BEAST Lab: A Practical Course on Experimental Evaluation of Diverse Modern HPC Architectures and Accelerators

Amir Raoofy
Technical University of Munich
Leibniz Supercomputing Centre
amir.raoofy@lrz.de

Bengisu Elis
Technical University of Munich
bengisu.elis@tum.de

Vincent Bode
Technical University of Munich
vincent.bode@tum.de

Minh Thanh Chung
Ludwig Maximilian University of Munich
minh.thanh.chung@ifi.lmu.de

Sergej Breiter
Ludwig Maximilian University of Munich
sergej.breiter@nm.ifi.lmu.de

Maron Schlemon
German Aerospace Center
maron.schlemon@dlr.de

Dennis-Florian Herr
Technical University of Munich
deflo.herr@tum.de

Karl Fuerlinger
Ludwig Maximilian University of Munich
karl.fuerlinger@ifi.lmu.de

Martin Schulz
Technical University of Munich
Leibniz Supercomputing Centre
schulzm@in.tum.de

Josef Weidendorfer
Technical University of Munich
Leibniz Supercomputing Centre
josef.weidendorfer@lrz.de

ABSTRACT

Giving students a good understanding how micro-architectural effects impact achievable performance of HPC workloads is essential for their education. It enables them to find effective optimization strategies and to reason about sensible approaches towards better efficiency. This paper describes a lab course held in collaboration between LRZ, LMU, and TUM. The course was born with a dual motivation in mind: filling a gap in educating students to become HPC experts, as well as understanding the stability and usability of emerging HPC programming models for recent CPU and GPU architectures with the help of students. We describe the course structure used to achieve these goals, resources made available to attract students, and experiences and statistics from running the course for six semesters. We conclude with an assessment of how successfully the lab course met the initially set vision.

KEYWORDS

High Performance Computing, Computer Architecture, Accelerator Architectures, Computer Science Education

1 INTRODUCTION

As Tier-0 compute centers in Europe move towards exascale, the education of students and users alike is gaining significant importance in order to be prepared to utilize the potential of these systems. Various valuable training materials and courses have been developed, including broad community training initiatives in the US (e.g., efforts in ECP [13, 14]) and Europe (e.g., activities in PRACE [8] and European Union's Horizon 2020 research and innovation program). These programs mainly cover mainstream HPC programming models, frequently target specific domains (e.g., ModSim or AI) for researchers beyond computer science, and often focus on platform-specific tools for better compute facility usage. A more general education for computer scientists from the architectural point of view is missing, which is, despite recent training methods and approaches, often lacking even in advanced university curricula. Those often stay either at the theoretical level, or focus only on high-level programming aspects of HPC systems.

Based on our experience working with HPC end-users in our compute center, we consider getting practical experience, in particular with focus on the micro-architecture of modern HPC platforms and programming models, a key aspect in HPC education, especially for computer science students and future HPC experts. This also must include architectural details, like differing processing elements and complex memory hierarchies.

In 2020, we introduced a new *practical lab* course focusing on the experimental evaluation of HPC architectures, called the BEAST Lab, after the BEAST testbed at LRZ which is used in the lab (BEAST is short for Bavarian Energy, Architecture, and Software Testbed). It is primarily targeted at Computer Science and Computer Engineering (CS and CE) students and has a strong focus on micro-architecture of modern HPC architectures and systems.

The BEAST Lab shares the spirit of the recent training methods and programs [4, 5, 9, 12] in providing resource access to students. Our primary teaching approach and educational strategy is to enable students to acquire a deep understanding of micro-architectural aspects of HPC platforms through practical experience on modern HPC platforms and experimental evaluation of their micro-architectural properties. To reach this goal, BEAST[1], the *Bavarian Energy, Architecture, and Software Testbed*, which is the LRZ test environment with the latest computer technologies available, enabling research and exploration of new computer technologies and (HPC) systems, plays a central role. In addition, the BEAST Lab aims to have a broader impact by enhancing collaborative research, utilizing specialized hardware, and involving students in HPC hardware evaluation and programming, through a multi-institute structure organized by LRZ, thereby complementing lectures and seminars at co-organizing universities, the Technical University of Munich and the Ludwig Maximilian University of Munich.

The BEAST Lab uses a multitude of diverse hardware platforms available in BEAST with the intention of enabling insights by highlighting the architectural differences as well as the differences in programming models designed to target this diverse hardware in practice. Therefore, the course is structured with practical assignments and projects requiring minimal implementation effort, but focusing on experimental evaluations, that teach how to conduct detailed performance analysis and interpret them. This enables students to explore and understand the architectural differences through their measurements. The assignments cover programming experiments on compute-bound and memory-bound kernels on CPUs and GPUs using common HPC node-level programming models (OpenMP, CUDA, and HIP), instruction-level and memory parallelism, branch prediction, and NUMA effects on CPUs.

The projects cover the same type of topics and experiments but need more effort in programming, enabling multi-threading and various optimizations such as low-level manual vectorization. These projects target a selection of conventional and fairly new applications such as multigrid solver [20], matrix profile computation [17], and interpolation kernels [18].

Finally, to assess the success of the lab course and to ensure that the lab material stays aligned with the teaching goals, we establish a survey for students, where the participants are asked to evaluate whether the goals are met. This also helps the instructors to gradually improve the course over time. In addition to quality assurance, the surveys serve as feedback from students on their experience on various hardware, software, tools and programming models. We collected these surveys over four semesters and discussed the results in the paper, e.g., that students voted OpenMP offloading the most intuitive and easy-to-learn model for GPU programming.

Overall in this paper, we make the following contributions:

- We discuss the benefits of the BEAST Lab from the perspective of organizing parties as well as students.
- We describe the teaching approach, including the course structure, resources made available to students, assignments, projects, and grading scheme.
- We summarize lessons learned from organizing the BEAST Lab for six semesters and provide insights from student surveys on experiences with various hardware and programming models collected over four semesters.

[1]BEAST is a wordplay: the more systems are coming in, the more the testbed environment looks like a beast to tame to get results. The corresponding logo shows a lion head as beast, relating to the lions in the Bavarian coat of arms (cf. Figure 1).

The survey shows that in most cases the students could explain measurements on the specific architectures and that overall the BEAST Lab helped the majority of students to properly evaluate and better understand modern HPC architectures, making this a highly successful lab course that has become a permanent offering.

2 MOTIVATION

The BEAST Lab was incepted with two primary goals in mind: filling a practical gap in educating students to become HPC experts, as well as acquiring experience and understanding the stability and usability of emerging HPC programming models for recent CPU and GPU architectures with the help of students. In addition to these aspects, the BEAST Lab is also designed with a broader impact in mind to strengthen the collaborative research between academic institutes and the Leibniz supercomputing center in Munich.

Motivation for the Compute Center: Evaluation of modern HPC hardware technologies and emerging HPC programming models is a critical research area for the LRZ, especially in the era of the Cambrian explosion of computing and novel architectures [15]. This research helps LRZ to acquire practical experience on the various modern systems, technologies, and approachability of programming models as well as the maturity of software and toolchains, and gain expertise to provide users with suggestions, recommendations, and best practices on how to write sustainable parallel codes for future generation machines in computing centers. However, the extensive variety of modern architectures requires allocating lots of internal resources. The BEAST Lab helps to attract, educate, and engage new students in this research area (similar to [22]). We received more than 20 inquiries from students to continue research after participating in the BEAST Lab. Additionally, the BEAST Lab helps motivate research teams and PhD candidates to engage in the evaluation of modern HPC architectures. 30 researchers (mainly PhD candidates) from the co-organizing universities are conducting research on BEAST; 5 of them actively contribute to the development of the BEAST Lab.

Motivation for Educators: The CS and CE study programs at TUM and LMU provide lectures around computer architecture both on a basic level in the first year of Bachelor as well as on a deeper level as selective courses in Master tracks. The latter discusses concepts, approaches, and design choices of the micro-architecture, mostly covering CPUs on a theoretical level, such as pipelining, out-of-order execution, or the memory hierarchy. The BEAST Lab helps to fill the practical gap in CS students' education regarding complex concepts and intricacies in the micro-architecture of modern HPC processors and accelerators. It drives understanding to observe the performance differences of code variants in practice, backed by discussion on the micro-architectural properties responsible for the observed effects. Furthermore, the variety of architectures available in BEAST allows for insightful comparisons, highlighting different design choices.

Motivation for Students: For students, the lab course provides the following benefits: 1) access to rather expensive, modern, and often rare HPC hardware, 2) gaining practical hands-on experience on these modern systems, and 3) engaging in state-of-the-art HPC hardware and system software research after gaining experience in the BEAST Lab. Additionally, the BEAST Lab also features invited talks by mainstream HPC vendors through which students can hear a light vendor roadmap and an overview of cutting-edge hardware and software technologies. We also provide a tour of the LRZ site, including a visit to the infrastructure and the compute cube of SuperMUC-NG, offering the students a unique chance to visit a top-tier HPC system.

3 AVAILABLE INFRASTRUCTURE FOR LAB

To stay up to date with new architectures and components, to contribute to the research in shaping the future of HPC systems, as well as to educate future HPC experts, the LRZ has established the "Future Computing" program funded by the Bavarian State Government: the cornerstone of this program is the development of test environments with the latest computer technologies, the "Bavarian Energy, Architecture and Software Testbed" or BEAST for short (Figure 1).

The "Future Computing" Program: LRZ serves academic researchers from Bavaria and Germany, mostly running scientific simulation codes from natural sciences. The main goal of the "Future Computing" program is to match available future technologies with the demands of LRZ users, starting with a set of benchmarks that reflect the typical application mix running on current systems. These benchmarks are then evaluated on the various architectures in the BEAST system to get an understanding of future technology and to answer questions like, "Which systems provide the best acceleration and efficiency for user codes considering a given budget?". The resulting insights are of significant help for the procurement of upcoming systems. Another important goal is to check the claims of vendors about the stability and versatility of the provided software stack, including support for parallel programming models. Finally, BEAST helps enable the porting/adaptation of in-house system tools for new architectures.

Figure 1: BEAST Testbed system at LRZ.

3.1 BEAST Hardware

Our lab course uses BEAST as the main education resource and provides the students with access to the latest HPC processors and accelerators. Through the access and working on BEAST testbed, students get a chance to work on processors and accelerators with similar architecture and capabilities to the top systems in Top500 list [6], including Frontier, Fugaku, LUMI, and Leonardo. The BEAST testbed offers a variety of architectures from different vendors and exposes different instruction sets. However, in the lab course, the following systems are used intensively.

Intel Icelake + Volta 100: Each node of this type features a two-socket Intel Xeon Platinum 8360Y CPUs with an Icelake micro-architecture. 72 cores are distributed over the two sockets (in two NUMA domains), where each is equipped with one NVIDIA Tesla V100 GPU (to be moved to A100 GPUs in the upcoming semester). We are providing access to two nodes of this type to our students.

ARM Thunder X2 + Volta 100: The second system type consists of two nodes with Thunder X2 CN9980 ARM CPUs. Each has two sockets with 32 cores per socket and four-way hyperthreading. Two V100 NVIDIA GPUs are connected to each node.

AMD Rome + AMD MI100: The third system type features two nodes with AMD EPYC 7742 Rome CPUs of Zen2 micro-architecture, each containing 64 cores distributed on two sockets. Each node is equipped with two AMD MI100 GPUs (MI50 GPUs were used in the early stages of the lab course). In the upcoming semesters, we are moving towards using AMD Milan systems with the Zen3 micro-architecture and AMD MI200 GPUs.

Fujitsu A64FX:. The fourth system features Fujitsu A64FX nodes with 64-bit Arm architecture with 512-bit vector implementation of ARMv8 SVE SIMD instruction set extensions. Each node has 48 cores distributed on 4 NUMA domains and a total of 32 GB HBM memory. There is no simultaneous multi-threading support.

3.2 BEAST Software Environment

BEAST offers a similar environment, such as the production systems of LRZ, running SLES and a SLURM instance (the scheduler is, however, not currently used in the lab as we allocate dedicated machines). However, as BEAST is part of the LRZ research infrastructure, many researchers from Munich universities and internal LRZ users actively use the infrastructure, and consequently, the software environment and configuration are subject to change based on the needs of the researchers.

For this reason, we use Spack [11] to maintain a dedicated software stack for the Lab course (of course, this stack can be reused for other purposes as well). For instance, various compilers with different backends, as well as tools (e.g., LIKWID), are installed using Spack. Other system software and tools, such as device drivers, vendor compilers, and runtime libraries, are installed system-wide and regularly updated. We also maintain a system-wide instance of Data Center Data Base (DCDB) [16] to monitor the outbound power consumption of nodes gathered from PDUs, which is used in parts of the lab for experiments on energy efficiency.

4 THE LAB STRUCTURE AND TEACHING APPROACH

Focus and Topic Coverage: BEAST Lab gives students opportunities to work on modern computing architectures. In particular, the course helps students understand how micro-architecture technologies affect applications' performance. The involved assignments and projects in BEAST Lab cover various aspects, such as thread-level parallelism, instruction-level parallelism, vectorization, memory access patterns, caches-memory hierarchy, NUMA effects, branch prediction, and power consumption. These are integrated through the general and experimental questions of assignments. Each assignment is linked to a micro-benchmark referring to the aspects that can impact overall performance. Students can perform experiments on CPU and GPU architectures.

Teaching Approach: Our teaching approach is primarily based on comparing architectural effects among different architectures. Students are assigned to work in a group to complete the assignments and projects. In each assignment, the groups work on a specific topic and conduct experiments on BEAST hardware. Before conducting experiments, they may have a conjecture about the results. After experiments, each group of students summarizes the results in a report and analyzes the performance evaluation of different architectures. Students are expected to understand architectural knobs, tuning, and optimization potentials affecting performance. We do not require students to put extra effort into programming from scratch. Instead, we provide the reference codes for the tasks and instructions for conducting experiments on BEAST platforms. With this approach, most of the time can be spent on investigating different optimization ideas and analyzing performance results.

Another aspect to consider for reducing the programming effort on the student side is the programming models. These models should make it easy to port from a sequential program to a parallel version and port to different platforms and architectures in BEAST. Therefore, we rely on OpenMP as the primary programming model for teaching as it is almost the only vendor-neutral option supporting all the deployment and acceleration in all BEAST platforms, including heterogeneous programming on GPUs and CPUs. Other advanced and specialized programming models, such as CUDA, or HIP, are also introduced to students. However, we consider these as bonuses if students try to make an effort.

Lecture Organization & Assignment Structure. The lab is structured into two main parts: In Part 1, students hear lectures on micro-architectural topics and work on weekly assignments. In Part 2, they work on projects with more complex coding tasks. Part 1 is organized with the weekly lab sessions as follows: sessions begin with a short lecture on a specific topic related to the upcoming assignment. This is followed by student presentations covering their experience and results from the previous assignment. In the end, we introduce the next assignment. In Part 2, the sessions mainly include an introduction to the project, open group progress discussions, and project presentations.

In the first lecture, we introduce the course organization and provide supporting materials to the students. This includes an overview of the available hardware and their architecture, software environment, and compilers in BEAST. Further weekly lectures cover an introduction to OpenMP, GPU architecture and OpenMP target offloading, memory hierarchy of multi-core CPUs, pipelining and branch prediction, and tools for tracing and profiling.

Overview of assignments: Following the first introductory lecture, BEAST LAB begins with a warm-up assignment providing instructions about accessing the system remotely and loading related libraries and compilers to familiarize students with the usage of our platform. After the warm-up assignment, seven assignments cover the mentioned topics. Each assignment is divided into smaller

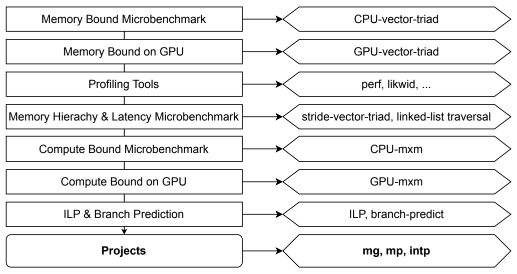

Figure 2: Overview of assignments & projects in BEAST Lab.

parts with specific tasks and questions, supposed to incrementally improve the understanding of the topics. Students are asked to explain the performance of certain codes and modifications, supported by appropriate visualizations. The assignment overview and order are shown in Figure 2. Their content is described as follows.

(1) **Vector Triad on CPU.** This assignment uses a vector triad to show how the available bandwidth of different levels in the memory hierarchy affects performance in memory-bound applications in dependence on their arithmetic intensity. The micro-benchmark is analyzed in sequential and parallel execution. Topics covered include compiler auto-vectorization, caching, thread pinning and OpenMP loop scheduling effects, NUMA effects and first touch policy.
(2) **Vector Triad on GPU.** Assignment 2 introduces students to OpenMP offloading in the context of GPUs. The tasks require analysis and evaluation of the GPU architecture with a focus on the memory hierarchy, i.e., the OpenMP thread and team scheduling on GPU processing elements, as well as the importance of memory coalescing and data transfers between host and device, which are explored.
(3) **Profiling Tools for CPU and GPU.** This assignment introduces students to various profiling tools for CPUs (Perf, Likwid, PAPI) and GPUs (nsys, ncu, rocprof, THAPI). Students learn to identify performance bottlenecks using code instrumentation and tools to measure performance events.
(4) **Memory Hierarchy and Memory Access Latency.** We use vector triad (a stride variant), a linked-list traversal, and a ping-pong code to introduce detailed performance aspects of the memory hierarchy and parallelism, such as prefetching, cache line efficiency, NUMA effects, cache coherence, and core-to-core latency.
(5) **Matrix Multiplication on CPU.** A dense matrix multiplication is used as a compute-bound micro-benchmark. We focus on memory access patterns, loop scheduling with OpenMP parallelization, cache blocking, (auto-)vectorization, and compiler optimizations. Additionally, we encourage the students to use roofline model for each architecture to further analyze their experiments on BEAST.
(6) **Matrix Multiplication on GPU.** Assignment 6 covers performance aspects of data transfers between host and device, matrix multiplication cache blocking and thread scheduling on GPUs, and introduces the CUDA programming model. Besides, students can measure energy efficiency using DCDB.

(7) **Pipelining and Branch Prediction.** This assignment introduces codes to measure properties of key aspects in modern pipelined microprocessor architectures, such as instruction latency, throughput, and branch prediction. Branch prediction capacities are analyzed via different types, i.e., conditional jumps, indirect jumps, and returns.

Final Projects: At the end of the lab course, student groups work on projects that put their knowledge and experience to the test. These projects are structured similarly to the weekly assignments but include more implementation workload around the topics discussed in the lab. Groups bid on the available platforms for their project and further implement specified applications targeting CPUs or GPUs using OpenMP, and optionally CUDA and HIP. They further and evaluate their implementations according to the provided instructions and report the results in a final presentation.

The content of these projects is described as follows:

(1) Multigrid is a well-established iterative approach for solving linear equations mainly associated with partial differential equations. This approach leverages a hierarchy of grids to accelerate the convergence of the iterative solver. In this project, students are provided with a sequential implementation of a full multi-grid solver that is based on Jacobi smoother, and it includes restriction and prolongation steps, as well as the typical so-called V and W cycle schemes. Groups implement vectorization, parallelization, and GPU offloading for target codes and investigate performance bottlenecks and efficiency.
(2) Automatic optimization and vectorization enabled by OpenMP directives or compiler flags may not improve performance in complex cases with complex access patterns. In such cases, manual implementation of SIMD is often required, potentially necessitating restructuring the algorithms to tap into parallel processing capabilities. In this project, students extend the implementation of a sinc interpolation function [18] with OpenMP as well as manual SIMD and evaluate their solutions on several CPU platforms in BEAST.
(3) In the last project, students work on a time series mining application, specifically matrix profile computation [17]. We provide students with the source code for the serial implementation of matrix profile computation. Students implement it by enabling efficient multi-threading, vectorization, and offloading and further study it in their selected target platforms in BEAST.

Teaching Support Structure, Student Feedback, and Evaluation Scheme: In terms of teaching support and student evaluation, we require students to work in groups with Git for their collaboration, assignment/project submission. For the discussion among students' groups, a common Zulip channel is created. Tutors also get involved in this channel to support students. Depending on the content of each assignment and project, deadlines can be in one or two weeks. After each assignment/project, a few groups are selected to present the reports of their performance results. Following that, we can question and discuss in detail how students understand the results and performance aspects corresponding to each micro-architecture.

5 CHALLENGES

Organizing the practical course with the above-mentioned structure comes with various organizational and technical challenges. These challenges include:

(1) Structuring suitable assignments and projects that are aligned with the primary goals of the lab course,
(2) Regular cross-institution coordination and planning,
(3) Coping with the gradual changes in hardware and software platforms, which requires additional effort in keeping a dedicated software environment for the assignments and projects to rely upon while also maintaining an ever-changing research software stack and environment,
(4) Better usage of scheduler for student jobs (overlapping allocations).

6 EVALUATIONS OF BEAST LAB SUCCESS

Over the course of four semesters between 2021 and 2023, we have asked students to evaluate their experiences in the BEAST Lab. While the two universities conduct generic course evaluations, we provided a special feedback form in addition that focuses on the syllabus and learning aspects specific to the BEAST Lab. The types of questions include rating on a scale, selecting from categories, and providing free-text answers. In total, we have received 60 responses, of which 12 are from students at the bachelor's level (all enrolled in computer science) and 48 at the master's level (in computer science and adjacent disciplines, such as computational science & engineering, robotics, cognition & intelligence, or games engineering).

6.1 Student Experience

On average, our students report having 4.2 years of programming experience, which is slightly above their average length of enrollment. Many students have previously taken courses in the subject field, with a majority (58%) having attended the lectures *Parallel Programming or Programming of Supercomputers*, 22% having attended the lecture *Advanced Computer Architecture*, 8% having attended a lab related to High Performance Computing or GPUs, and 12% having attended no prerequisite courses at all. Despite this, many students report having only little experience with both OpenMP and OpenMP Offloading. On a 1-5 scale (5 very experienced and 1 novice), the average student self-reports an experience of 2.4 for OpenMP and an experience of 1.3 for OpenMP Offloading, with 38% and 79% of students having had no respective prior experience at all, raising the question of how effective the adjacent courses teach OpenMP as a parallel programming technique.

6.2 Merits of OpenMP

OpenMP is by far the most popular parallelization technology in the lab, with almost two-thirds of students preferring its directive-based approach to a computing-kernel paradigm (25%) such as the one found in CUDA. Coming from a background of comparatively little experience, students find that OpenMP is reasonably easy to learn rated as 3.5/5, with OpenMP Offloading being somewhat more difficult at 2.8/5 on a scale of 1 very easy 5 very hard. This is likely due to the additional requirement of having to handle data movement. OpenMP loop constructs were rated as the most intuitive API for parallelization on CPU, followed by intrinsics

Table 1: Students' Opinions of Implementing Parallel Code on CPU and GPU on a 1-5 Scale.

	Debugging	Profiling	Difference
CPU	2.7	3.0	**0.3**
GPU	1.9	2.4	**0.5**
Difference	**-0.8**	**-0.6**	

Note: Generally, students prefer pro-filing over debugging and working on CPU over working on GPU.

before OpenMP SIMD. On GPUs, twice as many students preferred writing OpenMP Offloading code overwriting CUDA code, with barely any students preferring HIP. Figure 3 shows the top reasons that students like OpenMP and the most often cited criticisms. As expected, the ease of implementation and the learning curve are seen as big pluses for using OpenMP for on-node parallelization. However, students often find it difficult to understand what their code or the OpenMP library is doing at run time, which is reinforced by the observation that students also find it difficult to debug their parallel programs. To this day, despite the maturity of OpenMP and shared memory parallelism in general, these still appear to be common pain points for parallel program developers.

6.3 CPU and GPU

As part of the lab course, students work with both CPU and GPU technology. Unsurprisingly, students preferred working on CPU rather than GPU (with GPUs scoring around 20% lower in the ease of debugging criterion and 15% lower in the ease of profiling criterion) and found that finding bottlenecks was generally easier than debugging a program (Table 1). With a rating of only 1.9/5, debugging on GPU was by far the most painful for students. This is likely due to the fact that GPU debugging tools are not nearly as prevalent as their CPU counterparts. The easiest task was profiling on the CPU, but with a rating of 3.0/5 it can hardly be said that this is a seamless experience.

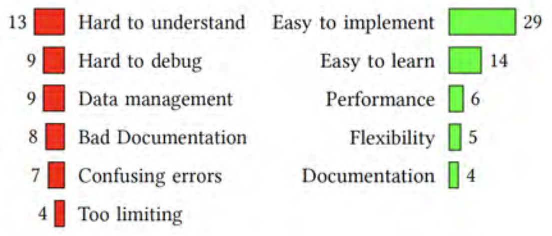

Figure 3: Advantages and disadvantages of using OpenMP according to student opinion. The biggest appeals of using OpenMP for students are how easy and fast it is to parallelize code and the learning curve. However, students often find it difficult to understand what their code or the OpenMP run-time is doing.

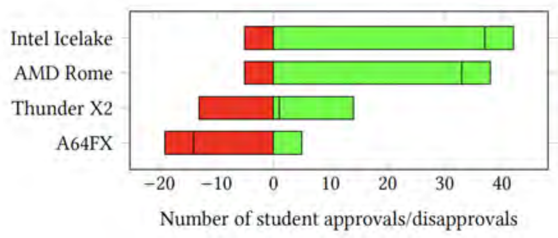

Figure 4: Architecture popularity among students. The x86-64-based systems (Icelake and Rome) outclass the ARM-based systems (A64FX and X2), with the A64FX, in particular, being the least popular. being the least popular.

6.4 Profilers

Students typically spend around 54% of their time parallelizing their application until they reach their first correct implementation (which includes debugging), and around 42% of their time in optimization (including profiling). In general, students find profilers reasonably useful when attempting to optimize code (3.5/5). The most popular profiler was Linux's Perf (3.4/5), with Likwid (3.1/5) and the GPU profilers (2.9/5) following suit.

6.5 Architectures

Throughout the lab, students work with four types of machines available in the BEAST cluster. One of the motivations for the LRZ to conduct the lab is to learn about the merits of different systems from a user's perspective, with students serving as stand-ins for actual scientific users. According to Figure 4, the students have a very clear opinion on which systems they like. The two x86-64-based systems, Intel Icelake and AMD Rome, come out far ahead of the ARM-based systems, Thunder X2 and the A64FX. Icelake wins out slightly over Rome, though this may be attributed to the system being slightly more recent, but both systems have a good approval rating of 62% and 55% of participants, respectively. The opinion on the Thunder X2 machines is neutral, with approximately equal counts of approval and disapproval leading to an approval rating

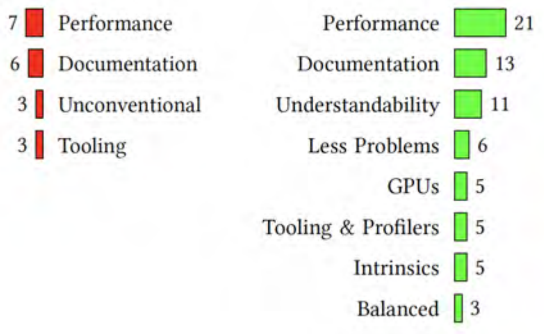

Figure 5: Top reasons why students like or dislike architectures found in BEAST. The most important factors for students are performance and available documentation.

of 2%. The biggest loser in terms of student opinion is clearly the A64FX, with a final approval rating of -23%, which is particularly unfortunate because the BEAST cluster has more A64FX machines available than any other type, which means crowding should have been less of an issue.

Figure 5 gives us clues as to why the approval ratings are the way they are. The top reason why students liked or disliked architecture was the performance they were able to achieve. For most students, this weighs in favor of x86-64 and against the A64FX, likely due to its unusual architecture. Documentation was the second most important factor, both in favor and against systems, with students remarking on the good documentation available for Icelake machines in particular. Furthermore, the ability to understand what was happening in hardware was important for students. In the end, when asked whether they think that they can choose which architecture to use given an algorithm, students were fairly confident (3.2/5) that they would be able to make a good decision. When asked whether they think they will be able to explain performance results, students were also fairly confident (3.3/5) that they would be able to explain the cause. Additionally, when asked what other architectures they would have been interested in, popular candidates included RISC-V systems, FPGAs, and AI-specific architectures.

6.6 Lab Evaluation

Last but not least, we asked students to evaluate the course itself. In general, students describe the difficulty of understanding the algorithms for the assignments as medium (3-5 hours) and for the projects as difficult (approximately 2 days). Some students felt that they could have benefited from more in-depth introductions to the algorithms, and it was found that self-guided research was the primary source of information for understanding concepts and developing solutions ahead of the provided course material and API/standards documentation. Further, students felt that the lab could be improved by additional feedback to students as well as sample solutions, indicating the desire to reduce the open-endedness slightly in favor of a more rigidly structured course. Overall, many participants thought that the course matched their expectations fairly well (3.7/5) and that the exercises helped them understand modern computer architecture (3.8/5). Many were also interested in further student work such as a thesis or guided research.

Throughout the semesters of the BEAST Lab, we gained a lot of experience with running the lab course and the surrounding infrastructure. We believe this course is unique among the offerings at our university, and based on student engagement in class and the provided feedback, we know that students have the opportunity to learn practical aspects of high performance computing not covered in the theoretical courses. We therefore believe that the BEAST Lab is a win for both the students taking the course and the staff supporting it.

7 RELATED WORK

Various valuable training programs, materials, and courses have been developed by the HPC community. In order to locate the scope of the BEAST Lab, we look at recent programs from three main perspectives: 1) the primary educational purpose, 2) the target audience, and 3) the teaching approach.

The Supercomputing Institute internship program at Los Alamos National Laboratory [22] provides a basis in cluster computing for undergraduate and graduate students, offering an educational experience to students and serves as an important recruitment tool for HPC field. RWTH Aachen University offers a software lab [19] to students, in which the students develop parallel code using modern tools while focusing on High-Performance Computing foundations and parallel programming skills. This lab also introduces a self-paced learning approach where status surveys, developer diaries, and group competitions are used to motivate and track students' progress. The German National Supercomputing Center HLRS has expanded its academic training programs to include courses for students, teachers, and professionals in response to the increasing demand from broader application domains and relevance of high-performance computing and simulations beyond academia[7]. Kokkos is presented as a generic parallel programming model suitable for the education of a broader audience, including academia, in the recent work of Sandia National Laboratory [3], where they introduce the best practices obtained from giving virtual classes on Kokkos. Michigan State University uses a flipped classroom model and a "hands-on" approach in teaching parallel programming to undergraduates in targeting STEM fields [4]. College of Meteorologic Oceanography at National University of Defense Technology in China offers a parallel computing course [2] tailored for atmospheric science majors, addressing challenges faced by non-computer science students in understanding parallel scalability. Texas A&M High Performance Research Computing (HPRC) [1] explored educational approaches in response to COVID-19 pandemic using virtual sessions and peer-learning environment.

Overall, in most cases, the existing programs focus on introducing the basics or mainstream HPC programming models and tools. Also, existing programs mainly target undergraduate or graduate student education or consider interdisciplinary domains (e.g., ModSim or AI) and, therefore, target a broad community of researchers beyond the computer science discipline.

The educational challenge of teaching high-performance computing in the face of rapid heterogeneous hardware innovation and adoption renders parts of textbooks obsolete [10]. Reed College uses a diverse heterogeneous hardware and software environment for computer science majors. Artificial Intelligence National Laboratory of Hungary, in collaboration with NVIDIA Deep Learning Institute, discusses challenges in accelerated heterogeneous parallel computing and deep learning education and presents the structure of their Instructor-led Workshops [12]. Magic Castle [9] of Jülich Supercomputing Centre in Germany creates the supercomputer experience in public or private Clouds enabling scalable HPC training through provisioning of virtual supporting infrastructures. Indiana University uses Jetstream National Science Foundation Cloud infrastructure [5], to provide practical HPC training experiences for both HPC administrators and users, covering concepts from basic command-line usage to advanced cluster management. FreeCompilerCamp.org of Lawrence Livermore National Laboratory [21] is an open online platform designed to train researchers in developing OpenMP compilers primarily to address the lack of training resources for researchers who are involved in the compiler and language development around OpenMP. Another worth-mentioning work is on teaching methods and hardware platforms used by

Purdue Research Computing for educating HPC system administrators [23]. Overall, we can conclude that providing resources to users for training is a central piece of the teaching method in most of the existing programs. Also, looking at ECP and PRACE education programs, we conclude that existing programs often target platform-specific tools and optimizations with the intention of better utilization of specific computing facilities.

BEAST Lab shares the spirit of recent training methods and programs in providing resource access to students and extends the scope of the micro-architecture of modern HPC platforms and programming models.

8 LESSON LEARNED, SUCCESS STORIES, AND DISCUSSIONS

Here we summarize a few of the lessons we learned from running the lab:

- most students registering for the lab are interested in recent technology. Getting access to expensive data center hardware motivates them to invest a lot of time in assignments.
- often it is difficult for students to come up with explanations for observed effects. Weekly meetings with a good mix of background information and in-depth discussion of detailed explanations are important for understanding.
- students always work more before deadlines. A job scheduler would not allow students to keep up with deadlines, so measurements may come from congested and overloaded systems. It is important to split up student groups to work on different hardware to reduce this effect.
- next to simple micro-benchmarks (Part 1), working on more complex code is important to see the bigger picture on the challenges of well-tuned HPC code. Examples: just tuning one kernel is not helpful (Amdahl's law will kick in early); on GPU it is important to keep data structures as long as possible on the accelerator.

We can identify two types of success stories. First, we see that Part 1 of the lab (discussing results of micro-benchmarks) helps students to get good results on optimizing real-world codes (Part 2 of the lab). Second, a lot of students come back for student work (bachelor's/master's theses) around BEAST. It is also nice to hear from nearside researchers that students cite their participation in the BEAST lab when searching for final theses. Finally, it is nice to have a colleague helping advise the BEAST lab now who joined the lab as a student before.

However, from the perspective of LRZ, it is important to discuss whether specific goals could be achieved. Especially valuable are experiences about (1) the maturity of software stacks, (2) the ease of getting into parallel programming with a given API/framework, and (3) whether programming models allow users to get the expected performance. In regards to (1), we wanted to understand the maturity of support for ARM. It was good to see that students had no issues with ARM compilers; the ecosystem around ARM seems to be so well supported nowadays that switching from x86 to ARM as host architecture for large HPC systems is not any risk. In regards to (2), we wanted to know if it is reasonable to start with OpenMP GPU support to get users into GPU programming. Experience from the lab shows definitely that it is beneficial for users to have an easy start. Once they see their code running on GPUs, it is relatively easy to dig deeper. However, the latter is required to get reasonable performance (3). Results from OpenMP Offloading often were disappointing.

9 CONCLUSION AND OUTLOOK

The BEAST Lab is a success story for all participating institutions, the universities, and the data center. While it is known among students to be challenging, we see students acquiring a lot of insights about CPU and GPU architectures, enabling them to come up with significant speedups for given HPC application codes. The lab successfully attracts good students. Furthermore, the "Future Computing" program benefits from the lab. First, students ask to do their Bachelor/Master theses with LRZ. Second, a lot of valuable experience is gathered with the help of the lab: it is helpful to know that, e.g., migrating users to a new ISA or to GPUs for the next leadership system can be done without much risk. And the experience becomes a guide for what kind of user training is required for successful migration to the next system.

REFERENCES

[1] Joseph Bungo and Daniel Wong. 2021. Bringing GPU Accelerated Computing and Deep Learning to the Classroom. *The Journal of Computational Science Education* 12 (Feb. 2021), 21–21. Issue 2.

[2] Juan Chen, Brett A. Becker, Youwen Ouyang, and Li Shen. 2021. What Influences Students' Understanding of Scalability Issues in Parallel Computing? *The Journal of Computational Science Education* 12 (Feb. 2021), 58–65. Issue 2.

[3] Jan Ciesko, David Poliakoff, Daisy S. Hollman, Christian C. Trott, and Damien Lebrun-Grandié. 2020. Towards Generic Parallel Programming in Computer Science Education with Kokkos. In *2020 IEEE/ACM Workshop on Education for High-Performance Computing (EduHPC)*. 35–42.

[4] Dirk Colbry. 2021. The Design of a Practical Flipped Classroom Model for Teaching Parallel Programming to Undergraduates. *The Journal of Computational Science Education* 12 (Feb. 2021), 41–45. Issue 2.

[5] Eric Coulter, Richard Knepper, and Jeremy Fischer. 2019. Programmable Education Infrastructure: Cloud resources as HPC Education Environments. *The Journal of Computational Science Education* 10 (Jan. 2019), 107–107. Issue 1.

[6] Jack Dongarra and Piotr Luszczek. 2011. *TOP500*. Springer US, Boston, MA, 2055–2057. https://doi.org/10.1007/978-0-387-09766-4_157

[7] Tibor Döpper, Bärbel Große-Wöhrmann, Doris Lindner, Darko Milakovic, et al. 2021. Expanding HLRS Academic HPC Simulation Training Programs to More Target Groups. *The Journal of Computational Science Education* 12 (Dec. 2021), 13–26. Issue 3.

[8] PRACE Training Events. 2023. Partnership for Advanced Computing in Europe. https://prace-ri.eu/training-support/training/. Accessed: 2023-08-10.

[9] Félix-Antoine Fortin and Alan Ó Cais. 2022. Magic Castle —Enabling Scalable HPC Training through Scalable Supporting Infrastructures. *The Journal of Computational Science Education* 13 (April 2022), 21–22. Issue 1.

[10] Eitan Frachtenberg. 2021. Experience and Practice Teaching an Undergraduate Course on Diverse Heterogeneous Architectures. In *2021 IEEE/ACM Ninth Workshop on Education for High Performance Computing (EduHPC)*. 1–8.

[11] Todd Gamblin, Matthew LeGendre, Michael R. Collette, Gregory L. Lee, Adam Moody, Bronis R. de Supinski, and Scott Futral. 2015. The Spack package manager: bringing order to HPC software chaos. In *SC '15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12. https://doi.org/10.1145/2807591.2807623

[12] Balint Gyires-Toth, Icsl Oz, and Joe Bungo. 2023. Teaching Accelerated Computing and Deep Learning at a Large-Scale with the NVIDIA Deep Learning Institute. *The Journal of Computational Science Education* 14 (July 2023), 23–30. Issue 1.

[13] Yun (Helen) He and Rebecca Hartman-Baker. 2022. Best Practices for NERSC Training. *The Journal of Computational Science Education* 13 (April 2022), 23–26. Issue 1.

[14] O. Marques and A. Barker. 2020. Training Efforts in the Exascale Computing Project. *Computing in Science &; Engineering* 22, 05 (2020), 103–107.

[15] Satoshi Matsuoka. 2018. Cambrian Explosion of Computing and Big Data in the Post-Moore Era. In *Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing (HPDC '18)*. Association for Computing Machinery, New York, NY, USA, 105. https://doi.org/10.1145/3208040.3225055

[16] Alessio Netti et al. 2019. From Facility to Application Sensor Data: Modular, Continuous and Holistic Monitoring with DCDB. In *Supercomputing 2019 (SC'19)*.

[17] Amir Raoofy, Roman Karlstetter, Dai Yang, Carsten Trinitis, and Martin Schulz. 2020. Time Series Mining at Petascale Performance. In *High Performance Computing: 35th International Conference, ISC High Performance 2020, Frankfurt/Main, Germany, June 22–25, 2020, Proceedings*. Springer-Verlag, Berlin, Heidelberg, 104–123.

[18] Maron Schlemon, Martin Schulz, and Rolf Scheiber. 2022. Resource-Constrained Optimizations For Synthetic Aperture Radar On-Board Image Processing. In *2022 IEEE High Performance Extreme Computing Conference (HPEC)*. 1–8.

[19] Christian Terboven, Julian Miller, Sandra Wienke, and Matthias S. Müller. 2020. Self-paced Learning in HPC Lab Courses. *The Journal of Computational Science Education* 11 (Jan. 2020), 61–67. Issue 1.

[20] Ulrich Trottenberg, Cornelis W. Oosterlee, and Anton Schüller. 2001. *Multigrid*. Texts in Applied Mathematics. Bd., Vol. 33. Academic Press, San Diego [u.a.]. With contributions by A. Brandt, P. Oswald and K. Stüben.

[21] Anjia Wang, Alok Mishra, Chunhua Liao, Yonghong Yan, and Barbara Chapman. 2020. FreeCompilerCamp.org: Training for OpenMP Compiler Development from Cloud. *The Journal of Computational Science Education* 11 (Jan. 2020), 53–60. Issue 1.

[22] J. Lowell Wofford and Cory Lueninghoener. 2020. The Supercomputer Institute: A Systems-Focused Approach to HPC Training and Education. *The Journal of Computational Science Education* 11 (Jan. 2020), 73–80. Issue 1.

[23] Alex Younts and Stephen Lien Harrell. 2020. Teaching HPC Systems Administrators. *The Journal of Computational Science Education* 11 (Jan. 2020), 100–105. Issue 1.