WebGPU: A Scalable Online Development Platform for GPU Programming Courses

Abdul Dakkak
Department of Computer Science
University of Illinois at Urbana-Champaign
Urbana, USA
dakkak@illinois.edu

Carl Pearson and Wen-mei Hwu
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign
Urbana, USA
{*pearson, w-hwu*}*@illinois.edu*

***Abstract*—The popularity of computer science classes offered through Massive Open On-line Courses (MOOCs) creates both opportunities and challenges. Programming-based classes need to provide consistent development infrastructures that are both scalable and user friendly to students. The "Heterogeneous Parallel Programming" class offered through Coursera teaches GPU programming and encountered these problems. We developed WebGPU – an online GPU development platform – providing students with a user friendly scalable GPU computing platform throughout the course. It has been used as the CUDA, OpenACC, and OpenCL programming environment for large Coursera courses, short-running summer schools, and traditional semester-long graduate and undergraduate courses. WebGPU has since replaced our traditional development infrastructure for the GPU classes offered at UIUC. This paper presents the original, revised, and upcoming WebGPU designs that address the requirements and challenges of offering sophisticated computing resources to a large, quickly-varying number of students.**

***Keywords*-GPU; CUDA; OpenCL; OpenACC; massive open online courses; programming education; online education**

I. Introduction

An emerging trend in post-secondary education is the availability of massive open online courses (MOOCs). These courses are typically free for students to enter and consist of video lectures, quizzes, problems, and exams delivered over the internet and consumed through a standard web browser. Delivering programming education this way comes with a specific challenge: a student must have access to the required programming tools and environments.

Exploiting the performance available in graphics processing units (GPUs) has become fundamental in making large data-parallel computations tractable. Extracting the performance from GPUs generally requires using a bulk-synchronous programming language like OpenCL or NVIDIA's CUDA and understanding architectural details.

As a result, more students and professionals are seeking learning resources for GPU computing. The University of Illinois at Urbana-Champaign (UIUC) has developed course material that explains the concepts and programming techniques for mapping algorithms into GPU programs that architecture and tools available. Similar materials was developed as single-core, multi-core, and distributed programming matured.

All GPU curriculum requires students to access systems with GPUs for programming assignments. There are three key challenges to overcome when providing such systems for a MOOC:

- Number of students and student-educator ratio. The environment cannot fall back on student/educator interaction (it must "just work").
- Variable computing capabilities available to students: few students will have GPUs for local development or low-latency internet connections.
- Decrease in participating students as time passes: Only 3.15%, see table I, of registered students in 2015 completed the full course.

For traditional courses, development environments are provided to students through loaner systems, computer labs, and/or educational/research clusters. These systems are not practical for delivering development environments in a MOOC setting. Loaner systems and computer labs are inaccessible to remote students, and providing access to a cluster ultimately requires over-provisioning of hardware (and therefore the cost). Furthermore, it is impossible to expect a consistent student GPU development environment.

WebGPU explores an efficient solution to these challenges. It is a scalable online GPU programming environment accessible though the web. By removing access to fully-featured system environment and developing targeted labs, WebGPU simplifies management without unduly impacting educational capability. The number of GPUs available through WebGPU can be dramatically fewer than the expected number of concurrent users, and can be dynamically scaled as the course participation changes.

The rest of this paper is organized as follows. Section II describes the challenges of providing computing resources for MOOCs. Section III describes the system architecture of WebGPU, with emphasis on security and scalability. Section IV describes how the instructors and students interact with WebGPU. Section V describes some of the courses that

This work was supported by the Starnet Center for Future Architecture Research (C-FAR), NVIDIA GPU Center of Excellence at UIUC, and the NSF Blue Waters Award (0725070).

The authors would also like to thank the Coursera community TAs that helped during the Heterogeneous Parallel Programming course offerings.

WebGPU has been used to deliver. Section VI describes some lessons learned in the development and deployment of WebGPU and an improved architecture design. Finally, section VIII concludes.

II. MOOC Challenges and WebGPU Approach

Between 2002 and 2011, online enrollment in degree-granting post-secondary institutions grew at an annual rate of 17.3%, from 1.6 million to 6.7 million students. At the same time, the total enrollment in the same institutions grew at an annual rate of 6.7% This reflects the broad agreement that online education is an important and sustainable part of pedagogy, useful to students, and online degrees are accepted in the workplace [1, pp.14-18].

MOOCs take the idea of online education to its farthest extent: fully-online classes with large enrollments, often without geographic restriction or requirements of prior admission into an academic institution. This presents three challenges when delivering programming education online.

A. Large Course Enrollments

Large enrollment causes management challenges that are not unique to MOOCs: a large course enrollment requires a large staff of educators to effectively administer the course. In traditional courses with enrollments of hundreds of students, there might be a single lecturer and a dozen teaching assistants. Large MOOCs like Duke University's *Think Again* [17] have tens or hundreds of thousands of active students. Providing a traditional teaching staff for a free course of this size is cost-prohibitive. WebGPU tackles this problem by taking advantage of being a completely-online development environment to reduce the amount of supported capabilities exposed to the students. Furthermore, WebGPU provides automated grading of the programming assignments to reduce the load on the teaching staff.

B. Variable Access to Computational Resources

The problem of large course enrollments is compounded in programming courses, which require development tools and environments as well as potentially specialized hardware. Assuming all students will have access to the tools and hardware and use them in a consistent way is not practical. MOOCs are not the only courses where this is a challenge – in traditional academic contexts an instructor often devotes the initial lectures and labs to describe the development environment and tools.

In traditional classrooms, instructors can usually leverage one or more of the following options to provide students with consistent computing environments:

1) **Individual/Loaner Systems:** Students may be provided with individual computer systems (laptops, development boards, etc.) upon which all development is done.
2) **Computer Labs:** Machines may be set up for student use in a fixed location. The student typically must be physically present to use one of these workstations, and they are shared with other students.
3) **Educational/Research Cluster:** Typical use takes place through a submission queuing system, where programs are submitted and scheduled onto cluster nodes based on their resource requirements and maximum run time.

The first two solutions are logistically impossible for a MOOC with global students. Remote cluster access seems like a plausible solution: clusters are already designed to support many concurrent remote users. Clusters are shared resources with particular OS and compiler versions that often are not compatible with the devices used in a structure GPU programming course. Also, the cost of provisioning of tens of thousands of users on an academic cluster is high.

In place of a command-line prompt, WebGPU requires a web browser, making it accessible to all students participating in the course. Any operating system and computing device may be used – in fact, around 2% of student logins to WebGPU are from tablets and smartphones. This also reduces the support required of the educators; when bugs do arise, educators can apply the fix by updating the website.

Table I
Registered users, completion rates, and issued certificates for the three years that Heterogeneous Parallel Programming has been offered through Coursera. Certification means that the student attended a proctored quiz.

Year	Registered Users	Completions	Completion Rate	Certificates Issued
2013	36896	2729	7.40%	-
2014	33818	1061	3.14%	286
2015	35940	1141	3.15%	442

C. Participation Reduces as Course Progresses

As with other MOOCs, participation in the course decreases as time goes on. Jordan [11] finds that the average MOOC completion rate is 15%. For the Heterogeneous Parallel Programming Coursera course offered using WebGPU, completion rates are shown in Table I.

This means that a statically-provisioned computing resource large enough for the beginning of the course will be mostly idle by the end of the course once many students have left. WebGPU builds on the cloud computing principle and handles this challenge through a modular architecture that separates web-server, database, and worker nodes so resources can then be scaled separately as demands require.

III. System Architecture

WebGPU is designed to be a fault tolerant system able to handle thousands of students submitting GPU tasks concurrently. Figure 1 shows the number of active students per hour from February 8^{th} 2015 to April 15^{th} 2015. WebGPU

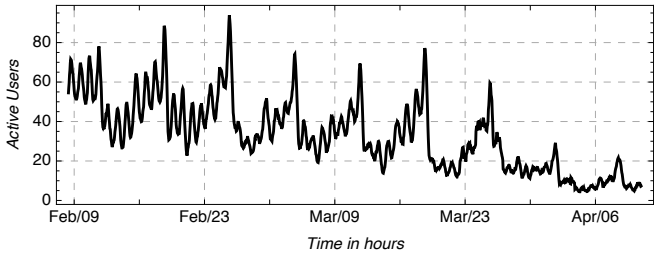

Figure 1. The number of active students per hour on WebGPU from February 8^{th} 2015 to April 15^{th} 2015. The number of active students varies from 112 on February 18^{th} to 8 on April 9^{th}. In the 2015 Heterogeneous Parallel Programming course, Thursday was the lab deadline. A spike occurs every Wednesday as students rush to complete the lab.

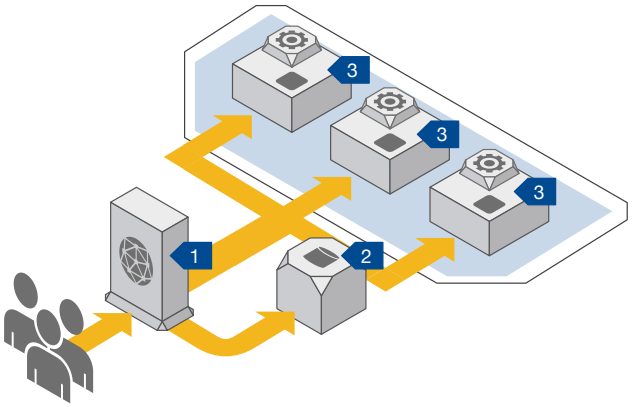

Figure 2. The architecture for WebGPU has three classes of services - web-servers ①, database servers ②, and workers ③. A web-server contains logic to accept user requests, store and retrieve information from a database, and dispatch jobs to a pool of workers. A worker accepts code from the web-server performs the compilation and execution of the user code in a sandbox environment. The results are sent to the web-server and relayed to the user.

design enabled it to scale through the fluctuations of students – thousands of users per day towards the start of the course to 200 users at the end. As can be seen, weekly spikes happen on Wednesdays, since the deadline is on Thursdays. We increased the number of GPUs available to WebGPU the day before the deadline.

Section II described three challenges in offering GPU computing resources to students enrolled in a MOOC: number of students, variable accessibility to GPUs, and dramatically shifting enrollment. Traditional HPC clusters could be used to provide students with the required resources, but come with several disadvantages:

- Significant resources expended on fair scheduling for arbitrary workloads, general filesystem and storage access, fine-grained user permissions, and other tools to support general use increase the administration overhead and complexity of use for the students.
- In 2013, when we first started to use WebGPU, GPUs were not as prevalent on HPC clusters. MOOC students would compete severely with other cluster users.
- HPC Clusters must be secure. Allowing anyone to sign up for the course without verification makes it impossible to share the cluster with any security-sensitive users.
- The practice of building and scheduling jobs on a cluster can be a deterrent to introductory students, and is of little pedagogical value for GPU parallelism.
- A large cluster for the students present at the beginning of the course would be idle at the end of the course.
- A separate assignment grading system needs to be created for students to submit their completed work.

This section describes the modular design WebGPU uses to avoid the above issues. Fundamentally, WebGPU consists of three types of nodes hosted on Amazon AWS [2]: web-servers, databases, and workers. Since these three node types are separate, each can be scaled as required. Figure 2 shows a high-level view of the system architecture.

A. Web-Server

The web-server ① generates the site's HTML code and handles user requests. It connects to the database and other external gradebooks. It automatically saves all student code, and their compilation and execution status, and previous attempts so that a user can backtrack to earlier versions of their code. This allows students to develop their code incrementally and explore the lab beyond the specified requirements. It also minimizes and extra work the students need to do in order to submit their work for grading. Finally, the web-server acts as an intermediary, dispatching jobs to a node in the pool of workers and relaying the results users.

B. Database Server

A database ② stores all user records such as user profile, program submissions, and grades. Initially WebGPU used a MySQL database but has since migrated to Amazon's Aurora database. The web-server maintains a connection pool to the database and records user submission activity.

C. GPU Worker Nodes

Upon a user program submission, the web-server ③ selects a single worker node and sends user code along with configurations specified by the lab. The worker node then compiles, executes, and evaluates the code using the datasets provided by the instructor. The results along with any error messages produced are sent back to the web-server which relays it to the user.

To maintain fairness, time limits are placed on the submission rate and on the duration of the compilation and execution of user code. The time limits can be adjusted on a per lab basis.

An additional task is for the worker node to send regular health checks to the web-server. The web-server would evict the worker from the pool of workers if a health check is not received within an allotted time.

D. Security

To maintain security we use a combination of black listing certain function calls at compile time and white listing the system calls allowed at runtime.

The black listing is performed on the user code. A textual scan on the unparsed code disallows certain strings such as `asm();` which introduces inlined assembly which may potentially escape any sandbox in place. This method rejects code which contains the black listed functions even within comments. If the black list search is run on the code after running the preprocessor, we can avoid false negatives, but few users found the false negatives a nuisance.

Execution of the user code must be done using unprivileged permissions and in a sandbox environment. We use `setuid` to execute the user code as unprivileged user who can only write to a unique temporary directory created for each compilation. For sandboxing, we utilize the Linux kernel's seccomp facilities introduced in 2.6.12. The seccomp-bpf extension allows us to provide a whitelist of posix calls that are allowed to be run by a process. The whitelist is provided by the instructor on a per lab basis.

Aside from the above, the physical separation of user code from the main logic on the web-server provides additional security. Since user code is only compiled and executed on the worker nodes, a user able to thwart our security measures would be confined to the worker node and cannot access critical data found on the database.

IV. Interface of WebGPU

This section describes how students and instructors interface with WebGPU. All students interact with WebGPU through standard web browsers. Instructors can manage existing course content through a web browser, but developing new content requires familiarity with the system deployment and Linux command line.

A. Student Actions

WebGPU restricts the available actions that a user can take to vastly reduce the per-user cost of maintaining the system. Broadly speaking, students can only take six actions:

1) **Edit code:** students are presented with an editor where they can develop their code. The editor provides autosave capabilities as well as syntax highlighting for the programming languages used in the course. Figure 3 shows a view of the editor.
2) **Code compilation:** students are presented with an interface to compile their code. Compilation is done on a worker node and errors are reported to the student.
3) **Run code against a provided dataset:** students can evaluate their code against instructor provided datasets. If a mismatch occurs between the computed and the expected values, the student is informed.
4) **Provide short-form answers to questions:** users are asked questions that are pertinent to their code. Unlike the multiple-choice quizzes, these are textual answers with not-necessarily one correct answer.
5) **Submit code for grading:** upon completion of the lab development, students submit their code for grading. This evaluates their program on all the datasets using the grading rubric defined.
6) **View code history:** code edits, code submissions, and grades are all saved and viewable by the students. Students can inspect and compare to previous codes.

These actions are sufficient to allow the student to develop solutions to the labs in a direct and natural fashion.

B. Lab Solution Development and Submission

A student can access five components of the lab material through a web browser.

1) **Description**: This is the manual for the lab, generated from the markdown-formatted [8] description described in section IV-E. The grading rubric is also shown to the students.
2) **Code**: This view consists of a text editor and compilation controls. Initially, the text editor contains any skeleton code of the solution. Figure 3 shows an example of a code view for the vector-addition Lab as rendered in Google's Chrome web browser.

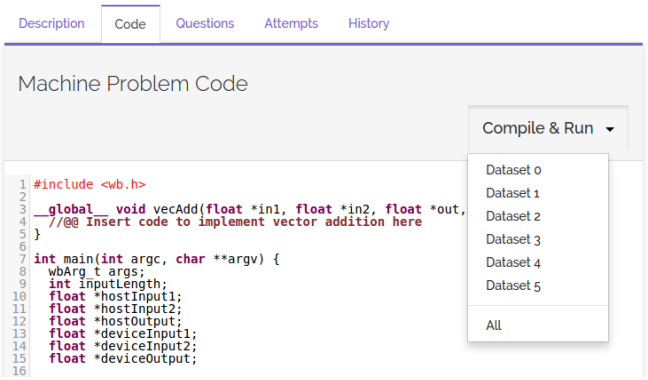

Figure 3. An example Code View, showing the drop-down menu with compilation controls. The student would be responsibly for filling out the vector-addition kernel `vecAdd` skeleton code. To test, the student can run against the various datasets in the drop-down menu. The `wb.h` WebGPU support library header is visible at the top of the code.

When the code is run against a test dataset (an *attempt*), the student is presented with any mismatches between the program result and the test dataset. Each attempt is stored under the Attempts view so the student can refer to it later. A student can generate a public link to their attempt once the lab deadline has passed.

3) **Questions**: This view shows any questions that the teaching team requires the students to answer. There is no system for automatic grading of questions.

4) **Attempts**: This view shows the result of every time the code has been run against one of the test data sets, as well as the result of the attempt. The student may view what the code looked like during that attempt, as well as any mismatches between the program result and the test datasets.
5) **History**: The entire revision history of the code is available in this view. Figure 4 shows an example view of coding history.

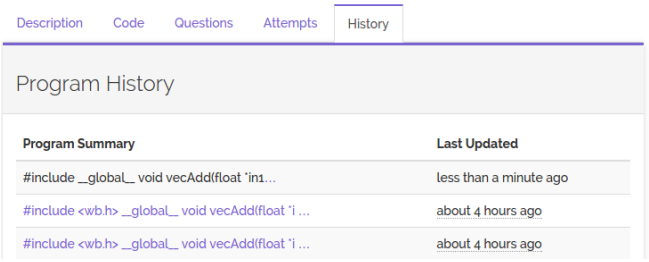

Figure 4. An example History View, showing a code snippet in the left column and the update time in the right column.

C. Optional Offline Development

The lab solution skeletons, test generators, and WebGPU library [5] are publicly available for students to develop their code offline. This requires that the student have access to a system with a C++ compiler, a CUDA compiler, and an NVIDIA GPU. The student must build the WebGPU support library, link it with the solution code, and test it with data produced by the generators. The build uses CMake script to make the build process portable across compilers and IDEs. Final submission must be done using WebGPU.

D. Peer Reviews

Since labs build on concepts in previous labs and reading (and not just writing) code is an important aspect of programming, WebGPU provides students the ability to evaluate each others' lab submissions. For example, in the second offering of Heterogeneous Parallel Programming, each student was assigned three other random students' labs with 10% of the lab's grade given to the completion of the peer reviews. Since WebGPU cannot evaluate the accuracy of the peer review, points were assigned for completing the peer review and did not impact student's grade.

Due to the random assignments, many students were offering reviews without receiving them. The high drop rate at the beginning of the course caused low probability of an active student being assigned an active peer reviewer. Complaints from users about not receiving peer reviews forced a decrease the weight of peer review to 5% and then phase out of the peer review in the third offering of the course. Students can share their attempts with others after the lab's deadline has passed. We are exploring an automated feedback approach for future offerings of the course.

E. Instructor Lab Creation

WebGPU is intended to be managed by its developers, and little emphasis was initially placed on a user-friendly lab creation system. Unlike the students who only need a web browser, lab creation requires a remote terminal and familiarity with system configuration. A lab is defined by:

1) **Lab Description**: a file in markdown [8] format. This description can include any text, images, and external links that are desired.
2) **Solution Skeleton**: This skeleton is starter code shown to the students when they first access the lab.
3) **Datasets**: Instructors provide both input and expected data for the lab. The files are used to check the correctness of student submissions.
4) **Short-answer Questions**: Questions to gauge students understanding of material can be specified, and there is no provision for automatically grading questions.
5) **Configuration Data**: A JSON file which describes the problem deadline, how to award points, the name of the Lab, and other similar information. Points are arbitrarily divided among datasets, short-answer questions, presence of keywords, and successful compilation.

F. Grading

WebGPU provides two complementary types of grading: automatic and instructor-driven. When a program is submitted, a student may specify the test datasets to run against. Results are returned to the student and stored in the database.

After students complete a submission, the system assigns a grade automatically and records it in the grade book (storing the grade in Coursera, for example). Instructors are provided an interface to override a grade.

Unlike lab creation (discussed in Section IV-E), the Instructor Tools are only accessible through a web browser. These are the tools that the teaching staff uses to manage grades and formal feedback for the labs.

Figure 5 shows the class roster view. This shows all students with a submission attempt for the Lab. Through the Roster interface, the instructor navigates to a student submission and review their code history, submission history, grades, and short-answer submissions. The instructor is able to comment on student's code and questions.

V. Teaching with WebGPU

This section briefly describes the content of courses that WebGPU has been used to provide GPU computing resources for. Most courses are taught in the CUDA programming language, but WebGPU also supports OpenCL, OpenACC, and MPI. Some of the WebGPU-hosted labs and the courses they are used for are listed in Table II.

From 2013-2015, the Heterogeneous Parallel Programming[9] Coursera course used WebGPU for programming and grading environment all students. This course serves as an introduction to GPU programming

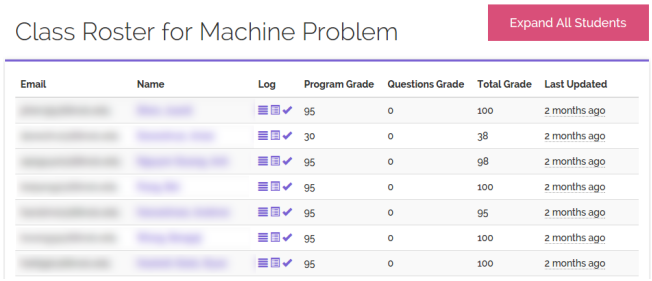

Figure 5. An example Roster View, showing student names and emails (blurred in this example), quick access to student attempts, coding history, and grades, the program grade, the short-answer question grade, the total grade, and the time of submission.

Table II
WEBGPU-HOSTED LABS AND THE COURSES THEY ARE USED FOR. **HPP** IS HETEROGENEOUS PARALLEL PROGRAMMING, **408** AND **598** ARE ECE 408 AND ECE 598 HK AT UIUC, AND **PUMPS** IS PROGRAMMING AND TUNING MASSIVELY PARALLEL SYSTEMS AT UPC BARCELONA.

Lab	Description	HPP	408	598	PUMPS
Device Query	Demo Lab to introduce WebGPU to students.	x	x	x	
Vector Addition	CUDA kernels.	x	x		
Basic Matrix Multiplication	Boundary checking and indexing.	x	x		
Tiled Matrix Multiplication	Introduce shared memory tiling.	x	x		
2D Convolution	Constant memory and shared memory.	x	x		
Reduction and Scan	Floating-point, work-efficiency, tree-like structures.	x	x		
Image Equalization	Atomic operations.	x	x		
OpenCL Vector Addition	OpenCL				x
Scatter to Gather	Transformation between scatter and gather.			x	x
Stencil	Register tiling and thread-coarsening.			x	
SGEMM	Register tiling and thread-coarsening.			x	
SPMV	Sparse matrix formats and performance effects.			x	x
Input Binning	Input Binning and performance effects.			x	x
BFS Queuing	Hierarchical queuing performance effects.			x	x
Multi-GPU Stencil with MPI	Multi-GPU programming and MPI.				x

techniques and hardware. The content has evolved yearly to reflect optimizations and techniques available for new generations of GPUs.

ECE 408 uses GPU programming and CUDA as a motivation for parallel programming and parallel algorithm fundamentals, and ECE 598HK provides more detailed and in-depth exposure to advanced algorithmic techniques. For both of these courses, WebGPU scales down in the number of worker nodes and serves as a development environment for a traditional course offering. WebGPU peer review was not used as the teaching team was able to fill that role because of the smaller class size.

Using WebGPU for ECE 598HK was offered simultaneously at three additional institutions: North Carolina State University, the University of Tennessee Knoxville, and Oklahoma University. Each institution had a local teaching staff to handle questions about the course content, but all Lab development was done in WebGPU and all lectures were delivered by video online from the University of Illinois. This style of offering is a hybrid between the MOOC model and the traditional course style.

WebGPU's restrictions on code run time, API calls, and lack of exposed file system make it inappropriate for open-ended course final projects. A traditional HPC cluster at the UIUC was used by students for both ECE 408 and ECE 598HK final projects. This is an area that has been targeted for improvement in future versions to the system.

WebGPU has also been used as the programming environment for PUMPS [21] an intensive week-long GPU-programming summer school. The rapid nature of this course exposed how requiring instructors to be familiar with WebGPU to deploy new labs is a significant limitation when instructors want to adjust course content quickly.

VI. WEBGPU 2.0 ARCHITECTURE

Student and instructor features continue to drive improvements to WebGPU's infrastructure. Based on our experience with WebGPU, as well as feedback from instructors at other institutions, we are in the process of rewriting WebGPU to make it usable for courses outside of UIUC.

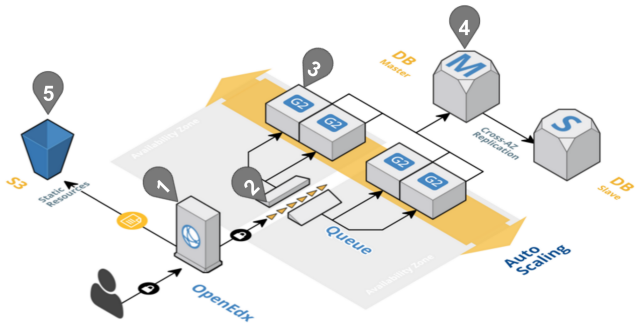

Figure 6. The new WebGPU architecture uses an OpenEdx ① frontend where we have developed an XBlock for programming. OpenEdx uses a message broker ② to queue jobs to the worker nodes ③. The worker nodes are automatically scaled and metrics and logging information on a replicated database ④. Lab datasets are stored on an Amazon S3 Bucket ⑤ which is accessible by both the OpenEdx instructor and the worker nodes.

A. System Architecture

WebGPU 2.0 is an infrastructure overhaul, utilizing new technologies not available 3-4 years ago. Figure 6 shows the current infrastructure. Like the previous design, there is a separation of subsystems.

We now use OpenEdx ① as an interface for instructors to author the labs and the students to develop the labs. This was a result of both instructors and students wanting the same site and interface for all course content – be it quizzes, videos, or programming labs.

OpenEdx communicates with a queue message broker server ② that can be replicated across Amazon availability zone – offering resiliency against faults and better response times for the students.

Worker nodes ③ poll the queue, accepting a job if the node meets the job requirements. This allows us to tag a lab as requiring Multi-GPU support or MPI support and dispatching jobs to the correct node. It also means that we do not need to provision our worker nodes to have the resources for the highest common multiple for the system requirements of the labs, thus reducing the overall Amazon charges.

Whereas the web-server pushed jobs to a worker node in the previous WebGPU architecture, the current requires the worker node to request a job from the queue. This means that we can more freely perform automatic scaling of the worker nodes in the current architecture.

Each worker node constantly monitors the system, performing necessary health checks, as well as validation of state. This information is stored in a replicated database ④. An information dashboard is available to the system administrators to track the system status.

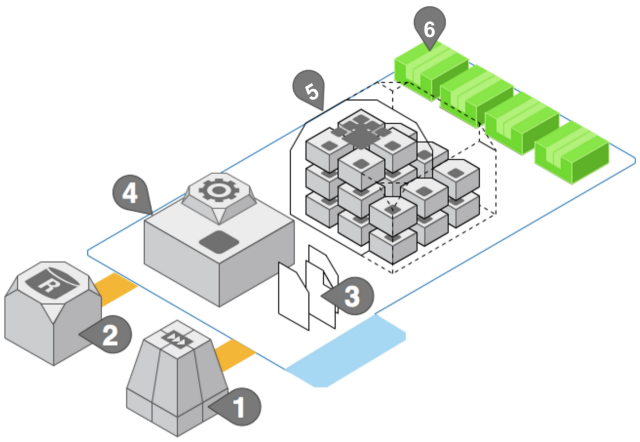

Figure 7. In WebGPU 2.0, a worker node architecture is connected to the job queue server ①, a database for recording metrics and logging information ②, a configuration file server ③. The main driver ④ connects these servers and maintains a pool of Docker [6], [19] containers ⑤. The docker containers are mapped onto physical GPUs ⑥ that are present on the system.

B. Worker Node Architecture

Figure 7 shows the internals of each node in the new architecture. Again, a worker node is connected to the queue ① and the external database ②. The worker node is also connected to a remote configuration system ③. This allows all worker nodes to be remotely configured uniformly.

A change in the remote configuration triggers the worker node to restart the main driver ④ which accepts work from the queue. The driver maintains a pool of Docker containers ⑤ which are mapped onto a fixed number of GPU nodes ⑥. Each time a job is accepted from the queue, the driver selects the appropriate Docker container (the containers are configured to have the essential tools required for the lab — a CUDA lab will not, for example, have the PGI OpenACC tools) and run the job in the container. Previous work [18] shows that docker containers do not add extra overhead when executing GPU code. Because we maintain a pool of containers, we can delete a container after a job complets and start a new container to replenish the pool.

VII. Related Work

A. Web-accessible Clusters

Lin [12] describes a web interface for cluster job distribution and multi-platform source handling in the context of providing parallel and distributed computing resources for a CS curriculum. The portal provides source file management as well as compilation of C, C++, and Java, and execution of the resulting binaries. Like WebGPU, this web interface allows non-expert students to use the development environment. WebGPU follows this design, but also tackles the underlying problem of system scalability in the context of large, fast changing, number of students.

B. Online IDEs and Compilers

Microsoft [13] has an online version of their Visual Studio integrated development environment (IDE), and Eclipse's Orion [7] is a similar IDE for Java. These tools are noteworthy in that they are full-featured development environments that can be used through a web browser. IDEs offer a wide range of sophisticated code-writing and analysis tools such as tab-completion, refactoring, and measures of complexity. For the simple educational programs used in programming courses these capabilities can detract from the core pedagogical focus. For that reason WebGPU does not include most of those capabilities. Furthermore, WebGPU tackles the problem of providing GPU resources to users.

Many simpler tools such as Coding Ground [20], CodeChef [3], and ideone [10], allow code to be edited, compiled, and run in a web browser without the level of sophisticated programmer assistance offered by a full IDE. These editors commonly offer code highlighting, simple automatic formatting, and simple tab-completion. WebGPU to offer tools for CUDA, OpenACC, OpenCL online programming, and specializes it to course management with the functions described in this paper.

Computing notebook environments combine simple code editing and execution with images, video, and rich text in a single page (a *notebook*) in a web browser. Wolfram Cloud [22] and IPython [14] are two such environments that support embedding and executing the Mathematica and Python languages, respectively. Wolfram Cloud is backed by cloud storage and includes methods for collaboratively editing notebooks. These tools demonstrate an excellent format for delivering code and examples to students, but there is no built-in provision for managing deadlines, grading, or providing GPU computing for students.

C. Online Programming Education Systems

qwikLabs [16] organizes learning material and activities into an online learning environment that is focused on cloud computing management. Labs take the form of instructions to guide users through a real-world use case on the actual environment they are learning about. Like WebGPU, qwikLabs only requires a web browser, but the full development

systems are exposed to the user to support qwikLab's goal of teaching a wide variety of cloud computing systems. WebGPU is designed for free GPU programming courses, so removing some flexibility to reduce cost is crucial so long as the restrictions do not impact the education experience.

Truong, Bancroft, and Roe [15] describe the Environment for Learning to Program (ELP), a fully-online programming environment used at Queensland University of Technology. Like WebGPU, the goal is to introduce programming to students without requiring them to familiarize themselves with the full development environment. ELP is also accessed through a web-browser, and utilizes server-side compilation. WebGPU expands on the same formula as ELP by tacking problems of scalability and providing GPUs to students.

VIII. Conclusion

This paper describes the design and interface of WebGPU, a scalable online development platform for GPU programming courses. The fundamental challenges of providing computing resources for a GPU programming MOOC is the number of students and the limited availability of GPUs and CUDA to the global population. WebGPU was developed at the University of Illinois at Urbana-Champaign to explore solutions to these challenges while providing a practical system for offering the Heterogeneous Parallel Programming course on Coursera.

By using standard web browsers, the broadest possible audience is able to participate in the course. In all, over 100,000 thousand students participated in the Coursera course over the last 3 years. By restricting the students' actions, a scalable system can offered on top of inexpensive commodity cloud resources, and making management possible for large number of students.

WebGPU provides an example of how sophisticated computing resources can be provided to a large global body of students with complexity that is manageable for a small team of educators and cost that is low enough for the course to be offered for free. Future work on WebGPU includes automated feedback to students and on-demand help/hints during development.

References

[1] I. E. Allen and J. Seaman, *Changing Course: Ten Years for Tracking Online Education in the United States*, ERIC. Retrieved from http://files.eric.ed.gov/fulltext/ED541571.pdf Dec 28, 2015.

[2] Amazon Web Services, Inc. *amazon web services*. https://aws.amazon.com

[3] CodeChef. *Code, compile, and run*. Retrieved from https://www.codechef.com/ide

[4] Coursera Inc. *Cousera*. Retr from https://www.coursera.org/ Jan 9, 2016.

[5] A. Dakkak. *libwb*. https://github.com/abduld/libwb

[6] Docker. *Docker* https://www.docker.com/

[7] Eclipse Foundation. *Orion*. https://orionhub.org

[8] J. Gruber. *Markdown* http://daringfireball.net/projects/markdown/

[9] W. Hwu. *Heterogeneous parallel programming* https://www.coursera.org/course/hetero

[10] Ideone *Ideone - online compilers and interpreters*. https://ideone.com/

[11] K. Jordan. *MOOC completion rates: The data*. Available at: http://www.katyjordan.com/MOOCproject. html.[Accessed: 27/08/2014], 2013.

[12] H. Lin. *Teaching parallel and distributed computing using a cluster computing portal*. 2013 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum. IEEE, 2013.

[13] Microsoft. *Visual Studio - Microsoft developer tools*. visualstudio.com

[14] F. Pérez and B. E. Granger. *IPython: a System for Interactive Scientific Computing*. Computing in Science and Engineering, vol. 9, no. 3, pp. 21-29, May/June 2007, doi:10.1109/MCSE.2007.53. URL: http://ipython.org

[15] N. Truong, P. Bancroft, and P. Roe. *A web based environment for learning to program*. Proceedings of the 26th Australasian computer science conference-Volume 16:255–264, 2003. Australian Computer Society, Inc.

[16] qwikLABS. *qwikLABS* https://qwiklabs.com/

[17] R. Riddle. *Premilinary results on Duke's third Coursera effort, "Think Again"*. https://cit.duke.edu/blog/2013/06/preliminary-results-on-dukes-third-coursera-effort-think-again. html.[Accessed: 13/01/2016], 2013.

[18] Špaček, František and Sohlich, Radomír and Dulík, Tomáš, *Docker as platform for assignments evaluation*, Procedia Engineering 100 (2015): 1665-1671.

[19] Haydel, N., Gesing, S., Taylor, I., Madey, G., Dakkak, A., de Gonzalo, S. and Hwu, W. , *Enhancing the Usability and Utilization of Accelerated Architectures via Docker*, UCC Cloud Challenge (8th IEEE/ACM International Conference on Utility and Cloud Computing), 7-10 December, 2015, St. Raphael Resort, Limassol, Cyprus.

[20] TutorialsPoint. *Coding ground*. http://www.tutorialspoint.com/codingground.htm

[21] Universitat Politecnica De Catalunya Barcelona. *Programming and tuning massively parallel systems*. http://bcw.ac.upc.edu/PUMPS2015/ html.

[22] Wolfram Research. *The Wolfram Cloud*. https://wolframcloud.com.