Measuring the Impact of HPC Training

Julian Miller
Chair for High Performance Computing
RWTH Aachen University
Aachen, Germany
miller@itc.rwth-aachen.de

Manuel Arenaz
Appentra Solutions, University of A Coruña
CITIC, Campus de Elviña
A Coruña, Spain
manuel.arenaz@appentra.com

***Abstract*—The demand for computational power is rising rapidly in science and engineering and is now expanding into areas from the social sciences and humanities. This is met with the use of large-scale, and increasingly complex, High-Performance Computing (HPC) systems. Programming such heterogeneous and quickly evolving systems efficiently has always been challenging, particularly for novice HPC programmers, but is continually increasing in complexity. HPC service providers are therefore growing their training offerings as part of their overall service provision. However, there is no standard for understanding the impact and effectiveness of this training. Traditional training and education metrics, such as examination, are typically not applicable. Additionally, the key criteria for service providers are not only knowledge gained but the impact that the knowledge has on HPC usage. To capture and evaluate the effectiveness of training, a novel methodology with two key components is proposed: progress productivity and training productivity. The methodology includes a process for deriving an activity-specific weighted selection of the metrics that comprise the productivity measures. The proposed productivity criteria could then be applied to improve the training of HPC application programmers taking the participant's pre-knowledge and focus areas into account.**

***Index Terms*—HPC education, training productivity, productivity metrics**

I. Introduction

The use of High-Performance Computing (HPC) is currently experiencing a period of rapid growth as interest in HPC-enabled technologies such as Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL), Big Data and Data Analytics grows worldwide. However, this rapid growth raises the pressure on HPC service providers, who need to ensure that their services are effectively utilized, including maximizing the overall machine usage and minimizing wasted CPU hours due to inaccurate and/or poorly performing software. In addition to the increased pressure on resources, the rapid expansion to new users means that HPC service providers need to address the knowledge gap due to the lack of expertise in writing HPC-enabled software. As a result, today's HPC training is considered a key part of HPC service provision.

It is becoming increasingly important to understand the return on investment (ROI) in HPC training for HPC service providers and their financiers to manage available resources effectively. In practice, measuring the ROI for HPC training is complex and difficult for several reasons. In professional training, business has typically emphasized measuring training effectiveness based on the resulting increased revenue or sales generation. In contrast, HPC training programs have traditionally put the emphasis on software performance, using assessments for benchmarking the relative performance across groups, years and countries.

More recently, the HPC community started to put attention on software portability as hardware rapidly changes and users move software between machines. Today, the great complexity of HPC software, coupled with a user base that is expanding more rapidly than expertise can be shared, is leading to a shift in focus from mainly performance and portability to including productivity, quality and sustainability [1]–[3] as well. This puts enormous pressure on the HPC training community which is evidenced by the increasing focus on HPC training at the major HPC conferences such as the SC conference series, and the inclusion of mandatory training programs in large HPC programs such as the Exascale Computing Project [4].

This paper addresses the challenge of quantifying the impact of HPC training by measuring the productivity of the learners. Productivity is a metric that relates the knowledge gain of the learner and the cost of providing this knowledge, where cost might be financial or effort-based. The main advantage of this approach is that it allows for the comparison of different training methods and/or the comparison across multiple groups of learners to understand knowledge improvements and the impact of different training approaches on different groups.

The main contribution of this paper is the proposal of a new methodology to measure productivity in HPC training, more specifically:

- The definition of a set of measurable productivity metrics adapted to the uniqueness of HPC training, and thus covering learning objectives related to science/engineering, software development and computer hardware.
- The presentation of two new metrics to quantify the productivity of HPC training activities in terms of the progress of the learners and their knowledge gain. The proposed metrics can be setup in different ways in order to capture the specific training goals of each activity (e.g., put more emphasis on performance or more emphasis on programmability).
- The application of the new methodology to measure productivity in real HPC training activities. In particular, the analysis of the series of hackathons CESGAHACK,

co-organized by the authors and the Supercomputing Center of Galicia (CESGA).

The rest of the paper is organized as follows: Section II describes the uniqueness of HPC training and its impact on productivity quantification. Section III proposes a set of productivity metrics that are relevant for HPC training and education. Section IV introduces the novel methodology and Section V presents a case study of its application to training events. Section VI provides a discussion on the merits and limitations of the methodology. Finally, Section VII presents conclusions and proposes future work.

II. Uniqueness of HPC Training Productivity

There has been extensive research on measuring scientific/research productivity [5]–[8] and software development productivity [9]–[11]. However, to the best knowledge of the authors, there is not at this time, a well-defined set of productivity metrics that applies to the uniqueness of HPC training and education programs.

According to the Organisation for Economic Co-operation and Development (OECD), 'productivity is commonly defined as a ratio of a volume measure of output to a volume measure of input use' [12]. Thus, the choice of an effective productivity measure typically depends on the purpose of the measure and on the available data. In the context of HPC training, effective output metrics include:

1) Improved software run-times (e.g., speedup, efficiency, improved strong and weak scaling);
2) Improved scientific output (e.g., papers, citations);
3) Increased problem size (e.g., maximum possible data-size);
4) Increased software functionality (e.g., addition of new algorithms/functionality that enables new/more science);
5) Increased uptake of parallel programming methods (e.g., the usage of advanced functionality from key parallel programming standards such as MPI, OpenMP etc., use of parallel I/O etc.);
6) Increased uptake of HPC services (e.g., new user communities, increase in demand for CPU hours);
7) Long-term impact especially since training participants are often slow to utilize the methods they learned in training in real-world applications;
8) Evaluation of the impact and efficiency of current training methods, including if those methods are suitable for the new user communities entering the HPC landscape;
9) Increased accessibility of training, as it targets learners with different skills and geographically distributed around the world in order to utilize unique HPC facilities remotely.

These output metrics can be related to the input volume which is typically the cost of providing the training. The advantage of such productivity metrics is that it enables the comparison of different types of training activities across multiple groups, years and countries.

However, the data collection of HPC training can be challenging. Assigning quantitative metrics for training and education has traditionally been limited to examinations of students, particularly in formal school and university education, profits/sales or similar financial outputs in business, and gross domestic profit, or similar, for countries or regional training programs. These solutions are not particularly applicable in the HPC sector where:

- trainees are not subject to formal examination;
- courses are typically run for individuals from different organizations;
- it is difficult to quantify a financial return for improvements in software quality that is not used for sales.

In the following, a key set of productivity metrics is proposed based on their applicability to real HPC training events, their strength in quantifying the training impact and their usability especially concerning the challenges associated with data collection.

III. Productivity Metrics in HPC Training

To gain a useful measure of training productivity, many of the general programming productivity metrics used in software engineering can be reused and adapted to the specifics of HPC software. Based on a thorough analysis of common productivity metrics [5]–[12] (c.f. Section II), the following main components of HPC training effectiveness are proposed:

1) The cost of providing the training (Section III-A);
2) The progress of training participants on specific activities during a course (Section III-B);
3) The knowledge improvement as the difference between pre-training and post-training knowledge (Section III-C);
4) The successful completion of tasks such as completing a set of required software improvements (Section III-D);
5) The long-term training impact (Section III-E).

The following sections outline a proposed methodology for measuring training cost. How to quantify and use these metrics is summarized in Table I, and the final methodology is derived in Section IV.

A. Cost

1) Identifying Cost: To quantify the productivity, the impact metrics above (components 2-5) need to be related to a cost. For the training organizer, this is the time involved in providing training or the total cost of running the event. For trainees, the cost is their individual cost of participating in the event, as either time (quantified in elapsed time or the cost of employment during those hours) and/or the financial cost of attending the event including travel, accommodation, fees etc.

2) Measuring Cost: While financial costs are typically easily determined, time-related metrics are more difficult to capture. For very time-confined training events, the event's schedule can be used as a time estimate. However, modern training typically embraces the trainee's individuality and responsibility in learning [13] which leads to large varieties in the trainees' efforts for an event. To measure these individual efforts, the use of developer diaries as, e.g., described in the Personal Software Process (PSP) [14] is recommended. Such journaling methods have been successfully applied to the

HPC domain including DARPA's HPCS program [15], [16] and in studies of Zhang and Hochstein [17] and Hochstein et al. [18]. Modern tools such as the EffortLog [19] or git hooks [20] can minimize the overhead for the data collection while maximizing the accuracy and comparability of the data.

B. Training Progress

1) Identifying Training Progress: Monitoring the progress of trainees during an event is essential to ensure that participants are receiving effective training and allow for corrective measures such as re-distributing mentoring efforts, or providing additional technical information/lecturing. In practice, this is often done, but the strategies used to identify those in need of targeted assistance may not be consistently applied. As HPC training programs become larger and more complex, it is useful to put in place a reproducible method for monitoring trainee progress. This provides the additional benefit of comparison between training events, provided the cohort size is sufficient to minimize limitations introduced by a variable initial knowledge level of participants.

2) Measuring Training Progress: To capture the training progress, well-defined milestones based on the goals of the training event can be used. To ensure effective use of this metric, the milestones must be clearly defined, understandable, and verifiable and appropriate to the goals of the training event. For example, 'complete working parallel version of code' can be obtained in many different ways. Depending on the goals of the training event, additional information might be added such as the tool used for parallelization, the specific code region that should be parallelized or a minimum speed-up that is expected.

One common approach is to first define the last milestone that ideally all participants should achieve at the end of the training event. An example could be that the trainee has completed a tuned parallel version that obtains a minimum speed-up. From there, intermediate steps can be split into additional milestones that are prerequisites to completing the final milestone. Two sets of milestones are provided in Appendix C as reference examples.

C. Knowledge Improvements

1) Identifying Knowledge Improvements: Improvements in knowledge of an individual are considered a key metric since there is a direct relationship between knowledge and improved output. Knowledge also provides a sustainable and long-lasting asset for a trainee. This work focuses on quantifiable changes in knowledge, i.e., propositional knowledge obtained through presentations, supplemental materials and best practice guides, and procedural knowledge obtained through the application of the concepts to real-world problems. The knowledge is comprised of a set of learning objectives provided by the trainer based on the focus of HPC training:

- $K_{science}$: The obtained scientific knowledge;
- K_{prog}: The obtained programming knowledge;
- K_{perf}: The obtained performance-oriented knowledge.

2) Measuring Knowledge Improvements: The knowledge of trainees can be quantified by benchmarking the knowledge immediately before and after the training. Previous work by the authors has used a participant knowledge survey [13], [21] that allows for high throughputs of self-reporting of knowledge concepts, rather than a specific test which removes the negative concerns with examinations [19]. Alternative approaches are 'mini-tests' such as that used by the International High Performance Computing Summer School [22], which provides an anonymous short test to assess participants knowledge of key concepts. Both approaches are valid, and the authors advocate using the most appropriate knowledge benchmarking method for the event.

However, for comparison across multiple training activities, it is important that the same knowledge benchmark is used. Once knowledge benchmarks as a percentage of complete/correct questions have been identified, the knowledge improvement, K_i can be quantified: $K_i = 1 - \frac{K_{\text{i-after}}}{K_{\text{i-before}}}$, with $K_{\text{i-before}}$ and $K_{\text{i-after}}$ being benchmarks taken before and after a training course. The overall knowledge improvement K is given by $K = \sum_{i=1}^{3}(\alpha_{K_i} \cdot K_i)$, with α_{K_i} being the weighting coefficients for the individual knowledge improvements. Additional information on knowledge benchmarking methods is provided in Appendix D.

D. Software Improvements

1) Identifying Software Improvements: Identifying metrics for improvements to HPC software beyond performance results has not yet been standardized and there are conflicting views on what constitutes improved software. This work focuses on metrics with high impact on the quality of HPC software. Furthermore, for comparability it is assumed that the coding exercises are identical, or at minimum, broadly similar. Thus, software complexity is not required for the proposed methodology even though it influences the cost/effort of modifying code as discussed above. The following set of key metrics to measure *Software Improvement*, S is proposed:

1) S_{func}: Percentage of completed functional requirements;
2) S_{ext}: Percentage of completed required calls to external libraries/external library interactions required (e.g., the need to replace particular functionality with a library routine, or the introduction of a particular parallelization library, such as MPI);
3) S_{log}: Size of the software in number of logical statements, I/O calls etc;
4) S_{eff}: Parallel efficiency of the software on a fixed number of processes: $E(N,P) = \frac{S(N,P)}{P} = \frac{T(N,1)}{P \times T(N,P)}$;
5) S_{procs}: Maximum number of processes that the software scales to while still achieving improved run-time;
6) S_{data}: Maximum data-set size achievable since many codes using HPC are limited not just by traditional time-based performance, but also by problem size;
7) S_{par}: Parallel fraction (as a function of run-time) provides the degree of parallelization of a code and can act

as the base for performance models such as Amdahl's law.

2) Measuring Software Improvements: It is important that the metrics described above are easily understood by those that are responsible for gathering (the evaluator) and providing (the trainee/trainer/other) data. In particular:

1) Measurement of the completed functional requirements will require clear specification of each functional requirement in accordance to the overall goals of a training event;
2) Measuring f_{ext} should focus on the requirements of the course and the tasks should be of small granularity;

For each of the metrics S_i defined above, a benchmark should be taken before ($S_{\text{i-before}}$) and after ($S_{\text{i-after}}$) a training course. For the metrics that focus on a percentage of completion, the initial benchmark may be zero. Detail on how to specify each of these metrics is given in Table I. The relative individual software improvements for a specific software improvement metric, S_i, can be calculated as: $S_i = 1 - \frac{S_{\text{i-after}}}{S_{\text{i-before}}}$ and the total software improvement, S, as: $S = \sum_{i=1}^{7}(\alpha_{S_i} \cdot S_i)$, with α_{S_i} being the weighting coefficients for the individual software improvements.

E. Long-term Productivity Metrics

1) Identifying Long-term Improvements: The overarching and long-term goal of scientific software is to enable the generation, collection, analysis and interpretation of data to generate new knowledge, positively impact human life, further human discovery and understanding, and produce scientific results. In the HPC community, these goals are sometimes overshadowed by performance-oriented results [23]. Nevertheless, measuring the scientific outcome of software, its impact, and the effort or cost of producing the results is of great importance. In the context of training, it is important to understand how training activities impact the long-term output of HPC. Thus, the following set of key metrics to measure *Long-term Improvement*, L is defined:

- L_{fund}: Relative increase of the future funding compared to the pre-training state;
- L_{cite}: Citation count of the science/software produced during the training;
- L_K: Retention of the obtained knowledge during the event as relative success in comparison the training knowledge directly after the training.

2) Measuring Long-term Improvements: Long-term results are abstract and as such difficult to measure. Moreover, it can be challenging to clearly attribute a specific outcome to a training event. Instead, the authors propose to use publicly available data such as the number of published papers, the associated citations, or the funding that a scientific software product produces. The degree of the impact of a specific training event can be estimated by questionnaires of the directly-involved scientists. For each of the metrics L_i defined above, a benchmark should be taken before ($L_{\text{i-before}}$) the training course and at a pre-defined time in the future ($L_{\text{i-future}}$). The overall long-term improvement L is given by $L = \sum_{i=1}^{3}(\alpha_{L_i} \cdot L_i)$, with α_{L_i} being the weighting coefficients for the individual long-term improvements.

IV. Quantifying HPC training productivity

The productivity of a training event can be generally defined as the ratio of the output of the event and the cost of the event. Using the quantities outlined in Section III above, two different approaches for quantifying training productivity emerge: *Progress Productivity* to measure the progress of reaching a pre-defined set of activity goals, and *Training Productivity* to measure the productivity of the completed training event to enable comparison across training events.

A. Progress Productivity

As discussed in Section III-B, the progress in training can be measured in terms of a pre-defined set of milestones. Thus, *Progress Productivity* (PP) can be formally defined as a ratio between the percentage of milestones completed during the training event and the cost of the event:

$$PP = \frac{\sum_t^T M_{t,comp}}{M \cdot T \cdot Cost}, \tag{1}$$

where T is the total number of trainees, M is the total number of milestones and $M_{t,comp}$ is the total number of milestones completed by the t-th trainee.

In order to track the progress of each trainee during the training event, the *Progress Productivity per trainee (PP_t)* is defined as follows:

$$PP_t = \frac{M_{t,comp}/M}{Cost_t}, \tag{2}$$

with $Cost_t$ being the training cost for the t-th trainee only.

Alternatively, in order to track the progress per milestone, the *Progress Productivity per milestone (PP_m)* is defined in similar manner:

$$PP_m = \frac{T_m/T}{Cost_m}, \tag{3}$$

where T_m is the number of trainees that completed the m-th milestone, and $Cost_m$ is the training cost for m-th milestone only.

B. Training Productivity

Using the standard definition of productivity, i.e., the ratio of output and cost, a *Training Productivity (TP)* metric can be defined where *Output* is the weighted sum of three components: the total human *Knowledge* improvements, K for all the trainees, the total *Software* improvements, and total *Long-term* improvements; and *Cost* is the associated cost of the training. The three components each consists of the weighted sum of their respective sub-metrics. Specifically:

$$TP = \frac{\sum(\alpha_{K_i} \cdot K_i) + \sum(\alpha_{S_i} \cdot S_i) + \sum(\alpha_{L_i} \cdot L_i)}{Cost}, \tag{4}$$

with α_{K_i}, α_{S_i}, and α_{L_i} being the weighting coefficients for the individual knowledge, software, and long-term improvements. The sum of the weighting coefficients equals one (i.e., $\sum \alpha_{K_i} + \sum \alpha_{S_i} + \sum \alpha_{L_i} = 1$).

TABLE I
HPC TRAINING PRODUCTIVITY METRICS.

Metric	Definition	Data Collection
Completed Milestones		
|_ **Milestones completed per trainee,** $M_{t,comp}$	Number of milestones completed by the trainee t.	Define set of milestones and monitor completion of all milestones for individual trainees.
|_ **Trainees that completed a milestone** m**,** T_m	Number of trainees that completed milestone m.	Define milestone m and monitor completion of milestones for all trainees.
Knowledge improvements, K		
|_ **Scientific knowledge,** $K_{science}$	Change in trainee's scientific knowledge.	Knowledge surveys, mini-tests before and after the training.
|_ **Programming knowledge,** K_{prog}	Change in trainee's programming knowledge.	Knowledge surveys, mini-tests before and after the training.
|_ **Performance-oriented knowledge,** K_{perf}	Change in trainee's performance-oriented knowledge.	Knowledge surveys, mini-tests before and after the training.
Software improvements, S		
|_ **Functional requirements,** S_{func}	Percentage completion of pre-determined set of functional requirements for the code.	Pre-determine a list of new functional requirements and record their completion at the end of the event.
|_ **External library interactions,** S_{ext}	Percentage completion of pre-determined set of external library requirements.	Pre-determine a list of new external library requirements.
|_ **Number of logical statements,** S_{log}	Comparison pre/post training of the total number of logical lines of code.	Calculate logical lines of code/logical statements using tools as described in, e.g., [24].
|_ **Parallel efficiency,** S_{eff}	Comparison pre/post training of the parallel efficiency of the code.	Calculate the parallel efficiency on a given problem size and specific number of processes.
|_ **Scaling: max processes,** S_{procs}	Comparison pre/post training of the number of processes achieving the lowest run-time.	Measure run-time with increasing process counts.
|_ **Max. achievable data-set size,** S_{data}	Comparison pre/post training of the max. data-set size that the software can use.	Measure the maximum usable data-set.
|_ **Parallel fraction,** S_{par}	Comparison pre/post training of the parallel fraction of the run-time.	Use the definition of 'serial fraction of the code' as specified in Amdahl's law.
Long-term improvements, L		
|_ **Funding,** L_{fund}	Relative increase of the future funding compared to the pre-training state.	Ratio of the future funding to the pre-training funding volume.
|_ **Citation,** L_{cite}	Citation count of the science/software produced during the training.	Count the citations based on publicly available data.
|_ **Knowledge,** L_K	Retention of the obtained knowledge compared to the post-training state.	Knowledge surveys, mini-tests before the training and at a future time.

Cost should be determined based on a cost metric that is self-consistent and relevant for the purposes of quantifying the effectiveness of training, as described in Section III-A. Crucially, the same cost measurement technique should be used for comparison across multiple events. Finally, given a specific training event, the weighting coefficients of Eq. (4) are set by the trainer in order to control the relevance of different training goals.

V. CASE STUDY

The proposed methodology was applied to the hackathons CESGAHACK3 (2018) and CESGAHACK4 (2019). This hackathon series was originally inspired by the popular Oak Ridge Leadership Computing Facility GPU hackathons [25] but evolved into an architecture-agnostic HPC hackathon with a focus on a methodological and structured way of teaching parallel programming. This shift in focus required intensive evaluation and iterative improvements to the quality of the training. Thus, it acts as a well-suited candidate for the proposed productivity methodology.

A. Methodological Setup

CESGAHACK3 was carried out in September 2018 with four teams and a total of $n = 8$ participants and CESGAHACK4 in March 2019 with four teams and $n = 6$ participants.

1) Cost: This case study focused on the effort in person-hours of all participants instead of financial costs as salary information was not obtained due to privacy reasons. Furthermore, additional costs for compute resources, location, travel etc. were mostly sponsored and, thus, not accounted for in this study. Both events span over five days, i.e., approximately 40 person-hours per trainee. The trainees reported their effort based on their own estimations. CESGAHACK3 was supported by eight mentors and CESGAHACK4 with six mentors with an approximate effort of 100 person-hours per mentor as estimated by mentor surveys. Furthermore, approx.

50 person hours were spent on data collection, analysis and post-hackathon surveys as tracked by the authors.

2) Training Progress: CESGAHACK3 introduced an innovative methodology of parallel programming which is based on parallel patterns. The trainees were guided by a clear structure of milestones which is split into two phases: The first phase contains the analysis and preparation of the code and the second phase the actual implementation and parallelization of the code. CESGAHACK3 used the milestones preparation, working version, and profiled version for the first phase which was extended in CESGAHACK4 with additional milestones for code refactoring and identification of parallel patterns. Towards the end of the analysis phase, a target hardware architecture was selected and a schema to implement and parallelize the application was derived. The implementation of this strategy was contained in phase two which was organized in two milestones: first-parallel version with programming model x and an optimized version with programming model x. This resulted in a typical set of five milestones for CESGAHACK3 and eight milestones for CESGAHACK4. Please see Appendix C for a complete list of the used milestones.

3) Knowledge Improvements: Knowledge improvement was the main goal of the events and was captured with a predefined set of learning goals which were examined via knowledge surveys. To capture the full domain of the learning objectives, Bloom's taxonomy [26] was applied and the questions of each category were distributed according to studies of Nuhfer and Knipp [21]. Furthermore, a set of at least 40 questions was used to obtain statistical meaningful data as detailed by Nuhfer and Knipp [27]. The weight of the knowledge impact was set to $\alpha_K = 0.6$. The full list of survey questions can be found in the supplemental material.

4) Software Improvements: The second major goal of the events was to parallelize the main parts of the application which was captured by S_{par} and weighted with $\alpha_{par} = 0.2$. This is followed by the parallel efficiency which enables the teams to increase their scientific outcomes, enable large simulations and apply for compute resources. Its weighting was set to $\alpha_{eff} = 0.2$. Since some of the applications targeted embedded devices, the parallel efficiency was evaluated with four processes and the maximal scaling metric was not investigated. The least important metrics for these events were the functional requirements, external library interactions, number of logical statements, and the max. achievable data-set size and were thus not weighted in overall training productivity. Nevertheless, data for these metrics was collected which can be found in the supplemental material.

5) Long-term Improvements: Long-term impact data could not be collected for the hackathons due to their recency.

B. Results

Table II shows the collected productivity metrics of both events. The participants completed the main milestones of both events with $\bar{M} = 3.75$ out of five milestones for CESGAHACK3 and $\bar{M} = 5$ out of eight milestones for CESGAHACK4. Moreover, the events showed $27.5\,\%$ and $44.3\,\%$ increase in the results of the knowledge survey. To investigate significance in this increase, Wilcoxon signed rank tests [28] were applied for each team and their pre- and post-knowledge survey results. It showed significant improvements in their obtained knowledge with p-values of 0.0117 for CESGAHack3 and 0.0277 for CESGAHack4.

The high standard deviations observed throughout the productivity metrics reflect the diversity in the teams. Combined with the high success rate in completing the milestones, knowledge increase and software improvements, high effectiveness of the training methodology for the adaptive and diverse needs of HPC training was observed. The innovations in the design of the training methodology showed clear maturing from CESGAHACK3 to 4 through significant increases in the productivity metrics. In summary, this lead to a large increase in the overall progress productivity of $44.2\,\%$ and training productivity of $70.3\,\%$ of CESGAHACK4 over 3.

VI. Discussion

This section discusses the applicability of methodology proposed in Section IV to real training environments. It bases on the authors' experience in a wide variety of training events including the case study of CESGAHACK 3 and 4.

A. Productivity Metrics

The progress productivity metric can be especially helpful in ensuring successful learning outcomes for all participants. By identifying trainees with a lower progress productivity within a particular cohort, corrective measures can be applied. Moreover, the methodology can be used to compare training methods or other factors, provided that other variables are kept the same. This can provide a useful tool for evaluating teaching styles and spur innovations in teaching methods as observed in the success story of the CESGAHACK series.

The size of a software is the most commonly used metric in software engineering [29] where its expressive power lies in the assumed linear dependency between size and effort. However, this dependency and therefore their applicability is widely discussed and criticized [30]–[33]. Similar findings were obtained for real-world HPC applications [34], [35] where the strong focus on performance required extensive efforts for parallelization and optimization of applications. Thus, the use of more advanced metrics such as functional size and technical size or HPC-specific metrics such as number of parallel regions/constructs/directives could be used. Furthermore, the software size could be combined with complexity metrics such as Halstead complexity [36], cyclomatic complexity [37] and Henry's and Kafura's information flow complexity [38] to better capture the effort required to develop HPC software.

While the longer-term impact of training is likely to be a key motivator in adopting this methodology, measuring knowledge retention and embedding after an event is a challenging task. Improvements greatly depend on the individual's other activities and attributing the impact of the activities can be challenging. Furthermore, observing trainees outside the enclosed training environment can be very time-consuming.

TABLE II

OVERVIEW OF THE COLLECTED DATA DURING CESGAHACK3 AND CESGAHACK4. THE METRICS ARE AVERAGED OVER ALL TRAINEES AND THE STANDARD DEVIATION FOR EACH METRIC IS GIVEN. $\bar{M}$ DENOTES THE AVERAGE NUMBER OF COMPLETED MILESTONES OVER ALL TRAINEES AND $\bar{PP}$ THE OVERALL PROGRESS PRODUCTIVITY. THE RESULTS OF THE PRE-KNOWLEDGE SURVEY ARE MEASURED IN $\bar{K}_{pre}$ AND THE RESULTS OF THE POST-KNOWLEDGE SURVEY IN $\bar{K}_{post}$ WHERE A RATING OF 3 MEANS THE ABILITY TO ANSWER THE QUESTION FOR GRADING PURPOSES AND 1 MEANS NOT ANSWERABLE BY THE TRAINEE. THE PARALLEL EFFICIENCY IS GIVEN BY $\bar{S}_{eff}$ FOR FOUR PROCESSES/THREADS AND THE FRACTION OF PARALLELIZED CODE IS DENOTED BY $\bar{S}_{par}$. THE RESULTING TRAINING PRODUCTIVITY IS GIVEN BY TP.

Metric	$\bar{M}$	$\bar{PP}$	$\bar{K}_{pre}$	$\bar{K}_{post}$	$\bar{S}_{eff}$	$\bar{S}_{par}$	TP
CESGAHACK3	3.75 ± 0.96	$3.37\,\text{e-}4\ \frac{\text{MS}}{\text{person hour}}$	1.71 ± 0.29	2.18 ± 0.09	$49.64 \pm 27.20\,\%$	$59.98 \pm 43.07\,\%$	$4.54\,\text{e-}2\ \frac{\%}{\text{person hour}}$
CESGAHACK4	5.0 ± 1.26	$4.86\,\text{e-}4\ \frac{\text{MS}}{\text{person hour}}$	1.58 ± 0.27	2.28 ± 0.33	$62.88 \pm 24.40\,\%$	$76.22 \pm 33.87\,\%$	$7.73\,\text{e-}2\ \frac{\%}{\text{person hour}}$

Post-training questionnaires and phone calls were carried out to capture longer-term impact. Similar methods are applied, e.g., by Barcelona Supercomputing Center [39]. Future work is required to formalize and standardize these which could lead to an extension on the long-term productivity metrics proposed in this work.

B. Obtaining Statistically Meaningful Data

While larger sample sizes typically result in greater statistical power of empirical test, HPC training typically features small number of participants and a large variety in the highly focused events. Moreover, software and knowledge improvements are both heavily dependent on the individual. For this reason, the authors advocate to thoroughly plan training programs and ensure that the settings for events are kept as similar as possible. Comparable training events can be identified from the event's goals or its derived milestones and weightings of training productivity metrics. Furthermore, comparable events can be found by sharing relevant data with other trainers.

When comparing within any group, the group size should be considered, especially the potential variability introduced by small group sizes. This can be mitigated by using an appropriate statistical test when comparing metrics. Particular care should be taken when comparing an individual within a particular cohort. As such, these measures are typically not a meaningful identifier of an individual's performance. Instead, formal examination or similar should be used to identify personal progress and attainment.

While the progress productivity metrics presented in Section IV are focused on single training events, they can be extended to compare the merits of different training methodologies. However, their applicability is limited since they do not take into account prior knowledge and experience. Thus, direct comparison of individuals or events is only possible where milestones are fixed and identical and the group of trainees is large enough. It is crucial that the results are only compared when the same methodology has been applied as shown in the case study. However, a comparison can still be carried-out across events that had individual priorities, by adjusting the weightings for consistency when considering multiple event comparisons.

In practice, the usage of the proposed methodoloy may add a significant overhead for HPC training organizers as observed in the CESGAHACK series. This is especially prevalent when its usage is first introduced or applied to the first editions of new training events. Thus, means to minimize the data collection overhead and to standardize the collected data are required, e.g., pre-defined data collection sheets. The data collection sheets used during CESGAHACK can be found in the supplemental material and can be adapted and extended for future training events. It contains progress, benchmarking and code analysis information and automatically calculates productivity metrics where applicable. To further reduce the overhead, estimations of metrics can be used in confined environments. This was successfully applied in, e.g, the case study where self-reported development effort was collected instead of the more precise development diaries.

VII. CONCLUSIONS AND FUTURE WORK

This paper presents a novel methodology to quantify productivity in HPC training and education, an area that has not yet been standardized. The methodology takes into account the complexity of the HPC community and the key goals of HPC training programs. The result is a combination of metrics that provide a useful tool for measuring progress during events, identifying needs of individual training participants, and also measuring the overall impact of training activities. The methodology was successfully applied in two real-world training activities with $n = 14$ participants and its applicability and usability were analyzed.

In future, the validation of the methodology needs to be extended to a wider variety of training events such as university courses (undergraduate and postgraduate), HPC training provided to professionals such as formal training, workshops, and tutorials. The authors anticipate that this methodology will evolve with the knowledge and experience gained from these events and that guidelines on the learning goals, expected progress, and productivity metrics and their weighting can be derived. The authors sincerely invite you to participate in this effort by applying the methodology to your own training events and by sharing experiences and collected productivity data.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the collaboration of the Supercomputing Center of Galicia (CESGA) in the organization of the CESGAHACK series of hackathons.

REFERENCES

[1] "Better scientific software," accessed: 2019-08-05. [Online]. Available: https://bssw.io/

[2] University of Edinburgh, "Software Sustainability Institute," accessed: 2019-08-05. [Online]. Available: https://www.software.ac.uk/

[3] "Software Carpentry," accessed: 2019-08-05. [Online]. Available: https://software-carpentry.org/

[4] "Exascale Computing Project Training," accessed: 2019-08-05. [Online]. Available: https://www.exascaleproject.org/training/

[5] B. F. Reskin, "Scientific productivity and the reward structure of science," *American sociological review*, pp. 491–504, 1977.

[6] J. O. Lanjouw and M. Schankerman, "Patent quality and research productivity: Measuring innovation with multiple indicators," *The Economic Journal*, vol. 114, no. 495, pp. 441–465, 2004.

[7] H. Dundar and D. R. Lewis, "Determinants of research productivity in higher education," *Research in higher education*, vol. 39, no. 6, pp. 607–631, 1998.

[8] P. Ramsden, "Describing and explaining research productivity," *Higher Education*, vol. 28, no. 2, pp. 207–226, 1994.

[9] B. W. Boehm, "Improving software productivity," in *Computer*. Citeseer, 1987.

[10] B. Kitchenham and E. Mendes, "Software productivity measurement using multiple size measures," *IEEE Transactions on Software Engineering*, vol. 30, no. 12, pp. 1023–1035, 2004.

[11] W. D. Yu, D. P. Smith, and S. T. Huang, "Software productivity measurements," in *Computer Software and Applications Conference, 1991. COMPSAC'91., Proceedings of the Fifteenth Annual International*. IEEE, 1991, pp. 558–564.

[12] Oecd, "OECD Productivity Manual: A Guide to the Measurement of Industry-Level and Aggregate Productivity Growth," OECD, Paris, Tech. Rep., 2001. [Online]. Available: http://www.csls.ca/ipm/2/schreyer-e.pdf

[13] K. R. Wirth and D. Perkins, "Knowledge Surveys: An Indispensable Course Design and Assessment Tool," *Innovations in the Scholarship of Teaching and Learning*, 2005.

[14] W. S. Humphrey, *A Discipline for Software Engineering*, 1st ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1995.

[15] J. Dongarra, R. Graybill, W. Harrod, R. Lucas, E. Lusk, P. Luszczek, J. Mcmahon, A. Snavely, J. Vetter, K. Yelick, S. Alam, R. Campbell, L. Carrington, T.-Y. Chen, O. Khalili, J. Meredith, and M. Tikir, "DARPA's HPCS Program: History, Models, Tools, Languages," in *Advances in COMPUTERS–High Performance Computing*, ser. Advances in Computers, M. V. Zelkowitz, Ed. Elsevier, 2008, vol. 72, pp. 1–100.

[16] V. R. Basili and M. V. Zelowitz, "The Use of Empirical Studies in the Development of High End Computing Applications," University of Maryland, Tech. Rep., 2009.

[17] M. Zhang and L. Hochstein, "Fitting a workflow model to captured development data," in *3rd International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 2009, pp. 179–190.

[18] L. Hochstein, V. R. Basili, M. V. Zelkowitz, J. K. Hollingsworth, and J. Carver, "Combining Self-reported and Automatic Data to Improve Programming Effort Measurement," in *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE-13)*. New York, NY, USA: ACM, 2005, pp. 356–365.

[19] S. Wienke, J. Miller, M. Schulz, and M. S. Müller, "Development effort estimation in hpc," in *High Performance Computing, Networking, Storage and Analysis, SC16: International Conference for*. IEEE, 2016, pp. 107–118.

[20] S. L. Harrell, J. Kitson, R. Bird, S. J. Pennycook, J. Sewall, D. Jacobsen, D. N. Asanza, A. Hsu, H. C. Carrillo, H. Kim *et al.*, "Effective performance portability," in *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 2018, pp. 24–36.

[21] E. Nuhfer and D. Knipp, "The Knowledge Survey: A Tool for All Reasons," *To Improve the Academy*, vol. 21, pp. 59–78, 2003.

[22] L. DeStefano and L. Rivera, "International Summer School on HPC Challenges in Computational Sciences: 2015 Summer School Evaluation Report," University of Illinois at Urbana-Champaign, Tech. Rep. September, 2015. [Online]. Available: https://www.ideals.illinois.edu/bitstream/handle/2142/88678/I-HPC-SS%20Report%202015_V2_ALL.pdf

[23] V. R. Basili, D. Cruzes, J. C. Carver, L. M. Hochstein, J. K. Hollingsworth, M. V. Zelkowitz, and F. Shull, "Understanding the high-performance-computing community," *IEEE software*, vol. 25, no. 4, pp. p29–36, 2008.

[24] V. Nguyen, S. Deeds-Rubin, T. Tan, and B. Boehm, "A sloc counting standard," in *Cocomo ii forum*, vol. 2007. Citeseer, 2007, pp. 1–16.

[25] S. Chandrasekaran, G. Juckeland, M. Lin, M. Otten, D. Pleiter, J. E. Stone, J. Lucio-Vega, M. Zingale, and F. Foertter, "Best Practices in Running Collaborative GPU Hackathons: Advancing Scientific Applications with a Sustained Impact," *Computing in Science Engineering*, vol. 20, no. 4, pp. 95–106, Jul 2018.

[26] B. S. Bloom, M. Engelhart, E. J. Furst, W. H. Hill, and D. Krathwohl, "1956, taxonomy of educational objectives: The classification of educational goals, handbook i: Cognitive domain," 1956.

[27] E. B. Nuhfer and D. Knipp, "Re: The use of a knowledge survey as an indicator of student learning in an introductory biology course," *CBE Life Sciences Education*, vol. 5, no. 4, pp. 313–314, 2006.

[28] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[29] V. R. B.-G. Caldiera and H. D. Rombach, "Goal question metric paradigm," *Encyclopedia of software engineering*, vol. 1, pp. 528–532, 1994.

[30] I. Christadler, G. Erbacci, and A. D. Simpson, *Facing the Multicore - Challenge II: Aspects of New Paradigms and Technologies in Parallel Computing*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ch. Performance and Productivity of New Programming Languages, pp. 24–35.

[31] F. P. Brooks Jr, *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Pearson Education India, 1995.

[32] S. Faulk, A. Porter, J. Gustafson, W. Tichy, P. Johnson, and L. Votta, "Measuring HPC Productivity," *International Journal of High Performance Computing Applications*, vol. 2004, pp. 459–473, 2004.

[33] J. Kepner, "High performance computing productivity model synthesis," *International Journal of High Performance Computing Applications*, vol. 18, no. 4, pp. 505–516, 2004.

[34] M. Nicolini, J. Miller, S. Wienke, M. Schlottke-Lakemper, M. Meinke, and M. S. Müller, "Software Cost Analysis of GPU-Accelerated Aeroacoustics Simulations in C++ with OpenACC," in *High Performance Computing*, M. Taufer, B. Mohr, and J. M. Kunkel, Eds. Cham: Springer International Publishing, 2016, pp. 524–543.

[35] J. Miller, S. Wienke, M. Schlottke-Lakemper, M. Meinke, and M. S. Müller, "Applicability of the software cost model COCOMO II to HPC projects," *International Journal of Computational Science and Engineering*, vol. 17, no. 3, pp. 283–296, 2018.

[36] M. H. Halstead *et al.*, *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY, 1977.

[37] T. J. McCabe, "A complexity measure," *IEEE Transactions on software Engineering*, no. 4, pp. 308–320, 1976.

[38] S. Henry and D. Kafura, "Software structure metrics based on information flow," *IEEE transactions on Software Engineering*, no. 5, pp. 510–518, 1981.

[39] N. Alexandrov and M.-R. Sancho, "Learning outcomes based evaluation of hpc professional training," *Procedia Computer Science*, vol. 108, pp. 2141–2150, 2017.

[40] "Count Lines of Code," accessed: 2019-08-05. [Online]. Available: https://github.com/AlDanial/cloc

[41] B. S. Bloom, M. D. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl, *Taxonomy of educational objectives: Handbook 1: Cognitive domain*. Longman Publishing Group, 1984.

[42] L. W. Anderson, D. R. Krathwohl, P. W. Airasian, K. A. Cruikshank, R. E. Mayer, P. R. Pintrich, J. Raths, and M. C. Wittrock, "A taxonomy for learning, teaching, and assessing: A revision of blooms taxonomy of educational objectives, abridged edition," *White Plains, NY: Longman*, 2001.

APPENDIX A
ARTEFACT DESCRIPTION APPENDIX

This paper does not contain or rely on, any computational results, so no artefacts are described here.

APPENDIX B
CASE STUDY SUPPLEMENTAL MATERIAL

The raw data of the case study is provided as supplemental material and the authors encourage the analysis and replication of the experiments carried out. The material includes the following productivity metrics:

- Development effort; self-reported by the trainees and mentors;
- Completed milestones per trainee ($M_{t,comp}$) and trainees that completed milestone m (T_m); mentors certified completion of milestones based on checklists;
- Pre- and post-knowledge (K); captured via knowledge surveys and the full set of questions is provided. See the next chapter for further details on how to adapt the knowledge surveys to other training events;
- Functional requirements (S_{func}); captured through observations of the mentors and questionnaires of the trainees;
- External library interactions (S_{ext}); captured through observations of the mentors and questionnaires of the trainees;
- Number of logical statements (S_{log}); captured through differentials between the initial and last version of the software and its analysis with the cloc tool [40];
- Parallel efficiency (S_{eff}); captured via data collection sheets, measurements were carried out with multiple repetitions and average run-times were used;
- Scaling: max processes (S_{procs}); captured via data collection sheets, measurements were carried out with multiple repetitions and average run-times were used;
- Max. achievable data-set size (S_{data}); captured through observations of the mentors and questionnaires of the trainees;
- Parallel fraction (S_{par}); captured via data collection sheets, calculations according to Amdahl's Law.

APPENDIX C
MILESTONES USED IN THE CASE STUDY

CESGAHACK3 used the following milestones:

- MS0: Preparing for the hackathon;
- MS1: Working serial version;
- MS2: Profiled serial version;
- MS3: Working OpenMP parallel version;
- MS4: Tuned OpenMP parallel version;
- MS5: Working OpenACC parallel version;
- MS6: Tuned OpenACC parallel version;
- MS7: Highly optimized OpenMP+OpenACC parallel version.

CESGAHACK4 used the following milestones:

- Phase 1: Analysis and preparation of the code;
  - MS0: Preparing for the parallelization of the code;
  - MS1: Working version;
  - MS2: Profiled version;
  - MS3: Refactoring the code for parallelization;
  - MS4: Identifying parallel patterns;
  - MS5: Selecting the target hardware;
- Phase 2: Implementation and parallelization of the code;
  - OpenMP parallel version;
    - MSOMP1: Working OpenMP parallel version;
    - MSOMP2: Tuned OpenMP parallel version.
  - OpenACC parallel version;
    - MSACC1: Working OpenACC parallel version;
    - MSACC2: Tuned OpenACC parallel version.
  - MPI parallel version;
    - MSMPI1: Working OpenACC parallel version;
    - MSMPI2: Tuned OpenACC parallel version.
  - Hybrid version.

APPENDIX D
KNOWLEDGE BENCHMARKING METHODOLOGIES

To assist with knowledge benchmarking, the authors propose two separate methods they have been implemented. The first is an informal, multi-choice quiz which can be completed online in a short amount of time. An example can be found in the Appendix of the 'International Summer School on HPC Challenges in Computational Sciences: 2015 Summer School Evaluation Report' [22].

The second example is the use of knowledge surveys in the authors' training events. To limit the time required for filling out the surveys (~30 minutes) while keeping statistical meaningfulness, surveys of 40 questions are suggested based on research by Nuhfer and Knipp [27]. Depending of the learning goals of the event, Bloom's taxonomy of the cognitive domain [41] (and its revised version [42]) can be helpful in designing a balanced survey. This methodology is described in [19] in detail and a full set of exemplary questions can be found in supplemental material. Moreover, a detailed example on how to introduce a knowledge survey to its participants can be found in [21].

Exemplary Recall Question (~27.5% of the questions)
Describe the term parallel efficiency.

Exemplary Comprehension Question (~27.5% of the questions)
A multiprocessor consists of 100 processors. If 10% of the code is sequential and 90% is parallelizable, what will be the maximum speedup when running this program on such a multiprocessor?

Exemplary Application Question (~15% of the questions)
Which of the following code snippets can be executed in parallel with minimum parallelization overhead?

Loop A:

```
for (h = 0; h < Adim; h++)
  hist[h] = 0;
```

Loop B:

```
for (h = 1; h < Adim; h++)
  hist[h] = hist[h] + hist[h-1];
```

Loop C:

```
for (h = 0; h < fDim; h++)
  hist[f[[h]] = hist[f[h]] + 1;
```

Exemplary Analytical Question (~15% of the questions) Consider the following pseudocode which operates on two arrays of real numbers d[N] and y[N+1]. Here, N is a large integer constant and array indices start at 1. If this was parallelized using OpenMP, explain where synchronization is required and why.

```
for (i = 1; i < N; i++) {
  y[i] = myfunction(i);
}
y[N+1] = 0.0;
sum = 0.0;
for (i = 1; i&lt;N; ++i) {
  d[i] = y[i+1] - y[i];
  sum = sum + d[i]*d[i];
}
printf("sum = ", sum);
```

Exemplary Synthesis Question (~7.5% of the questions) Write a program that computes the matrix subtraction where $C = A + B$ and $A, B, C \in \mathbb{R}^{N \times N}$ that is parallelized for a shared-memory NUMA system. You can use OpenMP or similar pseudo code. State explicitly which variables are private or shared and also the loop schedules. Aim for best performance without modifying the given algorithm. Explain briefly what you do for gaining good performance and why.

Exemplary Evaluation Question (~7.5% of the questions) Colors are defined by RGB (red-green-blue) values that are integers between 0 and 255. One `Color` implementation is given by the struct array below. Assume that we have 100 colors from which we modify the red component. The code below is running on a GPU with many threads. Comment on the correctness and performance of the code. If the code delivers unwanted results or its performance is improvable, give reasons for the behavior and how to fix or improve it.

```
struct {
  int rgb[3];
} *Color; // size 100
/**/
#pragma acc data copy(Color[0:100])
{
#pragma acc parallel loop
for (int i = 0; i < 100; i++)
  Color[i].rgb[0] *= 0.5;
}
```