rep., 2013.
[11] S. Shende and A. D. Malony, "The Tau Parallel Performance System," *IJHPCA*, vol. 20, no. 2, pp. 287–311, 2006.
[12] D. Boehme, T. Gamblin, D. Beckingsale, P.-T. Bremer, A. Gimenez, M. LeGendre, O. Pearce, and M. Schulz, "Caliper: performance introspection for hpc software stacks," in *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2016, pp. 550–560.
[13] Lawrence Livermore National Laboratory, "mpiP: A lightweight profiling library for MPI applications," https://hpc.llnl.gov/software/development-environment-software/mpip.
[14] S. Snyder, P. Carns, K. Harms, R. Ross, G. K. Lockwood, and N. J. Wright, "Modular hpc i/o characterization with darshan," in *2016 5th workshop on extreme-scale programming tools (ESPT)*. IEEE, 2016, pp. 9–17.
[15] A. Moody, G. Bronevetsky, K. Mohror, and B. R. De Supinski, "Design, modeling, and evaluation of a scalable multi-level checkpointing system," in *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2010, pp. 1–11.