Teaching supercomputing and software engineering skills to science and engineering students

Ludovic Räss

Laboratory of Hydraulics, Hydrology and Glaciology (VAW), ETH Zurich, Zurich, Switzerland
Swiss Federal Institute for Forest, Snow and Landscape Research (WSL), Birmensdorf, Switzerland

Mauro A. Werder

Swiss Federal Institute for Forest, Snow and Landscape Research (WSL), Birmensdorf, Switzerland
Laboratory of Hydraulics, Hydrology and Glaciology (VAW), ETH Zurich, Zurich, Switzerland

Ivan Utkin

Laboratory of Hydraulics, Hydrology and Glaciology (VAW), ETH Zurich, Zurich, Switzerland
Swiss Federal Institute for Forest, Snow and Landscape Research (WSL), Birmensdorf, Switzerland

Samuel Omlin

Swiss National Supercomputing Centre CSCS, Lugano, Switzerland

Abstract

We describe a newly created Master-level course about numerically solving partial differential equations (PDEs) on graphical processing units (GPUs), both on local machines, on high-performance computing (HPC) clusters and supercomputers. The course is aimed at *domain-science* students, which we broadly define as non-computer science students, such as earth-science, physics, engineering, etc. students. Besides the core content, the course aims also at teaching other essential skills needed for the domain scientist to successfully conduct numerical research, including software engineering skills (e.g., git and GitHub, testing, documentation), tools (e.g., VSCode and remote HPC access), teamwork and project management. The course is research-based as it closely follows the workflows, we use in our daily research activity. The course teaches its content through a hands-on, project-based approach with weekly assignments and two large projects as a core part of the course. We show that student grades and satisfaction is excellent, however, the high workload of the teachers warrants refinements in future iterations of the course.

Introduction

High-performance computing (HPC) takes an increasingly bigger role in modern science, utilising computers ranging between small local clusters to supercomputers. We, earth and fluid mechanics scientists, and many other domain scientists (in contrast to computer scientists) nowadays write code and run it on HPC-clusters. HPC changes over time, with the current standard moving towards using hardware accelerators such as graphics processing units (GPUs). But also on a smaller scale, workstations with one or several GPUs deliver compute capabilities similar to what entire research clusters would have less than a decade ago. Rapidly changing hardware also puts high requirements on scientific software development in order to leverage the constantly new capabilities efficiently (Dongarra, 2022).

As an example, scientific computing in earth sciences, which is the domain of the authors, leverages the continuous increase in the amount of observational data and available computing capabilities (Morra et al., 2021). This allows, for instance, to resolve all (or at least more) relevant physical processes from first principles instead of parameterising them and, thus, allows to test hypotheses on dynamical processes.

Advances in programming languages and scientific software packages make it possible for the domain scientist to now design and code HPC simulations without years of training. New languages such as Julia[1] (Bezanson et al., 2018) were designed to address the two-language barrier: namely allowing for a single codebase to serve both during prototyping and during production simulations. Previously, prototyping was typically done in a high-level but slow language, which then needed to be translated into a fast but low-level language for the production simulations. Skipping this step reduces a costly development cycle avoiding code duplication and errors when switching between higher-level prototyping codes and lower-level HPC production codes (Churavy et al., 2022). Coincidentally, these developments now also make it possible to teach students how to produce performant HPC-GPU applications within a just one semester long course.

We found that there was a gap, which we could fill, in ETH Zurich's (ETHZ) curriculum with respect to GPU based HPC with a focus on physics-based simulations. Thus, we designed and now twice taught a course which aims to spread the basics of HPC and supercomputing on GPUs to the (future) domain scientist, i.e., to science and engineering students. The course covers iterative algorithms for solving partial differential equations (PDEs) efficiently on GPUs (Räss et al., 2022), their implementation, performance considerations, without relying on external libraries as much as possible (with exception of visualisation and lower-level building blocks such as hardware specific software layers).

This manuscript is structured as follows: we first describe how we set up the course with focus both on a conceptual viewpoint as well as on technical aspects. We then discuss the outcomes of the course in terms of student participation, feedback, grades and handed in assignments and projects. The aim of this paper is to reflect on how our research-, project-, and tooling-based course achieves conveying the full skill set needed for our students to get started with computational domain science.

Course organisation and setup

To achieve our goal to introduce students to coding HPC simulations, we do not only teach them how to write numerical code but also project management skills and how to use software-engineering tooling, in the spirit of Software Carpentry (Wilson, 2006) and beyond (Smith, 2018). These computational competencies complement other skills our course emphasises on such as the use of supercomputing for domain science applications and the elaboration of a research process. We feel that all of these skills are equally important for a student to be successful as a computational domain scientist.

We pursue a research-oriented learning approach in the course. Healey (2005) writes "[…] students are likely to gain most benefit from research when they are actively involved in carrying out research projects, whether in part or in whole.". Healey (2005) furthermore decomposes the research-teaching nexus along three dimensions: (1) emphasis on research content versus research process, (2) students as audience or participants, and (3) teacher-focused or student-focused teaching. As stated in the above quote, one of the ways to produce a research-oriented teaching style is through projects. For instance, Pinho-Lopes and Macedo (2016) found that project-based learning models are well received by students. This approach

[1] http://www.julialang.org

is a form of active learning where the students develop the knowledge themselves rather than it being presented by the teachers (Prince & Felder, 2006).

A particular focus of our course is that we want to achieve that students feel *ownership* of the code they produced, the tools they learned and the projects they made during the course. To achieve this, we designed our course to be delivered in a *hands-on* manner. Besides the obvious, namely that the students need to write code for the course, hands-on also means to us that they need to learn and apply the skills and tools needed to successfully create scientific models and run simulations with them: software-engineering tools (e.g., version-control with git, GitHub, editors/IDEs, testing) and project management skills (e.g., writing documentation and reports, running simulations). We convey these skills via a project-based teaching approach.

Course structure

The course consists of 14 weekly lectures of 3h each spanning one semester. The course includes concise lecturing, weekly assignments, a project common among all students, and a personal final project. We use lectures as a basic course unit; typically, each lecture and its associated assignment cover one topic. Table 1 gives an overview of the covered material.

Lectures	Material taught		Exercises	
	main-topics	side-topics	main-topics	side-topics
Why Julia and GPUs	PDEs, GPUs	Tools for the job (Julia, Jupyter notebook)	Numerical solutions, predictive modelling	Visualisation
PDEs and physical processes	Solving PDEs	Git and version control	Solve basic physics (advection, diffusion, reaction)	Install Julia, create git repository
Solving elliptic PDEs	Fast iterative elliptic PDE solvers	Julia REPL and package manager	Implicit iterative solvers in 1D and 2D	Parametric study
Porous convection	A physical model for thermal porous convection	Julia's project environment	Thermal porous convection in 2D	2D visualisation
Parallel computing	Performance considerations, shared memory parallelisation	Unit testing in Julia	Performance evaluation	Unit testing
GPU computing	GPU architecture, array and kernel programming	Reference testing in Julia	Data transfers and optimisations, solving PDEs on GPUs	Unit and reference tests
xPU computing	The two-language barrier, backend portability	Continuous integration (CI) and GitHub Actions	3D thermal porous convection	Continuous integration and GitHub runners
MPI and distributed computing	Distributed memory computing, MPI	Getting started on a supercomputer	Distributed computing on GPUs	Running on a supercomputer
Advanced optimisations	Scientific applications' performance		Shared memory and registers manual tuning	
Projects	Solving PDEs on GPUs	Documentation, GitHub repository, continuous integration		

Table 1: Material and topics discussed during the course listed as main- or side-topics.

New material is introduced during the first 10 lectures, the remaining 4 lectures are reserved for the final project. The first two lectures provide the students the opportunity to familiarise themselves with the Julia language, differential equations and allow them to set up the tooling required for the class. To assure a smooth start, the coding related to the two first lectures happen in Jupyter notebooks hosted on a JupyterHub server (Granger & Pérez, 2021). Having a ready-to-use environment allows to quick-start the course avoiding the need to get per-person environment configurations. At lecture 3, we require everyone to have their personal Julia installation. Lectures 7-9 serve as a basis for the first, common project. Finally, during lectures 11-14 of the course, students work on a personal final project of their choice. They can choose among implementing domain specific PDEs they want to solve or applying performance optimisations to the codes from their first project, using the tooling and skills acquired during the course.

Each lecture contains a concise lecturing part used to introduce the basic concepts to be addressed. The new concepts are programmatically exemplified during the class by means of live-coding and small in-class exercises. Important concepts are practised using active learning sequences. The general approach is to provide an incremental build-up of knowledge and competencies throughout the course and provide to the student a set of skills they can further apply in the two projects (Table 1).

In summary, the course is founded on project-, tooling- and research-based teaching to convey the students the basis of HPC on GPUs.

Technical aspects

To provide a smooth learning experience to students we rely on a custom technical stack composed of specific hardware infrastructure, open-source software solutions and ETHZ-provided learning and communication platforms. This technical stack forms the backbone of our course and thus warrants a description. However, readers less interested in this aspect of the course should skip to the last paragraph of this section.

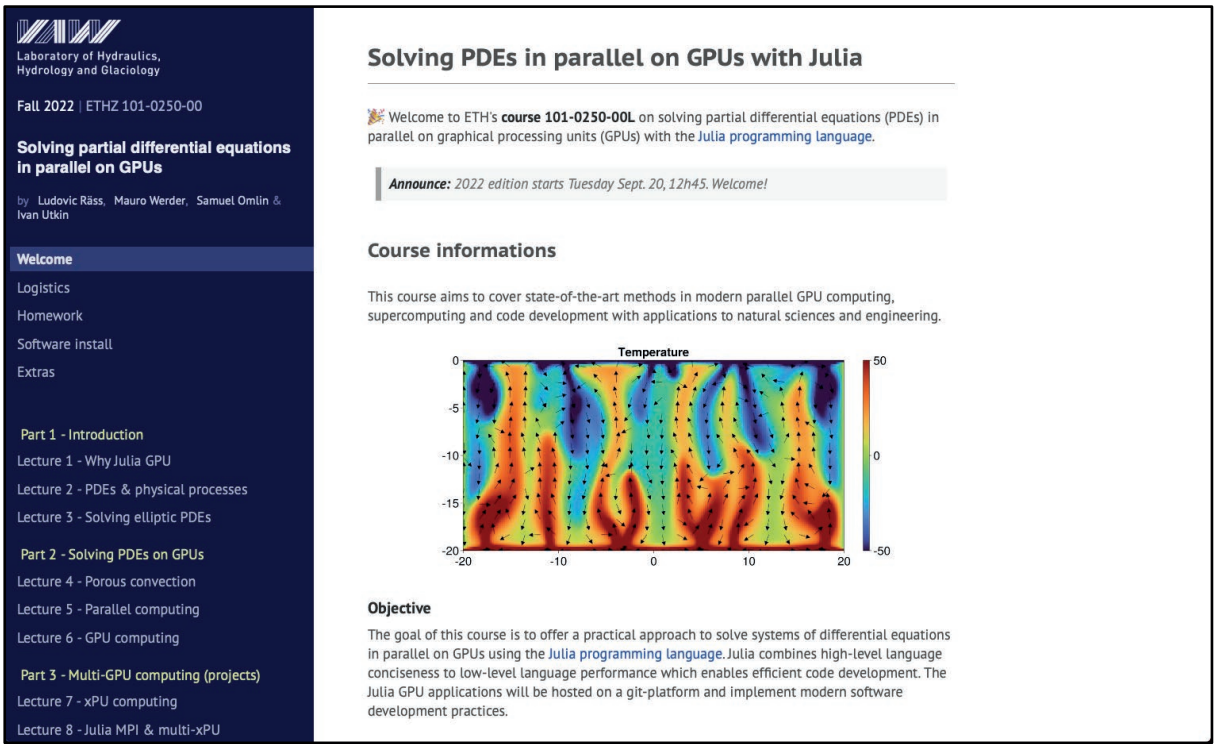

Figure 1: Screenshot of the 2022 edition course website https://pde-on-gpu.vaw.ethz.ch/ landing page.

The core of the course's technical stack is a GitHub (a code-hosting website) repository[2] including the course's website deployed using GitHub-pages at the custom ETHZ-provided address https://pde-on-gpu.vaw.ethz.ch/ (Fig. 1). We rely on a fully Julia-based stack combining literate programming[3], which is a programming technique combining text/documentation with the code (Knuth, 1984), and a static website generator[4] to generate the Jupyter notebooks, demo scripts and to deploy the content online; all is generated from a single master script. The notebook can further be turned into slides which can be presented with Jupyter using the RISE[5] plugin. The single-script approach avoids code and content duplicates which would lead to tedious maintenance and version tracking overhead. We automated most of the workflow by creating helper scripts that would trigger the deploy pipeline. During literate-code processing, we added the capability to parse for specific keywords which allows us to deploy assignments with hints or solutions depending on the needs, thus allowing us to keep the assignment and its solution in one master script.

We rely on a set of digital support applications, namely GitHub, Moodle (online learning platform), Zoom (video conferencing) and Matrix/Element[6] to enable the best experience and support a hybrid (in-person / remote) course format. Besides deploying the course website, we also use GitHub as a platform to handle students' weekly assignments and final projects. Hand-ins need to be pushed to GitHub prior to the deadline and a git commit hash is further uploaded to Moodle, a bot then downloads the student's code automatically to ETHZ servers. We rely on Moodle for ETHZ-only secure access such as sharing sensible information, collecting git commit hashes as well as its integrated JupyterHub. We use Zoom to broadcast and record all lectures and provide recordings on Moodle to allow students with conflicting schedules to still follow the course. We use Matrix/Element –an open standard for interoperable, decentralised, real-time communication– for instant messaging among course participants. The service, staff-chat and student-chat by ETHZ can be accessed via the cross-platform Element client. The class-related chat allows teachers to share general information with students via a "General" info channel and to run Q&As in a separate "Helpdesk" channel. The benefit of the approach avoids repetition of help as the Q&A is accessible to everyone enrolled in the class. Also, it pushes students to ask precise questions and we encourage the approach of using a minimal (not) working example in their posts that would precisely specify their issue. We also motivate students to take over the Q&A whenever possible such that they could provide help among each other and learn from it. The service being opt-in, everyone is free to filter the info to their needs.

To ensure a smooth start, we value that students can access a ready to go coding and learning environment. This step is important as getting the computing environment to be ready on the students' laptops may be very time consuming and would not provide a very motivating introduction for the course. Thus, for this step, we rely on JupyterHub. In the first edition of the course, we manually deployed a stripped-down version of JupyterHub[7] on one of our servers which the students could then access. Starting from the second edition of the course we used a JupyterHub instance which is now available from ETHZ with Moodle integration.

Among the goals of the course is the development of multi-GPU applications. Large-scale GPU resources are not so common, but the Swiss National Supercomputing Centre (CSCS) supported our course with 4'000 node hours compute time on the Piz Daint GPU supercomputer[8] in 2022. In 2021, we used the GPU computing resources offered by the Swiss Geocomputing Centre (Unil) where students could get free compute time on the Octopus

[2] https://github.com/eth-vaw-glaciology/course-101-0250-00
[3] https://github.com/fredrikekre/Literate.jl
[4] https://github.com/tlienart/Franklin.jl
[5] https://rise.readthedocs.io/en/stable/index.html
[6] https://matrix.org/
[7] https://tljh.jupyter.org/en/latest/
[8] https://www.cscs.ch/computers/piz-daint/

supercomputer. We could use the JupyterHub instance provided by the CSCS to get students smoothly started on Piz Daint including its GPUs, before transitioning to a more classical development and script-submission workflow on the supercomputer.

JupyterHub and notebooks provide a good quick-start environment but rapidly hit limitations in the way we do numerical modelling. Also, they cannot currently be used to efficiently launch processes in parallel and may lead to significant performance overhead. For this reason, we then transition to running Julia and simulations "locally", on students' laptops and on Piz Daint for the first and second parts of the course, respectively. This is seamlessly achieved thanks to Visual Studio Code (VSCode, a code editor)[9] and its Julia extension[10]. VSCode also features a Remote-SSH[11] extension which greatly simplifies the access to remote compute resources such as the Piz Daint supercomputer at CSCS. Using VSCode permitted all the students to have a ready to use local or remote Julia installation and we had not a single issue with students setting this up between two lectures over the course of a week (no support was needed from the teaching staff).

The Julia language consists of a standard library to which specific functionalities such as visualisation, maths-operations, parallelisation, or backend specific computations, can be added by using packages. We specifically use packages related to GPU computing and contribute to part of these. GPU-related packages such as CUDA.jl (Besard, Churavy, et al., 2019; Besard, Foket, et al., 2019) or AMDGPU.jl are grouped in the JuliaGPU[12] organisation. The building-blocks packages we further use in the course and contribute to are ParallelStencil.jl[13] (Omlin & Räss, 2022) and ImplicitGlobalGrid.jl[14] (Omlin et al., 2022).

In summary, the interactive course material is deployed to the course website directly from source-code scripts using an automated pipeline. The coding in class is started on Julia notebooks that are running on JupyterHub instances on ETHZ servers (CPU only) and on Piz Daint at CSCS (multi-GPU), this allows the students to hit the ground running and to not get bogged down with installation technicalities at the onset. In a second step, students transition to a local Julia installation on their laptops and run parallel Julia GPU scripts on Piz Daint outside of the notebook environment.

Evaluating student performance

We evaluate student performance in a composite way including weekly exercises, a common project, and a personal final project, contributing 30%, 35% and 35% to the final grade, respectively. Note that no examination is held.

During the first 6 lectures, students need to hand-in 5 out of 6 assignments which we use to give them prompt feedback and evaluate as well (Table 1). The main reason being to keep students motivated in the first weeks such that they can better appreciate the following steps. The project-work during lectures 7-9 is on the same project for the entire class which aims at bringing together the skills learned thus far, namely resolving multi-physics flow processes in 3D on multiple GPUs using the Julia language, in one self-contained unit. This first project also provides a consistent way to grade all students based on the same tasks presented to them with clear steps. The final project permits students to apply the skills they acquired during the course to their domain science, such as earth-sciences, engineering, or physics (see Table 1 for a summary of the course design).

[9] https://code.visualstudio.com/
[10] https://www.julia-vscode.org/
[11] https://code.visualstudio.com/docs/remote/ssh
[12] https://juliagpu.org/
[13] https://github.com/omlins/ParallelStencil.jl
[14] https://github.com/eth-cscs/ImplicitGlobalGrid.jl

The projects provide an ideal way to assess students' autonomy and ability to integrate the concepts learned from class with a clearly defined objective, namely, to write a Julia code that solves porous thermal convection in 3 dimensions (3D) and runs on 64 GPUs on the Piz Daint supercomputer at CSCS. Besides writing their own code and running it, students had to document their code using docstrings and code annotations, create unit and reference tests for their software, run the tests with continuous integration (CI) using GitHub Actions[15] and provide a report in the form of an enhanced README-file in their repository.

For their personal final projects, students would ideally try to identify differential equations relevant to their scientific interests and try to solve them using the newly learned methods. The evaluation of this final part is more challenging to ensure fairness between the different projects. We decided to value some formal items such as content and quality of the final report (an online README-file or automatic generated documentation) and give additional points for creativity and relevance. Project-based evaluations are interesting as they train students to manage their timing, allow them to organise their work with some freedom but also pushes them to report about their work in a concise way.

Teaching and learning in this course

Here we summarise our experiences from teaching two editions of the course and how we perceive the students were learning the material. The management of the course always went smoothly, although the workload is high, both for students and teaching staff. The course is very participative which demands extra resources on both students and teachers' side. We decided to limit the maximum number of participants to the course to 20 and 25 students in the first and second course edition, respectively, to ensure good quality and have enough bandwidth for supporting them. This choice paid off as we could provide that required support. We were also positively surprised that the broad variety in students' background (earth sciences, civil engineering, computer sciences, computational sciences and engineering, electrical engineering, mechanical engineering and robotics, mathematics, physics) did not hinder the teaching but rather provided material for constructive exchanges among students. According to the very positive student evaluations we got for these first two editions (Fig. 4), we can report that students enjoyed the practical approach towards demystifying GPU supercomputing. Students put in hard work and were enthusiastic about the course.

Assignments in the first 6 lectures of the course are important. They significantly help students to learn the material, both the core content as well as the additional skills and tools and push them to do the exercises during the course; this view is also shared by the students (Fig. 4d). They also provide a way for students to get feedback on their work assuming the teaching staff has enough time to promptly correct the hands-in and provide that feedback. The assignment correction is consuming about 8 to 10 hours of the teaching staff's resources on a weekly basis. For the second edition of the course, we hired a teaching assistant which significantly improved the early feedback we could give to the students about their exercises.

The final projects provided the students the opportunity to apply their GPU and HPC knowledge to solve a problem of interest in their scientific field. The projects required them to apply all the skills and tools learned in a integrated fashion. Most of the projects partly combined suggested topics with some variations to make them fit to the students' interests. Among the successful projects, we selected two projects from each year of the course to showcase here: In the first edition of the course, one student team tackled a 2D Navier-Stokes flow problem relying on a geometric multigrid solver running on Nvidia GPUs (Fig. 2a). Another student resolved shear heating activated shear-band formation due to thermomechanical coupling in 2D on GPUs (Fig. 2b). From the second edition of the course, we selected one project resolving acoustic

[15] https://docs.github.com/en/actions

wave propagation in 3D on multi-GPUs with an efficient convolutional perfectly matched layer (CPML) boundary condition implementation. This code will serve as the basis to perform full waveform inversions (Fig. 2c). The other highlight project from the second course edition resolves flow migration in viscously deforming porous media exhibiting the occurrence of solitary waves of porosity (Fig. 2d).

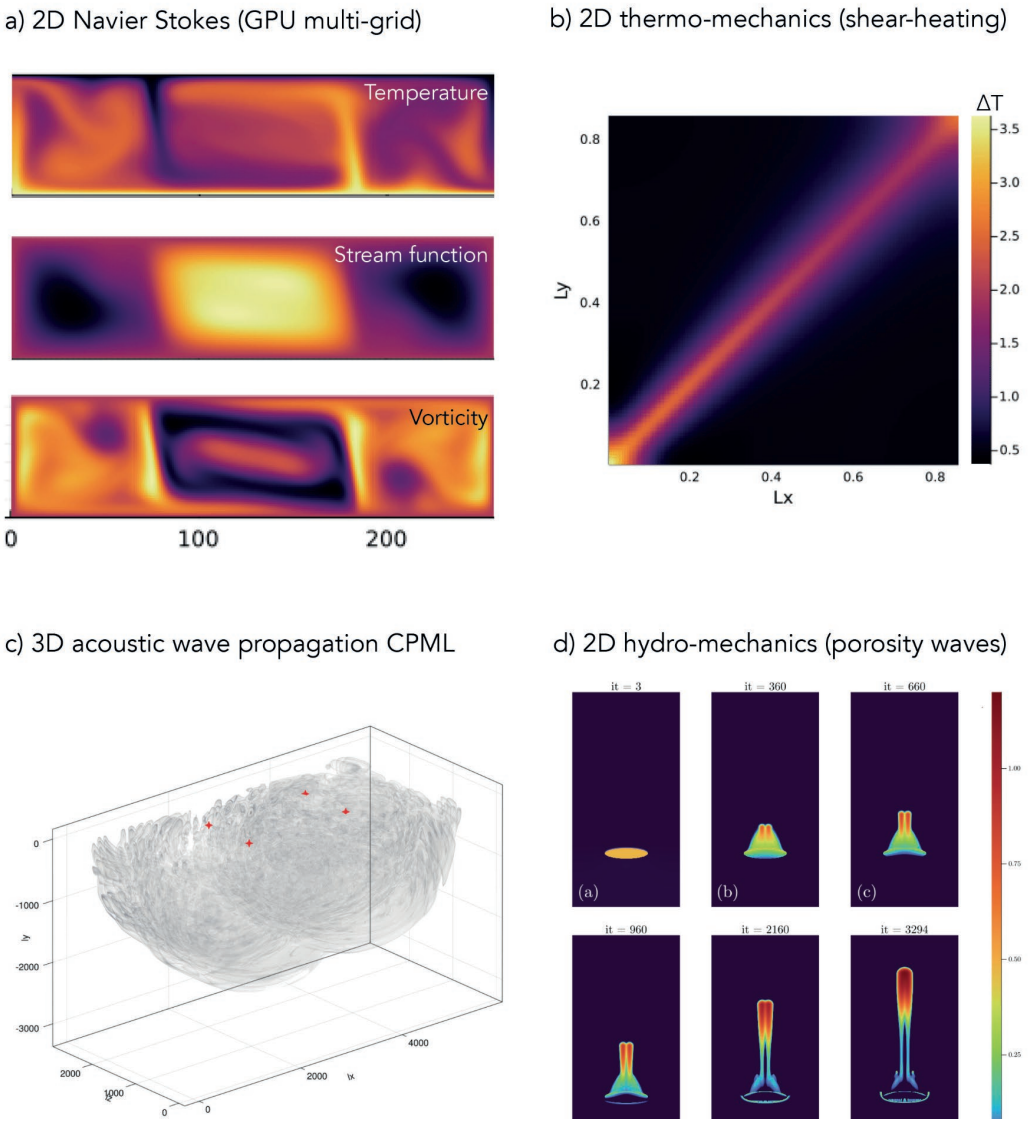

Figure 2: Selected student projects gallery. a) 2D Navier-Stokes simulation of convecting fluid on GPUs using a multi-grid solver[16]; b) 2D thermo-mechanical coupling leading to the formation of a ductile shear zone owing to shear-heating[17]; c) 3D multi-GPU acoustic wave propagation with a convolutional perfectly matched layer (CPML) boundary condition by Giacomo Aloisi[18]; and d) 2D hydro-mechanical coupling leading to the formation of solitary waves of porosity in deformable porous media by You Wu[19].

[16] https://github.com/ntselepidis/FinalProjectRepo.jl/blob/main/docs/part2.md
[17] https://github.com/YWang-east/course-101-0250-00-FinalProject/blob/main/docs/part2.md
[18] https://github.com/GiackAloZ/AcousticWaveCPML.jl
[19] https://github.com/youwuyou/HydroMech.jl

Throughout the course we ensure that the student exercises and projects also include and foster the additional skills we aim to teach (see Table 1 "side-topics"). The software engineering skills are directly tested as they are part of the exercises and projects: code hosting and exercise submission is using GitHub and git; the students are required to submit unit and reference tests; documentation is in form of readme documents; and running code on the HPC clusters is done using direct integration on the code-editor they are using. The students generally work in teams of two for the project work and thus practise teamwork skills; however, a direct assessment of these skills is not easy and not done within the course. Project management skills, such as report and documentation writing, keeping track and organising simulation outputs, are needed and integral work for both projects. Taken together with the actual numerical computing, these activities mean that student do practise all those additional skills and that they really have ownership of their final project as they produced everything in that project in a setting which could be used in a research environment.

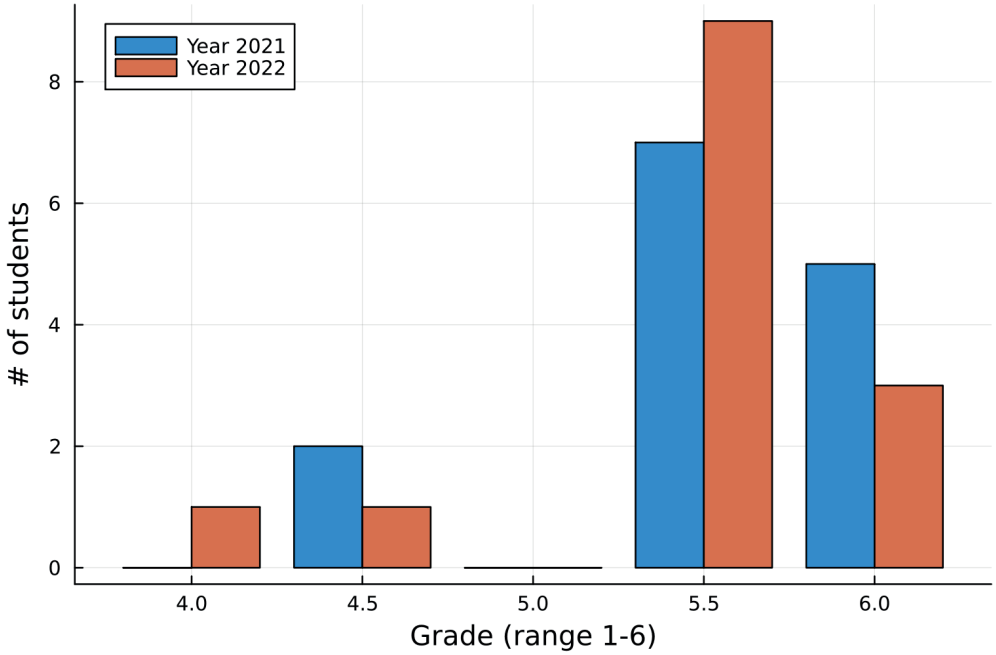

Figure 3: Student grades distribution for the two first editions of the course.

To gauge the student's perception of and happiness with the course, we rely on the standardised end-of-course survey of ETHZ and on informal feedback. Figure 4 shows a few selected answers from this survey, with the full survey available in the supplementary material. In general, the survey shows very positive evaluations from the students (Fig. 4a). Points of contentions are mostly related to workload and credit points (Fig. 4b), however, there is no clear signal there. The big effort on our side with preparation of material presented on the course website and the assignments are appreciated (Fig. 4c,d). Particularly good, we feel, is the positive feedback on the question whether the students could explain the material to a younger student (Fig. 4e). Also, informal feedback was good, with statements such as "not many courses like this are available to master students, so kudos to you!". We also perceive that the students learn a lot in this course, as is exemplified by some outstanding final project (Fig. 3) and that they are well motivated as exemplified by the close interaction with them.

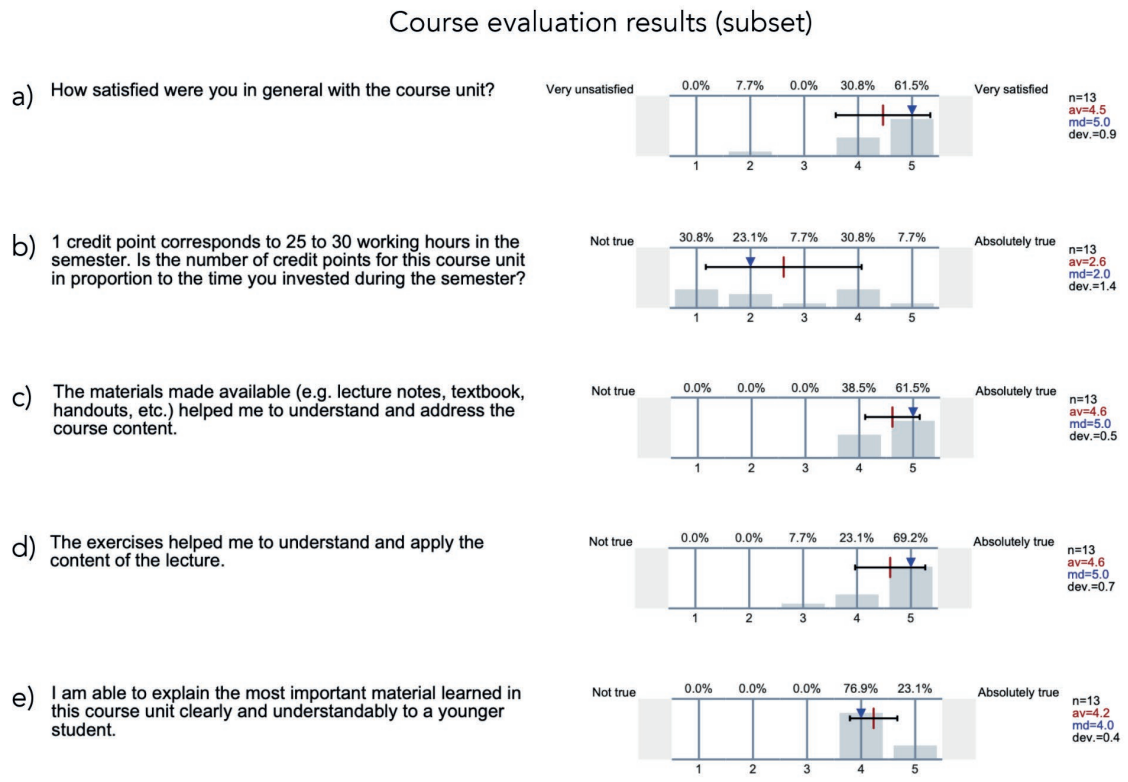

Figure 4: Course evaluation results for the 2022 edition. a) General satisfaction; b) Credit points versus workload satisfaction; c) Usefulness of support material; d) Exercises as knowledge transfer; and e) Ability to transfer acquired knowledge.

Student performance

The good grades reflect the students' involvement in the lecture (Fig. 3): all passed with more than five out of six receiving excellent grades. On average over the two first editions, we experienced about 20-25 inscriptions and 5 to 6 dropouts. These dropouts were related to too much workload in the whole of the students' curriculum and thus their need to reduce the number of courses.

Discussion

Teaching this course is a rewarding experience due to the combination of being able to teach subjects of high interest, namely GPU computing as well as software engineering tools, and due to the close interaction with highly motivated students. Thinking back to our days as students, this certainly would have been a course which we would have enjoyed taking and it would have helped jumpstart the scientific career we are now pursuing. The downside of the course is that it requires a high teacher workload, which needs to be reduced to keep this course sustainable in the long run.

The course is surprisingly well aligned to ETHZ's teaching strategy which states that "Research and teaching must be closely linked" and that "the acquisition of soft skills, […,] computational competencies, and the ability to analyse complex issues" (p.36) are key skills to be taught (Cantalou et al., 2021). The former being fulfilled as we are teaching the methodology, we employ during our research work (Healey, 2005). One of our intentions to teach this course is to prepare current and future Master and PhD students to work with us. Indeed, one current Master student completed the course as preparation for their thesis with us, and for one freshly

started PhD student the upcoming course in 2023 will be an integral part of their doctorate. Besides the research-oriented focus, the course aims at conveying many other skills such as software engineering, working in a team or project management through a project-based approach. Again, our motivation for this approach was that we want to pass on the knowledge and skills needed to succeed as a computational domain-scientist.

The technological advances over the last decade make a course on solving PDEs on GPUs possible. Numerical computing on GPU commenced in the early 2000s and started getting established with the first release of CUDA (the library to perform general purpose computations on Nvidia GPUs) in 2007 (Vuduc & Choi, 2013). However, until recently performing numerical computations on GPUs either required direct coding in low level languages, such as CUDA C, or the usage of GPUs via highly abstracted interfaces in high-level languages which either restricted capabilities, performance, or both. We feel that only the recent advances in the Julia programming language, made a course as ours possible by allowing to teach the low-level details whilst being easy enough to learn to fit into a semester course. Besides the GPU-computing capabilities, Julia provides a modern open-source software environment embracing many of the industry's best practices such as hosting of code in git-repositories (mostly GitHub) automated testing using continuous integration and having documentation (Hanson & Giordano, 2021). Thus, the Julia ecosystem serves as an ideal backdrop to teach those concepts to students.

Our current research uses the Julia language and its GPU package ecosystem and thus the course is directly based on our research and in the framework of Healey (2005), our course is indeed research-based. In his decomposition of the research-teaching nexus (see Introduction), our course focuses on: (1) the research processes rather than the research content, as we mainly teach a method on how to solve PDEs but no not focus on why one would want to solve those equations in the first place; (2) students in our course are definitely participants in many instances as the hands-on exercises and projects take up the largest part of the course; and (3) our teaching is student-focused, again through the many hands-on sequences, however, we do employ classical, teacher-focused lecturing to cover foundational material at a rapid pace.

One of the driving principles behind the design of this course was that we wanted the students to feel and take ownership of the products they created and the tools that they used (Pinho-Lopes & Macedo, 2016). With this we mean that (1) they develop their model from scratch without help of high-level packages or software, that (2) they use the tool we teach them (e.g. git, code editor, unit testing, HPC cluster interaction) self-reliantly and naturally by the end of the course, that (3) they conduct the final project from start to finish employing all their learnt skills and tools, and that (4) they help each other through team work or via help on the group-chat. Note that taking ownership is a process and that we guide the students towards it throughout the course. Indeed, Prince & Felder (2006) state that in problem-based teaching it is important to provide at first a scaffolding for the students which is then gradually removed as they acquire the needed skills. We feel that by the end of the final project they indeed own their work as they created it from the ground up with tools they know how to employ. The research and project-oriented learning approach enabled these developments and confirmed in our case that active learning and involving students in research-like projects was a positive experience (Pinho-Lopes & Macedo, 2016).

There are certainly challenges with this course which need to be addressed to take it into the future. The student-feedback needs to be prompt such that students can profit reliably from the material covered and such that they are ready to absorb new knowledge building on previous lecture content. This involves much manual labour, and the help of the teaching assistant was invaluable in the 2022 edition of the course. Going forward we aim to make assignment correction and feedback preparation less time consuming, for this we will investigate some auto-grading options or self-grading parts of the assignment by the students

themselves. One idea is that students code-review amongst each other (another important software engineering skill) and thus provide peer-feedback.

Regarding the class size, of note is that in the 2022 edition of the course, the waiting list was about as large as the number of spaces in the course; thus, expanding the course could be considered. Thinking big, this type of class is, in our opinion, at the forefront of supercomputing courses whilst teaching of supercomputing skills to Master-level students is very limited. We would expect this material to be of interest to a wide cohort of specialised students. Such a course expansion would require an increase of the teaching staff together with streamlining the assignment corrections as outlined above. Irrespective of whether the course size remains small or is scaled up, certainly, the course will keep on evolving including both its content and the used tools as both will be developed further by the research and open-source communities.

Conclusion

We designed a new course at ETH Zurich to teach domain scientists to numerically solve partial differential equations on graphical processing units using a high-performance/supercomputing approach. The course fills a gap in ETH Zurich's curriculum and builds upon recent advances in programming languages and scientific software packages making it now possible to develop such codes without years of training. In addition to numerical code, we also teach project management skills and software-engineering tools. This allows the students to take ownership of all their scientific workflow. We pursue a research-oriented learning approach with the methods mirroring the ones we employ in our research and with end-of-course projects closely following a research workflow. The computational competencies that are taught align well with ETH's teaching strategy and the course is highly rated by students.

Acknowledgement

We would like to thank the Swiss National Supercomputing Centre (CSCS) for donating 4'000 node hours compute time on the Piz Daint supercomputer and the Swiss Geocomputing Centre at the University of Lausanne for granting students compute time on the octopus supercomputer. We appreciate the JupyterHub server and other resources provided by ETHZ. This research has been supported by the Swiss University Conference and the Swiss Council of Federal Institutes of Technology through the Platform for Advanced Scientific Computing (PASC) program and the Swiss National Supercomputing Centre (CSCS project ID c23). This course would not be possible without the excellent Julia programming language and the unparalleled package ecosystem around GPU computing (https://juliagpu.org/). Furthermore, the tools we used for creating the website and assignments are essential to this course (Literate.jl, Franklin.jl and in particular Tibeaut Lienart, IJulia.jl). We also appreciate the other open-source software which makes this course possible such as JupyterHub, VSCode, GNU/Linux, Matrix/Element and many others. We thank our excellent teaching assistant Alexander Mandt of 2022 and all the future teaching assistants of the courses to come. We thank two anonymous reviewers and the editors for their helpful comments which improved this manuscript.

Bibliography

Besard, T., Churavy, V., Edelman, A. & Sutter, B. D. (2019). Rapid software prototyping for heterogeneous and distributed platforms. *Advances in Engineering Software*, *132*, 29–46. https://doi.org/10.1016/j.advengsoft.2019.02.002

Besard, T., Foket, C. & De Sutter, B. (2019). Effective Extensible Programming: Unleashing Julia on GPUs. *IEEE Transactions on Parallel and Distributed Systems*, *30*(4), 827–841. https://doi.org/10.1109/TPDS.2018.2872064

Bezanson, J., Chen, J., Chung, B., Karpinski, S., Shah, V. B., Vitek, J. & Zoubritzky, L. (2018). Julia: Dynamism and performance reconciled by design. *Proceedings of the ACM on Programming Languages*, *2*(OOPSLA), 1–23. https://doi.org/10.1145/3276490

Cantalou, J., Hierold, C., Buchli, A. & Klinger, R. (2021). *Strategy and Development Plan 2021–2024*. https://ethz.ch/content/dam/ethz/main/eth-zurich/portraet/Strategie/ETH_SEP_21-24_EN_Web.pdf

Churavy, V., Godoy, W. F., Bauer, C., Ranocha, H., Schlottke-Lakemper, M., Räss, L., Blaschke, J., Giordano, M., Schnetter, E., Omlin, S., Vetter, J. S. & Edelman, A. (2022). *Bridging HPC Communities through the Julia Programming Language*. https://doi.org/10.48550/ARXIV.2211.02740

Dongarra, J. J. (2022). The evolution of mathematical software. *Communications of the ACM*, *65*(12), 66–72. https://doi.org/10.1145/3554977

Granger, B. E. & Pérez, F. (2021). Jupyter: Thinking and Storytelling With Code and Data. *Computing in Science & Engineering*, *23*(2), 7–14. https://doi.org/10.1109/MCSE.2021.3059263

Hanson, E. P. & Giordano, M. (2021). *Code, docs, and tests: What's in the General registry?* https://julialang.org/blog/2021/08/general-survey/

Healey, M. (2005). Linking Research and Teaching to Benefit Student Learning. *Journal of Geography in Higher Education*, *29*(2), 183–201. https://doi.org/10.1080/03098260500130387

Knuth, D. E. (1984). Literate Programming. *The Computer Journal*, *27*(2), 97–111. https://doi.org/10.1093/comjnl/27.2.97

Morra, G., Bozdag, E., Knepley, M., Räss, L. & Vesselinov, V. (2021). A Tectonic Shift in Analytics and Computing Is Coming. *Eos*, *102*. https://doi.org/10.1029/2021EO159258

Omlin, S. & Räss, L. (2022). *High-performance xPU Stencil Computations in Julia* (arXiv:2211.15634). arXiv. https://doi.org/10.48550/arXiv.2211.15634

Omlin, S., Räss, L. & Utkin, I. (2022). *Distributed Parallelization of xPU Stencil Computations in Julia* (arXiv:2211.15716). arXiv. https://doi.org/10.48550/arXiv.2211.15716

Pinho-Lopes, M. & Macedo, J. (2016). Project-based learning in Geotechnics: Cooperative versus collaborative teamwork. *European Journal of Engineering Education*, *41*(1), 70–90. https://doi.org/10.1080/03043797.2015.1056099

Prince, M. J. & Felder, R. M. (2006). Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases. *Journal of Engineering Education*, *95*(2), 123–138. https://doi.org/10.1002/j.2168-9830.2006.tb00884.x

Räss, L., Utkin, I., Duretz, T., Omlin, S. & Podladchikov, Y. Y. (2022). Assessing the robustness and scalability of the accelerated pseudo-transient method. *Geoscientific Model Development*, *15*(14), 5757–5786. https://doi.org/10.5194/gmd-15-5757-2022

Smith, S. (2018). Beyond software carpentry. *Proceedings of the International Workshop on Software Engineering for Science*, 32–39. https://doi.org/10.1145/3194747.3194749

Vuduc, R. & Choi, J. (2013). A Brief History and Introduction to GPGPU. In X. Shi, V. Kindratenko, & C. Yang (Eds.), *Modern Accelerator Technologies for Geographic Information Science* (pp. 9–23). Springer US. https://doi.org/10.1007/978-1-4614-8745-6_2

Wilson, G. (2006). Software Carpentry: Getting Scientists to Write Better Code by Making Them More Productive. *Computing in Science & Engineering*, *8*(6), 66–69. https://doi.org/10.1109/MCSE.2006.122