Knoxcraft Terps: Parallel Minecraft for Novices, a Peachy Assignment

Jurdana Masuma Iqrah, Jaime Spacco, Michael Gerten
Department of Computer Science, Knox College
Galesburg, IL, USA
{jiqrah,jspacco,mgerten}@knox.edu

***Abstract*—Knoxcraft Terps is a CS0/1 assignment that extends students' previous experience with Logo-like programs into a 3D environment. It natively supports parallel execution, introducing students to core concepts of speed-up and diminishing returns in a visually engaging system. In a pilot run of the assignment, all students successfully created 3D structures, with many able to achieve speedups using the available parallel features. The assignment and documentation are available at https://github.com/jspacco/knoxcraftmod.**

***Index Terms*—Minecraft, PDC, CS1, CS2, peachy assignments, parallel & distributed computing**

I. Introduction

Peachy Parallel Assignments are an effort to crowd-source the creation of innovative assignments that cover parallel and distributed computing (PDC) topics. Peachy Assignments are typically solicited from the EduHPC and EduPar communities and judged on the following criteria:

- **Tested:** The assignment should have been tested with real students.
- **Adoptable:** The assignment should include materials so that it can be adopted. Ideally, it should cover widely-taught concepts using common languages/libraries.
- **Cool and inspirational:** Students should be motivated to work on the assignment and engage with the concepts. Ideally, it should be something they are inspired to share the results with friends.

The best assignments are then published in the workshop proceedings (e.g. [1], [2]) and posted to the Peachy Parallel Assignments webpage (https://tcpp.cs.gsu.edu/curriculum/?q=peachy). This paper presents one such assignment and the results of its pilot implementation at Knox College during the 25-26 academic year.

Knoxcraft Terps is a Minecraft mod and client-side library that lets students write Java or Python code inspired by Logo [3] in their IDE, then log into our specially modded Minecraft server to run their program and build 3D structures. Students can then improve the performance of their programs by splitting up the work into multiple conceptual "threads". Each thread creates an extra turtle that carries out its own work in an embarrassingly parallel fashion.

Knoxcraft Terps is approprate for students in CS1, as it requires only variables, method calls, and for loops. The threads operate like additional main methods.

II. Related Work

PDC is a key knowledge area in ACM's 2023 CS Curriculum [4] and is also required by the 2019 ABET accreditation criteria [5]. One popular approach is to offer specialized courses on PDC, such as Liu's 20 years of effort to teach PDC to college seniors [6], Yazici's course using "real life" applications to teach PDC [7], and courses on big data taught by Eckroth [8] and Saule [9]. Mullen taught PDC through a Massive Open Online Course (MOOC) [10].

A second approach is to include modules covering PDC content within existing courses [11] [12] [13] [14]. Peach Assignments are closest in spirit to a modular approach to teaching PDC.

III. Terp

Terp works as a Java library that students call. In their favorite IDE, students write Java code to create a *Terp* ("Terp" is short for Terrapin, a species of freshwater turtle, and also the mascot of the University of Maryland), to move their Terp in 3D using a few simple commands (forward, back, left, right, up, down, turn), and to set blocks (setBlock command). The code shown in Fig. 1 draws a simple pyramid made of obsidian.

The code is executed and uploaded to a Knoxcraft server, our specially modded Minecraft server. Students then log into the server and request a Terp (a modded vanilla Minecraft turtle) to run their program to produce structures such as Fig. 2.

The setup instructions, server code, student starter code, and a complete API of Terp commands are open-source and available at https://github.com/jspacco/knoxcraftmod under an MIT License. We will also run a server through Amazon Web Services for faculty.

IV. Parallel Terps

Terps place blocks fairly slowly. Each action takes at least 1 "tick", and Minecraft runs at approximately 20 ticks per second (TPS).

For faster performance, students can parallelize their code by creating additional "threads" of Terps. A Terp thread is analogous to a thread in parallel programming. For example, the code in Fig. 3 creates 4 threads (only one is shown to save space) to draw the flag of Mauritius, while Fig. 4 demonstrates the visualization of 4 Terps (threads) drawing the flag of

```
String serverUrl = "http://server.edu:8080";
String name = "minecraftUserName";
String programName = "pyramid";
String desc = "Draw a pyramid.";

Terp terp = new Terp(programName, desc);

for (int base=8; base>=0; base-=2){
  for (int i=0; i<base; i++) {
    for (int j=0; j<base; j++) {
      terp.forward();
      terp.setBlock(TerpBlockType.OBSIDIAN);
    }
    for (int j=0; j<base; j++) {
      terp.back();
    }
    terp.right();
  }
  for (int i=0; i<base; i++) {
    terp.left();
  }
  terp.forward();
  terp.right();
  terp.up();
}
TerpUploader.upload(serverUrl, terp, name);
```

Fig. 1. Sequential code for drawing the flag of Mauritius.

Fig. 2. A Terp executing the pyramid program. The Terp itself is a modded vanilla Minecraft turtle that only follows instructions from a student's code.

Mauritius in parallel. Because students can see multiple Terps working together, they can easily see how parallelizing their code leads to performance improvements.

V. Implementation & Key Features

We piloted Knoxcraft Terps for one assignment across five sections of CS-1 taught in Java in Fall 2025 and Winter 2026. Students do need to own Minecraft to complete the visualization part of the assignment. They can run other students' uploaded Terp programs, and can upload their code for other students to execute. As long as a portion of the class has Minecraft, students can pair up to execute their Terp programs in the Minecraft world.

The assignment we gave was very basic — build a 3D structure that has walls and a roof. Beyond that, students

```
ParallelTerp terp = new ParallelTerp(
    "parallelPyramid",
    "Draw pyramid in parallel");

int length = 12;
int width = 4;
terp.addThread(t -> {
  for (int i=0; i<width; i++) {
    for (int j=0; j<length; j++) {
      t.forward();
      t.setBlock(TerpBlockType.RED_WOOL);
    }
    for (int j=0; j<length; j++) {
      t.back();
    }
    t.right();
  }
});

// threads for blue, yellow, green stripes
// syntax is the same

String name = "minecraftPlayerName";
TerpUploader.upload(serverUrl, terp, name);
```

Fig. 3. Parallel code for drawing the flag of Mauritius; only one thread is shown to save space.

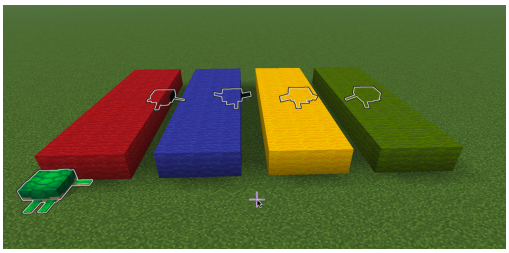

Fig. 4. In this parallelized drawing of the flag of Mauritius, we can see the outlines of four different Terps, each one drawing a different colored stripe.

were only limited by their imagination. During class, we were able to discuss improved speed as the main advantage of parallelizing their code. For example, we observed that four Terps yields approximately a factor of 4 speedup on small examples with a few dozen blocks. We also discussed the limitations of parallelism, such as the added complexity and diminishing performance for additional threads.

Knoxcraft Terps is a Forge-compliant Minecraft mod that is compatible with Minecraft 1.21.8 and Forge 58.0.6 at the time of publication; future releases will target future releases of Minecraft and Forge. This is important because, while the Minecraft modding ecosystem is sprawling, complex, and fast-moving, Forge is the oldest and most stable modding platform, and the safest target for stable future releases.

The mod disables most core features of Minecraft, such as mob spawning, block placement, block breaking, player combat, item usage, day/night cycle, portals, and weather. The world is always "superflat", the ground cannot be broken, and the only way students can place blocks is by writing and

running Terp code.

No student code actually executes on the server, which mitigates many security risks. Instead, students run their code through their IDE, which produces a JSON file containing every instruction the Terp should run in order. For the last line of their code, students call an `upload()` function that uploads the JSON to the server. When students then log into Minecraft and execute a Terp program, they are actually telling their Terp to execute a predetermined sequence of instructions.

Because the code runs client-side on the student's machine and JSON text files are actually uploaded to the server, Knoxcraft can support almost any programming language. Currently, we support Java, Python, and Blockly (basically Scratch), and it would be easy to add new languages.

The server can be hosted locally on a modest server, or in the cloud. Server setup instructions are available on https://github.com/jspacco/knoxcraftmod. We have hosted servers locally on one of our Linux servers, which handled about 70 students of which at most 10 were logged on at a time. Hosting a Minecraft server is not nearly as difficult as it may sound, and we can help you through the process. It takes about 10 minutes to get started.

We have also hosted servers on Amazon Web Services (AWS), and have sufficient AWS credits remaining to support a couple of faculty who are interested. Please contact us for details!

One final word of caution: This entire system only works with the Java Edition of Minecraft. It does NOT interoperate with Bedrock Edition, which is a C++ reimplementation that is not compatible with the vast majority of mods.

VI. Assessment

We evaluated students on very basic criteria: could they write code to build 2 structures, and was one of those structures recognizably a 3-dimensional object? During the lab session (70 minutes; whatever students do not finish during lab becomes homework for the next week), we encouraged students to think creatively and go beyond a minimal viable solution, such as just another flag similar to our sample code. Many students took the opportunity to create interesting structures; some students used block types that triggered interesting in-game effects, such as falling water and flowing lava.

In future implementations, we plan to add a more structured rubric that will encourage students to explore the parallel portion beyond their own personal interests.

VII. Evaluation

To evaluate the effectiveness of this assignment, we assigned it in 5 sections of our CS1 course: 3 sections with 70 students in Fall 2025, and 2 sections with 33 students total in Winter 2026, for a total of 103 students overall.

Upon completing the assignment, we asked students to complete a lab report with basic questions about how much time they spent on the assignment, and what they would improve for future versions. We used a 7-point Likert scale, with 7 being the most and 1 being the least. The first question we asked was their impression of the difficulty of programming their structures. Across the 69 student responses completed, the average was 5.04. This assignment was seen as difficult by them, with many students asking for additional examples of programs. We have expanded the set of programs covered when introducing the assignment to help alleviate these challenges. We additionally asked if the students had fun with the assignment, with an average score of 4.99. This matched the impression we had during the lab sessions with the assignment, where students were engaged and excited.

In Table I we report the number of students, successful student submissions, how many programs were strictly single threaded, how many used threads, how many total threads of execution students created, and the number and average number of instructions per program and per thread. Note that instruction counts are not the same as lines of code, since student-written code is effectively "unrolled" such that calling the `forward` method 100 times in a for loop becomes 100 calls to the `forward` method.

While we did not explicitly ask students to write parallel code or to measure speedups, we can give a rough estimate of what students might observe, at least in the aggregate. We see in Table I that in Fall 2025, 32 student programs created 187 threads, and each thread had about 347 instructions. Thus we estimate 5 threads per threaded program, which fits our data, we would expect about 1,700 total instructions per program. A Terp places blocks at a rate of about 20 per second, so placing 1,700 blocks with only one thread takes about 85 seconds, while splitting that work up among 5 Terp threads of about 340 instructions each takes only about 17 seconds. Saving a minute is a noticeable improvement for a CS1 student waiting for their 3D structure to complete.

Again, these are very high-level estimates, and we must remember that students were shown how to use threads of execution in Knoxcraft, but were not required to do so.

VIII. Discussion and Future Work

Students created many unique structures, often sharing them with other students near them in the classroom. This genuine student interest reinforced the idea of connecting programming to something many have interacted with in a recreational setting is an effective method to improve engagement. We included 2 representative examples of student creations, including a basketball court and a water well. These structures both met the requirements of the assignment, but also highlight the students' own creativity.

We also asked students what improvements they would like to see in future iterations of the assignment. Their responses, in general, asked for quality-of-life features and more examples to draw inspiration from. Quality-of-life, such as a method to undo the last Terp program, are in the development pipeline.

A change we intend to implement is a student showcase that lets students see what their peers have created in the assignment. This creates both an opportunity for students to see what past students have created and an incentive to create impressive structures themselves to be shown to future classes.

TABLE I
KNOXCRAFT TERPS PROGRAM STATISTICS BY TERM. "SINGLE PROGS" MEANS SINGLE-THREADED PROGRAMS, "THREADS" MEANS THE NUMBER OF THREADS OF EXECUTION STUDENTS INCLUDED IN THEIR CODE. "INSTRS" ARE THE NUMBER OF PROGRAM INSTRUCTIONS.

Term	Students	Single Progs	Threaded Progs	Threads	Instrs (single)	Instrs (threaded)	Avg Instrs per Single	Avg Instrs per Thread
Fall 25	70	258	32	187	262,142	64,902	1,016	347
Winter 26	33	89	22	157	109,579	98,408	1,231	627
Total	103	347	54	344	371,721	163,310	1,071	474

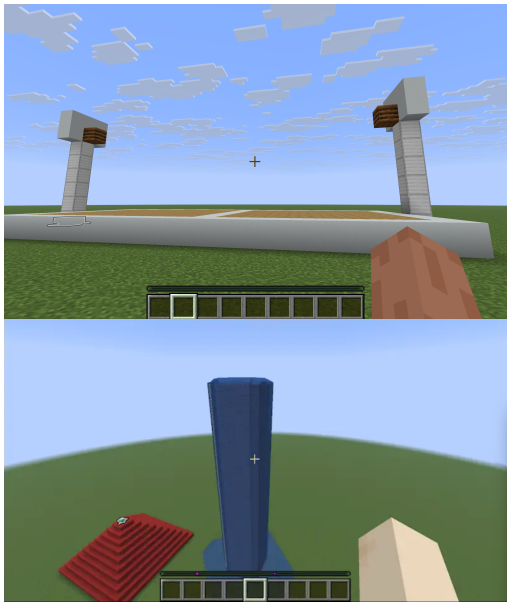

Fig. 5. Several student creations. One depicts a basketball court. The other is a well, utilizing water dynamics in-game.

The goal of these updates is to continue reducing the technical challenges students face while enabling them to work with parallel computing concepts.

Further, we intend to improve the example documentation to better demonstrate how students can utilize the assignment to learn about parallelism. One way we can better address the parallel concepts in the assignment is to make creating a parallel program a required portion. By making exploring parallelism a required part of the assessment, we can direct student effort to using these features. It remains the instructor's choice how much parallelism is required, allowing the assignment to be tailored to the course's needs.

IX. CONCLUSION

Knoxcraft is an exciting assignment that extends the ideas of Logo to a 3D environment. In our pilot study, students found the assignment to be highly engaging and created their own 3D structures. They were also able to parallelize their code to build structures faster. The assignment is freely available with support for adoption from the authors.

ACKNOWLEDGMENT

This work was supported by NSF grant award #2321020.

REFERENCES

[1] H. Bücker, J. Schoder, X. Suo, and D. Bunde, "Peachy parallel assignments (edupar 2025)," in *Proc. 15th NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar)*, 2025.

[2] C. Almeida, E. Shoop, D. García-Álvarez, A. Gonzalez-Escribano, D. Guerrero-Pantoja, C. Maloney, M. Pantoja, S. Rizzi, and D. Bunde, "Peachy parallel assignments (EduHPC 2025)," in *Proc. Workshop on Education for High-Performance Computing (EduHPC)*, 2025.

[3] S. Papert, *Mindstorms: children, computers, and powerful ideas*. USA: Basic Books, Inc., 1980.

[4] ACM/IEEE-CS/AAAI Joint Task Force on Computing Curricula, *Computer Science Curricula 2023 (CS2023)*. New York, NY: ACM, 2024. [Online]. Available: https://dl.acm.org/doi/book/10.1145/3664191

[5] ABET, Inc., "Criteria for accrediting computing programs, effective for review during the 2019–20 accreditation cycle," ABET, Tech. Rep., 2017. [Online]. Available: http://www.abet.org/wp-content/uploads/2017/11/CAC-Criteria-Version-2.0-2018-19.pdf

[6] J. Liu, "20 years of teaching parallel processing to computer science seniors," in *Proceedings of the Workshop on Education for High Performance Computing*. IEEE Press, 2016, pp. 7–13.

[7] A. Yazici, A. Mishra, and Z. Karakaya, "Teaching parallel computing concepts using real-life applications," *International Journal of Engineering Education*, vol. 32, no. 2, pp. 772–781, 2016.

[8] J. Eckroth", "A course on big data analytics," *Journal of Parallel and Distributed Computing*, vol. 118, pp. 166 – 176, 2018.

[9] E. Saule, "Experiences on teaching parallel and distributed computing for undergraduates," in *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2018.

[10] J. Mullen, C. Byun, V. Gadepally, S. Samsi, A. Reuther, and J. Kepner, "Learning by doing, high performance computing education in the mooc era," *Journal of Parallel and Distributed Computing*, vol. 105, pp. 105–115, 2017.

[11] R. Brown and E. Shoop, "Csinparallel and synergy for rapid incremental addition of pdc into cs curricula," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2012, pp. 1329–1334.

[12] D. W. Juedes and F. Drews, "Engineering a new curriculum: Experiences at ohio university in incorporating the ieee-tcpp curriculum initiative during a transition to semesters," in *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*. IEEE, 2012, pp. 1279–1282.

[13] S. K. Prasad, A. Gupta, K. Kant, A. Lumsdaine, D. Padua, Y. Robert, A. Rosenberg, A. Sussman, C. Weems *et al.*, "Literacy for all in parallel and distributed computing: guidelines for an undergraduate core curriculum," *CSI Journal of Computing*, vol. 1, no. 2, pp. 82–95, 2012.

[14] D. J. John and S. J. Thomas, "Parallel and distributed computing across the computer science curriculum," in *2014 IEEE International Parallel & Distributed Processing Symposium Workshops*. IEEE, 2014, pp. 1085–1090.