Self-paced Learning in HPC Lab Courses

Christian Terboven
Chair for High-Performance Computing
RWTH Aachen University, Germany
terboven@itc.rwth-aachen.de

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

Sandra Wienke
Chair for High-Performance Computing
RWTH Aachen University, Germany
wienke@itc.rwth-aachen.de

Matthias S. Müller
Chair for High-Performance Computing
RWTH Aachen University, Germany
mueller@itc.rwth-aachen.de

ABSTRACT

In a software lab, groups of students develop parallel code using modern tools, document the results and present their solutions. The learning objectives include the foundations of High-Performance Computing (HPC), such as the understanding of modern architectures, the development of parallel programming skills, and course-specific topics, like accelerator programming or cluster set-up.

In order to execute the labs successfully with limited personnel resources and still provide students with access to world-class HPC architectures, we developed a set of concepts to motivate students and to track their progress. This includes the learning status survey and the developer diary, which are presented in this work. We also report on our experiences with using innovative teaching concepts to incentivize students to optimize their codes, such as using competition among the groups. Our concepts enable us to track the effectiveness of our labs and to steer them for increasing sizes of diverse students.

We conclude that software labs are effective in adding practical experiences to HPC education. Our approach to hand out open tasks and to leave creative freedom in implementing the solutions enables the students to self-pace their learning process and to vary their investment of effort during the semester. Our effort and progress tracking ensures the achieving of the extensive learning objectives and enables our research on HPC programming productivity.

KEYWORDS

HPC education, software lab, parallel programming, programming effort, training productivity

1 MOTIVATION

With the intent to make the dedication of our chair–the High-Performance Computing (HPC)–popular among students, to attract the best and highly-motivated students, and in general to engage with students early on and to foster their skills, we have created a series of HPC software labs. These support a diverse group of students in self-paced learning and are meant to accompany theoretical education in the field of HPC with a practical component.

As a requirement to execute the labs successfully, we have to be able to stem the course with very limited personnel resources. Nevertheless, we want to give students the opportunity to work on world-class HPC architectures. To support the students to reach the given learning objectives, we developed a set of concepts to motivate students and to track their progress. This also enabled our research on HPC development productivity.

This paper presents our *learning status survey* and the *developer diary* to track the student's progress in achieving the learning objectives, and our approach to *enable the comparison* of different HPC cluster architectures or parallel programming models. We also report on our experiences with using innovative teaching concepts such as using a *competition* among students to motivate them to optimize their codes for performance and show the opinions that students have towards these concepts.

Thus, the paper is structured as follows: In Section 2, we describe the structure and content of three different kinds of software labs that we have conducted at the HPC chair of RWTH Aachen University. Section 3 summarizes the learning objectives of our labs–classified into general and course-specific goals. To motivate our students and increase the success rate, we have created various stimuli that are presented in Section 4. Section 5 covers the methodology on how we track development effort and progress. In Sections 6 and 7, we evaluate the software labs in terms of obtained knowledge, training productivity and programming models, as well as students' feedback based teaching evaluations. Finally, we conclude in Section 8.

2 HPC SOFTWARE LABS

Within the Computer Science curriculum at RWTH Aachen University, a software lab is a mandatory part of Bachelor studies and expected to be completed in the 4th or 5th semester. It teaches practical skills. As part of the actual work within a software lab, students have to come up with a precise outline of the task at hand, develop code using modern tools, document the results and prepare a final presentation. Special emphasis is put on the experience to work in a group, including the challenging tasks to self-organize the development project throughout the semester. At RWTH, students have the opportunity to select from 10 to 15 different software labs that are offered each semester. These span a wide range of topics,

© 2020 Journal of Computational Science Education
https://doi.org/10.22369/issn.2153-4136/11/1/10

covering the whole computer science domain at RWTH. While the students are not expected to have prior knowledge in the particular topic of a given software lab, they might have obtained knowledge based on their individual selection of optional courses. This diversity requires a flexible and self-paced approach towards HPC education. The following provides our concepts and experiences of the three different labs regularly carried out at our institute.

The first lab is called *Parallel Programming Models for Applications in the Area of High-Performance Computation* (PModels lab). Each group has to parallelize a serial code given as skeleton of a Conjugate Gradient (CG) solver for sparse matrices with three different parallel programming models on different hardware architectures. In the past, these were OpenMP for CPUs, and CUDA and OpenACC for GPUs. The different student groups start each with another programming model so that (amongst others) the chance to copy performance tuning steps from others is minimized. Before starting the course, the students get a basic introduction to these programming models. Then, they will work independently usually starting with a performance analysis (using corresponding tools) and investigating the hotspot of the application. It is key to efficiently parallelize this hotspot, i.e. the sparse matrix-vector product, to achieve good performance. For grading, basic parallel versions with reasonable performance are sufficient. However, as part of the competition, students are strongly encouraged to continue their performance tuning and apply tools and performance engineering of their codes until the end of the semester. This also includes thinking about (and implementing) performance models, new data structures for sparse matrices, minimization of data transfers, or other solutions to improve convergence of the solver. The final results are evaluated with respect to overall solving time for a given matrix.

The second lab is titled *Parallel Programming for Many-Core architectures with OpenMP* (OpenMP lab) that is offered every summer term. Each group has to solve three tasks and provide implementations with the parallel programming model OpenMP. In all tasks, the students are encouraged to use performance and correctness tools to analyze their intermediate and final solution. Task one is the parallelization of a *sparse-matrix-vector-multiplication.* The identification of the loop that has to be parallelized and the implementation of a first-parallel version is rather simple, but the main challenge is to achieve a load balance among the participating threads since the non-zero entries are irregularly distributed among the matrix. Thus, the students have to identify that the number of nonzero elements has to be the same for each thread, not the number of rows in the matrix. Further improvements of the execution performance have to be achieved by enabling SIMD vectorization and aligned allocation of the data. Task two is a *merge-sort* code, for which tasking has to be employed due to the recursive nature of the algorithm. Again, a first-parallel version can be implemented with little effort, but the challenge is to find a cut-off strategy to limit the overhead induced by the recursion. Both tasks have to be implemented for a many-core architecture with two different kinds of memory, namely regular DDR memory and high-bandwidth memory, and the students have to decide where the data elements have to be placed. Furthermore, both kinds of memory exhibit NUMA characteristics that have to be respected appropriately. In task three, a *k-means* code has to be ported to an accelerator (GPU) architecture. In this task, the effort to implement an initial-parallel version is higher, because the offloading requires more programming work and is more error-prone than the constructs that have to be used in the first two tasks. The key performance optimization challenge is to minimize the data transfer between host and GPU. Furthermore, the students have to understand which type of kernel and data volumes are profitable when offloaded to a GPU and deliver a performance improvement over the parallel execution on the host. In summary, the solution of the three tasks teaches students the key skills to program for contemporary HPC architectures and to perform performance analysis and optimization on these architectures.

The third lab is titled *HPC Cluster Challenge* (Cluster lab) and is offered every winter term. Each group receives a box of hardware with the task to construct a cluster from its contents. Afterwards, they have to parallelize and optimizing a Jacobian solver code aiming towards fully utilizing their respective cluster. The four sets of hardware for the four different groups are very different, so that the groups' experiences can be compared at the end of the software lab. All groups receive the same network equipment, which is pre-configured to allow connections to the university network and, thus, enabling remote work. We provide an overview lecture about what constitutes a cluster and provide general hints (not step-by-step instructions) on how to configure the network, a shared filesystem, etc. The first group receives a set of Intel-based desktop PCs with NVIDIA GPU cards. In consequence, the code has to exploit message-passing, multi-core and many-core parallelism. Although this is the most standard hardware, we found the groups are challenged to choose from the different configuration options and descriptions found online. The second group receives a set of NVIDIA Jetson boards, and again message-passing, multi-core, and many-core parallelism has to be exploited and there are different possible software configurations. The third group receives a set of Huawei Kirin boards. These are limited to message-passing and multi-core parallelism because the programming options for the so-called AI engine is not well-documented. The fourth group receives a set of 64-bit Banana Pi board, and again these are limited to message-passing and multi-core parallelism, this time because of limited capabilities of the graphics processor. In summary, the hardware ranges from low-power ARM-based SoCs to desktops equipped with GPUs. In all cases, the hybrid parallelization has to employ message-passing between the nodes, threading with each node and partly offloading to exploit accelerator units. The results are compared with respect to effort and price-performance.

Table 1 provides an overview of the three labs discussed in this work with the number of participating students, the group size and their average semester. It is noteworthy that we have improved our material and methodology (cf. Section 5) over time by taking feedback and new insights into account. To this end, we (still) used manual developer diaries in summer 2015 instead of the more advanced electronic approach introduced in Section 5. Furthermore, we focused on oral attestations and final grades to evaluate the students' gained knowledge in summer 2015. In later semesters, we added knowledge surveys (cf. Section 5) to extend and improve this concept.

Table 1: Overview of the three software labs with the number of participating students, the group size and their average semester.

Term	Lab	# Students	# Groups	Semester
Summer 2015	PModels	14	7	4.5
Summer 2016	PModels	12	6	4.1
Summer 2017	OpenMP	18	6	5.4
Winter 2017	Cluster	15	4	5.5
Summer 2018	OpenMP	17	5	4.4
Winter 2018	Cluster	15	4	6.4
Summer 2019	OpenMP	16	4	4.8

3 LEARNING OBJECTIVES

The learning objectives of our labs can be classified into a general and a course-specific set of goals. The generic foundation of our HPC education lies in a thorough understanding of modern multi- and many-core processor architectures including CPUs (with various instruction set architectures) and accelerators. This foundation is paired with theoretical knowledge including parallelism, scalability and performance modeling to form analytical and assessment skills for a wide range of hardware architectures. We build upon this foundation by teaching software engineering skills and best practices geared towards developing parallel software. These include generic skills such as software requirements and design, documentation, version control and development diaries (cf. Section 5) as well as more specific tasks such as the correctness of parallel programs, debugging and performance analysis. Furthermore, we foster self-organization and collaboration through team work. Presentations of the results teach the students the visualization and description of software solutions, performance results and algorithms.

After completing the PModels lab, the students have a general understanding of shared-memory and GPU programming using different parallel programming models, i.e., OpenMP, CUDA and OpenACC. They know about methodologies to leverage the available parallelism and can clearly identify differences between low-level and high-level programming approaches. Moreover, students have an idea how to treat sparsity and how to apply optimization techniques to typical numerical solver such as the CG.

The goals of the OpenMP lab are similar: The students have a broad understanding of shared-memory and accelerator programming with OpenMP and its various techniques to map parallelism to hardware such as the concepts fork-join, tasking and accelerator offloading. Furthermore, the students are able to make profound decisions on how to parallelize scientific tasks for specific hardware architectures and optimize its hardware utilization.

The HPC Cluster Challenge lab focuses on a broad understanding of HPC clusters including the structure of clusters, networks, shared storage and the cluster management. A key goal is the understanding and analysis of power demands and energy efficiency of clusters. On the software side, the objective is to port and parallelize scientific tasks to a target cluster and the design and implementation of the task with suitable programming models (mainly OpenMP, MPI, CUDA-C/C++, OpenCL).

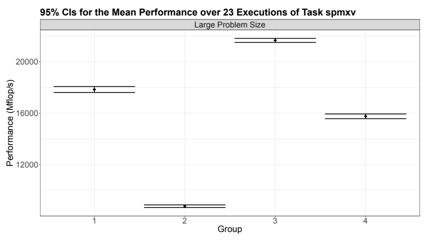
4 STIMULI

We use various stimuli in our labs to increase the success rate of the learning objectives while fostering creative solutions. We found that some of the tasks are especially challenging to derive from the objectives without providing a step-by-step guide which would hinder self-pacing, creativity and planning aspects. Thus, we define generic tasks based on the expected outcome such as 'implementing, parallelizing and optimizing a specific algorithm for a target architecture' and combine these with stimuli to increase the achievement of the learning objectives while fostering creative solutions.

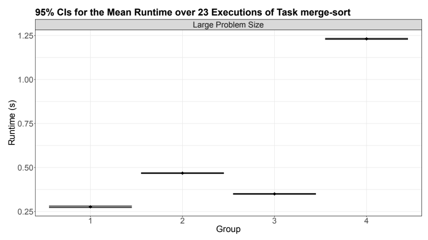
The main stimulus we use is competition through group work in which prices (HPC-related books) are awarded to the team which achieves the fastest solution for all three tasks of the OpenMP lab, or all three parallel CG code versions in the PModels lab. As was outlined above, each lab partitions the students into three to five groups. Each group has to solve the same tasks in the same order, but the competition successfully ensures that solutions are not freely exchanged between the groups. In all three software labs, the solution of a task results in a parallel program for which in the execution on the target architecture the time can be measured. Each tasks offers a certain degree of freedom in the solution and the execution parameters so that the performance results differ between the groups. The winning group is determined via the formula-1 system: for each task, the fastest solution is awarded 25 points, the second fastest is 18 points, then 15 and 12 points. This ensures a fair and thrilling competition even in the presence of one group delivering a much better or worse solution than the rest in one particular task. In no instance of the software labs we have observed a single group clearly dominating the competition.

Figure 1 represents a typical result of the competition. It shows the result of all three tasks from the competition in summer term 2019. For each of the four groups, the resulting runtime of 23 repetitions is plotted. All four groups have delivered a working solution, applied the correct techniques to achieve reproducible performance with little variation, and achieved results in the same performance class. The third group won the competition since they achieved the highest throughput for task 1, the second-lowest runtime for task 2 and the lowest runtime for task 3.

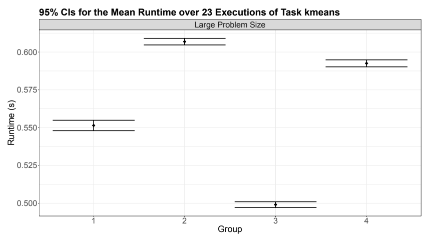
Furthermore, we award creative solutions through presentation time during the oral attestations and the final presentations with all students. In order to expose the work of the HPC Cluster Challenge software lab to the IT Center, which also operates national HPC infrastructure, we selected a public and frequently visited space in between two building parts for the setup the clusters. In consequence, the clusters are on public displays and interested visitors can see the systems in operation, include power measurements, and the students at work. While this is certainly not comparable to the public display of the Cluster Challenge activities at ISC or SC conferences, the students reported that they like that atmosphere after a certain time of getting used to it.



(a) Task 1



(b) Task 2



(c) Task 3

Figure 1: Competition results of the OpenMP lab in summer 2019.

5 EFFORT AND PROGRESS TRACKING

One of our main learning objectives is the ability to document and present software projects in an especially performance-oriented context. These goals are coupled with our own motivation of investigating the progress of the students and the effectiveness of our labs. Therefore, we established a thorough data collection methodology which is used throughout the labs. While we do not grade the quality of said data, we regularly collect the data and provide students with feedback on how to improve their documentation skills.

We use the collected data to steer the lab for increasing sizes of divers groups of students as well as to develop and assess innovations in teaching techniques. The main output of our teaching curses is knowledge obtained as measured by the degree of achieved learning objectives while the input (cost) is the effort invested by the students in completing the course. Thus, the overall training productivity TP is quantified by $\frac{\text{Degree of achieved learning objectives}}{\text{Training effort}}$. Furthermore, the data supports research into programming productivity on a increasingly heterogeneous set of computing hardware and programming models. The main output is hereby the achieved performance of the implemented parallel software: programming productivity $PP = \frac{\text{Achieved performance}}{\text{Training effort}}$. For better comparison among a cohort, we typically normalize the obtained data.

The following provides our methodology for collecting the three main productivity metrics knowledge, performance and training effort. The learning objectives include propositional knowledge obtained through preparational and supplemental materials such as programming courses, lectures[1], literature, best practice guides, etc. and procedural knowledge obtained through completion of design, implementation, programming etc. tasks. We use knowledge surveys (KS) [4, 7] where students rate their confidence in solving tasks on a three-point scale to quantify the changes in knowledge. This allows for a much higher assessment throughput than in traditional tests and KS provide a comprehensive self-assessment to the students. We assess our set of learning objectives with 40–50 tasks (approx. 30 minutes to answer) which has shown to be preferable regarding the overhead for the students and their participation rates. The KS are combined with oral attestation to capture additional learning objectives such as team-oriented skills, software-engineering methods and decision processes. The resulting confidence ratings of the KS are combined with average grades from all oral attestations (typically 2–3 over the course of the lab).

The performance of the software is typically captured by runtime or throughput on a specific system. Hereby, the system consists of pre-defined types of HPC cluster nodes for the OpenMP and PModels lab or the self-built cluster for the Cluster lab. The students may use all available resources of the system towards their performance goals. To increase the reliability of the collected performance data, we use multiple benchmark repetitions coupled with mean and standard deviations. To simplify the data collection, we provided run scripts, makefile targets and data collection spreadsheets.

The cost of the training is the effort in person-hours which consists mainly of the time attending the lab and the development effort for completing the task(s). While the attendance time is clearly defined, lots of the development is carried out outside the lab's presence hours. Thus, we use development diaries to record the quantity and type of effort carried out by the students. To maximize the accuracy and comparability of the data while minimizing the intrusion of the data collection, we develope the electronic development diary EffortLog[2] [6]. It uses strict input forms, precise questionnaires and fixed intervals (60-minutes has proven well) to achieve highly accurate data. Large efforts were recently put into increasing the usability of the tool which include a simplified layout, auto-completion, notifications to further improve data quality and summaries of the current project including performance results. Moreover, experience has shown that an operating system agnostic implementation with a minimal set of dependencies is key for the usability of the tool. The recorded effort data is related to the achieved performance by reminding the students to collect performance data and append it to the development activities. The resulting data contains the development of the achieved performance over the development effort as shown in the evaluation in Chapter 6. The main challenge of the tool is that it is currently not well-integrated into the typical development tool chain of the students. Harrell et al. [1] target this challenge by integrating data collection into git hooks. While this method promises high adaption through commonly used tooling, the accuracy of this method needs to be investigated further especially for student setups where we have observed that version control is often used sparsely and with low commit frequency. We intend to investigate the integration of

[1]See the list of courses and lectures of the HPC group of the RWTH Aachen University: https://www.i12.rwth-aachen.de

[2]The sources are publicly available on Github: https://github.com/RWTH-HPC/effort-log

Table 2: Overview of the collected knowledge data during the OpenMP lab in summer 2018 and 2019. A KS rating of 3 means the ability to answer the question for grading purposes and 1 means not answerable by the trainee.

	2018		2019	
	pre-KS	post-KS	pre-KS	post-KS
Mean	1.97 ± 0.22	2.40 ± 0.30	1.32 ± 0.27	2.35 ± 0.33
Median	1.95	2.40	1.20	2.40

such tools or the combination of both kind of tools into a future version of our labs.

6 EVALUATION OF THE LABS

This section provides an overview of the results obtained with the data collected during the labs including investigations into the obtained knowledge and training productivity, as well as, differences in parallel programming models. The obtained knowledge is mainly captured by knowledge surveys carried out before and after the lab as described in Section 5. The participation is voluntarily and does not contribute to the grading of the labs. This protects the privacy of the students but often leads to incomplete data sets. Therefore, only the meaningful data is discussed which was obtained from the OpenMP labs in summer 2018 and 2019. We collected 19 valid surveys (6 people finished both pre- and post-KS) in summer 2018 and 12 valid surveys (4 people finished both pre- and post-KS) in summer 2019. The observed mean, median and standard deviations are provided in Table 2. The labs show to be very effective in achieving the learning objectives by a large increase in the confidence rating of the post-KS over the pre-KS for both labs. To investigate statistical significance of this data, we applied one-sided Wilcoxon signed rank tests to the collected data of people completing both pre- and post-KS. It shows statistically significant increases in the knowledge of the students with p-values of 0.00014 and 0.01046 for the OpenMP labs 2018 and 2019 respectively.

The collected productivity data of the students opens up wide areas of research into HPC programming productivity such as the estimation of software costs of HPC projects. While most of this research is ongoing and will require more data, some early results can be found in [2, 3, 6]. Figure 2 provides an example for the analyses carried out on the productivity data collected during the OpenMP lab in summer 2019. It shows the normalized performance (in relation to the best-effort solution) over the normalized development effort (in relation to each group's total effort). The anonymized data is provided by four groups of students for three tasks. It shows two distinct developments of the performance over the effort: A linear increase and a step-wise increase in performance. Linear increases are typical for groups working on small, incremental changes or only parts of the code which can lead to missed tuning opportunities. The collected data aids in identifying these groups early-on and supporting them in identifying the main performance limiters. Step-wise increases are often observed in groups who radically change parts of their code, algorithm or their launch configuration. A few of these changes (1–2 for these small projects) often lead to large performance increases while most of the changes do not provide performance benefits. Our research focuses on modeling these functions, the understanding of the triggers for a sharp change and estimation methods.

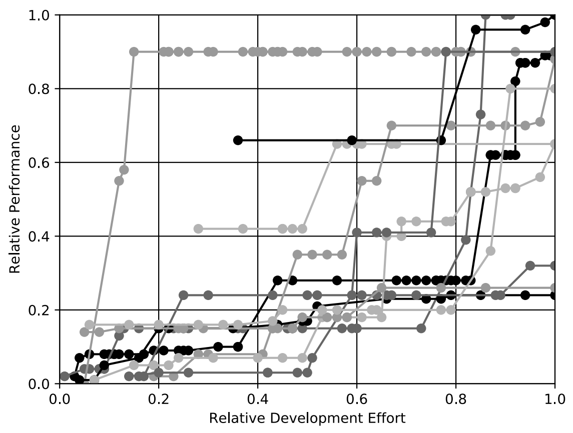


Figure 2: Achieved performance over the development effort of $n = 12$ solutions during the OpenMP lab.

	OpenMP	OpenACC	CUDA
OpenMP		0.6812	0.0737
OpenACC	0.0737		0.0210
CUDA	0.0161	0.2598	

effort

runtime

Figure 3: p-values of one-sided Wilcoxon rank sum test with respect to students' development effort (upper triangle) and runtime (lower triangle) [5]. Significant differences (on a 5 % significance level) are marked in grey. Results are based on (valid) data, i.e. 11 student teams, from the PModels labs.

To this end, it is important to also investigate different impact factors on development time and runtime. For example, results from the PModels labs show that the choice of the parallel programming model may affect the productivity [5]. Figure 3 illustrates the significant differences between OpenMP, OpenACC and CUDA in terms of development effort (upper triangle) and runtime (lower triangle). It expresses corresponding p-values in the way that the row item is significantly lower than the column item (on a 5 % significance level) where results are based on the one-sided signed Wilcoxon rank sum method. We find that the effort to use OpenACC is significantly lower than the one needed for CUDA programming. In contrast, the runtime achieved when using CUDA is significantly lower than with OpenACC. For the comparison of other programming models, we cannot draw any conclusions with this data. Nevertheless, this kind of evaluation of software lab data supports the hypothesis that parallel programming models affect productivity results so that this factor should be kept fixed for future investigations of other impact factors.

Table 3: Teaching evaluation results averaged over all our software labs (except winter 2017 and summer 2019). BG = necessary background knowledge available, SOL = able to solve exercises alone or contribute to solutions in a group, MOT = exercises motivate student to solve them. Results given as percentage of students answering the question with 1 = strongly agree, ..., 5 = strongly disagree.

scale	BG [%]	SOL [%]	MOT [%]
n	69	71	69
1	37.7	67.6	47.8
2	34.8	23.9	33.3
3	15.9	7.0	14.5
4	5.8	1.4	2.9
5	5.8	0	1.4
average	2.09	1.45	1.82

7 STUDENT FEEDBACK (BASED ON TEACHING EVALUATIONS)

To investigate whether our teaching concepts appeal to the students, we take students' feedback into account. For that, we use the results of the official teaching evaluations that RWTH Aachen University implements for each course at the end of a semester. These teaching evaluations consist of three main parts: statistical information, questions on a Likert scale (mostly ranging from 1 = strongly agree to 5 = strongly disagree), and free-form fields where students have the chance to mention anything that they liked and disliked about the course. Since the questions in all lab teaching evaluations are (mostly) the same, we can easily average or aggregate the results over all our software labs. The only exceptions are the evaluations in winter 2017 and summer 2019 where some of the questions did not appear. Thus, these questions have a reduced population (cf. the corresponding values of *n*). During our software labs, the students get a dedicated time slot to fill out the corresponding questionnaires. Nevertheless, participation is voluntary so that the number of responses *n* may differ for each question. For this evaluation, we focus on questions that may reflect the effectiveness of our teaching concepts (instead of presenting the complete results).

In our seven labs, we supervised 107 students from which 103 took part in the teaching evaluation. These students are mostly, i.e. 87 %, in their second or third year of the Bachelor program in computer science. Furthermore, 72.5 % of the voting students, i.e., students who rated the question with one and two, denoted that they have the necessary background knowledge to complete this course where the average scoring is at 2.09 (cf. *BG* in Table 3). Thus, we assume that the students' feedback stems mainly from their experiences gained throughout our software labs—instead to their (missing) pre-knowledge.

First, we evaluate the overall concept of our software labs. Here, the students have rated the PModels lab on average with 1.8 and 1.5 in summer 2015 and summer 2016, respectively. The OpenMP lab was scored on average with 1.4 in summer 2017, with 1.2 in summer 2018, and 1.6 in summer 2019. The Cluster lab has been rated with 1.7 in winter 2017 and 1.9 in winter 2018. For comparison, we (only) have the average scores across all courses within the computer science department at RWTH Aachen University (for 2016 and 2017) available. In summer 2016, this overall average was at 1.8, in summer 2017 at 2.0, and, in winter 2017 at 1.9. To this end, our corresponding software labs are better rated than the average. Nevertheless, this interpretations should be taken with care since the computer science average also includes (compulsory) lectures that usually score worse than labs or seminars.

Table 4: Teaching evaluation results aggregated aver all our software labs. Answers to the questions what students particularly liked or disliked, respectively, about the lab (in free-text form). Top three answers are presented if they have more than one vote.

like		dislike	
topic	#	topic	#
independent working/flexibility	11	unclear goal	14
concept of tasks	7	little instructions	11
competition	6		

Getting more specific, we look at the feedback for our concept of tasks. 81.2 % of the voting students state that the exercises motivate them to solve the tasks. The corresponding question is rated with an average score of 1.82—as given by *MOT* in Table 3. Moreover, seven students particularly praise this concept in the free-form comments of the questionnaire (cf. Table 4). The competition as part of the exercises concept is explicitly mentioned positively six times. Contrarily, the fourteen comments that the goals of the software labs are not clearly given (cf. Table 4) is thought-provoking. Thus, we are continuously improving our corresponding material and goal statements without limiting the student's creative freedom in solving the tasks.

Finally, we investigate the concept of independent working and self-paced learning. From the free-text comments in Table 4, we see that it is received with mixed feelings. Eleven students particularly mention that they like the independent working and the flexibility that comes with self-paced learning. On the other hand, eleven students state that they do not like working without detailed instructions and developing (performance tuning) steps themselves. Assuming different learning types and the fact that students are not very familiar with this way of working through other courses, we still find these results a balanced relationship. This is especially true considering that 91.5 % of the voting students rated that they were (still) able to solve the exercises alone or contribute to solutions in a group (cf. *SOL* in Table 3) with an extremely good average score of 1.45. Correspondingly, 94.4 % of the voting students find the degree of difficulty appropriate (cf. Table 6).

Overall, students work mostly between one and five hours for the software lab outside of the classroom sessions. Taking the respective median from the different time intervals in Table 5, this comes to an average of 4.1 hours. As comparison, this is more than double

Table 5: Teaching evaluation results averaged over all our software labs. Time for preparation and follow up work given as percentage of students answering the question ($n = 99$).

time	[%]
< 1 hr	0
1 to 3 hrs	26.3
3 to 5 hrs	49.5
5 to 7 hrs	18.2
7 to 9 hrs	3.0
> 9 hrs	3.0

Table 6: Teaching evaluation results averaged over all our software labs (except winter 2017 and summer 2019). Degree of difficulty reported, given as percentage of students answering the question ($n = 71$).

degree	[%]
appropriate	94.4
too difficult	5.6
too easy	0

the time that students spend for exercises attached to regular HPC lectures taking place in the same semesters as the labs. Since grading does not require the best performing code version, these results indicate that students make use of the provided flexibility and are motivated to spend extra time for scoring well in the competition.

Given the students' feedback from the official teaching evaluations, we conclude to continue with our concept of self-paced learning while improving our material, e.g., with respect to elaborating on the goals of the software labs.

8 CONCLUSION

Software labs are effective in adding practical experiences to the HPC education and in enabling access to and hands-on experiences on world-class HPC systems. Our approach to hand out open tasks and to leave creative freedom in implementing the solutions enables the students to self-pace their learning process and to vary their investment of effort during the semester. These conclusions are also supported by students' feedback given through teaching evaluations. Our effort and progress tracking ensures the achieving of the extensive learning objectives and enables our research on HPC programming productivity.

REFERENCES

[1] Stephen Lien Harrell, Joy Kitson, Robert Bird, Simon John Pennycook, Jason Sewall, Douglas Jacobsen, David Neill Asanza, Abaigail Hsu, Hector Carrillo Carrillo, Hessoo Kim, et al. 2018. Effective performance portability. In *2018 IEEE/ACM International Workshop on Performance, Portability and Productivity in HPC (P3HPC)*. IEEE, 24–36.

[2] Julian Miller, Sandra Wienke, Michael Schlottke-Lakemper, Matthias Meinke, and Matthias S Müller. 2018. Applicability of the software cost model COCOMO II to HPC projects. *International Journal of Computational Science and Engineering* 17, 3 (2018), 283–296.

[3] Marco Nicolini, Julian Miller, Sandra Wienke, Michael Schlottke-Lakemper, Matthias Meinke, and Matthias S Müller. 2016. Software cost analysis of GPU-accelerated aeroacoustics simulations in C++ with OpenACC. In *International Conference on High Performance Computing*. Springer, 524–543.

[4] Edward Nuhfer and Delores Knipp. 2003. 4: The knowledge survey: A tool for all reasons. *To improve the academy* 21, 1 (2003), 59–78.

[5] Sandra Wienke. 2017. *Productivity and Software Development Effort Estimation in High-Performance Computing; 1. Edition.* Dissertation. RWTH Aachen University, Aachen. https://doi.org/10.18154/RWTH-2017-10649 Apprimus Verlag, Published on the publication server of RWTH Aachen University 2018.

[6] Sandra Wienke, Julian Miller, Martin Schulz, and Matthias S Müller. 2016. Development effort estimation in HPC. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 107–118.

[7] Karl R Wirth and Dexter Perkins. 2005. Knowledge surveys: An indispensable course design and assessment tool. *Innovations in the Scholarship of Teaching and Learning* (2005), 1–12.

A ARTIFACT DESCRIPTION: SELF-PACED LEARNING IN HPC LAB COURSES

A.1 Abstract

This paper does not contain or rely on any computational results.