Integrating interactive performance analysis in Jupyter Notebooks for parallel programming education

1st Lena Oden
Computer Engineering
University of Hagen
Hagen, Germany
lena.oden@fernuni-hagen.de

2nd Klaus Nölp
Computer Engineering
University of Hagen
Hagen, Germany
klaus.noelp@fernuni-hagen.de

3rd Philipp Brauner
Human-Computer Interaction Center
RWTH Aachen University
Aachen, Germany
brauner@comm.rwth-aachen.de

***Abstract*—Understanding the performance behavior of parallel applications is important in many ways, but doing so is not easy. Most open source analysis tools are written for the command line. We are building on these proven tools to provide an interactive performance analysis experience within Jupyter Notebooks when developing parallel code with MPI, OpenMP, or both. Our solution makes it possible to measure the execution time, perform profiling and tracing, and visualize the results within the notebooks. For ease of use, it provides both a graphical JupyterLab extension and a C++ API. The JupyterLab extension shows a dialog where the user can select the type of analysis and its parameters. Internally, this tool uses Score-P, Scalasca, and Cube to generate profiling and tracing data. This tight integration gives students easy access to profiling tools and helps them better understand concepts such as benchmarking, scalability and performance bottlenecks. In addition to the technical development, the article presents hands-on exercises from our well-established parallel programming course. We conclude with a qualitative and quantitative evaluation with 19 students, which shows a positive effect of the tools on the students' perceived competence.**

***Index Terms*—jupyter, parallel programming, performance analysis, interactive programming, high performance computing**

I. Introduction

When teaching parallel programming, it is important to cover not only coding and parallel algorithms, but also aspects such as parallel performance, scalability, and performance analysis. This requires integrating benchmarking and scalability testing into the curriculum and exercises, allowing students to gain hands-on experience with these topics.

Learning parallel programming, particularly with OpenMP and MPI, frequently demands a significant additional effort, especially for undergraduate students who may not be familiar with all the concepts involved. Often, our undergrad students lack familiarity with the preferred programming languages, such as C or C++, and may not have knowledge of `bash` and `ssh`, which can be discouraging in their pursuit of understanding parallel algorithms. Additionally, mastering profiling techniques introduces further complexity, as it involves learning new tools, that are sometimes poorly documented.

To overcome this, we use Jupyter Notebooks to teach parallel programming, focusing on OpenMP and MPI, because this environment has many benefits:

- **Access via web browser**: Our system can be accessed through a user-friendly web interface, provided by JupyterHub and JupyterLab.
- **Combination of text and code**: Jupyter Notebooks support rich text, which allows creating an easy-to-read combination of theoretical background, instructions, and code in a single file. Additionally, students can write and run code snippets directly in the notebook, reinforcing their programming skills.
- **Reproducibility**: Jupyter Notebooks support the concept of reproducibility. Students can share their work with peers and instructors, allowing others to replicate their experiments and verify results.
- **Feedback and assessment**: Instructors can provide feedback directly within Jupyter Notebooks, making it easy to assess students' work and provide guidance for improvement.

However, while this approach is excellent for teaching parallel programming and parallel algorithms, the performance of interpreted C/C++ code is lower compared to compiled code, making it unsuitable for an in-depth performance analysis.

Additionally, when it comes to performance analysis, these tools are typically not integrated into the Jupyter environment, which necessitates students to resort to external tools. Visual analysis, for example, may require the installation of additional software, which, as we have learned, can be an obstacle for students to solve a task independently.

To address these limitations, our goal was to create an integrated learning environment for parallel programming, based on JupyterLab. This environment includes

- Jupyter Notebooks for parallel programming: Specialized notebooks for teaching MPI and OpenMP in C/C++.
- Benchmarking support: Integration of benchmarking tools for strong and weak scaling tests, allowing students to assess their program's performance comprehensively.

These tests are automated and avoid the interpretation overhead.

- Performance analysis: Profiling and tracing tools are seamlessly integrated to help students gain insights into their code's performance.

Also, we developed a set of Jupyter Notebooks to support these objectives. For students who prefer an alternative approach, we offer access to C files and Makefiles to solve the programming exercises.

Our platform is installed on a university server, ensuring accessibility for all students. Alternatively, a local version can be easily installed using Docker, Podman, or Conda, making it available to a wider audience and allowing students to work on their own machine while still using the tools we provide.

In this work, we want to introduce our environment and the usage in our parallel programming course together with an evaluation from the students.

II. Related work

JupyterLabs and Jupyter Notebooks are widely used to give users web-based access to high performance computing systems, often as an alternative to SSH. Several large computing centres implement a Jupyter instance on their systems and report overall good experiences [14, 10].

In [9], an open-access course is presented that teaches the fundamentals of high performance computing (HPC) using Jupyter Notebooks. However, unlike our work, it is based on Python and does not include performance analysis.

In [4] Jupyter Notebooks are used as interactive textbooks to teach parallel programming. Most of the course is taught in C++. They use the Jupyter `%%writefile` "magic" command to write the contents of a Jupyter cell to a file, combined with the ability to execute shell commands with the `!` prefix.

A similar approach is used, for example, in [17] to run MPI programs from Jupyter Notebooks. Our notebooks use C++ directly with the Cling C++ interpreter. In addition, none of this work addresses automatic performance analysis, tracing or profiling in teaching.

Previous research in [20] explores the use of a Python kernel integrated with Score-P bindings to enable tracing and profiling of Python code within Jupyter Notebooks. However, this approach requires the use of external visualization tools such as Vampir [12] and CubeGUI [7] to analyse the results.

III. Jupyter Notebooks for parallel programming in C/C++

Jupyter Notebooks have gained immense popularity, particularly in the scientific community, among data scientists and Python programmers. Although their use in parallel programming, especially with OpenMP and MPI, is not yet widespread, they offer significant advantages for educational purposes and interactive work.

The Xeus framework serves as the foundation for multiple Jupyter kernels, including those for languages such as C++, Python, and Julia [13]. Of particular interest is the `xeus-cling` Jupyter kernel, which enables the interactive use and interpretation of C++ code [11]. This kernel relies on Cling, an interactive C++ interpreter, built upon the robust LLVM and Clang libraries.

```
#include <omp.h>
#pragma cling load("libomp.so")
#pragma omp parallel
printf("I am thread %d \n",
  omp_get_thread_num());
```

Fig. 1: Integrating OpenMP in a Jupyter Notebook

```
#include <mpi.h>
#pragma cling load("libmpi.so")
```

```
%%executable test.bin -- -lmpi
MPI_Init(NULL, NULL);
int rank;
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
printf("I am %d\n", rank);
MPI_Finalize();
```

```
!mpirun -n 2 ./test.bin
```

Fig. 2: Integrating MPI in a Jupyter Notebook using three code cells

To enable OpenMP within the kernel, the *kernel.json* file must be configured to include the `-fopenmp` flag, thereby supporting all OpenMP 4.5 features [3]. A simple OpenMP example is provided in Figure 1.

While OpenMP leverages threads for parallelism, MPI relies on processes. Employing multiple parallel kernels in Jupyter may introduce complexities during startup, potentially resulting in deadlocks and scalability testing challenges[1].

To circumvent these issues, our approach leverages the `xeus-cling` kernel's cell magic `%%executable`. This feature compiles code from input cells into an executable binary. The `%%executable` cell serves as the main entry point for the program. The generated binary can be executed within a Jupyter Notebook using `mpirun`, for example by two processes as shown in Figure 2. However, functions and non-parallel runs can be executed using the interpreter, providing an interactive programming environment.

In our course, we use notebooks with both OpenMP and MPI to teach students parallel programming. This is not ideal for measuring parallel performance, as interpreted code (even if it is written in C/C++) has lower performance than compiled code. Also, profiling tools, which allow a deeper analysis of the code, are not easy to use. Therefore, in the next section, we describe how we overcome these limitations.

[1]For instance, see https://ipyparallel.readthedocs.io/en/latest/reference/mpi.html

IV. Performance analysis and benchmarking in Jupyter Notebooks

To integrate benchmarking and performance analysis into the Jupyter environment, we establish the following requirements:

1) Code for both benchmarks and profiling should be compiled to mitigate the bottleneck caused by interpretation (For most profilers, re-compilation is necessary).
2) Users can adjust the *problem size* and the *number of parallel execution units* (threads or processes) for benchmarking.
3) Users can mark multiple timing regions.
4) Visualization should be seamlessly integrated into the Jupyter environment.
5) All immediate results should be accessible to the user.
6) The tool should be accessible via a graphical JupyterLab extension and via a C++ API.

To address these needs, we convert Jupyter Notebooks into compilable files. Consequently, our performance tool initially employs `nbconvert` to transform the notebook into a C++ file. However, the generated file lacks a `main` function, making it unsuitable for direct compilation.

Instead, our tool processes the C++ file by searching for the user-defined marker comments `// start_main` and `// end_main`. These markers are ignored during cell execution but aid the profiling tool in identifying essential code regions. It is allowed to extend the main function across multiple cells. Any functions and variables declared before `// start_main` are included in the C++ file, while content after `// end_main` is ignored.

The resulting file can be compiled for profiling or tracing, but additional steps are required for benchmarking.

A. Adding markers for benchmarking

For accurate timing, it is crucial to add timing functions into the code. Students are responsible for determining which regions to measure by adding comments to the code, that begin with `// start_` or `// end_`. Figure 3 provides an example.

```
// start_main
int n = 1024;
// start_timing
MatrixMatrixMult(n);
// end_timing
// end_main
```

Fig. 3: Special line comments enclose the `main` function and timing regions

The performance tool scans the file to identify these timing markers, includes functions to record elapsed time, and writes the timing results to a file for subsequent analysis. A single pair of timing markers can be reused multiple times within the same notebook; during analysis, such instances will be numbered. It is important to note that markers with identical names should not be nested within each other. This can be avoided by adopting custom naming conventions, such as `// start_total` or `// start_communication`.

B. Benchmarking different problem sizes and parallel execution units

One of the primary challenges is the definition of various problem sizes, as required during scalability testing. There is no one-size-fits-all solution due to the diverse ways problems can be defined. As a result, we adopt a flexible yet straightforward approach that can be customized for different use cases.

For automatic benchmarking purposes, each notebook should include the line `int n = ... ;`. When parsing the code to insert the timing functions, the performance tool identifies this line and replaces it with the user-defined problem size.

This approach ensures that the cell can execute correctly even without the presence of the performance tool. As the definition of the problem size is always necessary, we cannot rely on marker comments for this purpose.

The benchmark tool generates a new C++ file for each problem definition, which is subsequently compiled and executed. To vary the number of execution units, we leverage the `OMP_NUM_THREADS` environment variable in OpenMP and employ `mpirun` to initiate multiple parallel processes. To ensure accurate measurements, users should avoid manually specifying the number of threads using an OpenMP function or directive.

C. Profiling and Tracing

In addition to performance analysis, our tools encompass capabilities for profiling and tracing. We utilize Score-P/Scalasca [18] for these purposes internally. The process involves recompiling the generated source code file with specific commands essential for code instrumentation. After executing the application and collecting profiling data, the generated visualization allows for straightforward analysis.

While our tool does not exploit the full range of functionalities these instruments offer, our objective is to provide students with an introductory experience of profiling that is comprehensive yet not overwhelming.

D. C++ API

The C++ API enables students to evaluate the performance of Jupyter Notebook code. Students start by creating an instance of the performance class. For example, a student can instantiate it with `performance p("mm.ipynb")`, where the input is either a notebook or a C++ source file. Remarkably, this instantiation can occur within the `mm.ipynb` notebook itself, facilitating the concurrent analysis and development of code.

The API's execution model is automatically determined (`mpi`, `openmp`, `hybrid`, `serial`). It chooses `mpi` for `MPI_Init`, `openmp` for `pragma omp`, and `hybrid` if both are present. Alternatively, students can set the model manually like `performance p("mm.ipynb", "mpi")`.

In section V-B we give more examples of using the API while demonstrating the visualization.

E. Engaging with the JupyterLab extension GUI

The extension makes the API accessible from the JupyterLab environment. It introduces a new menu option in JupyterLab (Figure 4) and clicking it will open a new panel (Figure 5).

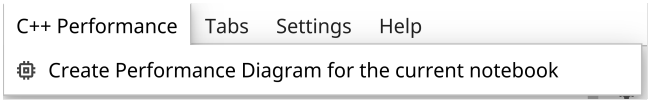

Fig. 4: Menu option to start the JuypterLab extension

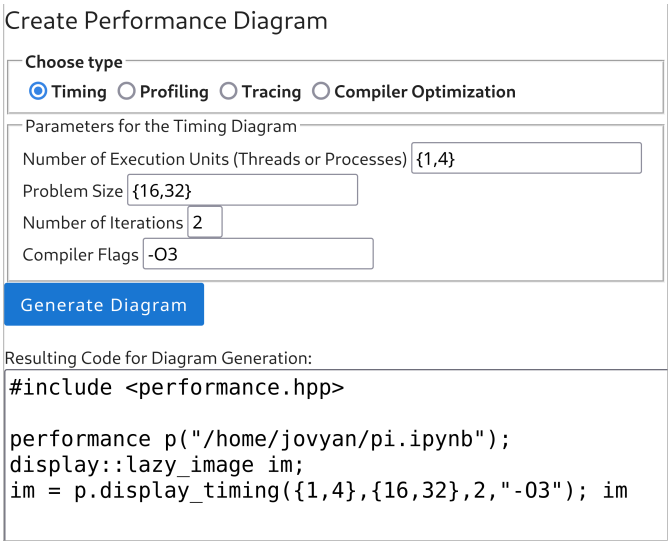

Fig. 5: Panel to start the timing analysis from the JuypterLab extension

GUI interactions trigger updates to an editable input field. This allows students to view or change the API calls that will be executed. When generating plots, a new C++ kernel runs the entire notebook, including the code in the editable input field. The results are then displayed in a new Jupyter panel.

V. Using the tool for teaching

In the following, we will introduce our parallel programming course and give some examples of how the tool is used for teaching.

A. Parallel programming at a distance learning university

Our university is a distance learning university. In contrast to local universities, the students are typically older and pursue their studies alongside full-time jobs, which contributes to a more diverse student body.

One of the primary philosophies of our university is to enable students to study asynchronously, allowing them to balance their education with their employment. This approach includes occasional, non-mandatory video meetings with instructors, but the majority of the learning materials should facilitate self-study and should be self-explanatory. Therefore, videos and example notebooks are created to explain the usage of the tool.

Additionally, the students are located in different parts of Germany and even around the world, which poses challenges for collaborative work. However, it should still be encouraged as it is an important skill for future computer scientists. All materials provided to students are designed with these considerations in mind.

The course "Parallel Programming" is designed for Bachelor students in computer science, but is also open to students from various Master's programs [15]. This is a common approach in Germany, since not all Master students have a computer science background. The course is divided into a total of six units:

1) Introduction to parallel programming
2) Approaches to parallelization
3) Parallel programming with OpenMP
4) Parallel programming with MPI
5) Parallel algorithms
6) Big data (Hadoop, Spark)

Each course unit provides accompanying texts explaining the fundamentals and voluntary exercises.

Based on experience, many students tend to work on the exercises later, resulting in non-submission. Sample solutions are provided for reference. The course is being concluded with an oral exam, which can be scheduled throughout the year. The grade of this exam is also the final grade of the module.

While we provide Jupyter Notebooks for nearly all parts, the primary focus of this work is on the notebooks and tools provided for course units 3, 4, and 5. Not all students have knowledge of C++ or Jupyter. Therefore, we provide additional materials such as video tutorials and programming exercises.

B. Visualization of performance results

The examples presented below demonstrate exemplary output for the different types of analyses. Similar visualizations can be created using the provided example notebooks. Please refer to the link at the end of the article to learn more (VII).

All outputs, including source code files, measurement results, and profiling and tracing data, are stored in a dedicated folder within the students' user space, allowing them direct access to the data for further analysis.

1) Automatic benchmarking (Timing): Exercise 3 includes the calculation of π via numerical integration, teaching students synchronization mechanisms like reduction operations, critical regions, and atomic operations. A part of the Jupyter Notebook for this exercise is illustrated in Figure 6.

```
int n = 8192;
double pi;
// start_critical
pi = CalcPi_critical(n);
// end_critical
// start_reduction
pi = CalcPi_reduction(n);
// end_reduction
// start_atomic
pi = CalcPi_atomic(n);
// end_atomic
```

Fig. 6: Code example from the third exercise (compute π)

In this example, we use three methods to calculate π: `reduce`, `atomic`, and `critical`. Consequently, we use three timing markers for measurement. Students can run benchmarks either by using the provided JupyterLab extension or by calling the `display_timing` function within the same notebook, as shown in Figure 7

```
timingIm = p.display_timing({1,2,4,8},
  {(1<<20),(1<<21),(1<<23),(1<<24)}, 4);
timingIm
```

Fig. 7: Code example to run the benchmark (compute π)

In this example, we run the application with 1, 2, 4 and 8 threads for and the problem size `n` equal to 2^{20}, 2^{21}, 2^{23} and 2^{24}. Each measurement is repeated four times. The tool automatically runs benchmarks for all parameter combinations. It generates and displays plots that show how the application scales, as illustrated in Figure 8.

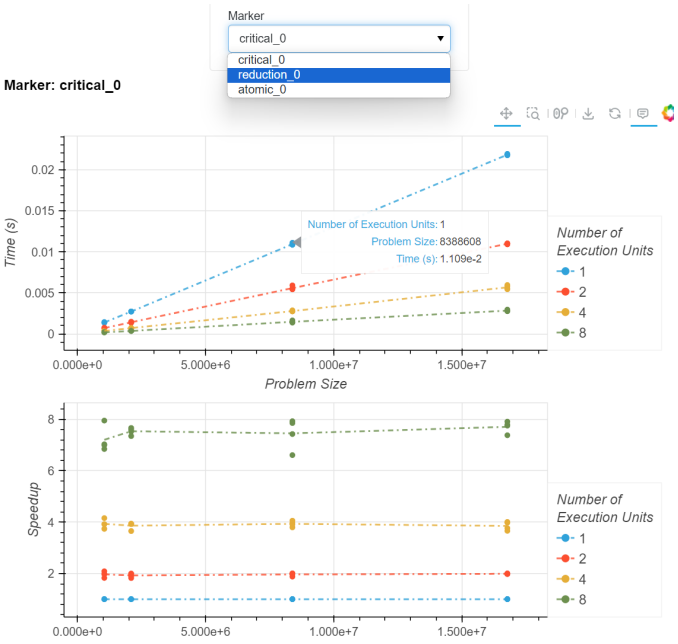

Fig. 8: Automatic benchmarking output of the π calculation with OpenMP

Both plots are interactive; students can zoom in and out and view absolute numerical values by using the mouse.

2) Profiling: In one task in exercise 5, students are asked to implement a sparse matrix vector multiplication using the CSR format. The OpenMP version is used to compare different scheduling algorithms. Various raw matrices are provided for evaluation. The helper function to select a matrix is shown in Figure 9. It turns the problem size `n` into a matrix filename, allowing automatic benchmarking.

In Figure 10 we zoom in on the profiling data. Students can understand the connection between OpenMP's implicit barriers and for-loops by looking at function names, line numbers and an optional call graph that can be generated.

Note that threads 1-3 wait significantly longer than thread 0, indicating an uneven load distribution when dynamic scheduling is used.

```
std::string choose_file(int n) {
  if (n == 0) return "c-22.mtx";
  else if (n == 1) return "parabolic_fem.mtx";
  ...
  return ""; }
```

Fig. 9: Helper function to convert a problem size into a matrix filename

3) Tracing: In exercise 4, students have to parallelize prime number calculations with MPI, implementing three versions. The first two use a controller-worker pattern, with the controller process dispatching numbers (1) or intervals (2) to workers. The third uses static task distribution among MPI processes. Figure 11 displays the tracing output produced by the profiling tool for the initial version. The Figure also shows a part of the Jupyter Notebook where the API function is called.

We zoom in to highlight a relevant section and select only the relevant functions. Students can observe that a lot of time is spent in communication rather than in computation. Notably, worker process 1 experiences a prolonged blockage in the `MPI_Recv` function. This is identified as the bottleneck of the application, guiding students towards the optimized second version.

VI. Evaluation

In order to understand if the students in our parallel programming course use the tool, how they use it, and how the tool could be improved, we conducted an evaluation of the tool. In the following, we present the design of our study and the analysis of the results.

A. Method

We developed a survey to understand our user's reported use of the tool [6], because no logging of the tool's usage and linking to exam results is possible due to privacy reasons (GDPR).

The questionnaire consisted of three main parts: First, we asked about the participants' demographics (age in years and

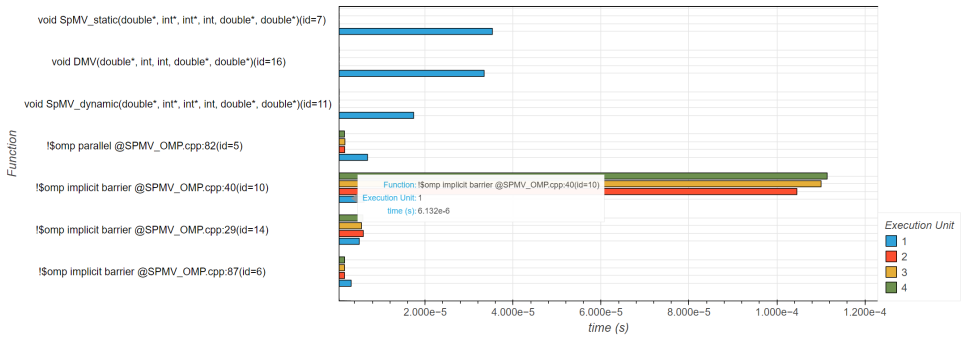

Fig. 10: Profiling output of the sparse matrix vector multiplication with OpenMP (dynamic scheduling)

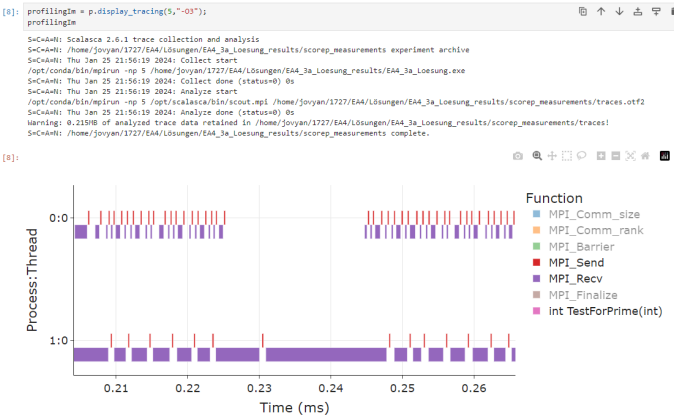

Fig. 11: Tracing output of the prime number calculation with MPI

gender) and study program. The second block included the expertise of the participants and the evaluation of the course. We asked for knowledge before and after the course (pre-post design). Specifically, we asked about general programming expertise and expertise in C++ and – being the learning objectives of the course – about their competency in parallel programming, benchmarking, and profiling on 5-point scales ranging from *very low* to *very high*. Note that we used a single survey *after* the course, hence the reported previous knowledge may be subject to cognitive biases. We further asked if they used the provided tools for benchmarking and profiling on a scale ranging from *not at all* to *whenever possible*, and if they experimented with the provided tools beyond the assignments. Third, we asked two open-ended questions on both positive and negative aspects of the course and the provided tools.

We used LimeSurvey (www.limesurvey.org) to create the questionnaire, and we invited the students to participate using the course management system. Two reminders were sent to increase participation. The survey was anonymous and voluntary and as an incentive, we gave away three vouchers of €10 each among all completed questionnaires.

Sample: 29 participants started the questionnaire. After removing cases with only a few responses (i.e., only reopened to question items on the first of five pages) we had 19 completed responses that we analyze in the following. Four of the 19 participants were women (21.1%), 14 were men (73.7%), and 1 participant did not disclose the gender (5.3%). The age ranged from 23–66 years (median 35 years) 10 were enrolled in a Bachelor's degree (Computer Science and Business Informatics) and 9 were enrolled in a Master's program (Applied Computer Science and Business informatics).

B. Results

First, we present the results from the quantitative evaluation of the course.

From the 19 participants, 5 (26.3%) reported no additional experimentation beyond the assignments, while 13 (68.4%) reported some and an additional participant (5.3%) reported heavy experimentation.

15 (78.9%) reported to have used the server from the university, 8 (42.2%) used their own environment on their own computer, and 2 (10.5%) used the provided Docker container on their computer.

Regarding the used tools for benchmarking and profiling, our sample was evenly split in users and non-users. For benchmarking, 10 participants (52.7%) reported to have not or seldom used the provided tools, while 9 reported to have used the tools often, mostly, or whenever possible (with the latter being the 31.6%). For profiling, 9 participants (47.4%) reported to have not or only sometimes have used the provided tools, whereas 10 participants (52.7%) reported to have used the tools often, mostly, or whenever possible (again, the last answer "whenever possible" was the most frequent one, 31.6%). For the following analysis, we split the sample in the groups of users and non-users of the tools.

1) Tool use and learning outcome: We measured knowledge before and after the course (both reported in the same survey) and found a strong and significant increase in perceived knowledge ($T_{(df=17)} = -12.9$, $p < .001$, $d = 3.05$) with an average increase of 1.21 points on the 5-point knowledge scale. The increase was particularly strong in the areas of parallel programming, performance analyses, and profiling. As this result should be expected for a university course on parallel programming, we further investigated how the perceived competencies relate to the use of the provided tools.

We calculated a linear regression with tool use (see above) and subjective competency before the course as independent variables and subjective competency after the course as dependent variable. Table I shows the significant regression model that explains over 52% of the variance in subjective competency after the course ($F_{(2,15)} = 10.3$, $r^2_{adj} = .523$, $p = .002 < .05$). Obviously, prior competency has a significant and strong influence on the outcome ($\beta = .602$, $t = 3.51$, $p = .003 < .05$) and, as indicated by the significant intercept, the participants report higher competency after the course.

More importantly in our context, using the provided tools had an equally strong (similar β-coefficients) and positive impact on the perceived competency after the course ($\beta = .687$, $t = 2.05$, $p = .059 > .05$), although the threshold of statistical significance was narrowly missed [2].

Lastly, we checked if using the tools is unrelated to the participants' demographics. In our sample, tool use was not related to age ($\tau = .017$, $p = .934 > .05$) and gender ($\tau = -.149$, $p = .520$), hence we assume that the tools do not pose barriers for certain user demographics.

TABLE I: Regression results for competence after the course as dependent and tool use and competence before the course as independent variables. The sig. model explains $r^2_{adj} = 52.3\%$ of the variance in competence after the course.

Predictor	**Estimate**	**SE**	**t**	**p**	**Std.** β
(Intercept)	2.233	0.289	7.73	$< .001$	–
TOOLUSE	0.417	0.204	2.05	.059	0.687
COMPETENCE (pre)	0.540	0.154	3.51	.003	0.602

In summary, the findings suggest that using the provided performance analysis tools has a positive influence on the (alas only subjectively measured) competence of the students.

2) Qualitative Feedback: In addition to the quantitative analysis, we asked the students about aspects that they found positive and negative about the course concept and the tools provided using open-ended questions. In the following, we present a thematic analysis [16] and quotes of the key aspects mentioned by the students. The feedback addresses the structure of the teaching unit and the exercises, as well as the provided Jupyter Notebooks.

Regarding the course content, the script received praise for being consistent and concise ("*[The script] is consistent and crisp with an eye for the essentials.*"). Also, the first four exercise assignments were seen as well-designed and challenging. However, some students reported to have struggled with part 5 but found it manageable.

As for the provided Jupyter Notebooks, the students appreciated the uniform development environment and their ease of use ("*A standardised development environment for all students makes sense and is fun.*", "*I considered setting up a local environment, but the considerable extra work involved put me off.*"), the ability to generate profiling results in the Notebooks ("*I particularly liked the ability to quickly generate profiling results from parallel programs.*"), and the integrated assignments ("*The intuitive operation and the already integrated assignments.*").

Yet, some participants encountered issues with the Jupyter environment and preferred local development ("*I'm just not a fan of Jupyter Notebooks. Especially the variant for C++ was picky towards some constructs.*"), occasional performance issues ("*It was quite slow, depending on the time of day.*"), and they suggested more documentation.

Summarizing, the feedback reflects a mix of appreciation for the course content, alignment of the assignments, and functional aspects of the Jupyter Notebooks, as well as technical challenges and time requirements for certain assignments.

C. Limitations

While the study reveals some interesting findings, it is not without its limitations. The main limitation of this evaluation stems from the small number of participants in the course and consequently in the survey (elective course in a specialization area, voluntary participation in the survey), as well as the diverse population of students at our university. The generalizability of the findings is limited, as this exploratory study focused on a pragmatic evaluation of the course concept and the provided tools with the goal of their future improvement.

Instead of a formal pre-post design, we captured the perceived competency before and after the course in a single survey. As the students could voluntarily decide to use or not use the tools, the results may be biased by their self-efficacy [2]. Yet, the study suggests a positive effect of the provided tools, even if accounted for prior competency. Consequently, the didactic concept and effect of the tools should be more thoroughly investigated in a field experiment, probably also including students from different universities [6].

VII. Conclusion

The new performance tool integrates into JupyterLab and enables C/C++ performance analysis in an interactive environment. Our evaluation shows that the students liked the tool, and it helped them to get a better understanding of parallel programming, benchmarking, and performance analysis. However, since not all students liked Jupyter Notebooks, we will still provide "traditional" materials like code skeletons and Makefiles which can be used outside Jupyter.

In the future, the C++ wrapper class could be translated to different programming languages and the python visualizations could be extended to support more analysis tools. In the future, we also plan a new study with designed pre- and post-tests to gauge the learning outcomes.

In part, our work tries to address problems caused by poor documentation and user interface design, and we encourage performance analysis tool developers to simplify the setup process, provide examples, and follow best practices of user interface design (e.g. [5]) and technical documentation writing (e.g. Diátaxis [8] or the Good Docs Project[19]).

[2]A model based only on tool use as independent variable is significant and also shows a strong positive effect of tool use on subjective competency ($r^2_{adj} = .204$, $\beta = .983$, $t = 2.37$, $p = .030 < .05$)

Future research should investigate the impact of easily accessible performance measurement tools and their integration in different educational settings.

Beyond training computer scientists, there is also great potential and demand in other domains, such as the transformation of production [1]. Here, non-computer scientists must be enabled to process large amounts of data efficiently, which is only possible through suitable, easy to learn, and easy to use tools for benchmarking and profiling.

Availability of data and materials

The source code and data generated during the current study are available at https://zenodo.org/records/10573107.

Acknowledgements

We would like to thank all the students who gave us feedback on the course concept and our tools in the questionnaire and beyond.

Funded by the Stifterverband für die Deutsche Wissenschaft e.V.

Funded by the federal state of North Rhine-Westphalia.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC- 2023 Internet of Production – 390621612.

References

[1] Philipp Brauner et al. "A Computer Science Perspective on Digital Transformation in Production". In: *ACM Transactions on Internet of Things* 3.2 (2022), pp. 1–32. ISSN: 2691-1914. DOI: 10.1145/3502265.

[2] Philipp Brauner et al. "The Effect of Tangible Artifacts, Gender and Subjective Technical Competence on Teaching Programming to Seventh Graders". In: *Proceedings of the 4th International Conference on Informatics in Secondary Schools (ISSEP 2010), LNCS 5941*. Ed. by Juraj Hromkovic, R. Královič, and J. Vahrenhold. Zurich, Switzerland: Springer-Verlag Berlin Heidelberg, 2010, pp. 61–71. DOI: 10.1007/978-3-642-11376-5_7.

[3] Clang. *OpenMP Support - Clang 9 documentation*. URL: https://releases.llvm.org/9.0.1/tools/clang/docs/OpenMPSupport.html. (accessed: 04.03.2024).

[4] Dirk Colbry. "The Design of a Practical Flipped Classroom Model for Teaching Parallel Programming to Undergraduates". In: *Journal of Computational Science* 12.2 (2021).

[5] *Command Line Interface Guidelines*. URL: https://clig.dev/#guidelines. (accessed: 04.03.2024).

[6] Catherine Courage and Kathy Baxter. *Understanding Your Users – A Practical Guide to User Requirements Methods, Tools & Techniques*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 2005. ISBN: 1558609350. DOI: 10.1016/B978-1-55860-935-8.X5029-5.

[7] *CubeGUI User Guide*. URL: https://apps.fz-juelich.de/scalasca/releases/cube/4.8/docs/CubeUserGuide.pdf. (accessed: 04.03.2024).

[8] *Diátaxis - A systematic approach to technical documentation authoring*. URL: https://diataxis.fr/. (accessed: 04.03.2024).

[9] Ben Glick and Jens Mache. "Using jupyter notebooks to learn high-performance computing". In: *J. Comput. Sci. Coll.* 34.1 (Oct. 2018), pp. 180–188. ISSN: 1937-4771.

[10] Jens Henrik Göbbert et al. "Enabling interactive supercomputing at JSC lessons learned". In: *International Conference on High Performance Computing*. Springer. 2018, pp. 669–677.

[11] Loic Gouarin Johan Mabille and Sylvain Corlay. *Xeus Cling*. URL: https://xeus-cling.readthedocs.io/en/latest/. (accessed: 04.03.2024).

[12] Andreas Knüpfer et al. "The vampir performance analysis tool-set". In: *Tools for High Performance Computing: Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing, July 2008, HLRS, Stuttgart*. Springer. 2008, pp. 139–155.

[13] Johan Mabille and Sylvain Corlay. *Xeus*. URL: https://xeus.readthedocs.io/en/latest/. (accessed: 04.03.2024).

[14] Michael Milligan. "Jupyter as Common Technology Platform for Interactive HPC Services". In: PEARC '18. Pittsburgh, PA, USA: Association for Computing Machinery, 2018. ISBN: 9781450364461.

[15] *Module des Studiengangs Praktische Informatik M.Sc.* URL: https://www.fernuni-hagen.de/mi/studium/msc_prinformatik/module.shtml. (accessed: 04.03.2024).

[16] Kimberly A. Neuendorf. *The Content Analysis Guidebook*. SAGE Publications, 2017. ISBN: 9781412979474.

[17] Pambayun Savira, Thomas Marrinan, and Michael E Papka. "Writing, Running, and Analyzing Large-scale Scientific Simulations with Jupyter Notebooks". In: *2021 IEEE 11th Symposium on Large Data Analysis and Visualization (LDAV)*. IEEE. 2021, pp. 90–91.

[18] *Score-P: the Scalable Performance Measurement Infrastructure for Parallel Codes*. URL: https://perftools.pages.jsc.fz-juelich.de/cicd/scorep/tags/latest/html/. (accessed: 04.03.2024).

[19] *The Good Docs Project*. URL: https://thegooddocsproject.dev/. (accessed: 04.03.2024).

[20] Elias Werner et al. "Bridging between Data Science and Performance Analysis: Tracing of Jupyter Notebooks". In: *The First International Conference on AI-ML-Systems*. 2021, pp. 1–7.