

A Free Online Textbook and Interactive Exercises for Introducing Parallel Computing

Suzanne J. Matthews

U.S. Military Academy

West Point, NY, USA

Email: suzanne.matthews@westpoint.edu

Tia Newhall

Swarthmore College

Swarthmore, PA, USA

Email: newhall@cs.swarthmore.edu

Kevin C. Webb

Swarthmore College

Swarthmore, PA, USA

Email: kwebb@cs.swarthmore.edu

Abstract—We present the parallel and distributed computing (PDC) content of *Dive into Systems*, a free, online textbook that covers a broad range of computer systems topics. *Dive into Systems* introduces many topics in the TCPP curriculum including two chapters devoted to parallel computing. Our textbook is written assuming only a CS1 background of the reader, and it is designed to be used as a primary textbook for a range of courses. As a supplement to our textbook, we developed online interactive learning resources that provide practice to students learning its content. These resources include interactive end-of-chapter exercises and novel Ask me Another components, including four specific to PDC. Ask me Another components generate new instances of their question type, allowing a user to generate a virtually infinite number of practice questions. We also present results of a multi-institutional study that piloted one of our PDC Ask me Another components.

I. INTRODUCTION

Expensive textbooks represent a financial barrier to students, limiting access to those who can afford them. Free online textbooks help level the playing field by ensuring that their content is readily available to anyone with an Internet connection. While there are many such free resources for introductory programming (for example [1], [2]), there is a dearth of such options for introducing computer systems and parallel computing.

We present *Dive into Systems* [3], a free online textbook with interactive exercises that introduces a broad range of computer systems and parallel computing topics. While it is primarily designed to be used as the principal textbook in a computer systems course, it also serves as a valuable resource for background reading in upper-level courses. The online format of our textbook allows us to seamlessly update content and to easily support students who are learning in remote environments, a need that was highlighted by the 2019 COVID pandemic.

Dive into Systems is one of a very few free online resources that covers parallel computing assuming only a CS1 background of the reader, and it's the only one that covers a large range of different systems topics at this introductory level. Our book introduces many topics in the NSF/IEEE-TCPP curriculum [4], and it focuses in depth on shared memory parallelism as the reader's first introduction to parallel computing. The coverage demonstrates how to introduce parallel content throughout a systems course, making it easier for instructors to incorporate PDC into their curricula.

The online format of our textbook makes it ideal for adding interactive learning tools, which we incorporate in three forms. First, we added interactive exercises for readers to practice their understanding of the book's material. Second, we developed several novel Ask me Another components, four specific to PDC topics, that provide a user the ability to generate an effectively infinite number of practice question instances. Finally, we constructed standalone visualization tools that support other TCPP topics, like ASM Visualizer [5], which helps students trace complex blocks of assembly code. Collectively, these online interactive tools provide learners hands-on practice with challenging systems (and PDC) topics.

To ensure that the exercises were widely applicable to a large range of courses and institutions, we recruited faculty from various institutions to be our exercise developers. Since 2022, we have run an exercise development program under our NSF grant [6] to add a broad array of interactive exercises to *Dive into Systems*. The exercise developer program is just one example of many ways in which we have incorporated external experts into the development of *Dive into Systems*. Throughout the process of writing the textbook and developing its associated learning resources, we sought input and reviewers of our work, with the goal of making these resources applicable to the larger community and to a broad range of uses.

In this paper, we focus on *Dive into Systems*'s PDC topics and PDC-specific interactive learning resources. Our book's primary PDC coverage is in two chapters dedicated to parallel computing. One chapter focuses on shared memory parallelism, which we use to first introduce the reader to parallelism and to provide some depth of coverage to promote parallel thinking skills. Prior chapters build up to shared memory parallelism with earlier coverage of multicore processors, processes, virtual memory, and concurrency, making shared memory parallelism a natural next step. Another chapter presents a "looking ahead" overview of other types of PDC to expose the reader to other parallel paradigms. Although our book is specifically targeted to introductory computer systems and parallel courses, it has been used by a large variety of courses, including in some upper-level CS courses. Currently, *Dive into Systems* is in use by over 65 different institutions.

II. RELATED WORK

We were initially motivated to create *Dive into Systems* due to our own difficulty finding a single textbook that covered all of the topics (PDC and others) that we wished to cover in our introductory, required systems courses. While there are several textbooks [7]–[10] that cover PDC, their costs may be prohibitive to students [11], or they are designed for use in more advanced courses.

Existing systems textbooks that are freely available, while excellent, are also written for an audience with more than a CS 1 background [12]–[16], and therefore they are not suitable to be used as the primary textbook in an introductory computer systems course. For example, *Operating Systems in Three Easy Pieces* [15] and *ThinkOS* [17] are designed for use in an operating systems course, while *Intermediate PDC* [16] is designed for a dedicated parallel computing course. Other excellent free online systems textbooks (such as *Computer Science from the Bottom Up* [18]) either do not cover parallelism, or are not designed to be used as primary course textbooks [12], [13], [19], [20].

Consequently, faculty wishing to incorporate PDC into systems courses traditionally needed to cobble together PDC coverage using multiple sources. For example, the CDER [21] project provides community-aggregated PDC courseware [21], while the CSinParallel [22] project offer curated modules for learning parallelism. LearnPDC.org [23] contains several Raspberry Pi-based modules for learning PDC content, while PDC Unplugged [24] offers a repository of unplugged activities. However, practice exercises are largely absent from these materials. Filling these gaps and coming up with novel exercises can be burdensome to faculty new to teaching PDC.

Online exercises for practicing PDC concepts do exist, although they are generally rare; both the *PDC for Beginners* [19] and *Intermediate PDC* [16] textbooks are built using the Runestone [25] framework, and have several excellent custom Runestone exercises that allow learners to explore and run PDC programs in the browser. However, the number of exercises that allow students to “guess and check” core knowledge topics is currently limited in these texts.

Dive into Systems, in addition to being a free online computer systems textbook with introductory-level coverage of PDC, also comes with a rich collection of interactive end-of-chapter exercises [26] designed to help readers gauge their level of understanding and their application of chapter topics. Our exercises are built using the Runestone [25] framework, and appear as a companion booklet to the main text. As such, they can be used in conjunction with *Dive into Systems* or independently. The book’s exercises also include several of our custom PDC Ask me Another components that provide nearly endless practice to the reader on the question topic of the component. The automatic question generation functionality of PrairieLearn [27] most closely resembles our work, but at present, PrairieLearn’s PDC content is very limited.

III. DIVE INTO SYSTEMS OVERVIEW

The *Dive into Systems* textbook introduces a broad range of computer organization, architecture, systems, and parallel computing topics. Because of the potentially large range of different uses of our book, we worked to limit chapter dependencies as much as possible so that an instructor can mix and match content to best fit the needs of their course. In the preface of *Dive into Systems* we suggest how chapters can be used in different types of courses, and we discuss our own experiences on how we have integrated the textbook into required systems courses at our institutions in prior work [3], [28].

The set of topics we cover, and the depth in which we cover them, is in part guided by the introductory nature of our textbook, leaving more advanced coverage to textbooks that are specifically designed for upper-level OS, compilers, architecture, parallel, or distributed computing courses. The presentation of content is also guided by three main themes that unify the set of topics and the foci of their coverage across chapters. The first main theme is *how a computer system is designed to run programs*. In exploring this theme, the book takes the reader through a vertical slice of a computer system, explaining how a program expressed in a high-level programming language is executed by the low-level circuitry of the computer hardware, from translation to assembly, binary representation, CPU and system architecture design, and OS abstractions for running sequential and parallel programs. The second unifying theme is *how to evaluate systems costs associated with a program’s performance*. We discuss this theme in the context of the memory hierarchy and CPU caching, in evaluating the performance of OS overheads including its support for multiprogramming, and in the context of parallel computing performance and overheads. The final unifying theme is *how to leverage the power of parallel computers*. Chapter 14 introduces shared memory parallelism and includes detailed coverage of Pthread programming and synchronization. Chapter 15 is a “looking ahead” chapter that provides an overview of other types of parallel systems and parallel programming models, namely accelerators and CUDA, distributed memory and MPI, and Cloud and MapReduce. In addition, we cover many other PDC topics in other chapters, primarily in our coverage of computer architecture, CPU caching, and operating systems.

We specifically chose the shared memory model to introduce parallelism in our textbook, primarily due to the ubiquitous nature of multicore architecture. Our discussion of threads and shared memory also follows naturally from other content in earlier chapters such as multicore architectures, processes, concurrency, and virtual memory. Another reason for choosing shared memory parallelism as our focus is that it is generally viewed as a model that is easier to understand than message passing, so we feel that it is a good choice for a first introduction to parallel computing and parallel thinking. We also choose to focus primarily on a single parallel paradigm to allow us to go more in depth on this one model vs higher-

TABLE I
2020 NSF/IEEE-TCPP COVERAGE OF *Dive into Systems*

TCPP Topic Area (sub area): specific topics	Book Chapters
Pervasive: asynchrony, concurrency and dependency, locality, performance	Ch. 5, 11, 13, 14
Arch (Classes of Parallelism): Flynn’s taxonomy, ILP, SIMD/vector, pipelines, MIMD, multi-threading, multicore, heterogeneous, cache coherence	Ch. 5, 11, 14, 15
Arch (Underlying Mechanisms): caching, thread safety, consistency, coherence, false sharing, interrupts, processes, virtualization	Ch. 5, 11, 13, 14
Arch (Performance, Scaling & Others): IPC, fault tolerance, IEE 754 standard	Ch. 4, 5, 15
Programming (Paradigms and Notations): Concurrency and parallelism, SIMD, Shared Memory, Distributed Memory, Client-Server, Peer-to-Peer, Task/Thread Spawning, SPMD, Data Parallel, MapReduce	Ch. 14, 15
Programming (Semantics): Tasks/Threads, Synchronization, Deadlock, Race conditions	Ch. 14
Programming (Performance): computation decomposition strategies, program transformations, false sharing, data locality, performance metrics	Ch. 11, 12, 14
Algorithms (Performance): speedup, efficiency, scalability, MapReduce, sorting	Ch. 14, 15

level coverage of many types of parallelism. We believe that our approach best promotes the initial development of the reader’s parallel thinking skills and provides them with a deeper development of PDC skills through our coverage of race conditions, synchronization, deadlock, and parallel performance. The coverage of synchronization includes examples that require mutual exclusion, barrier, and producer-consumer-type synchronization.

Table I summarizes the parallel coverage in *Dive into Systems* as aligned to the NSF/IEEE-TCPP topic areas [4]. We expand on the details of our primary coverage in Section IV.

Initial Reaction to *Dive into Systems*

In 2021, we ran an early adopter program [3] in which faculty across the United States piloted the beta version of our textbook as the primary text in their classes. As part of the program, we surveyed both students and faculty participants to gather feedback on our textbook’s use at different institutions and for different types of classes. The reception was overwhelmingly positive; 89.4% of faculty felt that *Dive into Systems* fit their courses either “Well” or “Extremely Well”, while 92.3% of students liked the book more or equally as much as other textbooks. Both faculty and students appreciated the clarity of presentation and the engaging nature of both the examples and figures. These results, along with our continued experiences using the book and informal feedback from external faculty, have revealed an enthusiastically positive sentiment toward its content and its fit for introductory systems courses.

IV. MAIN PDC CONTENT

The main PDC content in our textbook is in Chapter 14, which introduces shared memory parallelism. Chapter

15 examines other parallel architectures and programming models. In addition, PDC coverage is interspersed in earlier chapters, and our book covers many other foundational topics recommended by the NSF/IEEE-TCPP curriculum. Throughout chapters 14 and 15, we repeatedly use two examples to illustrate unifying themes in parallelism. The first is *ScalarMultiply*, in which the elements of a large vector are multiplied by the same scalar value in parallel. It is a classic candidate for data parallelism, and we use it to illustrate how the same operation can be parallelized over multiple PDC architectures. The second is *CountSort*, a linear sorting algorithm, that we use to explore more complex threading concepts such as race conditions, mutexes, false sharing and thread safety.

A. Shared Memory Parallelism

Chapter 14 (leveraging shared memory in the multicore era) covers multicore architectures and how to program them. We start by introducing Moore’s law and the power wall and discuss their impact on processor design, leading to the development of multicore and manycore processors. We present heterogeneity in our discussion of how not all cores are equal, discussing CPU cores vs GPU cores along with the performance and efficiency cores used in many of today’s architectures. We also show students how to learn about the multicore processors on their systems by running the `lscpu` command.

Next, we move on to a general discussion of processes vs threads and illustrate how processes and threads execute differently on multicore systems, motivating the power of multithreading. We also introduce parallel thinking through our presentation of *ScalarMultiply*, describing how this example can be parallelized using the threading model.

Having introduced the idea of a thread, we next cover POSIX threads (Pthreads) programming starting with a simple *HelloWorld* threading example, which we use to introduce the primary functions for thread creation, joining, and the concept of a thread function. We also describe how to run multithreaded programs on the command line and highlight the nondeterministic nature of thread execution. We then show a Pthreads implementation of *ScalarMultiply*, using this example to highlight thread mechanics, including creating and joining threads, passing multiple arguments to threads, and load balancing them.

Next we use the *CountSort* example to reinforce how to design a parallel program using threads, and also to illustrate and identify race conditions. We continue to use *CountSort* to discuss the performance consequences of adding mutexes and mutex placement in multithreaded programs. Additionally, there are separate sections in this chapter that discuss deadlock, thread safety, and other synchronization primitives, such as condition variables, barriers and semaphores. We also cover several performance topics such as speedup, efficiency, Amdahl’s Law, scalability, and Gustaffson-Barsis law. Near the end of the chapter, we again use *CountSort* to identify and resolve an instance of false sharing, reintroducing cache coherence in the process.

Lastly, we revisit the `ScalarMultiply` example at the end of the chapter to illustrate how it would be implemented using OpenMP.

B. Looking Ahead: Other Parallel Systems

Chapter 15 (*Looking Ahead: Other Parallel Systems and Parallel Programming Models*) is the book's final chapter. Rather than focusing in depth into a single parallel paradigm, it aims to provide students a broad overview of the mechanisms and applications of parallel computing beyond a shared memory model. The chapter begins by introducing Flynn's taxonomy, after which it branches into three sections that each characterize different parallel models. All three sections demonstrate their respective topics using the same `ScalarMultiply` example, which helps illustrate similarities and differences across the different parallel paradigms.

The first section introduces heterogeneous computing in the form of GPUs, FPGAs, and ASICs. We primarily focus on GPUs, using the NVIDIA architecture as an exemplar. We describe general-purpose GPU computing and the CUDA programming language and how it relates to GPU hardware. We then discuss an example `ScalarMultiply` CUDA program and briefly examine CUDA thread synchronization.

The second section focuses on message passing and distributed memory systems. We begin with discussing distributed architecture, comparing and contrasting COTS clusters and supercomputers. We continue with a discussion of different distributed computing communication models, such as Client-Server, Pipeline, Boss/Worker and Peer to Peer. Next, we give a brief overview of messaging protocols before discussing the Message Passing Interface (MPI) model in detail. After presenting a simple MPI HelloWorld program, we illustrate how the `ScalarMultiply` program is parallelized, including its use of MPI data transfer functions.

The final section broadly introduces the problem of processing massive amounts of data in parallel. We begin with a brief synopsis of cloud computing and the service models associated with it. We then explain the MapReduce programming model with examples of `ScalarMultiply` and the classic example WordCount. The section concludes with an analysis of fault tolerance, a characterization of two popular MapReduce implementations (Hadoop and Spark) and a forward-looking overview of exascale and edge computing.

C. Other PDC coverage

Additional PDC topics are covered throughout the book, most notably in three chapters (Chapter 5 on computer architecture, Chapter 11 on the memory hierarchy and caching, and Chapter 13 on operating systems). The Architecture chapter introduces the Von Neumann architecture and focuses on how a single core processor is built, starting from basic logic gates to its clock-driven execution of program instructions. We cover some parallel architecture topics in later sections, including two sections on pipelining and a section that provides an overview of some parallelism in CPU architectures, such as ILP, vector processors, VLIW, and superscalar, multicore,

and hardware multithreading. The introduction of multicore processors in this early chapter of our textbook is important as it allows us to revisit issues related to shared memory architectures in later chapters.

In the first pipelining section, we introduce the idea of pipelining using our simple 4-stage single-core CPU to illustrate how multiple instructions can be simultaneously executed in a pipelined manner to achieve a higher IPC rate. In a second section, we introduce some advanced pipelining issues using a 5-stage pipeline to discuss control and data hazards and discuss how these can lead to pipeline stalls.

In the Memory Hierarchy Chapter we cover CPU caching, focusing on the motivation for caching and the mechanism part of how addresses are used to look up cache entries and how direct mapped and set associated caches are organized. We present an LRU replacement policy and its implementation for 2-way set associative cache, but we leave detailed policy coverage to upper-level textbooks. In a later section, we discuss support for multicore caching, focusing on cache coherency. We start by revisiting multicore architecture, using a simplified example on a system with private L1 caches and a shared L2 cache. With this model, we introduce the cache coherency problem, and then discuss cache coherency protocols, introducing snooping and directory based approaches. We then present the details of a simple snooping protocol (MSI) to illustrate how a snooping protocol is implemented to ensure cache coherency. We additionally introduce the problem of false sharing, but we defer discussing it in detail until our main chapter on shared memory parallelism (Chapter 14).

Our Operating Systems chapter introduces several PDC topics. Its main focus is defining the OS and its role in efficiently running programs on computer hardware. Again, we focus on the mechanism parts of the OS more than on policies that we leave to textbooks for upper-level OS courses. In our discussion of how the OS works, we introduce asynchrony, interrupts, traps, and handlers. The main content covers the OS's process and virtual memory abstractions. Our coverage of processes includes an introduction to multiprogramming and concurrency. We discuss process states, process creation, and the process hierarchy. We present system calls related to process creation (`fork`, `exec`, `wait`, `exit`), and discuss the concurrent execution of several C code examples. Additionally, we introduce some OS abstractions for interprocess communication including pipes and sockets, shared memory segments, and signals with a detailed example of a `SIGCHLD` signal handler. The coverage of processes, concurrency, and virtual memory in this chapter provides a foundation for the next chapter on shared memory parallelism that begins by defining the OS's thread abstraction and describing multi-threaded parallel program execution on multicore processors.

V. INTERACTIVE EXERCISES

Each book chapter includes a set of interactive exercises. The online nature of our textbook makes it an ideal platform for adding interactive exercises that allow readers to test their understanding and application of book topics. Exercises span

all levels of Bloom’s taxonomy in order to provide readers with practice at all levels of their learning a chapter’s content.

Over the past four years we have run an NSF-funded [6] Exercise Developers Program to develop book exercises with help from the larger CS community. Involving external experts in developing the questions for the book’s exercises was an important goal. It brought to this process a diverse set of ideas and perspectives, and we feel that it resulted in a much stronger and more broadly applicable set of questions than we could have produced on our own.

Through our four-year Exercise Developer’s program (concluded in June of 2026), we recruited a different group of faculty experts for each chapter or set of related chapters. We provided a stipend to most participants from our NSF grant, but we also had many participants volunteer in this effort. Participants met with us monthly to discuss the chapter subtopic focus for the month and to brainstorm some starting ideas for question topics. Exercise developers then wrote questions for exercises, creating a pool of questions from which we selected most to implement as interactive exercises to each chapter, and others we added to our book’s instructor portal. We edited questions to ensure common phrasing across all book exercises, and with the help of Swarthmore students, we implemented them in Runestone. The exercises make use of Runestone’s interactive question types for fill-in-the-blank, multiple choice, short answer, and parsons problems. The resulting set of interactive book exercises are primarily from questions that our exercise developers wrote, with a few written by us.

Additionally, as part of this effort, and with the help of Swarthmore students, we extended Runestone’s core functionality to include novel Ask me Another question applets (*components*). Each component has a “Generate Another Question” button that a user can press to generate another unique instance of the question type – a user may press this button as many times as they need to practice a specific concept. We also developed a stand-alone assembly code tracing tool, ASM Visualizer [5], that is used by some of our interactive exercises *Help* content to help explain a question’s answer. The interactive exercises are implemented in our custom build of Runestone [25], and employ a common look-and-feel to make it easy for a user to learn how to interact with them.

A. Ask me Another Components

In parallel with our work with the exercise developers, we, along with several Swarthmore students, designed and implemented the Ask me Another components as extensions to Runestone. We developed 34 components for a wide range of computer systems topics, including several that are specific to PDC content. Each component defines an interactive question format that supports automatically generating any number of question instances. Most use heuristics in combination with some randomization to generate good question instances of the component’s question type. Our use of heuristics, some of which are fairly complex, allows us to create components that generate questions testing a deeper application of concepts at

Instructions: This question intends to guide you through understanding parent-child relationships between processes. For the code snippet shown below (assuming all calls to `fork()` succeed), answer **how many times** each letter is printed with `printf()`.

Configure question: You can generate the C code block from several modes:

Select a mode: 2

```

if (fork()) {
    printf("a");
} else {
    printf("b");
    fork();
}
if (fork() == 0) {
    printf("c");
}
printf("d");

```

How many times will **a** print?

How many times will **b** print?

How many times will **c** print?

How many times will **d** print?

Generate another question Show process hierarchy Help Check answer

Process Hierarchy Graph: Each node represents a process. The text within each node indicates what the process prints.

Click on the code above to see how the tree is built step by step.

```

graph TD
    ad((ad)) --> bd((bd))
    ad --> cd1((cd))
    bd --> d((d))
    bd --> cd2((cd))
    d --> cd3((cd))

```

Fig. 1. Process Component with `fork` and the Process Hierarchy. Shown with the “Show process hierarchy” button selected.

higher levels of complexity than could be done using randomization alone. All components are configurable, allowing a user to start with simpler generated question instances and build up to generate more challenging instances of the component’s question type.

We have four components that are most related to PDC content: two in the Operating Systems chapter test the reader’s understanding of processes and concurrency; and two in the Shared Memory Parallelism chapter test the reader’s understanding of parallel execution of threads, race conditions, and synchronization.

The first process component (shown in Figure 1) generates questions that consist of C code with calls to `fork` and `printf` and ask the user how many processes execute each `printf` statement. It stresses the reader’s understanding of process creation, the process hierarchy, and concurrency. This component includes an interactive visualization of the process hierarchy that is created from the code’s execution. It is designed to help a user answer the question or understand the answer. A user selects the *Show process hierarchy* button to see the process hierarchy (shown expanded below the question in Figure 1). If a user clicks on individual statements in the generated question code, the visualization displays the process

Instructions: In this exercise, you'll analyze a C program that utilizes `fork`, `wait`, and `exit` system calls to manage process creation and synchronization. Your task is to determine which of the provided output sequences could be produced by the program, considering the possible concurrency allowed by `fork` and `wait`. Please tick all of the possible output print sequences that could be produced by the program.

Configure question: You can generate the C code block from several modes:
 Select a mode: 1

```
fork();
if (fork()) {
    printf("a");
    wait(NULL);
    printf("b");
} else {
    printf("c");
    exit();
}
printf("d");
```

☐ cdaabdcdb
☐ abacbccdd
☐ caacbbdd
☐ caadcbbdd

Generate another question
Show process timeline
Help
Check answer

Process Timeline Graph: This diagram shows the execution timeline of processes. Moving along the horizontal axis to the right represents time elapsed, and each horizontal contiguous line represents the sequence of a process execution. Vertical arrows represents the fork-wait-exit relationship between different processes.

Fig. 2. Process Component with `fork`, `wait`, `exit` and the Process Timeline. Shown with the “Show process timeline” button selected.

hierarchy at that point in the execution—a user can start at the first statement and click statements one at a time to see how each part of the process hierarchy is created from the calls to `fork` in the question code.

The second process component (shown in Figure 2) generates C code with calls to `fork` and `printf` like the first, but it also includes calls to `exit` and `wait` in the generated question code. This component asks the user to select all possible orderings of output statements from a list of options generated with the question. It further stresses process concurrency and how calls to `wait` and `exit` affect the concurrent actions of the parent and child processes. When a user selects the *Show process timeline* button, the component displays a process timeline visualization designed to help them answer the question (shown expanded in Figure 2). The process timeline represents `fork`, `wait`, `exit`, and `printf` events of the processes over time, illustrating when processes execute concurrently, `exit`, and `wait` for other processes.

Both components are configurable with different mode levels to specify the complexity of the generated code (the configuration options are at the top of each component before the generated question). A user generally starts with mode 1 and builds up to higher modes. Mode 1 questions contain fewer calls to `fork`, fewer nested calls, and fewer added `else` clauses with generated `if(fork)` statements than in higher modes. The user selects the *Check answer* button to receive

Instructions: Given the global variables and code for two threads below, select all possible values of the variables after the threads execute in parallel.

Configure question:
 Select a mode: 1

- Mode 1: Each if/else body contains one expression.
- Mode 2: If/else bodies can contain two expressions.

Please select what type of question to generate: Multiple Choice

Shared Global Variables:

```
int x = 6;
int pthread_mutex_lock(pthread_mutex_t *mutex);
```

Thread 1:

```
int y = 4;
if (y >= x) {
    y = y + 8;
} else {
    pthread_mutex_lock(&mutex);
    x = x - 8;
    pthread_mutex_unlock(&mutex);
}
print("%d %d", x, y);
return NULL;
```

Thread 2:

```
int y = 4;
if (y >= x) {
    y = y + 8;
} else {
    pthread_mutex_lock(&mutex);
    x = x - 8;
    pthread_mutex_unlock(&mutex);
}
print("%d %d", x, y);
return NULL;
```

Select all the possible final states:

	x	y (thread1)	y (thread2)
☐	-2	4	12
☐	-2	12	4
☐	-10	4	4
☐	-10	4	12

Generate another question
Check answer

Fig. 3. Threads and Race Conditions Component with mutex

some feedback about their answer if it is incorrect. To generate a new question instance to try, a user selects the *Generate another question* button, which is at the bottom left, below the question, of all Ask me Another components.

The process components require fairly complicated heuristics to generate questions that are both feasible and interestingly different across generated instances. We developed a grammar to use for generating question code. We use some randomization with weighted probabilities, the weights in part based on the question mode level, to select or expand different grammar constructs in the generated C code. We then parse the resulting question code to produce a graph representing when processes are created by other processes, when they print, when they wait for exited processes, and when they exit. By processing the graph, we compute all possible outputs generated by the concurrent processes in order to compute the answer to the generated question. We also use the graph to generate the interactive process hierarchy and process timeline visualizations.

We have two components that test a reader’s understanding of the parallel execution of threads, race conditions, and synchronization. Each component generates Pthread function code for two threads. The code includes global and local variable accesses and asks questions about the possible values of the variables that result from the execution of the code. One component (shown in Figure 3) adds mutex synchronization to the generated code. Both components are configurable with different difficulty levels and with different answer modes. In one answer mode the user enters all possible outputs produced from the execution. In the other answer mode, the component presents the user with several output sequence options, and the user selects all sequences that are possible from the code’s execution.

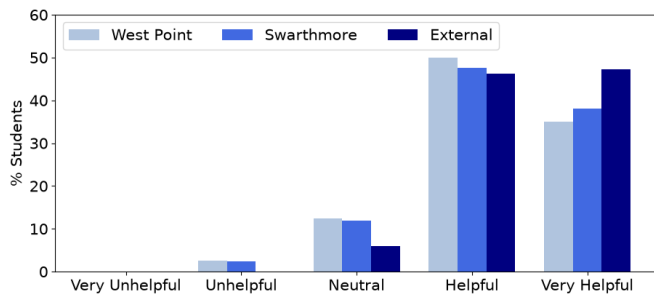


Fig. 4. Student responses to the question, “How helpful was the component for studying and learning about process creation?”

We use a technique similar to the one we use in the process components to generate Pthread question code. Our question generation heuristics use some randomness with some bounding values to ensure that we generate an interesting mix of code sequences for different question instances, including generating different types of data races (RW, WR, WW) and occasionally generating questions without race conditions. We also need to be careful when adding calls to lock and unlock mutex variables into the code to ensure that we produce deadlock-free code and good variations of the placement of synchronization code to generate an interesting variety of question instances. To compute the set of answers to the generated question, we process the question code to create a solution tree of all possible execution orderings. In the answer mode where a user selects correct output sequences from a set of options (the answer mode shown in Figure 3) we choose some valid output sequences and generate one or two interesting invalid sequences using the solution tree.

B. Pilot Study: Process Component

In academic year 2025-2026, we piloted one of the process Ask me Another components at several institutions. This was part of a larger study that broadly piloted several different components. Our study was authorized by the West Point Human Research Protection Program office, which was the institution of record for this study, and separately approved by West Point’s Institute Research Council. Instructors at West Point, Swarthmore, and 13 other institutions across the United States piloted a version of the component shown in Figure 1 in their classes, using a custom module that we created to introduce students to it in the context of the lesson.

After trying out the component and being encouraged to use the *Generate another question* button, students were given an optional two-question survey that asked them how helpful they found the component and provided them an opportunity to share additional open-ended feedback. Faculty also were given a link to an optional survey about their own experiences using the module in their courses.

We received 298 responses to our students survey that are summarized in Figure 4. At our own institutions, 85% of the West Point students ($n = 40$) and 85.7% of the Swarthmore students ($n = 42$) ranked the process Ask me Another component as being either “helpful” or “very helpful” for

learning or studying process creation. At external institutions, 93.5% of the students gave the process component those same rankings.

Common themes in the responses included students expressing appreciation for the helpfulness of the component, and the usefulness of the Generate Another Question button. Said one student, “Using the “generate another question” component is actually quite helpful for studying process creation because it forces you to practice the same core ideas in different contexts. Each new question reshuffles the forks, conditions, and exits, so you’re not just memorizing one solution”. Students also expressed appreciation for the “Show process hierarchy” button: “I found the process hierarchy visual to be helpful with understanding fully what’s going on with the process and [it] made it easy to clear up misunderstandings”.

All the faculty who piloted the exercise and responded to our survey ($n = 11$) indicated that they found that the process component either fit “well” or “extremely well” with the topics taught in their course, with 54% agreeing with the latter. Said one faculty member, “I liked that the processes exercise emphasized the functionality of `fork()` independently of the `exec()` functions; it ensures that students really internalize that the execution of the nearly identical processes concurrently”.

We recognize that the same topic is not taught the same universally across all schools, possibly resulting in some users finding the component a less good fit for their learning. A common theme in both the student and faculty feedback was to have better Help functionality that explains *why* an answer is wrong. Improving this feature is an avenue of our future work.

VI. CONCLUSION

Dive into Systems offers a large assortment of freely-available parallel and distributed content that requires only a CS 1 background of the reader. Students who have used the book report overwhelmingly positive feedback for the book’s clarity, figures, numerous examples, and for its free availability. Instructors appreciate the book’s broad topic coverage and flexibility for use across courses. Currently, our textbook has been adopted at over 65 institutions; of the 38 faculty who have volunteered to provide detailed information about their use of our text in our informal user survey, 92.1% report using our textbook as the primary textbook in their introductory systems courses.

The book has been strengthened by community involvement throughout the project. Being an online book, we can rapidly make updates or correct bugs as readers find them. Most recently, the community has helped us to develop a large set of interactive practice exercises, including suggestions for some Ask me Another components that we ended up developing. Students find the interactive exercises and the Ask me Another components helpful for learning, and instructors believe they fit well in their courses. Ultimately, we believe that *Dive into Systems* and its supplemental interactive learning resources have filled an important niche in making introductory systems and PDC content available to all.

ACKNOWLEDGMENTS

This work is supported in part by NSF (NSF:DUE:IUSE grant 2141722 and IAA 214181). We thank colleagues who volunteered as chapter editors of the beta version of our textbook, who participated in our early adopter program piloting our book in their courses, and who participated in our exercise developers program (the PDC exercise developers include Jeannie Albrecht, John Barr, David Bunde, Danielle Ellsworth, Sunjae Park, Kathleen Riley, John Rieffel, Christine Riley, John Stratton, Brian Toone, and Dee Weikle). We also thank Swarthmore students (Zhengfei Li, Bohou Zang, Akiva Sherin, Yue Zhang, Jasmine Fan, Jay Iyer, Xiaoya Yuan, Khang Cao, Kuzi Maveria, and Arys Aikyn) for their help implementing and testing interactive exercises, and designing and implementing the Ask me Another components and other interactive tools used by our textbook. Special thanks to William Bailey, David Bunde, John DeGood, Danielle Ellsworth, Shervin Hajiamini, Ed Harcourt, Bill Kerney, Zachary Kurmas, Jeffrey La Marche, Willard Rafnsson, Dee Weikle, Richard Weiss, and Jennifer Wong-Ma for piloting the process Ask Me Another component in their classrooms. Our webpage has a full list of contributors to *Dive into Systems* [29].

The opinions expressed in this paper are solely of the authors and do not represent those of the U.S. Military Academy, the U.S. Army or the U.S. Government.

REFERENCES

- [1] C. Severance, “Python for everybody: Exploring data using python 3,” 2016. [Online]. Available: <https://www.py4e.com/>
- [2] C. M. Allen B. Downey, Jeffrey Elkner, “Think python: How to think like a computer scientist,” 2002. [Online]. Available: <https://greentepress.com/wp/think-python/>
- [3] S. J. Matthews, T. Newhall, and K. C. Webb, “Dive into systems: A free, online textbook for introducing computer systems,” in *Proceedings of the 52nd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE ’21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1110–1116. [Online]. Available: <https://doi.org/10.1145/3408877.3432514>
- [4] S. K. Prasad, T. Estrada, S. Ghafoor, A. Gupta, K. Kant, C. Stunkel, A. Sussman, R. Vaidyanathan, C. Weems, K. Agrawal, M. Barnas, D. Brown, R. Bryant, D. P. Bunde, C. Busch, D. Deb, E. Freudenthal, J. Jaja, M. Parashar, C. Phillips, B. Robey, A. Rosenberg, E. Saule, and C. Shen, “Nsf/ieee-tcpp curriculum initiative on parallel and distributed computing - core topics for undergraduates, version 2.0,” 2020. [Online]. Available: <https://tcpp.cs.gsu.edu/curriculum/>
- [5] T. Newhall, K. C. Webb, I. Romea, E. Stavits, and S. J. Matthews, “ASM Visualizer: A Learning Tool for Assembly Programming,” in *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V.1*, ser. SIGCSE TS 2025. New York, NY, USA: Association for Computing Machinery, 2025, p. 840–846. [Online]. Available: <https://doi.org/10.1145/3641554.3701793>
- [6] NSF:DUE:IUSE grant number 2141722 and IAA 2141814, “Developing Dynamic and Interactive Materials to Teach Computing Systems Concepts to All Students,” 2022–2027.
- [7] P. Pacheco and M. Malensek, *An Introduction to Parallel Programming*. Morgan Kaufmann, 2021.
- [8] Ananth Grama and Anshul Gupta and George Karypis and Vipin Kumar, *Introduction to Parallel Computing (2nd Edition)*. Pearson, 2003.
- [9] Umakishore Ramachandran and William Leahy, *Computer Systems: An Integrated Approach to Architecture and Operating Systems*. Pearson, 2010.
- [10] Randal Bryant and David O’Hallaron, *Computer Systems: A Programmer’s Perspective (3rd Edition)*. Pearson, 2015.
- [11] Will Jarvis, “A Textbook Giant Is Going ‘Digital First.’ That Might Not Be Good for Affordability.” *The Chronicle of Higher Education*, July 2019.
- [12] S. K. Prasad, A. Gupta, A. Rosenberg, A. Sussman, and C. C. Weems, *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*. Morgan Kaufmann, 2015.
- [13] —, *Topics in Parallel and Distributed Computing: Enhancing the Undergraduate Curriculum: Performance, Concurrency, and Programming on Modern Platforms*. Springer International, 2018.
- [14] M. V. Steen and A. S. Tanenbaum, *Distributed Systems, 4th Edition*. distributed-systems.net, 2023.
- [15] R. H. Arpaci-Dusseau and A. C. Arpaci-Dusseau, *Operating Systems: Three Easy Pieces*, 1st ed. Arpaci-Dusseau Books, November 2023.
- [16] E. Shoop, R. Brown, S. J. Matthews, and J. C. Adams, “Interactive textbooks for parallel and distributed computing across the undergraduate cs curriculum,” in *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (EduPar’24)*, 2024, pp. 377–384.
- [17] A. B. Downey, “Think OS: A brief introduction to operating systems,” 2015. [Online]. Available: <https://greentepress.com/thinkos/html/>
- [18] I. Wienand, “Computer science from the bottom up,” 2024. [Online]. Available: <https://greentepress.com/thinkos/html/index.html>
- [19] J. Adams, R. Brown, S. Matthews, and E. Shoop, “Parallel Computing for Beginners 2.0,” <https://www.learnpdc.org/PDCBeginners2e/>, 2022.
- [20] E. Shoop, “Intermediate Parallel and Distributed Computing,” <https://www.learnpdc.org/IntermediatePDC/>, 2023.
- [21] CDER, “CDER courseware search,” 2018. [Online]. Available: https://tcpp.cs.gsu.edu/curriculum/?q=courseware_management
- [22] J. Adams, R. Brown, S. Matthews, and E. Shoop, “CS in Parallel,” <https://csinparallel.org/>, 2019.
- [23] CSinParallel, “Learnpdc: Free hands-on materials for learning pdc,” 2023. [Online]. Available: <https://www.learnpdc.org/>
- [24] S. J. Matthews, “PDCunplugged: A Free Repository of Unplugged Parallel Distributed Computing Activities,” in *2020 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2020.
- [25] 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>
- [26] Dive into Systems, “Dive Into Systems: Exercises.” [Online]. Available: <https://diveintosystems.org/exercises/>
- [27] S. Poulsen, G. L. Herman, M. Silva, M. Fowler, D. H. Smith, L. Porter, N. Ritschel, C. Zilles, and M. West, “Enabling Open Educational Resource Adoption through Integrated Sharing in PrairieLearn,” in *Proceedings of the 57th ACM Technical Symposium on Computer Science Education V.1*, ser. SIGCSE TS 2026. New York, NY, USA: Association for Computing Machinery, 2026, p. 859–865. [Online]. Available: <https://doi.org/10.1145/3770762.3772503>
- [28] T. Newhall, K. C. Webb, V. Chaganti, and A. Danner, “An Introductory-level Undergraduate CS Course that Introduces Parallel Computing,” *Journal of Parallel and Distributed Computing*, vol. 199, p. 105044, 2025.
- [29] Dive into Systems, “Dive into systems webpage.” [Online]. Available: <https://diveintosystems.org/>

Appendix: Artifact Description

Artifact Description (AD)

I. OVERVIEW OF CONTRIBUTIONS AND ARTIFACTS

A. Paper's Main Contributions

The main contributions of this paper are follows:

- C_1 A free online textbook introducing a broad range of systems topics including an introduction to parallel computing. It assumes only a CS1 background of the reader and designed to be useful to instructors teaching a range of courses.
- C_2 Free interactive exercises for testing student knowledge of PDC and other systems concepts.
- C_3 Novel PDC-related "Ask Me Another" interactive components that allow infinite practice on specific types of questions.

B. Computational Artifacts

The Dive into Systems textbook and its interactive resources are free for use at the following url: <https://www.diveintosystems.org>. Below we give a detailed list of all the artifacts related to this paper.

- A_1 Textbook: <https://diveintosystems.org/book/>.
- A_2 Exercises: <https://diveintosystems.org/exercises/>.

The table below illustrates how the computational artifacts support the paper's primary contributions and other related paper elements:

Artifact ID	Contributions Supported	Related Paper Elements
A_1	C_1	NA
A_2	C_2, C_3	Figures 1, 2, 3, 4

II. ARTIFACT IDENTIFICATION

A. Computational Artifact A_1

Relation To Contributions

The book is the primary way that readers learn about PDC information. It is designed to be used in the context of a required introductory systems course or as a supplement in a dedicated PDC course.

Expected Results

Readers will increase their knowledge of PDC topics.

Expected Reproduction Time (in Minutes)

The time required to access this book is dependent on the reader's Internet connection speed and how quickly they read.

Artifact Setup (incl. Inputs)

The book is available for free on the Internet, and accessible through a browser. No setup is involved, beyond having access to a computer with a Browser.

Hardware: The user must have a computer with a working Internet connection.

Software: The user must have access to a web browser on their computer.

Datasets / Inputs: N/A

Installation and Deployment: The user is not required to install any additional software to use the book.

Artifact Execution

To use the book, simply visit the link in your web browser. The book renders on standard web browsers as well as mobile devices.

Artifact Analysis (incl. Outputs)

N/A

B. Computational Artifact A_2

Relation To Contributions

Interactive and "Ask me Another" exercises cover the range of systems and PDC topics presented in the textbook. The interactive exercises contain a wide variety of interactive exercise types, including matching, multiple choice, fill in the blank, and parsons problems. The "Ask me Another" custom components allow a learner to receive seemingly infinite practice on hard PDC topics.

Expected Results

Learners using the interactive exercises and "Ask Me Another" receive real-time feedback on their understanding of PDC topics.

Expected Reproduction Time (in Minutes)

The time required to run through the exercises will depend highly on what a reader has learned from the textbook. If a user gets a question wrong, they will receive feedback to help them answer correctly. The time also depends on how frequently a user chooses the "Generate Another Question" button to create a new instance of the question type for additional practice.

Artifact Setup (incl. Inputs)

The interactive exercises are available for free on the Internet, and accessible through a browser. No additional setup is required.

Hardware: The user must have a computer with a working Internet connection.

Software: The user must have access to a web browser on their computer.

Datasets / Inputs: Our IRB prevents us from sharing any data related to our study, due to the privacy risks involved.

Installation and Deployment: The user is not required to install any additional software to use the exercises.

Artifact Execution

To use the exercises, simply visit the link in your web browser.

Artifact Analysis (incl. Outputs)

In academic year 2025-2026, we piloted one of the process Ask me Another components at 15 institutions across the United States, in the context of a larger study that broadly piloted a larger set of components. We received 298 responses to our students survey that are summarized in Figure 4.