Lightning Talks of EduHPC 2021

Henry A. Gabb[1], Alexandru Nicolau[2], Satish Puri[3], Michael D. Shah[4], Rahul Toppur[4], Neftali Watkinson[5], Weijia Xu[6], Hui Zhang[7]
[1] Intel Corporation, USA
[2] Donald Bren School of Information and Computer Sciences, University of California, Irvine, USA
[3] Department of Computer Science, Marquette University, USA
[4] Khoury College of Computer Sciences, Northeastern University, USA
[5] Department of Computer Science and Engineering, University of California, Riverside, USA
[6] Texas Advanced Computing Center, The University of Texas at Austin, USA
[7] Department of Computer Science, University of Louisville, USA

***Abstract* – The EduHPC lightning talks are an opportunity for educators to discuss early results. These talks are compiled into this paper. The EduHPC 2021 lightning talks cover diverse pedagogical topics in parallel and distributed computing: teaching instruction-level parallelism in the context of computer organization, parallelizing cipher algorithms using single-instruction multiple threads, teaching edge computing to undergraduates, and deploying education-as-a-service on HPC resources.**

***Index Terms* – computer science education, high performance computing education, education tools, parallel and distributed computing curriculum**

I. Introduction

High-performance computing (HPC) used to be confined to supercomputing practitioners solving large computational problems. Consequently, it was considered a graduate-level topic. Today, multiprocessor systems are ubiquitous. Everything from the most powerful supercomputers, servers, and workstations to laptops and mobile devices contain multicore processors. Heterogeneous systems that contain a mix of general-purpose processors, integrated and discrete graphics processors, and other specialized processors are also becoming more common. Because every developer has access to multiprocessor systems, it is imperative that undergraduates are taught parallel and distributed computing (PDC) concepts as early as possible.

It is also important that educators have access to the latest pedagogical research on effectively teaching PDC, course materials to build PDC courses, and computing resources to support PDC courses. The Workshop on Education for High-Performance Computing (EduHPC) has become to primary venue to exchange ideas to improve PDC education. The EduHPC lightning talks provide an opportunity for PDC educators to present their preliminary work. The four lightning talks from EduHPC 2021 touch a range of topics:

1. Teaching single instruction, multiple data (SIMD) parallelism using the Intel vector intrinsic functions is presented in the context of a computer organization course. Students are also introduced to heterogeneous parallel programming using image manipulation coding exercises in NVIDIA CUDA.
2. Four introductory programming modules are described for teaching single instruction multiple threads (SIMT) parallelism using NVIDIA CUDA. Students are introduced to heterogeneous parallel programming concepts using cipher cracking and image steganography coding exercises.
3. Edge computing is an increasingly important PDC topic that touches several fields of study: artificial intelligence, cloud computing, embedded systems, Internet of Things, computer networks, and data ethics and privacy. Experiences teaching undergraduates the design principles of edge-based applications are discussed.
4. Teaching PDC topics increasingly requires HPC resources that are not designed to support education, which increases the burden on instructors. Experiences and practices using HPC resources at the Texas Advanced Computing Center (TACC) are discussed. In particular, a web-based application framework that improves HPC accessibility for instructors and their students and delivers education-as-a-service on TACC systems.

Section II describes the SIMD talk, Section III describes SIMT talk, Section IV describes the edge computing talk, and Section V describes education-as-a-service talk.

II. Teaching SIMD Instructions using Intel Intrinsics in a Computer Organization Course

By: Satish Puri

This lightning talk is based on experience of teaching Computer Organization and Design class at Marquette University for second year undergraduate students in Computer Science. The class follows the "Computer Organization and Design" book written by Patterson and Hennessey [1]. Topics covered are ARM assembly programming, logic operations, integer and floating-point representation, memory hierarchy,

multi-cores and graphics processing units. Vector instructions and Single Instruction Multiple Data (SIMD) are covered after the chapter on ARM instruction set architecture. First, I will mention the context and motivation.

Third chapter of the course book is "Arithmetic for Computers." This chapter has content on subword parallelism. The book chapter shows matrix multiplication functions with and without using Intel Intrinsics [2]. In the twelve pages of the book chapter, ARM and x86 SIMD extensions are covered. Using Advanced Vector Extensions (AVX) in x86 instruction set, up to 3.85X speedup is mentioned for matrix multiplication compared to sequential version in C. The book did not choose ARM Neon instructions to teach subword parallelism through programming example. This motivated the use of x86 extensions in my course module. Rather than directly teaching SIMD parallelism for matrix multiplication, simpler examples were created for teaching.

Motivation: In the book, subword parallelism appears towards the end of the chapter (subsections 3.6 to 3.8). The book chapter does not provide a complete program, only a function is given to illustrate subword parallelism. So, the motivation was to complement the existing material with additional contents like lecture slides, programming labs, quizzes and homework assignments targeted towards teaching subword parallelism in more detail. Additional content on vector data types, SIMD registers, intrinsic functions and memory alignment was developed.

In this class, the Single Instruction Multiple Data (SIMD) concept is illustrated by showing scalar and vector assembly pseudocodes to do vector addition where two input arrays are added element-wise to produce an output array. Embarrassingly parallel tasks like image manipulation are described to motivate SIMD processing. SIMD functions are used in C code first and then assembly code produced by a compiler is shown. First, Intel Intrinsics were covered, and then, towards the end of the course, GPU architecture was covered with a simple CUDA program.

Intrinsics are C functions to perform data movement, logic, and arithmetic computations. An intrinsic function is a function available for use in a given programming language whose implementation is handled specially by the compiler. Intrinsics get translated to vector instructions by compiler. There are a few benefits of starting with intrinsic functions rather than vector instructions directly. First of all, writing a program using C functions is easier. ARM instruction set was taught in assembly level following the format of the book. So, programming exercises for SIMD instructions based on x86 instruction set architecture would have been difficult. Therefore, rather than teaching second (x86) assembly language, teaching intrinsics was a simpler approach. To write a full-fledged program, it is easier to use C functions instead of vector instructions in assembly. Basics of C language were covered in a class and a lab before this module. Examples of Advanced Vector Extensions (AVX) were shown in class. Programming assignments and laboratory assignments were based on AVX as well.

The lectures were separated into two distinct parts:

- Part 1
  - Conceptual ideas on subword parallelism and SIMD instructions.
  - A few intrinsic functions to load, store and compute.
- Part 2
  - Loop transformation to show how to apply intrinsics in a for-loop. An example was shown to demonstrate loop unrolling technique. The example showed how unrolling four iterations of a for-loop makes it easy to apply intrinsic function for double precision floating point type where width of the register is 256 bits.
  - Aligned memory allocation.
  - Demonstration on Intel processors.
  - Quiz, lab problems, and homework assignments.

The following table shows one-to-one mapping between a data type in C (scalar) and a corresponding data type in AVX (vector) that was covered in class.

C data type	AVX data type
int	__m256i
float	__m256
double	__m256d

The following table shows a few AVX intrinsic functions and the corresponding assembly instructions covered in the class.

AVX intrinsic function	Assembly instruction
_mm256_load_pd	vmovapd
_mm256_store_pd	vmovapd
_mm256_mul_pd	mvulpd
_mm256_broadcast_sd	vbroadcastsd
_mm256_set_pd	initialization

The following code illustrates the intrinsic functions to load, store, and compute operations on array elements.

```
double *arr = {1.0, 2.0, 3.0, 4.0,
               5.0, 6.0, 7.0, 8.0};

// Load first four numbers from array arr
__m256d a0 = _mm256_load_pd(arr);

// Load last four numbers from array arr
__m256d a4 = _mm256_load_pd(arr + 4);

// Perform four pairwise additions
__m256d sum = _mm256_add_pd(a0, a4);
double arr[4] = {0, 0, 0, 0};

// Copy the contents of sum to arr
_mm256_store_pd(arr, sum);
```

GCC compiler was used to compile and generate the assembly code to see the vector instructions corresponding to the functions used in C program with intrinsics. The *immintrin.h* header file is required for compilation. The *-mavx* flag must be used in compilation command. Running the code is same as an ordinary C program. Students were asked to inspect the assembly code produced by compiler using *-O2* and *-O3* flags.

For standard C-based matrix multiplication, the code generated by the compiler was shown first and then the code generated using the function with intrinsics was shown to compare the vector instructions in the two versions. The parts of the code where compiler generated sub-optimal vector code was highlighted. Following this exemplar, students repeated this exercise for the C program they had written. Students compared the assembly code generated by regular C program with optimized assembly code generated when intrinsics were used.

The difference in vectorization was explained as follows. In suboptimal code, compiler-generated code had XMM registers with 128 bits width. In the version with intrinsics, compiler generated code had YMM registers with 256 bits width. Wider registers have the benefit of packing more data elements in a single register. For data movement, *vmovsd* was generated in the sub-optimal code instead of *vmovapd*. *s* stands for scalar in *vmovsd*. *p* stands for packed in *vmovapd*. The last character *d* is for double precision.

vmovsd loads one element into XMM register, while *vmovapd* loads four elements into YMM register. Similarly, *vmulsd* was generated by compiler in the sub-optimal code instead of *vmulpd*. In short, the improvement is attributed to wider registers and parallel load of four array elements into the registers using AVX. The scalar version loads one element into the register. For matrix multiplication, this results in more than 3x speedup difference between the two versions on an Intel processor.

A. Sample Quiz Questions

Q1. The fastest ARM processor Fugaku by Fujitsu has a SIMD register which is 2048 bits wide. How many double precision floating point data can be stored in one SIMD register?

Q2. What does the array *arr* contain after the code has been executed as shown in Code Listing 1?

Q3. Let us assume that array *arr* is stored in a block of memory with starting location $(3200)_{10}$ and we are trying to use the load function, as shown below:

```
__m256d first = _mm256_load_pd(arr + 1);
```

Will this line of code work? Question 3 is designed to test the memory alignment issue. Memory alignment issue can lead to program crash. In the question above, 32-byte (256 bits) alignment requirement is violated because address of (*arr*+1) is 3208, which is not completely divisible by 32.

B. Programming Exercises

The course materials containing programming exercises covered in laboratory, quizzes and homework problems has been shared on GitHub [3]. Lecture video has been shared online on YouTube [4].

The programming lab question was polynomial evaluation. A sequential program was provided, and the task was to do SIMD parallelization of the loops by adding intrinsic functions. After finishing the coding part, the next task was to benchmark the execution time and calculate GFLOPS and speedup with respect to baseline. Image manipulation example using intrinsics was considered but eventually not used because of higher complexity of the code. One common problem that was encountered in the lab by a few students was misspelling in intrinsics which was creating compilation errors.

III. ATTACK AT DAWN: CRACKING CODES WITH CUDA AND TEACHING PARALLELISM

By: Rahul Toppur and Michael D. Shah

With the introduction of CUDA in 2007 from NVIDIA, GPGPU programming saw an increase in usage in both graphics and scientific computing domains [5]. When introducing GPGPU programming, educational content has often been dedicated to full courses in the domains of image processing, rendering graphics, and image classification [6]. However, GPGPU programming can be used to solve a wide variety of parallel problems as researchers and industry tackle computationally expensive tasks in new domains, including computer security, cryptography, and machine learning.

The motivation of this courseware is to approach learning GPGPU programming with example lectures, labs, and assignments in the security domain. We present four modules with specific learning objectives that are compact and designed to be inserted within a normal semester's introduction to computer security course. The course is developed for an asynchronous audience and has accompanying lecture videos and presentation slides. All materials (including lectures and videos) can be found at this public GitHub repository [7].

A. Course Methodology

The course materials provided emphasize students' ability to both measure and visualize changes to their program's performance as they learn. For example, students experiment by varying the number of threads and blocks that they spawn to see the performance impact on their programs. In matrix addition, students write the sequential and parallel versions before doing a performance case study. They use the UNIX *time* utility and vary the input of two *NxN* matrices to generate their own data. Finally, they create a graph to visualize the performance of their C and CUDA implementation.

During each assignment, we encourage students to recognize the underlying parallelization opportunity, and to see what portions of the problem can be done utilizing data parallelism in a separate thread. Assignments such as breaking the Caesar and Affine ciphers, performing image processing (such as implementing sepia and gray scale filters), and image steganography (the practice of hiding secret messages in images) all help to hammer this point home.

B. Equipment

The materials provide guidance and encourage students to use their own CUDA-enabled NVIDIA graphics card and the associated *nvcc* compiler in order to run their programs. However, we wanted to make sure this content was accessible to all students, especially in the instance where GPGPU computing is not available in the curriculum because of a lack of computing resources. Thus, Google Colab is used within lecture demonstrations. Colab, along with the associated plugin, allows students to use NVIDIA GPUs to run their programs. We provide a monorepo on GitHub which students can clone to complete their own assignments with the associated starter code.

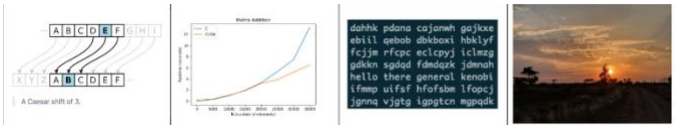

Figure 1. In Module 1 (far left), students work on Caesar ciphers while also reviewing memory management in C. In Module 2 (middle left), students are introduced to CUDA and perform a performance case study for matrix addition comparing CPU versus CUDA implementations. In Module 3 (middle right), students revisit the Caesar cipher using CUDA to decipher a message. In Module 4 (far right), a message is hidden in this image using least-significant-bit steganography.

C. Course Overview

The course is broken up into the following four modules (illustrated in Figure 1, each with their corresponding assignments and learning objectives):

- Module 1: A review of C programming. Students review data structures through linked lists and use a brute-force attack to crack a message encrypted with the Caesar cipher.
    - Review C syntax, File I/O, pointers, and data structures.
- Module 2: CUDA fundamentals. Students are introduced to making simple kernel calls using CUDA. Topics such as *cudaMalloc*, *cudaMemcpy*, and *cudaFree* are introduced here, and students observe the effects of the CUDA thread model (using combinations of blocks and threads) on vector addition. We also observe the performance difference between the C and CUDA implementation for matrix addition by varying the number of inputs. Students gather their own data and create a graph to visualize the performance.
    - Introduction to making kernel calls.
    - Cover memory management, *cudaMalloc*, *cudaMemcpy*, thread model.
    - Use *nvprof* to profile CUDA programs.
    - Write both the C and CUDA implementation to measure, visualize, and compare performance.
    - Compare runtimes for matrix addition between sequential and parallel approach.
- Module 3: Ciphers revisited. Students extend the ideas of Module 1 by now using CUDA to break a new message encrypted with the Caesar cipher. We also introduce the Affine cipher in this module.
    - Recognize the underlying parallelization idea when given a new problem.
    - Recognize a use-case for Single Instruction, Multiple Threads (SIMT).
    - Spawn the appropriate number of blocks and threads to cover the given problem space.
- Module 4: Image processing. Students work on applying different types of filters (such as grayscale, brighten, and darken) to get familiar with images. Students also work with image steganography, where they encode and decode a given message using least significant bit steganography.
    - Perform bitwise operations to manipulate data.
    - Understand least-significant-bit steganography.

D. Preliminary Assessment

We performed a pilot study with two students to gather feedback on this short course.

1) Setup

After students completed the course, we asked them to complete a short survey. The survey asked subjective questions on a scale from 1-5 about their enjoyment of the course, course engagement, and if the lecture videos and assignments contributed to their understanding. A short assessment also followed, which asked comprehension questions about the thread model and memory management in CUDA.

One student was able to make it half-way through Module 3 (the image-processing module) while another made it through Module 2 (vector addition and matrix addition). Student feedback suggested that the lecture videos and assignments strongly contributed to their understanding of the material. Both students thought that they had a firm understanding of memory management and the CUDA thread model after completing the course and got full scores on the assessment portion of the course feedback survey.

- *The Course 2 assignment was pretty good, I would spend a tiny bit more time walking through memory allocation on lecture 2 though." Student Feedback*

E. Conclusions

In this paper we have presented our curriculum for embedding a series of introductory GPGPU programming modules within a security domain. We believe this course could be used as optional supplemental material within an existing course to facilitate students to learn GPGPU programming. The existing modules provide students a starting point with CUDA, and instructors have freedom to expand the curriculum with additional modules or projects.

There remain possibilities to use different languages (such as PyCUDA) that may also integrate well in other security courses if language choice presents a challenge. In addition, we may introduce future modules utilizing GPUs for brute-force

password cracking against MD5 hashes or exploring side-channel attacks.

IV. Teaching Edge Computing as an Undergraduate Course

By: Neftali Watkinson and Alexandru Nicolau

Courses on Edge Computing can be found as part of professional certifications. After surveying undergraduate programs in Computer Science and Computer Engineering, we have not found an active offering of an Edge Computing course for undergraduate students. We believe that University of California – Irvine is the first school to offer it with full curricular credits.

The idea for CS190 started as the culmination of a 2-year project adopting the Curriculum Initiative (TCPP) [8] into introductory programming courses. Out of this effort we reported that undergraduate students could successfully understand basic concepts of parallel computing, dependence analysis and vectorization [9]. We taught these concepts using C++ and Python. For the latter, we relied on Numba for generating optimized code through a Just-in-Time (JIT) compiler [10]. We also designed NumbaSummarizer, a wrapper for Numba that facilitated vectorization and dependence reports for Python users [11]. The motivation for teaching CS190 is to go beyond the conceptual learning of parallel computing and provide students with the opportunity to apply and learn practical concepts of high-performance computing (HPC) into open ended assignments and projects.

The class was offered online during the Spring quarter of 2020. For programming and implementation assignments, the students relied on a JupyterHub server and a Raspberry Pi (RPi) 3B+ with a camera installed. We provided them with a USB Coral TPU which contains a Tensor Processing Unit for efficient and optimized tensor-driven inference. This would allow for their RPis to do image recognition on live video with a faster frame rate than using the RPi's processor alone (25 FPS v. 5 FPS approximately).

The class met two times a week for 100 minutes each day (equivalent to two 50-minute sessions). We formatted so that the content would continuously flip between sessions focused on theoretical concepts and sessions that were purely practical and showcased tutorial-like lectures where students interacted with useful tools for their final project. The students could meet with the TA or the lecturer whenever was needed, and they had to meet twice with the TA to report on their progress for the final project. The content of the class was loosely based on the book "Fog and edge computing: principles and paradigms" [12].

A. Course Content

Disregarding midterms and holidays, there were 15 active sessions shifting between theory-centered lectures and practice sessions. The entire lesson schedule was as follows:

1. Introduction to Edge Computing: An overview of the course's content and logistics, key concepts in Edge Computing, and identifying the different layers of and Edge Computing network.
2. Setting up your tools: In this session students get step by step assistance in setting up their RPis, running code provided on the Coral TPU, and how to access the Jupyter Hub server.
3. Performance challenges in Edge Computing: A focused look into the role that computational performance and energy use has in edge devices, including a description of common techniques for doing effective computation and tools that can be used to monitor performance and energy usage.
4. Profiling performance: Students learn how to measure running time, energy usage, memory usage and number of executed instructions.
5. General Purpose Processors in Edge Computing: A detailed comparison between different philosophies behind computing architecture design, an overview of Reduced (RISC) and Complex (CISC) instruction set architectures, and common optimizations found in Intel and ARM general purpose processors.
6. Numba: A primer on using Numba to optimize Python and using NumbaSummarizer to output vectorization reports.
7. Special Purpose Processors in Edge Computing: In this lecture we describe the different types of architectures that may be found at the edge including GPU, TPU, FPGA, and ARM's big.LITTLE architectures.
8. Dependence analysis: In this session students get to practice dependence analysis and loop transformation to improve running performance and memory usage.
9. Middleware for Edge Computing: This is a flipped classroom experience where students research different types of middleware and then present them to their classmates.
10. Brokerless MOMs: In this session students learn to setup a basic Message-Oriented Middleware (MOM) without a need for brokers.
11. Kernel optimization: For this session, students use their knowledge and experience optimizing code to optimize some common kernels for image transformations targeting their Raspberry Pi in preparation for a related assignment.
12. Security, Privacy and Ethics: For this lecture students get to participate in a conversation regarding the importance of subscribing to a code of ethics. Additionally, we introduce them to relevant law regarding the use and capture of personal data.
13. Setting up a cloud service: Students learned how to setup a cloud service with Google Cloud using trial accounts and how to interact with their RPi using that service.
14. Edge Computing's state of the art: For the final theory centered lecture the students are introduced to novel approaches to common tasks at the Edge of the network.
15. Final project showcase: Students get to demo their final projects.

For the final project the students developed a full edge computing network. The system's design is open ended,

students had to come up with the idea for the system. However, the minimum requirements were as follows:

- Students had to implement all three levels of the Edge computing network (Fog, Cloud, and Edge).
- The Edge layer should have at least two devices interacting with each other either directly or through a middleware.
- The scope should be properly documented and broader than the demo itself. In other words, students had to describe how the full-fledged system would look like if they had the time and resources to fully develop it.
- The final delivery should have a report on how feedback was incorporated into the project and deployment instructions for the demo as well as relevant code.

Some of the submission we received included a trash sorting system that used the cloud to keep track of how much of the trash sorted was recyclable, a system that detected if people were practicing social distancing, a smart thermostat system that used the cloud service to look at the predicted weather and adjust according to it, a system for controlling watering and nutrient addition in a greenhouse, a smart store system that kept track of product inside a physical shopping cart and suggested other products based on the user's history, a security system that granted access based on face detection, among others. All teams managed to turn in their final deliveries.

B. Conclusion and Feedback

We had 61 students, 1 lecturer and 1 teaching assistant (TA). For the final project we had 15 groups with an average of 4 students per group. Although the course was intended for seniors, we did not enforce a prerequisite. 23 students reported they were junior level and eight sophomores. The average final grade was 95%. There was 10% of extra credit and the grade was not curved.

Reading through student feedback, some common positive responses were regarding the content of the course. Students mentioned that the topics are relevant and immediately applicable in the professional world, the assignments invited them to go beyond the minimum requirements and the final project was something they could feel proud of and include it in their professional portfolio. Among the negative feedback, ideas that repeated were that the open-ended nature of some assignments left room for confusion and ambiguity, working in teams was difficult (we believe this is because they had to work remotely) and they would have liked more content on some of the topics.

We believe that an in-classroom offering would improve the experience with working in teams. We also want to make it a dual offering for Computer Engineering to ensure that we have a diverse classroom where students could benefit from each other's experience when working together.

While designing the course we noticed that it is very easy to let topics such as machine learning, IoT, or Cloud Computing take the spotlight. it is important that when adopting this course, the instructor keeps it balanced and lets the students pursue the topics they want to learn more about.

V. TOWARDS ENABLING EDUCATION AS A SERVICE ON HIGH PERFORMANCE COMPUTING RESOURCE

By: Weijia Xu and Hui Zhang

There is an increasing demand of supporting educational activities directly with high performance computing (HPC) resources in recent years. This is in concert with the increasing adoption of data-driven analytics fueled by massive amounts of complex data produced by businesses, scientific applications, government agencies and social applications across various disciplinary fields. Data-driven analytics have the potential to help users gain new insights for decision support, scientific discovery [13]. However, data-driven analytics often require computing resources only available from remote HPC resources. Hence there is an increasing demand of learning and utilizing remote HPC resources along with an increasing demand of using HPC to facilitate classroom education across many disciplinary fields.

Since 2018, the authors have organized over a dozen training workshops at Texas Advanced Computing Center (TACC) and/or in conjunction with leading conferences such as Supercomputing conferences (2018–2020) using computational resources at TACC. The topics of those training workshops ranges including scalable machine learning, big data analysis tools, and distributed deep learning tutorials. The enrollments of those training events have been steadily increasing over the year.

During the same period, the number of requests to use HPC resource at TACC in the classroom is also increased dramatically. Those requests are motivated to support a wide variety of topics, from traditional computing fields, such as teaching distributed methods in computer science and data science program, to domain sciences such as introducing large data set and latest methods in biology and digital humanities. Therefore, education and training on effectively using HPC resources become essential requirements disciplinary fields.

There are many unique challenges to directly support educational tasks with the HPC resources. We summarize three main challenges: resource provisions, credential management, and configurable content as follows.

Resource Provisions Challenge Cyberinfrastructures are usually designed as shared computing resources. Computing nodes are provisioned dynamically upon individual *job request* by users. Each *job request*, once submitted, will be placed in a queue and managed by a job management software. This resource provisioning model is not well suited to support educational activities where all students need access and use the system for a short period time. Since each *job request* is submitted independently by each student, there is an uncertainty on when one's job can run especially when available computing resources are limited. Due to different levels of familiarity, background and learning pace, it is often the case that some students will wait longer than others, which is disruptive to the pace of the entire class.

Credential Management Challenge As a shared infrastructure designed for many users, it is necessary to have strict and complex access control mechanisms. Taking the resource hosted at TACC as an example, the process not only requiring each user

to set up an account prior to the class, but also requiring the instructor to associate student accounts with an appropriate project allocation. Complicating this process further is the adoption of multiple-factor authentication requirement. These security measures require student and instructor preparation well ahead the class time. Otherwise, it will take significant amount time to setup individual accounts during the class. In some scenarios, such as given an open tutorial at a conference, it is impractical to set up accounts for participants on the spot.

Content Customization Need Education activities often need to be tailored and customized each time. Even for the same topic and content, small updates may be needed each time. Some examples include, to update data sets used, to update exemplar code and exercises, to make adjustment based on classroom size etc. It is common that one seemingly small change may require additional updates. On the other hand, there are also similarities on the abstracted tasks needed for different educational activities. Hence there is a potential need to improve the robustness and reusability through a modularized customizable design support.

A. *Approaches and Novelty*

To address those challenges and to better support various education tasks, we propose the development and utilization of a user driven configurable web application framework, named IDOLS [1]. An overview of the system architecture of our framework and startup workflow is shown in Figure 2. Here *application owner* refers to the instructor who will start the web application service which is used to support various educational activities on the HPC cluster. A*pplication user* refers to students who will access and utilize the web application service started by the instructors. Hence, students will interact with clusters through the web application interface set up by the instructors rather than access the computing cluster directly.

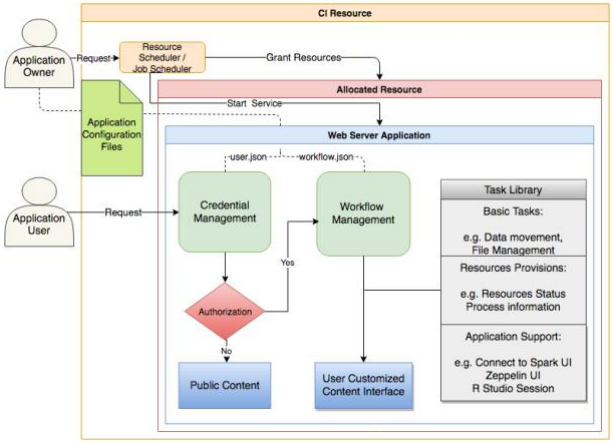

Figure 2. Overview of framework architecture and access workflow

There are four key components of this web application framework: credential management [14], application management, pre-defined task libraries and application configuration files. These components can address the resources provision challenge, credential management challenge, and customization needs described previously. On resources provision, instructors can request resources and start the web service application before the class. So, there is no need for students to request resources during the class. Since the web applications have its own dynamic credential management through which temporary credentials can be generated and assigned to the students on demand. The temporary credential can be linked to pre-generated training account on the cluster. The application configuration files are the central piece to enable further customization and updates. Most of the features and the web application content can be dynamically customized through configuration files [13]. For more details, please refer to [15].

B. *Use Cases and Takeaways*

We have experimented with three use case scenariols: supporting training workshops with dynamic generated credentials; supporting of teaching parallel computing in the classroom; and supporting audio collection studies in digital humanities.

Supporting training workshops with dynamic generated credentials Many of our tutorial training sessions uses Jupyter Notebook to deliver examples and exercises to students. Preivously, we need hand out prepared training accounts to participants and ask student to start each Jupyter notebook session on their own. Each job will need go through the job queue and wait for resources to be ready. With the new approach, we can prestart Jupyter notebook sessions and assign one to each student upon request on demand. Both starting sessions, temporary credential generation and requests can be handled by prebuilt tasks in IDOLS. This new approach can reduce the wait time for students and imporve roubstness.

Supporting of teaching parallel computing in the classroom Dr. Zhang has incorporate the framework with his class for teaching different parallel R models for undergraduate students at University of Louisville as well as a data science gateway for K-12 teachers to learn data science techniques and practice big data principles with R language. Most of the audiences are new to data analytics and are not yet equipped to take advantages of the advanced tools by HPC. Different parallel models are prebuilt as modules for students to experiment. This approach can reduce the needs to introduce additional background knowledge on HPC environment and help students focusing on different parallel models [16].

Supporting audio collection studies in digital humanities This use cases concerns StoryCorps *Las Historias* audio collection. This audio collection could be used as a subject in different social science courses and studies. However, the collection is about 400GB makes it hard to distribute. We have built a web application interface using IDOLS to facilitate it access. In this model, the audio collection is stored on HPC, our web application serves as an interface for students and instructors to access. In addition to access and listen to the audio files, we also integrate with deep learning and natural language processing methods which can generate transcripts from audio and run meta-analysis for students. The interface can ease the access to these large data collections and also augmented with machine learning methods for social science students and researchers to interact [17].

ACKNOWLEDGMENTS

This work was supported by funding from National Science Foundation (Award# 1726816).

REFERENCES

[1] Patterson D.A. and Hennessy J.L. (2016). *Computer Organization and Design (ARM Edition): The Hardware/Software Interface.* Morgan Kaufmann.

[2] Intel Intrinsics Guide, https://software.intel.com/sites/landingpage/IntrinsicsGuide/.

[3] Puri S. Teaching Intel intrinsics for SIMD parallelism, https://github.com/satishphd/Teaching-Intel-Intrinsics-for-SIMD-Parallelism.

[4] Puri S. Intel intrinsic functions for SIMD parallelism (vectorization), https://www.youtube.com/watch?v=KABGmD7BJ28t=2750s.

[5] Sanders J. and Kandrot E. (2010). *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional.

[6] Fenwick G. and Norris C. (2020). GPGPU Programming for CS Undergraduates, *Proceedings of the 2020 ACM Southeast Conference*.

[7] Toppur R. Introduction to CUDA, https://github.com/rahultoppur/CUDA.

[8] S. K. Prasad S.K. et al. (2011). NSF/IEEE-TCPP curriculum initiative on parallel and distributed computing: Core topics for undergraduates, *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*.

[9] Watkinson N., Shivam A., Chen Z., Veidenbaum A., and Nicolau A. (2017). Using data dependence analysis and loop transformations to teach vectorization, *2017 International Conference on Computational Science and Computational Intelligence (CSCI)*.

[10] Watkinson N., Shivam A., Nicolau A., and Veidenbaum A. (2019). Teaching parallel computing and dependence analysis with python, *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[11] Watkinson N., Tai P., Nicolau A., and Veidenbaum A. (2020). Numbasummarizer: A python library for simplified vectorization reports," *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[12] Buyya R. and Srirama S.N. (2019). *Fog and Edge Computing: Principles and Paradigms.* John Wiley & Sons.

[13] Yige W., Huang R., and Xu W. (2018). Authentication with user driven web application for accessing remote resources, *ACM Proceedings of the Practice and Experience on Advanced Research Computing*.

[14] Xu W., Huang R., and Yige W. (2018). Enabling user driven web applications on remote computing resources, *2018 IEEE World Congress on Services*.

[15] Xu W., Huang R., and Yige W. (2018). Enabling user driven big data application on remote computing resources, *2018 IEEE International Conference on Big Data*.

[16] Subramanian R. and Zhang H. (2019). Parallel R computing on the web, *2019 IEEE International Conference on Big Data*.

[17] Xu W. et al. (2020). A study of spoken audio processing using machine learning for libraries, archives and museums (LAM), *2020 IEEE International Conference on Big Data*.