Interactive textbooks for parallel and distributed computing across the undergraduate CS curriculum

Elizabeth Shoop
Macalester College
shoop@macalester.edu

Richard Brown
St. Olaf College
rab@stolaf.edu

Suzanne J. Matthews
United States Military Academy
suzanne.matthews@westpoint.edu

Joel C. Adams
Calvin University
adams@calvin.edu

***Abstract*—It has been a decade since the ACM/IEEE CS2013 Curriculum guidelines recommended that all CS students learn about parallel and distributed computing (PDC). But few textbooks for "core" CS courses–especially first-year courses–include coverage of PDC topics. To fill this gap, we have written free, online, beginner- and intermediate-level PDC textbooks, containing interactive C/C++ OpenMP, MPI, mpi4py, CUDA, and OpenACC code examples that students can run and modify directly in the browser. The books address a serious challenge to teaching PDC concepts, namely, easy access to the powerful hardware needed for observing patterns and scalability. This paper describes the content of these textbooks and the underlying infrastructure that make them possible. We believe the described textbooks fill a critical gap in PDC education and will be very useful for the community.**

***Index Terms*—C, C++, computing, education, interactive, MPI, OpenACC, OpenMP, parallel, software, textbook**

I. Introduction

Teaching parallel and distributed computing (PDC) is an increasing requirement in undergraduate computing curricula. Beyond the ubiquity of multicore processors in everyday computers, students interact with computer systems employing distributed communication patterns on a daily basis. Recognizing the need for PDC in the undergraduate curriculum, the *ACM/IEEE CS Curricula 2013* guidelines added PDC to the CS body of knowledge [1], and the *Accreditation Board for Engineering and Technology* (ABET) has required all ABET-accredited CS program graduates to learn about PDC since 2019 [2].

One curricular strategy for exposing all undergraduate CS students to PDC is to incorporate PDC topics throughout the CS curriculum [3], an approach that we might describe as *PDC early and often*. Rather than expose CS majors to PDC via a single required third or fourth year course–by which time sequential thinking is a difficult-to-break habit–the PDC early and often approach includes at least a week of PDC coverage in each of the "core" CS courses that all majors take. By consistently exposing students to PDC topics in each "core" CS course, this approach lets students experience the broad applicability of PDC across the discipline of computing.

Unfortunately, few "core" CS course textbooks incorporate any PDC topics, especially first-year course texts. Instructors wishing to add PDC coverage to such courses have had to develop their own supplemental PDC materials, or coordinate access to PDC resources. While large scale resources such as the NSF-funded ACCESS program make it possible for instructors to coordinate access, the process is often too time-consuming to justify its use for a one week PDC unit in a core CS course.

To alleviate this problem, we have developed *free, online, interactive PDC textbooks* that allow readers to learn about and run PDC code on any web browser. The texts are *interactive*, allowing readers to "tinker" with the PDC code examples: students can run the provided code and view the results, modify the code and then re-run it to see the effects of their modifications, and so on. The books also include other interactive elements, such as videos and "guess and check" questions, to help readers gauge their understanding.

Currently, two texts are available on our learnpdc.org site:

- *PDC for Beginners*, for first- or second-year courses, and
- *Intermediate PDC*, for more advanced courses.

Each book may be used as the course text for a standalone PDC course, or as a supplemental text for a CS "core" course. To support the latter use case, each book is organized into platform-specific sections that can be used as in-class activities or as assigned readings. These textbooks currently contain C and C++ code examples with sections devoted to the OpenMP (shared-memory), MPI (distributed-memory), CUDA (GPU), and OpenACC (GPU) parallel platforms, and Python examples for MPI. There is an example for Python multiprocessing in the *PDC for Beginners* book, but the code example is static. All code examples provided in the books are available for download from our GitHub repository [4]. Note that C and C++ are *compiled* languages. Our inclusion of interactive compiled-language code examples for PDC distinguish our books from other interactive texts that only support interpreted languages or single-core execution. Our implementation of the front- and back-end infrastructure needed to support compiled-language interactivity for PDC is a key contribution of this work.

The rest of this paper provides a detailed examination of these books. In Section II, we discuss work related to this project. In Section III, we describe the contents of the two books. In Section IV, we describe the underlying infrastructure we created to realize our goal of compiled-language interactive execution. In Section V, we present our future plans and concluding remarks.

Funded in part by the National Science Foundation and the Department of Defense

II. Related Work

There are platforms that allow students to learn both compiled and interpreted languages through interactive code examples. Here, we survey some that have been used for teaching high performance computing (HPC) and PDC.

Originally designed for Python coding, a Jupyter Notebook [5] is an open-source system that has a front-end browser-based "notebook" document editor connected to a back-end server that can execute blocks of code within the document. The documents that students use, which can contain textual explanations along with code blocks, are portable and can be shared. A Jupyter Notebook App that enables such code execution can be installed locally or can be installed on a server. Glick and Mache [6] described the use of these notebooks for teaching HPC to undergraduates. They used an extension called NBGrader [7] that enabled them to create assignments by labeling portions of documents as problems, where code solutions are hidden from students until they complete them. The authors created several notebooks with PDC platform extensions to a server at their institution, and successfully used the notebooks to teach their HPC course.

Google has created a free hosted Jupyter Notebook service called Collaboratory, or Colab [8], which improves the ease of use of Jupyter-like notebooks as it eliminates the need to set up a service for running various types of code examples. While the basic Google Colab service has limited computing resources available, due to a primary focus on Python programming and A.I., it does enable users to access a GPU. Xu recently reported success in using Colab in an upper level PDC course for teaching CUDA [9].

Runestone [10] is an open source platform for creating interactive web-based textbooks. The interactivity comes in the form of traditional kinds of problems, such as multiple choice and fill-in-the blank, but also newer kinds specific to computer science, such as Parsons problems and Active Code block examples that readers can edit and run, with output displayed below the code. Among other languages, the Runestone system enables basic C and C++ code examples to be compiled with gcc/g++ and run on a remote open-source server called Jobe (job engine) [11]. Jobe has a RESTful API interface and executes given code on demand, with timeouts and other configurable limitations.

Runestone is designed to enable authors to write books of their own in reStructuredText[1], to enable researchers to extend the code base, and to serve up books on their own servers. We took advantage of these capabilities to build our own version of the Runestone server from a fork of the original code base from early 2023, and we created our own version of the Jobe service that enables us to compile and execute C, C++, and Python code for PDC examples using popular PDC libraries on a multicore server with GPU hardware that we maintain. Using our revised version of Runestone, we have thus far authored two PDC textbooks.

[1]The newest version of Runestone is transitioning from reStructuredText to preText, but we started this work before that transition.

III. About the Books

A. PDC for Beginners

PDC for Beginners is a free, interactive, on-line textbook designed to teach early computing students the basics of parallel and distributed computing. The book assumes that readers have only a CS1 foundation in Python and/or C. In addition to text and code blocks, it uses short video segments featuring visualizations and analogies to introduce PDC concepts to students. Unlike a traditional textbook, *PDC for Beginners*' chapters can be used in a piecemeal fashion to "inject" parallelism into different early CS courses.

Each chapter follows a "crawl", "walk", "run" approach. First, unplugged activities and analogies are used to introduce PDC beginners to some concept at a high level (the "crawl" phase). Figure 1 shows one such analogy (the "pizza eating" analogy) for introducing students to processing elements (PEs, or "pizza eaters" in our analogy).

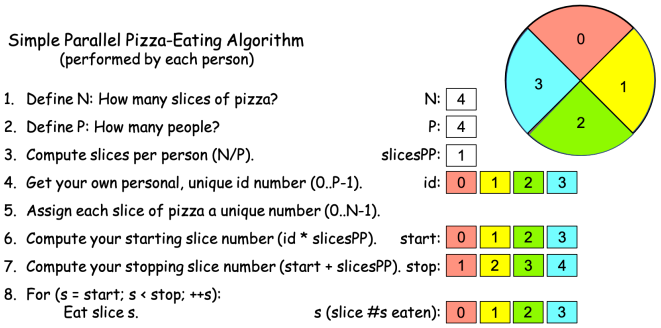

Fig. 1. A Sample Analogy

Once the reader understands a high-level concept, we then present one or more simple code examples, first in the form of small code examples called patternlets [12], which demonstrate the essence of a particular parallel pattern, followed by other approachable examples (the "walk" phase). Lastly, we present a larger, more complex example that shows how the concept can be applied in a real-world context (the"run" phase). These examples are all interactive, allowing the reader to run and/or modify the code and view the results in their browser. In Figure 2, the green 'Save and Run' button compiles, links, and executes the OpenMP C code on a multicore server; that server returns the results, which are then displayed dynamically below the code example.

Runestone also allows such interactive examples to be paired with brief assessments; this allows the reader to check their understanding, as can be seen in the lower portion of Figure 2.

To run interactive examples in MPI, we created a custom Runestone Active Code block that allows the reader to specify the flags passed to the `mpirun` command. Figure 3 shows a screenshot of an interactive MPI example that uses this feature: Readers may run this example and control the number of MPI processes by directly modifying the argument passed to `mpirun` via the `-np` flag.

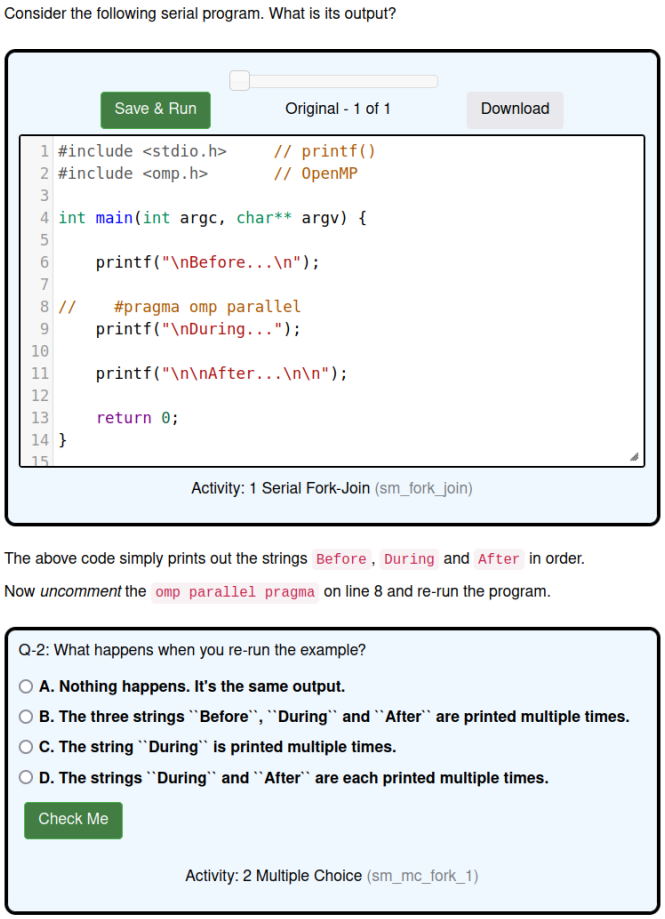

Fig. 2. An Interactive Code Example and Assessment

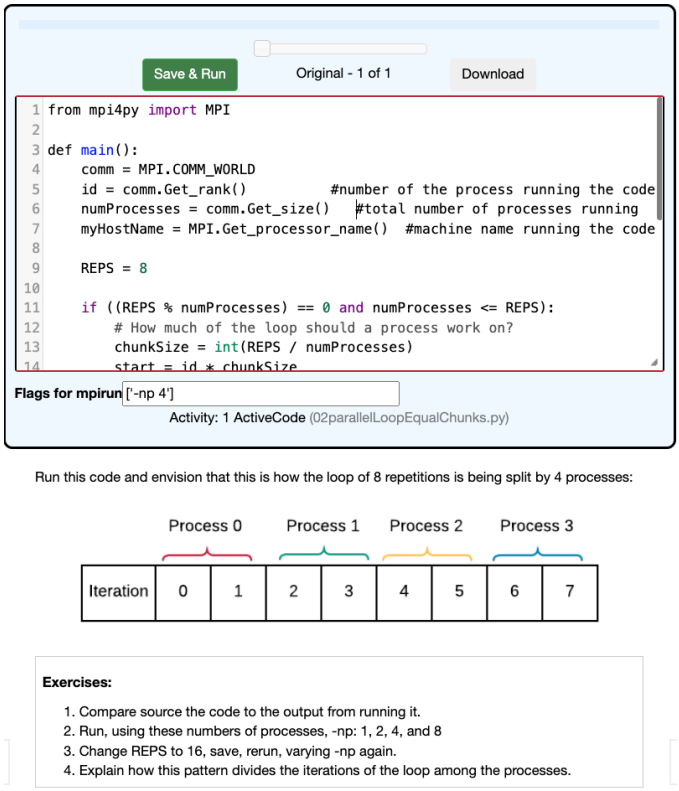

Fig. 3. An Interactive Code Example with MPI

Organization: The *PDC for Beginners* book is organized into the following chapters:

0. PDC Introduction
   Provides foundations of PDC concepts, including types of hardware, how we think about and measure performance, and what types of programming strategies we use. Purely text and figures and agnostic of any particular software library, this represents a 'crawl' chapter that students can refer back to when trying examples in further chapters.
1. Shared Memory Programming with OpenMP
   Provides C and C++ examples, starting with simple patternlet programs [12] (crawl), then classic examples such as trapezoidal rule integration using the reduction pattern (walk), and ending with a simplified drug design program (run).
2. Message Passing using Python
   Contains examples written in Python for mpi4py, starting with simple example demonstrating classic PDC message passing patterns (crawl), then what can go wrong and slightly more complex examples with exercises (walk), and finally showing an example of simulations of a forest fire (run).
3. Common algorithmic patterns
   Highlights two contributions: 1) how parallel selection works in both OpenMP and MPI, and 2) an example using the shared queue of Python multiprocessing.
4. GPU Computing Basics with CUDA C
   Provides the crawl/walk approach to the CUDA GPU programming model, including small code examples, then uses two versions of a vector addition example to let students run.
5. A Deeper Dive into CUDA
   Extends from the previous basics chapter to show the effects of block size choices, how to time code, and measuring performance.
6. Applications
   Here we have 2 larger applications with detailed descriptions, implemented in more than one PDC library so the types of platforms can be compared: 1) the AI Minimax algorithm in serial, OpenMP and MPI, and 2) matrix multiplication in serial, OpenMP and CUDA.

Guest Authors: To provide different perspectives, we invited additional authors to contribute sections to *PDC for Beginners* on common algorithmic patterns and applications. Contributing authors were asked to follow the same crawl, walk, run approach in their sections of the book, though in more condensed form. Three additional authors contributed to the

PDC for Beginners text, writing sections on the Shared Queue data structure, Parallel Selection, and a parallelization of the Minimax algorithm, commonly used for AI applications. All submissions were reviewed and edited before being included in the book.

Use Cases: Unlike a traditional textbook, we do not anticipate that this book be adopted in full in a particular core CS course. Rather, we imagine that individual chapters or sections be used as required reading or used for in-class activities in different early CS courses. For elective or upper-level courses focusing more heavily on PDC, content in this introductory book can be used to introduce PDC concepts before moving on to the *Intermediate PDC* book described below.

Here are some suggested scenarios for using the *PDC for Beginners* text:

- Most students should find chapter 0 useful as background reading to introduce PDC concepts. In particular, sections 0.3 and 0.4 describe PDC concepts using analogies instead of code and provides a conceptual basis for the code examples in subsequent chapters.
- Students in CS1, CS2 or Systems who have some familiarity with a C-family language should be able to actively try Chapter 1 in a class period.
- Any students at any level who can read Python should be able to actively work through the examples in Chapter 2. In that chapter we also suggest how the sections can be split over class or lab periods.
- Upper-level courses that cover message passing may also choose to adopt content from chapter 2 because the Python examples are fairly straightforward.
- An algorithms course may want to delve into a subset of the sections in Chapter 3 in their course after assigning some of the background reading from Chapter 0. Also, the application in Chapter 6, section 1 described below might be of interest.
- Chapters 4 and 5 (CUDA programming on GPUs) are likely applicable to students who have some experience with C or C++. Chapter 4 is designed to be able to be attempted in a 1.5 hour lab. Chapter 5 carries on from Chapter 4 if you want to spend more time getting into a few more details of CUDA programming. We have used these in both systems courses and as a lead-in to OpenACC in an advanced course.
- In 6.1, the AI minimax algorithm is described, along with parallel implementations in OpenMP, so the shared memory from chapter 1 is necessary background. Chapter 2 on MPI is background for the MPI example presented here.
- In 6.2, the matrix multiplication application implementations in OpenMP and CUDA could be follow-on activities after either Chapter 1 or 4.

B. Intermediate PDC

Whether trying to introduce parallelism early in the undergraduate CS curriculum or not, many departments still choose to offer an upper-level undergraduate course focusing on PDC and/or high performance computing (HPC). For such courses, we have developed a second textbook. However, it is also possible to use portions of the book in other upper-level courses that are not focused on PDC.

Since this book is meant for more advanced students, we modify the approach for introducing material. As in *PDC for Beginners*, *Intermediate PDC* begins with an overview of common patterns in parallel programming, grounded by past research [13], [14]. Through the chapters, we emphasize what patterns are used when introducing the active examples. The chapters start with more of a "walk" focus, though with plenty of examples, since advanced students should have the background to comprehend them. Next, students progress to the "run" phase with more complex examples and suggested projects.

Organization: This book's current chapters are described below. This book is in active development, and we plan to add more chapters in the future (see Future Work section).

1. PDC Patterns
   Parallel patterns are organized into a diagram and broken down and described, in the context of both shared memory with OpenMP and message passing with MPI.
2. Shared Memory Patterns with OpenMP
   Provides active examples of most low-level patterns described for shared memory systems, using C OpenMP code. Also shows how each code example fits into the patterns diagram from chapter 1.
3. Random Number Generators for PDC
   Presents short examples using a parallel-safe pseudo random number generator (PRNG) library that guarantees uniform distribution and reproducible results regardless of number of PEs used. Includes diagrams showing two common approaches.
4. Message Passing Parallel Patterns
   Provides active examples of some low-level patterns described for distributed systems, using C MPI code. As in Chapter 2, shows how the examples match with the patterns diagram of Chapter 1.
5. Message Passing: Combining Patterns
   Digs deeper into message passing examples by combining multiple patterns in each example.
6. Message Passing Example Applications
   Presents full MPI applications written in C, one of which uses the PRNG library from Chapter 3.
7. OpenACC Basics
   Compares sequential, OpenMP, multicore (using OpenACC), and GPU (using OpenACC) solutions to the classic vector addition problem.
8. OpenACC: Next Steps
   Presents approachable OpenACC examples of 2-dimensional matrix calculations on the GPU, along with suggested exercises. Presents scalability results that let the reader directly experience the benefits of this technology.

Use Cases: The *Intermediate PDC* book may be used during a full semester/quarter PDC course. Instructors who wish to emphasize particular technologies may pick certain chapters to fit their needs. Here are some additional suggested scenarios for using sections of the book outside of that context:

- Chapter 1 introduces parallel computing patterns, and serves as background reading for subsequent chapters.
- Instructors in a sophomore level Systems course using C who want to include a week or more on shared memory parallel computing may use Chapter 2 in a class-lab setting. The chapter has 16 examples that start from base principles and progress to more complex and nuanced concepts; instructors can stop after the first 8 examples and students will have tried the most important concepts.
- Instructors who wish to cover message passing but prefer C to the Python examples in the beginner book can use Chapter 4 as a classroom activity. For those wishing to go further, Chapter 5 provides additional content.

C. Our Experiences

While creating these books we have taken opportunities to use portions of them in our courses and in our undergraduate research experiences. Shoop used *PDC for Beginners* as a two-day intervention in a sophomore-level systems course taught by a colleague. Reading from Chapter 0 was assigned as background before class, and students completed the exercises in Chapter 1 in a one-hour class. The following class day, Shoop provided a brief presentation about the GPU and its programming model. She then was able to demonstrate the matrix multiplication example from chapter 6.2, showing the remarkable additional speedup and scalability of the CUDA version over the already reasonably scalable OpenMP version. This type of intervention inpressed the students and made them interested in taking the advanced PDC course the next semester, which some did. In that PDC course, Shoop used a great deal of *PDC for Beginners* as starting material, then transitioned to the *Intermediate PDC* book. Not all of it was completed before the course ended, but many of the book's examples (in a different format) were used for activities. Matthews did a similar guest appearance in a Systems course lab, where she used the CUDA material from Chapter 4 as an activity. She has also used portions of the book as starting points for students working on independent research projects who need to learn to use a particular HPC platform.

IV. Implementation Details

In order for the interactive content of these two books to work, it was necessary to design and build the "behind the scenes" components needed to run PDC code in the browser. While Runestone Interactive [10] supports a significant amount of interactivity within books, its Jobe server for performing computations in compiled languages does not natively support the ability to run the parallel and distributed code we require. To allow instructors and students to compile and execute PDC code interactively within Runestone, we:

1) Updated the open source Runestone code base to support Active Code blocks for PDC languages and libraries (the front end of the web application).
2) Extended Jobe to become an intermediary service in the case of PDC, which forwards requested computations to a custom-built "execpdc" backend compute server.
3) Wrote chapters and recruited authors to write sections for the beginner book.
4) Procured hardware to host the books and servers.

We now describe these activities in greater detail.

A. Front end changes to Runestone

Our front end changes to the Runestone code provide new options for our textbook authors, and are not directly visible to the instructors and students who use our interactive books. Our goal for redesigning and improving Runestone Active Code blocks (also known as Activecode directives) was to provide an extensible way to handle multiple PDC platforms, with different compilers and runtime capabilities. An Active Code directive supporting PDC compilers needs to determine: (i) that the code is for a PDC technology (versus a traditional serial job), and (ii) which PDC compiler to use.

The top of Figure 2 shows an example of how a Runestone Activecode directive renders in the book. This particular example shows a snippet of OpenMP code. Note that students and instructors see and interact with the code as shown there.

Textbook authors write the books in restructured text and use a special Runestone directive called an Activecode block to produce these. Since OpenMP is built-in on all modern versions of gcc (and Runestone and Jobe run gcc natively), this particular code block can technically be implemented using the original Runestone Activecode directive (see Figure 4A) by including the C linker flag `-fopenmp` in the directive's associated `:linkerargs:` tag in the reStructuredText (rst) file. Runestone makes the design choice to hide linker arguments from students to make learning easier in texts for introductory programming courses. We followed this design philosophy of hiding this information for our textbook examples.

Figure 4A depicts the original way of writing C-language Active Code blocks in Runestone, and Figure 4B shows our revision for sending PDC details to our modified Jobe service. Notice we have changed the `:language:` directive from `C` to a more generic `pdc` and added a new directive for the compiler called `:compiler:`, enabling an author to explicitly choose a compiler. Thus far, we have added support for the following tags for the `:compiler:` directive for our code blocks:

- `gcc` and `g++` for OpenMP,
- `mpicc` and `mpi4py` for MPI,
- `nvcc` and `nvc++` for CUDA,
- `pgcc` and `pgc++` for OpenACC

The compiler the author chooses also provides a clue to the back-end server for the hardware that should be used (see Section IV-B for more details).

Runestone's C Active Code blocks will reveal command line arguments if authors add them, and we make use of

```
.. activecode:: sm_fork_join
   :language: C

   :linkargs: ['-fopenmp']
   :enabledownload:
   :caption: Serial Fork-Join

   #include // printf()
   #include // OpenMP

   int main(int argc, char** argv) {

      printf("\nBefore...\n");

      // #pragma omp parallel
      printf("\nDuring...");

      printf("\n\nAfter...\n\n");

      return 0;
   }
```

A. Original Runestone Active Code block

```
.. activecode:: sm_fork_join
   :language: pdc
   :compiler: 'gcc'
   :linkargs: ['-fopenmp']
   :enabledownload:
   :caption: Serial Fork-Join

   #include // printf()
   #include // OpenMP

   int main(int argc, char** argv) {

      printf("\nBefore...\n");

      // #pragma omp parallel
      printf("\nDuring...");

      printf("\n\nAfter...\n\n");

      return 0;
   }
```

B. Redesigned PDC Active Code block

Fig. 4. Changes for PDC to Runestone Active Code blocks

them often in the intermediate book. In addition to letting non-novice readers update command line arguments, it is sometimes instructive to let readers experiment with compiler flags, so we deviated from the original Runestone design choice (which hides them) and chose to expose them.

Figure 5 shows a web rendering of this for an OpenACC example. The block allows authors to set and display default command line arguments for C/C++ active code examples along with any compiler flags. Users can change and re-run the example with modified arguments or flags.

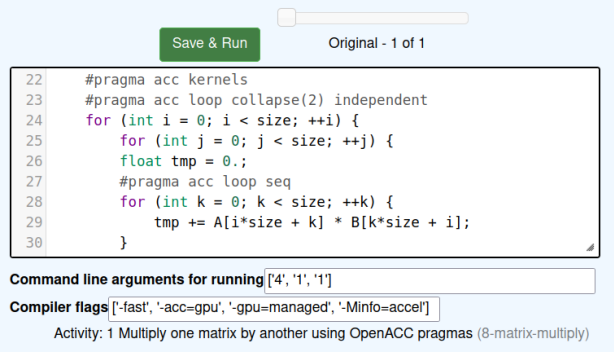

Fig. 5. Exposing the compiler flags to readers

Figure 6 shows what the beginning of the Active Code block in the underlying rst file looks like for the example in Figure 5. Note the use of the `:runargs:` tag for command line and the `:compileargs:` tag for compiler flags. These Active Code blocks also allow for default compiler options within the service that compiles the code, so for beginner examples we can hide these by eliminating the `:compileargs:` tag in the rst code. Similarly, when examples don't call for special command line arguments, we don't use them and they aren't rendered in the book.

These changes to Runestone's Active Code block allow us to compile and run code from multiple PDC compilers. MPI was an interesting challenge because the executable needs to

```
.. activecode:: 8-matrix-multiply
   :language: pdc
   :compiler: 'pgcc'
   :compileargs: ['-fast', '-acc=gpu', '-gpu=managed', '-Minfo=accel']
   :linkargs: ['-fopenmp', '-lm']
   :runargs: ['4', '1', '1']
   :caption: Multiply one matrix by another using OpenACC pragmas
```

Fig. 6. Exposing the compiler flags to readers

be run via `mpirun`, which can have its own flags. Since Runestone's Activecode directive already had a tag called `:interpreterargs:` (originally designed for Java and Python), we chose to adapt that for the flags that go with `mpirun`. When present in the code block, these are also rendered in the book, so they are visible to the reader for MPI C/C++ or mpi4py code. The only MPI flag we have used so far is `-np`, to indicate how many processes to use.

B. *Jobe as an intermediate service, and back end execution*

In general, parallel and distributed computations may call for a bewildering range of hardware systems (large multicore hosts, networked clusters of high-performance nodes, accelerators such as GPUs, etc.) and software platforms (compiled and/or interpreted languages, plus libraries such as MPI, OpenMP, Cilk+, etc.), as well as heterogeneous combinations of these (e.g., MPI jobs involving OpenMP and/or CUDA on each node). Even our beginners text assumes support for C and Python, MPI and OpenMP, and CUDA and OpenACC for GPU computing. More advanced books could benefit from support for additional hardware/software platforms.

Traditional Runestone books focus on a single software platform (e.g., Python or C++ language, or a database management system). In many cases, Runestone implements those computations directly within the user's browser (e.g., using a web assembly implementation of Python, or a subset of standard Python sufficient for that book). Code in Active Code blocks for compiled languages (e.g., C or C++) are sent directly to a Jobe server, which traditionally runs in a single-core container that has support for Jobe's pre-configured programming languages. This constrains the kinds of programs that an unmodified Jobe server can perform.

By contrast, we want readers of our PDC Runestone books to learn from Active Code examples depicting specialized PDC computations. For example, a beginners exercise might have a student explore scalability by running an Active Code block and varying the number of threads or processes; exercises for more advanced students might compare scalability of PDC computations on a multi-core machine vs. a cluster system.

To support such a wide range of computational options, we created a custom Jobe extension for PDC computations, which forwards `pdc` computations to an "execpdc" backend server that executes code from an Active Code block in a hardware+software platform specified by that block. Figure 7 depicts the Runestone server and our extended Jobe server in green ovals, and an "execpdc" backend server in a yellow

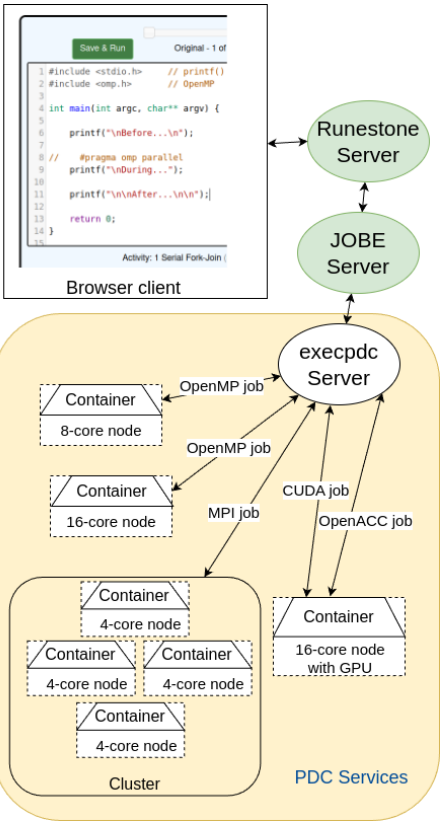

Fig. 7. Design of new Runestone system

region, capable of performing computations on diverse hardware+software systems.

Our design, in which Jobe becomes an intermediate server rather than a compute server, requires minimal modification to Jobe itself: we add just one custom Jobe extension. This extension parses Runestone options given in an Active Code block (describing a desired hardware+software system, and per-job parameters such as compiler flags or number of threads/processes), then forwards that desired computation to an "execpdc" backend server for execution. By separating this functionality, the machine running our modified Jobe server does not itself require any specialized PDC compute capabilities. Also, the "execpdc" backend server can be configured as needed (e.g., to accommodate a particular library or version of CUDA), without further modifications to Runestone or Jobe.

C. Authoring book material

Our books are authored in reStructuredText, and maintained in GitHub as a series of `.rst` files. Typically, each section of each chapter is its own file. Figures 4 and 6 provide a glimpse of the Active Code blocks in such files. There is a learning curve with rst, but we had prior experience from writing PDC module materials, plus Shoop has written another Runestone book, which reduced the learning curve. We found it fairly straightforward to add other active-learning features, such as the multiple choice question shown in Figure 2.

We welcome ideas from the community about contributing to this project going forward; we give recognition in the book and assign a DOI to each contributor's work. We are happy to take ideas and help with the authoring of the rst files.

D. Hosting

After creating material in reStructuredText files, we use the Runestone system to build the web-based books from these files. The building process creates fully active HTML and Javascript files that can be hosted on any web server, or the book can be hosted on a complete Runestone server, which has a DBMS to hold information about user logins, what activities they have completed, and many other features.

Since we are still developing material and the books will continue to evolve, we chose to host the books on a web server rather than a complete Runestone server. Currently, the front-end component of the book resides on a small VM on a CS department system at Calvin University; the green oval labeled "Runestone Server" in figure 7 is a simple Apache web server. In the future we plan to incorporate a full Runestone Server's capabilities, along with other enhancements described below. Likewise, all of the PDC services shown in the yellow / bottom portion of Figure 7 currently reside on a single 96-core Calvin University server.

V. Conclusions and Future Work

We have presented two free, online, interactive PDC textbooks that students and faculty can use now to learn about PDC concepts. Our primary contributions are as follows: (i) we–and invited guests–authored two brand-new textbooks that cover beginning and intermediary PDC content; (ii) we extended Runestone Activecode directives with features needed to support PDC code; (iii) we built an enhanced version of the Jobe server as an intermediate service; and (iv) we built an "execpdc" back-end server that permits PDC code to be compiled and executed on a variety of parallel platforms from within the browser. Our book chapters employ a modular design, making them useful in variety of curricular contexts. Thanks to our back-end design, it is relatively straightforward to add support for additional languages or systems.

The textbook's front-end and back-end work well for the current level of use, but the back-end's scalability is ultimately limited by its 96-core host. To avoid this limitation, we are currently transitioning the back-end to the Google Cloud Platform. Using the cloud will enable us to spin up resources on demand and scale up during times of heavy use. Thanks to funding from the Department of Defense (DoD), the PDC computing resources for these books should be available for at least 5 years after this transition is complete.

We plan to continue adding material to the *Intermediate PDC* book, which we have authored entirely ourselves so far. We intend to add more examples for most compilers and platforms, and we also plan to include a chapter devoted to suggestions for projects or assignments for a PDC course. We hope to reach out to authors of Peachy Parallel Assignments [15] to include some of those submissions.

We also plan to use the same system to host a new advanced PDC book, whose development is currently underway. We expect to add OpenMP 5 support for GPU computations, an alternative to OpenACC, and we have begun exploring Chapel examples that we may include in this book, as well as OpenCilk [16] examples that emphasize performance engineering. We welcome contributions from the PDC community for this work; to participate, please contact the authors.

ACKNOWLEDGMENTS

We are grateful for the contributions of Dorian Arnold, Steven Bogaerts, and John Rieffel who guest-authored sections of *PDC for Beginners*.

This work was supported by the National Science Foundation, DUE-1822480/1822486/1855761 and by additional funding from the Department of Defense. Research was sponsored by the United States Military Academy (USMA) and was accomplished under Cooperative Agreement Number W911NF-23-2-0044. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

REFERENCES

[1] Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, *Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*. New York, NY, USA: Association for Computing Machinery, 2013.

[2] M. J. Oudshoorn, S. Thomas, R. K. Raj, and A. Parrish, "Understanding the new ABET computer science criteria," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 429–434. [Online]. Available: https://doi.org/10.1145/3159450.3159534

[3] R. Brown, E. Shoop, J. Adams, C. Clifton, M. Gardner, M. Haupt, and P. Hinsbeeck, "Strategies for preparing computer science students for the multicore world," in *Proceedings of the 2010 ITiCSE working group reports on Working group reports*, ser. ITiCSE-WGR '10. New York, NY, USA: ACM, 2010, pp. 97–115, ACM ID: 1971689.

[4] "CSinParallel Project Code Repository," https://github.com/csinparallel/CSinParallel, accessed: 01-24-2024.

[5] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows." *Elpub*, vol. 2016, pp. 87–90, 2016.

[6] B. Glick and J. Mache, "Using jupyter notebooks to learn high-performance computing," *J. Comput. Sci. Coll.*, vol. 34, no. 1, p. 180–188, oct 2018.

[7] J. B. Hamrick, "Creating and grading ipython/jupyter notebook assignments with nbgrader," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*, 2016, pp. 242–242.

[8] "Collaboratory FAQ," https://research.google.com/colaboratory/faq.html, accessed: 01-20-2024.

[9] Z. Xu, "Teaching heterogeneous and parallel computing with google colab and raspberry pi clusters," in *Proceedings of the SC '23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis*, ser. SC-W '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 308–313. [Online]. Available: https://doi.org/10.1145/3624062.3624095

[10] B. J. Ericson and B. N. Miller, "Runestone: A platform for free, on-line, and interactive ebooks," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1012–1018. [Online]. Available: https://doi.org/10.1145/3328778.3366950

[11] R. Lobb, "JOBE," https://github.com/trampgeek/jobe, 2018, accessed: 01-20-2024.

[12] J. C. Adams, "Injecting parallel computing into cs2," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 277–282. [Online]. Available: https://doi.org/10.1145/2538862.2538883

[13] T. G. Mattson, B. Sanders, and B. Massingill, *Patterns for parallel programming*. Pearson Education, 2004.

[14] K. Keutzer, B. L. Massingill, T. G. Mattson, and B. A. Sanders, "A design pattern language for engineering (parallel) software: merging the plpp and opl projects," in *Proceedings of the 2010 Workshop on Parallel Programming Patterns*, ser. ParaPLoP '10. New York, NY, USA: Association for Computing Machinery, 2010. [Online]. Available: https://doi.org/10.1145/1953611.1953620

[15] "Peachy Assignments," https://tcpp.cs.gsu.edu/curriculum/?q=peachy, accessed: 01-20-2024.

[16] "OpenCilk," https://www.opencilk.org/, accessed: 01-24-2024.