Traditional and AI Tools for Teaching Concurrency

Prasun Dewan
Department of Computer Science
University of North Carolina
Chapel Hill, USA
dewan@cs.unc.edu

***Abstract*— Today, AI tools are generally considered as education disrupters. In this paper, we put them in context with more traditional tools, showing how they complement the pedagogical potential of the former. We motivate a set of specific novel ways in which state-of-the-art tools, individually and together, can influence the teaching of concurrency. The pedagogy tasks we consider are illustrating concepts, creating motivating and debuggable assignments, assessing the runtime behavior and source code of solutions manually and automatically, generating model solutions for code and essay questions, discussing conceptual questions in class, and being aware of in-progress work. We use examples from past courses and training sessions in which we have been involved to illustrate the potential and actual influence of tools on these tasks. Some of the tools we consider are popular tools such as interactive programming environments and chat tools - we show novel uses of them. Some of the others such as testing and visualization tools are in-use novel tools - we discuss how they been used. The final group consists of AI tools such as ChatGPT 3.5 and 4.0 - we discuss their potential and how they can be integrated with traditional tools to realize this potential. We also show that version 4.0 has a better understanding of advanced concepts in synchronization and coordination than version 3.5, and both have a remarkable ability to understand concepts in concurrency, which can be expected to grow with advances in AI.**

Keywords— concurrency, visualization, automatic assessment, student progress, clicker system, ChatGPT

I. Introduction

The teaching of programming inherently involves the use of tools – to edit, compile, and run the worked instructor examples and student assignment solutions. Several other general-purpose popular tools supplement the programming environment. These include forums such as Piazza, testing frameworks such as JUnit, grading servers such as Gradescope [1], presentation-creation tools such as PowerPoint, and presentation-sharing and chat tools such as Zoom.

When a set of entities works well, it is attractive to consider adding more variety to it. Several kinds of niche tools have been explored for teaching programming such as user-interface generators [2] and difficulty-detection tools [3]. These are niche/novel tools in that they are not mainstream but have been used effectively by a relatively small subset of teachers – sometimes only the inventors of these tools.

The recent release of ChatGPT has furthered interest in tool-based pedagogy. It has been shown that ChatGPT can solve many introductory problems [4], leading to fears of plagiarism - not from humans but from AI tools. On the positive side, ChatGPT's potential to aid pedagogy is attracting increasing interest. Examples of such interest include explaining error messages [5] and creating programming exercises [6].

In this paper, we focus on concurrent programming. We motivate a set of specific novel ways in which state-of-the-art tools, individually and together, can influence the teaching of concurrency. We use examples from past courses and training sessions in which we have been involved to illustrate the potential and actual influence of tools on these tasks.

All of these examples involve the use of Java thread mechanisms to teach concurrency. Moreover, all of the educational tools developed by us that are discussed here are implemented in Java. However, the nature of these examples and tools is not Java-specific. Therefore, we try to keep most of this discussion at an abstract, language-independent level.

The pedagogy tasks we consider are illustrating concepts, creating motivating and debuggable assignments, assessing the runtime behavior and source code of solutions manually and automatically, generating model solutions for code and essay questions, discussing conceptual questions in class, and being aware of in-progress work.

Some of the tools we consider are popular tools such as interactive programming and chat tools - we show novel uses of them. Some of the others such as testing and visualization tools are in-use novel tools (for teaching concurrency) - we discuss how they been used. The final group consists of AI tools such as ChatGPT - we discuss their potential.

Many of the individual novel uses of the traditional tools have been discussed in previous papers [7-10]. The contribution of this paper is a tool-based focus that puts a diverse set of tools relevant to concurrency in context, showing the way they relate to each other. In the rest of the paper, we discuss these tools and the roles they can play in teaching concurrency.

II. Physical World Simulations

Our work on concurrency has focused on three kinds of concepts; concurrent thread execution, thread synchronization, and thread coordination. Concurrent thread execution is creation of independent thread stacks, and executing the root procedures of the stacks. Thread coordination is the ability of threads to wait until they are signaled by other threads. The signal may be broadcast to all threads waiting in a queue or sent only to the thread at the front of the queue. Thread synchronization ensures that shared resources are locked when threads manipulate them. Low-level thread coordination mechanisms such as semaphores can be used to also implement synchronization. In modern

This work was funded in part by NSF awards OAC 1829752 and 1924059.

object-oriented languages such as Java, Python, and C#, higher-level mechanisms are provided to lock selected instance or static methods of a class.

The physical world, consisting of a variety of concurrent actors, provides analogies to illustrate these fundamental concepts. In a virtual simulation of this world, concurrent actors would correspond directly to concurrent threads. The actors taking turns to access shared resources such as a traffic intersection or coordinating with each other to perform a group task such as running a relay correspond to threads synchronizing and coordinating, respectively. As a result, well-understood aspects of the physical world can be used to understand analogous concepts in program multithreading.

In a worked example used by us, we made two different threads animate two different shuttles concurrently, thereby visually illustrating concurrency. To illustrate synchronization, we created two threads that animate a single shuttle, which corresponds to two different pilots flying the same shuttle. When the threads are synchronized, they take turns animating the shuttle, with one thread's animation following the other thread's animation. When they are not synchronized, the shuttle oscillates between the points in the trajectories computed by the two animation loops. In a third variation of this example, multiple threads, controlling different shuttles, coordinate with each other. A thread that starts its animation sends a signal to any waiting thread when the shuttle reaches a certain height. A later thread waits until such a signal is received by it before starting its animation. The internal queue kept by the underlying waiting and signaling mechanisms is used to determine which thread receives a signal from another thread.

III. UI Generator

It is possible, of course, to build custom special-purpose, shuttle-specific, tools to create the above animations. It is more attractive, however, to create a general framework to create arbitrary concurrent animations. Such a framework could be used by students to implement assignments that create such simulations.

Such animations not only illustrate concurrency concepts but also provide real-world motivations for them. In some of our course offerings, students implemented a series of layered assignments that. together, simulated the bridge scene from the movie *Monty Python and the Holy Grail*. Assignments 1 to 9 implemented the static scene shown in Fig. 1. Assignment 10 forked threads that made avatars of Arthur, Galahad, Robin, and Lancelot move independently. To exercise synchronization, in Assignment 11, they synchronized accesses by multiple threads that manipulate the same knight. To exercise thread coordination, in Assignment 12, they created threads that made the knights march to a beat set by the guard, using coordination mechanisms Animating threads were also created by some students (for no credit) to make the knights fall into the gorge on failing to answer a question.

These animations are also excellent debugging and manual assessment tools - errors in thread behavior are reflected in the simulations. For example, the knights in Fig. 1 would not move at the same time if the corresponding threads do not execute concurrently to animate them. Similarly, the knight threads would not march to the beat set by the guard thread if they did not wait for broadcasting signals from the latter.

We have used a user-interface generator developed by us, called ObjectEditor (used also to teach sequential concepts [2]) as a framework to easily create such simulations in concurrency-based assignments. Given an arbitrary observable object, ObjectEditor automatically displays the object, and keeps its visual representation consistent with its internal state. (An observable object [11] calls notification methods in registered observer objects whenever the state of the object changes.) For example, given an observable object representing the shuttle, ObjectEditor automatically moves the image when notification methods invoked in it indicate that the coordinates of the underlying object have changed. Similarly, given a set of observable objects representing the bridge scene, ObjectEditor automatically animates them in response to state-change notifications. In addition to creating up-to-date visual representations of objects, ObjectEditor also provides menus and buttons to invoke methods in the object – a feature we see in the two sections below.

In [7], we give examples of several other motivating layered concurrent assignments students have implemented using ObjectEditor, which include simulating Halloween and a scene from the *The Wizard of Oz*

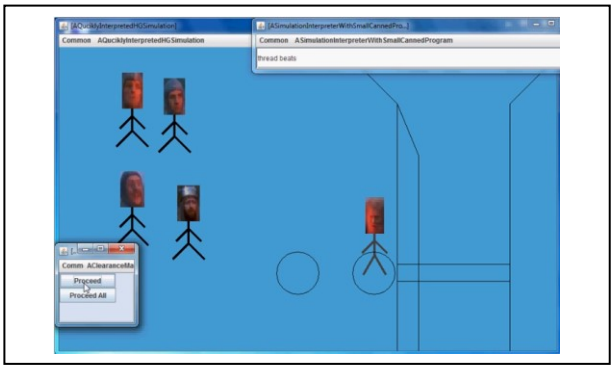


Fig. 1. Holy Grail Bridge Project

IV. Internal Thread-Based Structures

Simulations visualize the behavior of thread-based operations, but not the underlying data structures that define the behavior. Two important such data structures are the stack associated with each thread, and the queues in which waiting threads are put.

Fortunately, modern interactive environments visualize stacks. By asking students to put appropriate breakpoints, we have allowed them to see that each thread is associated with its own stack and program counter, as illustrated in Fig. 2. In this Java-based example, they can see that the root procedure of the stack of each thread forked by them is the Java `run()` method.

To visualize queues, we have developed our own tool, which provides a `Proceed` button to signal any thread in the visualized queue. Fig. 3 illustrates how the tool can be used to understand waiting and signaling. The simulation starts with two threads, animating independent shuttles, in the displayed queue. This state simulates two physical shuttles queued to enter a physical launching pad. When the `Proceed` button is pressed, the first

thread is removed from the queue and starts animating its shuttle. Clicking the `Proceed` button, of course, corresponds to an air traffic controller giving clearance to the next waiting shuttle to take off.

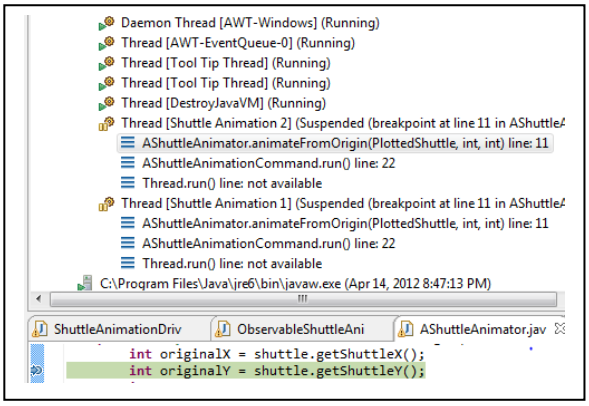


Fig. 2. Multiple Stacks Shown for Different Threads

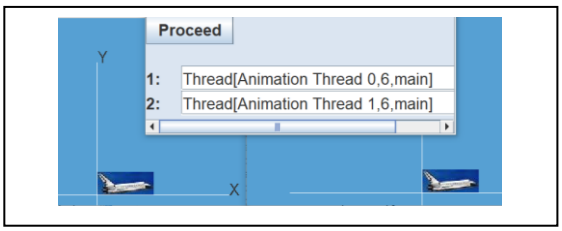


Fig. 3. Interactive Queue Manipulation and Visualization

This tool is layered on top of ObectEditor in that it uses the latter to visualize the queue and invoke the signaling method in response to pressing of the `Proceed` button.

V. Active Grading Management → Manual Assesment

As mentioned above, animating simulations can be used by a human grader to assess solutions to concurrency assignments, as bugs in the use of these mechanisms are reflected in the simulations. Thus, if the simulations are correct, the code is likely to be correct and does not need to be examined. If they are incorrect, then the code can be studied to identify the errors. As students themselves can detect such errors by viewing the simulations, it should be rare for the concurrency semantics of the assignments to be not correct.

Web-based systems such as Gradescope [1] have shown that manual assessment can be speeded by a grading management system that automates rubric management and navigation to the student submissions. However, because they are web-based, they cannot directly run interactive programs that create simulations to be examined by the human grader.

Using ObjectEditor, we have developed a grading management [10] that can directly execute assignment code. While our grading management system was developed independently of Gradescope, it offers similar features to automate rubric management and navigate among student submissions. Fig. 4 shows parts of it being used to manually grade the concurrency features (4-7) of an assignment, which also included non-concurrency features (1-3).

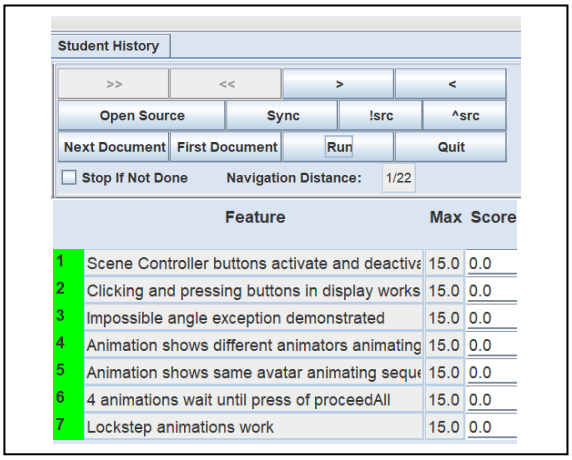


Fig. 4. Interactive Submisson Assesment

Unlike Gradescope, our system is specialized for program grading and is not web-based. This allows us to make it "active" by providing a `Run` command, which can be invoked by a human grader to execute the graded submission and interact with it. Fig. 1 shows the result of invoking this command from the user interface of Fig. 4. This command is, of course, crucial to grade visualized concurrency.

VI. Oberverving Tests

If concurrency is visualized and all visible state changes are observable, then it should be possible to write test cases that automatically check consistency constraints. These tests can observe all state change notifications, and based on their order and the threads that sent them, determine if some or all concurrency requirements are met. For instance, if:

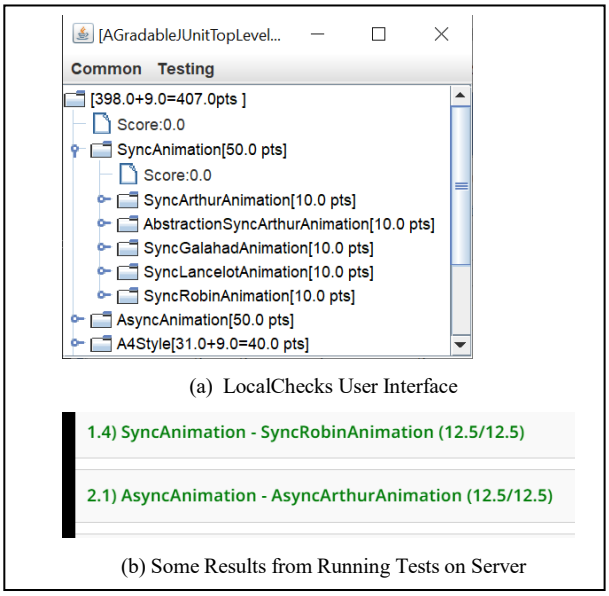
(a) all notifications are sent by a single thread, then a test processing them can flag a lack of concurrency,

(b) notifications by different threads about changes to a shared object are interleaved, then the test can indicate lack of synchronization,

(c) notifications by different coordinating threads are not in the expected order, then the test can assume coordination mistakes.

Based on this key idea, we have developed an extension of the JUnit framework that can test visualized observable concurrency, which is discussed in depth in [10]. We have used these tests to check the concurrency requirements of a variety of assignments.

These tests can be loaded into the Gradescope server to support automatic grading of final solutions (Fig. 5 (b)). More important, they can be executed locally by students on partial solutions, using an interactive tool developed by us, called LocalChecks (Fig. 5(a)). Again, this tool uses ObjectEditor to create its user-interface. Students have used this tool as an instructor agent that checks which requirements have been met, and gives hints, through appropriate error messages, on how the unmet requirements should be satisfied.



(a) LocalChecks User Interface

(b) Some Results from Running Tests on Server

Fig. 5. Executing Tests Locally and on Server

If a test were to directly observe changes, then it would be multithreaded, as a call to a notification method could be executed by one of many threads is the tested program. Multi-threaded programs are, in general, more complicated than single-threaded ones. Therefore, we have developed the abstraction of a problem-independent event management system to allow problem-dependent tests to be single-threaded. The system itself is multi-threaded, and stores the streams of notifications received from different threads in a data structure, which can be queried by single-threaded tests to check problem-specific requirements.

Before we built our testing framework, we used our interactive active grading management system to test all concurrency requirements of simulations. After building our testing framework, we wrote tests for all of these requirements. However, we continued to use the active grading management system to (a) check the tests for false positives and negatives, (b) evaluate the quality of the animations, and (c) ensure that students had written code that demonstrated that the concurrency (and other) requirements had been met. We do not have tests for checking their demonstration code.

Our grader management system can, like Gradescope, run the tests also, and allow the human grader to override test results, if necessary. These tests, however, are run on the computer of the human grader when they do the grading, and not on a server when the students submit their code for grading. We used to also have the ability to run the tests in our server, but ensuring this server can handle the load and is always available was a problem. Hence, currently, the tests are all run on Gradescope.

VII. Observable Prints

The approach of observer tests assumes concurrent manipulation of observable objects. Thus, this approach will not work when the assignment solution is not coded in an object-oriented language or when the manipulated data are not observable objects.

To overcome these limitations, we have added a layer above our event management system for testing programs that follow the fork-join model [12]. In this model, a dispatcher thread distributes its work to a set of worker threads it forks, which deposit their results in a data structure shared with the dispatcher. The dispatcher waits for them to finish, and then collects and prints the results. Typically, the input and output data are stored in arrays, which are not observable objects in any language with which we are familiar. Moreover, the dispatcher thread receives all input and produces all output. In a correct implementation of this model, the relationship between the output and input is independent of the number of threads forked. Thus, the traditional approach of using this relationship to test a program cannot be used to check concurrency requirements.

Our layer for testing such programs is based on two main ideas. First, we intercept the prints using a custom observable object, which can be observed by our event management system. Second, we require each worker thread to print intermediate results. As a result, the event management system can store concurrency-related notifications from all threads, which can then be queried by problem-specific tests to check the concurrency requirements. By providing appropriate abstract classes for implementing these tests, we ensure that the tests are responsible only for the "what" of fork-join testing – the "how" is taken care of by the abstract classes and the event management system.

The fork-join framework and the problems it has been used to test are described in depth in [12]. Fig. 6 shows some of the error messages produced by tests written using this framework. These tests check if a program correctly uses the fork-join model to find odd numbers in a set of random numbers.

```
Forked threads do not execute concurrently.
Between the first and last output of each forked
thread, there is no other thread output.
```

a) Fork-Join Problem

```
Imbalanced thread load: Max thread iterations(4)
- min thread iterations(1) = 3. It should be <= 1
```

b) Imbalanced Load Problem

```
Computed total number of odd numbers 1 != expected total 3.
You may have race conditions.
```

(c) Race Condition Problem

Fig. 6. Error Messages Identifying Problems with Concurrent Program

We see here how tests can act as instructor agents, clarifying requirements, identifying bugs, and giving hints on how to fix them. The three messages indicate, respectively, that the programmer is expected to create worker threads that execute concurrently, ensure that the load of worker threads is balanced, and of course, compute the correct number. They also indicate that if the programmer has tried to meet these three requirements, then the program has bugs. The second message indicates that fairness can be achieved by ensuring that the difference between the maximum and minimum load is <=1. The third message indicates that race conditions may have

caused the final result to be wrong. It is up to the programmer of the checks to generate only those hints that are expected to contribute to learning.

VIII. Problem-Specific Source Checkers

There could be several causes for the fork-join problem above (Fig. 6(a)). For instance, the programmer (a) may not have forked the threads, or (b) joined each forked thread before forking the next thread. A check that looks only at the runtime input/output of a program cannot determine which of many problems in the source cause unexpected runtime behavior. Even the hints in the race condition problem (Fig 6(c)) is a guess of the test writer. Therefore, it is attractive to build a tool that examines the source code to identify potential problems.

One example of such a tool is Checkstyle [13] - an Eclipse plugin that finds problem-independent Java style violations. We have created an extension of this plugin, called Hermes, to find coding issues based on a problem-dependent specification of the expected code structure. Our extended plugin can check if a class or a method instantiates a particular class or calls a method in a particular class. In addition, it can check if a class implements a particular interface. The plugin, like Checksyle, gives error messages as source code is edited. In addition, it writes the error messages to a file that can be read by our JUnit tests. As a result, these tests can check both runtime behavior (Fig. 6) and also source (Fig. 7).

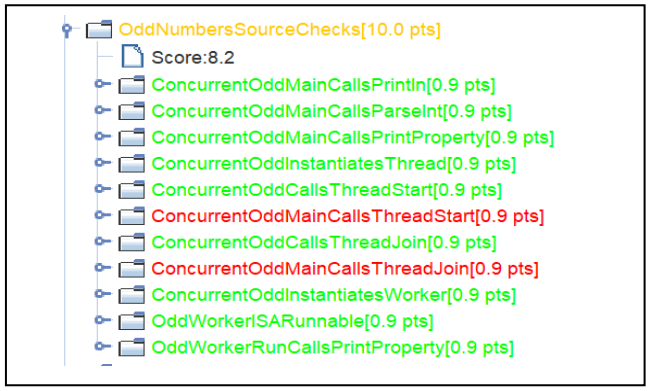


Fig. 7. Source Check Results on Given Worked Example

Often students given a programming assignment say they do not know where to start or ask an instructor if they are on the right path. The source checks give them an idea of the code that must be included in a correct solution. The names of the tests in Fig. 7 are mostly self-explanatory. They indicate source-code milestones that must be met to solve the problem.

As we see from the names, some of these checks test if certain classes are instantiated. Most of the checks test if calls to certain methods are included in the code. They provide *class-level* and *method-level source granularity* based on whether they specify the class or method that makes the call. A check is green or red based on whether it passed or not. These tests check both that (a) correctness constraints are met (e.g. the call to `Thread start` to start threads), and (b) tracing constraints are met (e.g. the call to `printProperty` to print intermediate results).

IX. AI Tools for Code Questions

Hermes cannot check the logic embedded in the source code. This is consistent with all algorithmic source-code checkers such as compilers, interpreters, and style checkers. The reason is that the halting problem says that it is impossible to even determine if a program halts – checking if it halts with the correct solution is a more difficult problem.

AI tools, on the other hand, can make inferences that are not guaranteed to be correct. ChatGPT is arguably the most attractive of such tools, and therefore, has been used to investigate its potential for understanding the logic of sample sequential assignments [4]. Therefore, it is attractive to determine its potential for finding problems with concurrency logic.

Today, ChatGPT comes in two versions, 3.5 and 4.0. We gave both the source code of an erroneous program for concurrently finding odd numbers in a set of random numbers. This program had comments describing what the program should do. We asked both versions to correct it. In an in-class exercise reported in [14], students were given the same task. We will refer to this program as the erroneous odd number program.

ChatGPT 3.5 found two of the three errors in the program, as shown by excerpts of its response given below:

Bug in **`fairThreadRemainderSize`** method: This method is intended to distribute the remainder of work among the threads fairly. The current implementation only assigns work to the first thread (**`aThreadIndex == 0`**). This is incorrect. To distribute the work fairly, you should use the following code: ....

Bug in **`incrementTotalOddNumbers`** method: The issue here is that multiple threads can read **`totalNumberOddNumbers`** at the same time, and they may all increment it and then write it back, resulting in some updates being lost. To fix this, you need to make this method synchronized. Uncomment the synchronized keyword in the method signature like this: ...

ChatGPT 4.0 found also the third bug, indicated by the following excerpt:

`forkAndJoinThreads`: Separated the forking and joining of threads into two loops to ensure that all threads start before they are joined. This allows them to run concurrently, which is the intended behavior of fork-join concurrency.

ChatGPT 3.5 was also able to find this bug, but only after it was explicitly given the prompt: *"There is also a bug in forkAndJoinThreads(). Do you see it?"* The comments in the program did identify all procedures with the bugs – version 3.5 however did not find them sufficient to identify the third bug.

To further test ChatGPT's ability to understand and compose concurrent programs, we gave version 3.5 a program that sequentially computes prime numbers, and has no concurrency code. We asked it to make it concurrent. It correctly performed this task, reusing the sequential code as much as was possible. In an exercise reported in [8], trainees were given the same task.

The success of ChatGPT with these two concurrency-based training problems should not be surprising, at least in retrospect.

Usually, the declarative mechanisms of OpenMP are sufficient to code concurrency in such problems, and these mechanisms are used in very stylized ways in these problems. As a result, the concurrency semantics of these problems vary less than their sequential semantics. Thus, given enough training examples, AI tools should be able to fix, compose, and evaluate concurrency aspects of these problems more easily than sequential semantics. The fact that version 4.0 was more successful than 3.5 indicates that the number of concurrency problems fed to AI tools has recently increased, and we can expect this trend to continue with the increase in importance of concurrent programming.

This is a serious issue if students use AI to solve the problems. On the other hand, if they do not directly use AI to do so – which can be the results of following an honor code or doing the problem in class under supervision – then these examples also show the potential of AI to generate model solutions and give automatic hints to the students. Moreover, like the hand-crafted runtime and source checks mentioned above, they can be used to grade solutions based on the bugs they find. They are an attractive alternative to the former as they do not require problem-specific test code to be written. An additional advantage is that, unlike the fork-join tests discussed above, they do not require students to put tracing prints in their code.

Current AI tools such as ChatGPT can be directly used by instructors to generate model solutions, after students have submitted their work. It is not clear, however, how they should be used to give hints or grade solutions. If students directly use them, then they may get hints together with the fixes. One solution is to integrate AI tools with hand-crafted tests. These tests can use the API provided by an AI tool try and filter out the hints from the AI tool's response, before giving them to the students. Such filtering can also be used for grading.

A fundamental problem with these uses is the lack of reliability of AI tools. On the other hand, we can expect finding and fixing bugs in generated model solutions takes less time than generating the code and associated explanations from scratch. Judging the correctness of hints can be the responsibility of the students, or these hints can be mediated through the instructors. Again, validating and correcting hints can be expected to take less work than generating them, with explanations, from scratch. False positives in grading errors are not that serious as regrade requests, which can be expected to be rare, can be handled manually. False negatives are the main problem. If the alternative is manual grading, then we expect AI assignment grading to be less error-prone and result in fewer false negatives. Problem-specific hand-crafted tests, though requiring more work, can always be expected to be preferable to AI assignment grading.

A secondary issue arises from the fact that there is, currently, a monetary cost associated with the use of ChatGPT. Other competing AI tools can be expected to also charge for API use. If the checks are executed on a commercial sever such as Gradescope, it is not clear how to pay for this cost. If these are executed on the computer of the students or the graders, then the students and graders would have to incur this cost, which they are likely to accept. Thus, we are considering changes to our testing infrastructure that use AI tools when the tests are run locally, as part of LocalChecks (Fig. 6(a)), or our active grading management system (Fig. 7), but not when they are run on Gradescope (Fig. 6(b)).

The sample problems we gave to ChatGPT were small – the solutions fit in one Java file. It is unlikely for AI tools to be able to handle layered assignments such as the Holy Grail problem, as each of the assignment descriptions refers to and builds on the ones before it. If AI-based plagiarism is an issue, then such layered assignments become attractive. However, such assignments also may not benefit from the pedagogical potential of AI.

X. AI Tools For Conceptual Questions

In the exercise reported in [14], students were asked to not only correct the erroneous odd number program, but also answer quiz questions on it. Given the ability of ChatGPT to provide natural language descriptions of concurrent programs, the bugs in them, and the fixes it makes to them, it is attractive to consider its potential to answer and grade quiz questions about a program.

Three of the quiz questions asked the influence on the output of the three fixes, individually and collectively. They did not ask for the exact output, but only an abstract description of how it would change. Version 3.5 gave correct answers to all of them without giving the exact output. One quiz question tested if the answerer knew that synchronizing the only caller, A, of method B, is equivalent to synchronizing B, when A and B are static methods. Version 3.5 got that correct also. The last question checked if the answerer understood that if A and B are instance and static methods, respectively, then synchronizing B is not equivalent to synchronizing A. The reason is that calling a static synchronized method on a class locks all synchronized static methods of the class, while calling a synchronized instance method on an object locks all synchronized instance methods of the instance. Version 3.5 got that wrong. It did, however, get the right answer when it was asked if it had considered the fact that A and B were instance and static methods, respectively. Version 4.0 got all questions correct without additional prompts.

Students who did the exercise of fixing the buggy odd numbers problem were also given a similar exam question. Two of the parts of the questions asked students to give an exact output when a method manipulating a shared variable was synchronized and when it was not. Amazingly, version 3.5 was able to compute an exact output from the source code for not only the synchronized version but also the unsynchronized one (Fig, 8), which requires considering race conditions.

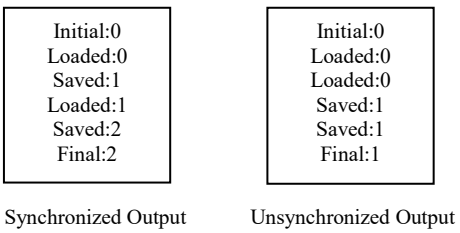


Fig. 8. Output with and without synchronization

A third question required, like the analogous quiz question, the understanding that synchronizations of static and instance

methods are different. Not surprisingly, again, 3.5 got this question wrong and 4.0 got it right, showing at least consistency.

The last question involved synchronizing two mutually dependent methods, which caused a deadlock. Version 3.5 did not see the deadlock, and neither did version 4.0. When 4.0 was told that there would be a deadlock, it agreed, gave the correct reason for it, and computed the correct output. Version 3.5 did not agree at first, and repeated its initial answer. When told again that there was a deadlock, it agreed, but gave the wrong reason, and did not give the output. When asked again for the output, it said it could not compute it. Only one student answered the deadlock question correctly in the exam, which can be explained by the fact that students had not seen this concept explicitly before, even though they were told how synchronized methods work. Version 3.5 seems to be like most students.

This example shows the ability of ChatGPT to generate model answers to essay questions. We gave version 3.5 also some correct and incorrect answers to the questions above, and asked it to grade them. It found fewer mistakes with incorrect/irrelevant answers than the human grader. But with advances, AI tools can be expected to generate model essay answers more correctly as well as grade them more critically. Given that the instructor – the author - did not make an effort to generate model answers for the quiz/exam questions, and took a whole day to grade the 40 quiz/exam submissions, both capabilities would be extremely useful.

XI. Chat + Discussion Model→ Active Learning

Conceptual questions can be asked not only in quizzes and exams, but also during lectures, as a form of active learning. Several clicker systems are used today to help gather multiple-choice answers from students. We have coupled Zoom chats with an innovative discussion model to create a new form of a clicker system for essay answers. In this model, the instructor first poses the question to the whole class. Students are asked to compose their answers by writing a chat message. When they are finished, they are asked to raise their hands, but not post the message until the instructor asks them to do so. The instructor waits for a certain number of raised hands before asking them to commit their answers, and then discusses each answer.

Fig. 9 illustrates this approach by showing some of the answers to the following concurrency-based question posed in a class on PDC (Parallel and Distributed Computing):

What are the reasons for making a server multi-threaded? Why is this particularly important for a compute-intensive task such as determining prime factors?

We tried this strategy by accident during the Covid pandemic when the instructor's audio connection to the students became unusable in one class. Today, even in in-person classes, we prefer this approach to the traditional approach of receiving audio answers from students who have volunteered to answer the question by physically raising their hands. There are several reasons for this preference:

1. There is a written record of who tried to answer a question, which is used to give them class participation points. They receive a fixed number of points as long as they try to answer the question – the quality of the answer does not matter. The written record of questions and corresponding answers can also be used to quickly review the material before an exam or assignment.
2. We have found that there is less participation in audio-based interaction. Many more students will enter chat answers than volunteer to raise their hands, especially when points will be awarded for trying to answer the question. Usually, at most 10 percent of the class raises hands, while about 75% of the class enters chat answers.
3. It is possible to know how many students are thinking of the same answer. In the traditional approach, students will lower their hands or not raise their hands after another student gives an answer similar to what they are thinking.
4. Students have more time to think of answers, which increases both the answer quality and number.

12:38:59: Multiple threads are created for different clients because clients are independent from each other. Multiple threads avoid the impacts of one client on the other.

12:39:00: Multiple threads can help a server parallelize computation from multiple clients. This could speed up task completion.

12:39:00: Dividing up tasks into discrete threads allows the server to have more control over the requests and the resource usage involved with processing said requests. Threads could be controlled using facilities provided by the OS to ensure that particularly resource-intensive tasks do not choke the system.

12:39:01: Multiple threads allow for parallelization, which can massively speed up tasks such as prime factorization and prevent any single client from blocking you if you end up waiting on them.

12:39:01: So it doesn't have to wait for the previous request from the first client to work on the request of the second client. If it has to execute one after the other, with a time intensive task, clients are going to have to wait longer for their information which isn't good. What client 2 is requesting shouldnt have any impact on the time it takes client 1 to get their information..

12:39:03: You need multiple threads with multiple clients so that one client isn't blocked by the other during processing

Fig. 9. Chat-Based Class Discussion

A main disadvantage of this approach is that interaction is less fluid as only the instructor speaks – while posing the question and discussing the answers. Another problem is scalability – in a class with more than 40 students, it has not been possible to discuss the answer of each student.

A solution to this problem is suggested by the example interaction shown in Fig 9. There are several duplicate answers – for instance, many of them say that the performance of the server can increase by processing concurrent requests on multiple processors concurrently. This duplication makes it attractive to use an AI tool to cluster answers and discuss a representative from each cluster. The ability of ChatGPT to grade some of the quiz answers gives us hope that such clustering may be possible.

XII. Logging and In-Progress Awareneness

Our runtime and source checks log actions taken by the students to run them. This information can be used to provide

instructors with awareness of in-progress work of students [7-9]. This information, in turn, can be used by instructors to determine the inherent difficulty of a problem, whether some students are having more than usual difficulty with some problems, and whether lack of progress of some students is due to not having spent enough time on the problem.

Fig. 10 shows a visualization of the logs from three classes that did the Holy Grail layered assignments. It divides all the tests into four categories: Threads, Synchronization, Coordination, and Other, based on which aspect of the problem they were checking. It shows the distribution of the number of attempts (test executions) [7] required to pass tests in the four compared categories. Given a point in the X axis, representing a certain number of attempts, the bar on the Y axis indicates the number of students who made that many attempts in each of the four categories. The figure graphically shows, that concurrency topics resulted in more high numbers (> 4), and the highest numbers were associated with coordination and synchronization. Thus, it gives the instructor an idea of the relative difficulty of these topics and the fraction of students facing each level of difficulty.

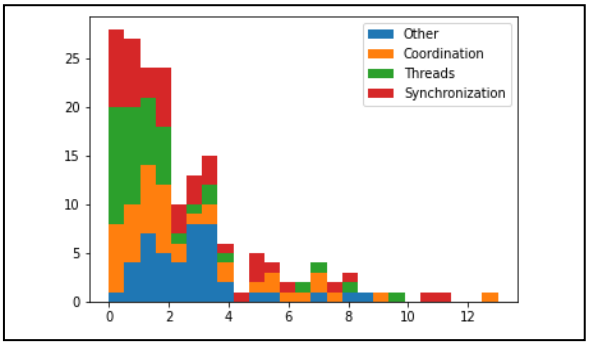


Fig. 10. Distribution of Test Attempts, 2016-2018

This graph was created after all three classes had finished their work. We are currently building a tool that allows log visualization to be created in real-time. Such a visualization could be used, for instance, to give unsolicited help to the students on the bottom right facing more than usual difficulty with coordination and synchronization. One hurdle to achieving this goal is that the logs, by default, do not give the identity of the students. Instead a unique "fake name" is used for each student, which the student can customize. One solution is to broadcast a message to the whole class indicating the willingness of the instructor to help students with certain fake names with specified tests. Students with these names encountering difficulty with that check can then solicit help.

XIII. Conclusions and Future Work

This paper addresses a variety of pedagogical tasks related to concurrency – illustrating concepts, creating motivating and debuggable assignments, assessing the runtime behavior and source code of solutions manually and automatically, generating model solutions for code and essay questions, discussing conceptual questions in class, and being aware of in-progress work. It shows that these tasks can benefit from a combination of AI and a large variety of traditional tools, which include user-interface generators, programming environments, queue visualizers, active grading management systems, automatic source and runtime checks, logging and visualization tools, and chat systems.

Further work is needed for creating traditional automatic assessment tools for languages other than Java, integrating them with AI tools, developing traditional tools for real-time visualization of in-progress work, and developing tools to detect difficulty and offer semi-automatic help to students not making expected progress. This paper provides a basis for investigating these directions.

Acknowledgment

Thanks to Sheikh Ghafoor for inviting the author to write this paper for the EduHIPC'23 workshop.

References

[1] Arjun Singh, S.K., Kevin Gutowski, Pieter Abbeel. Gradescope: A Fast, Flexible, and Fair System for Scalable Assessment of Handwritten Work. in Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale. 2017. ACM.

[2] Dewan, P. How a Language-based GUI Generator Can Influence the Teaching of Object-Oriented Programming. in Proc. ACM SIGCSE. 2012.

[3] Carter, J., P. Dewan, and M. Pichilinani. Towards Incremental Separation of Surmountable and Insurmountable Programming Difficulties. in Proc. SIGCSE. 2015. ACM.

[4] Finnie-Ansley, J., P. Denny, B.A. Becker, A. Luxton-Reilly, and J. Prather. The robots are coming: Exploring the implications of openai codex on introductory programming. in Proceedings of the 24th Australasian Computing Education Conference. 2022.

[5] Leinonen, J., A. Hellas, S. Sarsa, B. Reeves, P. Denny, J. Prather, and B.A. Becker. Using large language models to enhance programming error messages. in Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1. 2023.

[6] Sarsa, S., P. Denny, A. Hellas, and J. Leinonen. Automatic generation of programming exercises and code explanations using large language models. in Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1. 2022.

[7] Dewan, P., S. George, A. Wortas, and J. Do, Techniques and Tools for Visually Introducing Freshmen to Object-Based Thread Abstractions Journal of Parallel and Distributed Computing, 2021. 157.

[8] Dewan, P., A. Worley, S. George, F. Yanaga, A. Wortas, J. Juschuk, M. Rogers, and S.K. Ghafoor, Hands-On, Instructor-Light, Checked and Tracked Training of Trainers in Java Fork-Join Abstractions., in IEEE 29th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW). 2022, IEEE. p. 28-35.

[9] Dewan, P., S. George, B. Gu, Z. Liu, H. Wang, and A. Wortas. Broad Awareness of Unseen Work on a Concurrency-Based Assignment. in 2021 IEEE 28th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW). 2021. IEEE.

[10] Dewan, P., A. Wortas, Z. Liu, S. George, B. Gu, and H. Wang. Automating Testing of Visual Observed Concurrency. in 2021 IEEE/ACM Ninth Workshop on Education for High Performance Computing (EduHPC). 2021. IEEE.

[11] Gamma, E., R. Helm, R. Johnson, and J. Vlissedes, Design Patterns, Elements of Object-Oriented Software. 1995, Reading, MA.: Addison Wesley, 1995.

[12] Dewan, P. Infrastructure for Writing Fork-Join Tests. in Proceedings of the SC'23 Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis. 2023.

[13] Checkstyle, C. Checkstyle. 2015; Available from: http://checkstyle.sourceforge.net/.

[14] Dewan, P., Lecture-less Java-Threads Training in an Hour? , in IEEE 30th International Conference on High Performance Computing, Data and Analytics Workshop (HiPCW).. 2023, IEEE.