HELPUS: How Entry Level Programmers Use Synchronization

MIDN 1/C Matthew Friedel
Computer Engineering
United States Naval Academy
Annapolis, Maryland
mfriedel2021@gmail.com

Asst. Prof. Christian DeLozier
Computer Engineering
United States Naval Academy
Annapolis, MD
delozier@usna.edu

Asst. Prof. James Shey
Computer Engineering
United States Naval Academy
Annapolis, MD
shey@usna.edu

***Abstract*—Parallel programming is a powerful tool that can be used to increase performance in most applications, however the skillset required to program in parallel is unavailable to novice programmers. HELPUS is a visual programming language (VPL) that directly represents parallel programming concepts of traditional locks and execution dependencies. It is presented alongside a series of games that represent various parallel challenges faced in real-world applications. By working through the games using the VPL, novice programmers learn to recognize opportunities for adding parallelism in real-world applications.**

I. Introduction

A. *Performance vs Productivity*

Performance and productivity are the two main measurements of the usefulness of a programming language or tool. Performance is a measure of the processing power that it takes to run a specific calculation, while productivity is a measure of the effort that it takes to create a specific calculation. Typically, the performance or productivity of a programming language is measured in relation to another widely used language. An example that highlights the difference between these is ”Simulate a Sine wave over one period in C++ and in Matlab.” The resulting C++ program would use less processing power and execute faster than the Matlab program while still producing the same result, which indicates that C++ has a higher performance. The effort that it takes to create these two pieces of code, however, would be very different. Creating the C++ code would require writing several more lines and take longer, while the Matlab code would could be written in very few lines in a fraction of the time, which indicates that Matlab has a higher productivity.

B. *Novice Programmers*

Previous studies of parallel productivity involve studying post-grad students and programmers with extensive programming experience. This is because the majority of new parallel programming tools are aimed around increasing the performance and productivity of High Performance Computing (HPC) machines such as supercomputers. Over the last few decades, however, multi-core processors have become so widespread that almost every computer is a multi-core system. This has created a need for languages that have the ability to write parallel code that are accessible to novice programmers. Since the average novice programmer will not be writing parallel programs that take hours or days to execute, it is more important that the parallel languages they use have a more accessible syntax and are more focused around productivity than performance. This demographic has the most to gain by having access to a simpler parallel programming language.

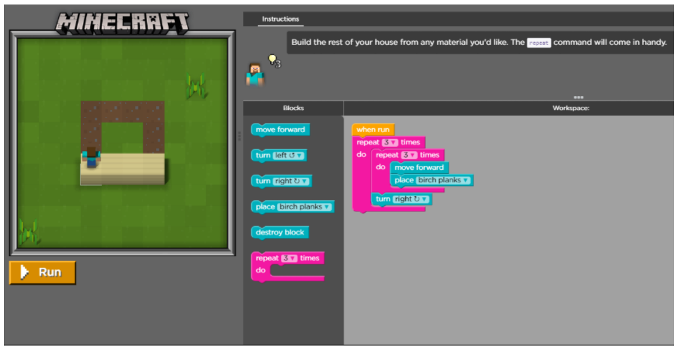
One of the main benefits in studying novice programmers will be their use of intuitive thinking. In parallel-programming classes, students are taught basic patterns of parallel problems that make it easier to create solutions. Since novice programmers will not have been taught those series of patterns, the ease of a programming language’s syntax to suggest possible solutions will be highlighted in the productivity measurement. Another benefit of studying novice programmers will be to highlight problems that might not appear in advanced programmers’ code. Any problems that consistently appear in several novices’ solutions will highlight what could be improved in future parallel programming languages and tools.

One of the main challenges in studying novice programmers will be the variance of their skill. Due to the wide range of abilities that subjects can bring to the table, it is possible the a measured difference in productivity could actually be due to the difference in the programmers’ ability to think about problems and not the difference in the use of the programming tools. Another challenge in studying novice programmers will be the significant increase in beginner struggles such as syntax errors.

II. Productivity Measurement Method

A. *Hour of Code Format*

The inspiration for the way this project measures productivity comes from hourofcode.com. This website teaches people the basics of programming using a visual programming language (VPL). At the start of a game, the user is presented with a programming challenge and blocks that each represent a line of code. The user must solve the challenge by manipulating the number and order of each of the blocks to create a program. At the beginning, the user is only given a small number of simple blocks. As they progress through the challenges, they are gradually given more blocks and more complicated problems. By the end of an hour, the user has been introduced to most of the major programming concepts that can be applied to any language.



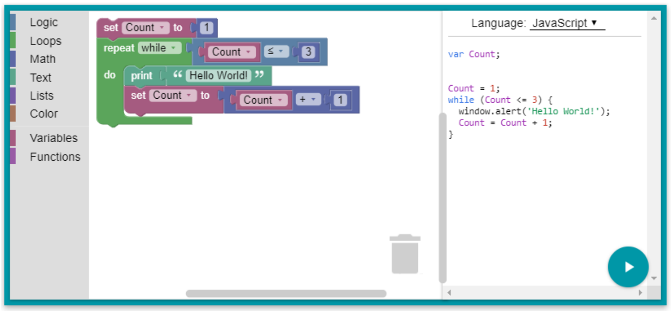
Build the rest of your house from any material you'd like. The repeat command will come in handy.

Fig. 1. Minecraft Hour of Code Game

While Hour of Code is based around teaching basic programming to people who have no programming experience, This project is based around teaching the basics of *parallel* programming to novice programmers. This project accomplishes this by creating an environment similar to Hour of Code for teaching parallel programming. It will also use a visual programming language that introduces blocks and problems, however the blocks will be representative of traditional locks and execution dependencies. The use of a VPL will mean that users will not be able to make syntax errors and the code will always be able to compile and run, which will reduce the amount of coding mistakes that novice programmers can make.

B. Blockly

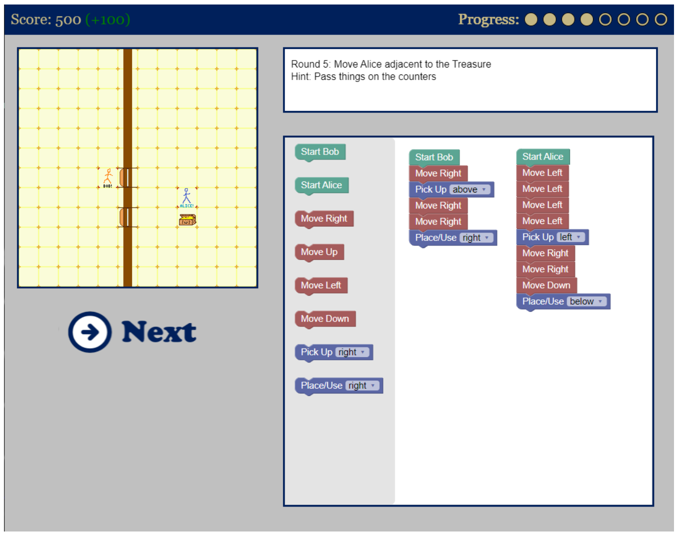
Blockly is an open-source library from Google that adds a VPL editor to a website that allows the user to compile and run code on their client from a browser. As an open-source platform, it has the built-in ability to create custom blocks. This will be the main method of creating the VPL. It will also make it possible to passively measure productivity metrics such as how long it took the user to create a solution, how many attempts they went through, and how many extra lines of code they created and later deleted.



```
var Count;


Count = 1;
while (Count <= 3) {
  window.alert('Hello World!');
  Count = Count + 1;
}
```

Fig. 2. Example of a Blockly editor



Round 5: Move Alice adjacent to the Treasure
Hint: Pass things on the counters

Fig. 3. HELPUS Game Layout

III. Implementation

A. HELPUS Layout

HELPUS is broken up into two sides. The left side presents the game board, and the right side contains the VPL as well as a description of the challenge the user must complete. The user is able to add and manipulate blocks in the coding window on the right side in order to solve the puzzle on the left.

B. Game Layout

Users begin the first round with only one character on the board and limited blocks available. This allows the user to learn the mechanics of the game easier. As they progress, a second character is added to the game which the user controls at the same time. More blocks are also added to the game which allow the characters to interact with other features on the board. As users are being introduced to all of the VPL commands that they can use, rounds are presented that represent various parallel challenges. Figure 3 shows one of the intermediate rounds that represents a two-step pipeline problem. The two characters are on either side of a partition, but are able to pass things across a counter. The task that is presented requires the first character to complete a task which must be handed off to the second character to complete.

C. Relationship between Board Elements and Programming Concepts

The game board is a 2D grid which represents a 2D array of memory addresses. Each character on the board represents a different core within a processor that accesses the same memory. Programs that control both characters simultaneously represent two different threads that each execute on a different processor at the same time. Generally, both characters are able to move around the entire board, however sometimes the game will have partitions placed on the screen that characters cannot pass through. This represents memory allocations that are only available to a specific process or thread. The partitions, however, contain spaces that both characters can access, which represents shared memory allocations.

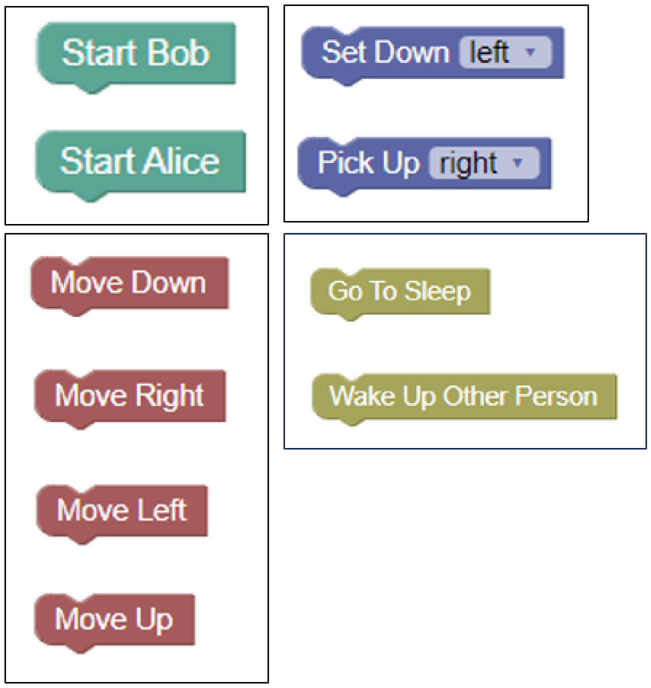


Fig. 4. Blocks available to the User

D. *Available Blocks*

The HELPUS VPL contains four categories of blocks that are added to the Blockly API. The first set are the Start blocks. These blocks delineate the different threads that the user creates. A Start block is given to the user for each character. When the game is run, each character's program begins at the same time. The second set of blocks are the movement blocks. These allow the characters to move around the 2D board to access different items. The character's movement is analogous to a memory pointer moving to various memory locations. The third set of blocks are the Pick Up/Set Down blocks. These allow the characters to interact with the board. Each character's inventory represents localized memory. The pick up/set down blocks are the way in which the character either simultaneously locks a specific item and moves it to their localized memory or unlocks a specific item and moves it to shared memory. These directly represent traditional memory lock and release commands. The fourth set of blocks are the Sleep/Wake Up blocks. If the user wants a character to stop at a certain point and wait for the other character to finish a task, the user inserts a Sleep command at the stopping point and a Wake Up command at the point in the other user's code that they want to wait to be completed. These blocks allow the user to explicitly state any dependency that exists in their code.

IV. Future Work

A. *Round Creation*

The next step in completing the HELPUS VPL is to create more game rounds. Currently, the game only has eight designed rounds that represent single-process, embarrassingly parallel, and two-step pipeline problems And introduce all of the VPL Blocks. Rounds should be created to represent further parallel patterns, such as replicable tasks and event-based coordination. Another other feature that could be added are heterogeneous computing. This would be represented by giving each of the characters a specific specialty that only they are able to complete. Another feature that could be added is forcing errors. Currently the language is deterministic, meaning that the exact same result will occur every time the program is run. The first step in forcing errors would be to add randomization of when each of the characters start and how long each command takes. Next, the ability for the VPL to detect potential conflicts would need to be added and used to decide when each of the characters should begin to force that conflict.

B. *Fielding*

Initially, the game will be fielded to other students and professors in the ECE department on an individual basis. This will allow a game developer to monitor their progress in-person to understand what aspects of the VPL need to be tweaked as well as real-time feedback from the players of the game. Following these initial tests and subsequent revisions, the game will be fielded to students who are currently in, or have recently completed, the SY110 and EC310 courses. This group will encompass students who have had at least an introduction to programming, but will still be novice programmers and will have not encountered parallel programming.

Acknowledgement

I would like to thank Prof. Devietti, Prof. Nelson, MIDN 1/C Michael Hamilton, MIDN 1/C Steven Hess, MIDN 2/C Rosemary Weston, and MIDN 3/C Johnathan Whorton who helped provide feedback and recommendations on initial versions of the project that directed the creation of many features.

References

[1] C. DeLozier, B. Lucia, A. Eizenberg, J. Devietti. *SOFRITAS: Serializable Ordering-Free REgions for Increasing Thread Atomicity Scalably*, University of Pennsylvania, March 2018
[2] E. Pasternak, R. Fenichel, A. Marshall. *Tips ofr Creating a Block Language with Blockly*, Google, 2017
[3] V. Pankratius, A. Adl-Tabatabai *A Study of Transactional Memory vs. Locks in Practice*
[4] L. Hochstein, V. Basili, M. Zelkowitz, J. Hollingsworth, J. Carver *Combining self-reported and automatic data to improve programming effort measurement*, University of Maryland, 2005
[5] L. Hochstein, J. Carver, F. Shull, S. Asgari. *Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers*, University of Maryland, 2005

V. Appendix

The HELPUS game can be accessed at this website: helpususna.altervista.org