Parallel Program Comprehension: A Mental Model Approach

Leah Bidlake **Eric Aubanel** **Daniel Voyer**
Faculty of Computer Science, Faculty of Computer Science, Department of Psychology
University of New Brunswick
leah.bidlake@unb.ca, aubanel@unb.ca, voyer@unb.ca

Abstract

Empirical research on mental model representations formed by programmers during parallel program comprehension is important for informing the development of effective tools and instructional practices including model based learning and visualizations. This work builds upon our initial pilot study in expanding the research on mental models and program comprehension to include parallel programmers. The goals of the study were to validate the stimulus set, consisting of programs written in C using OpenMP directives, and to determine the type of information included in expert parallel programmers' mental models formed during the comprehension process. The task used to stimulate the comprehension process was determining the presence of data races. Participants' responses to the data race task and the level of confidence in their responses were analyzed to determine the validity of the stimuli. Responses to questions about the programs were analyzed to determine the type of information that was included in participants' mental models and the type of models (situation and execution) participants may have formed. The results of the experiment indicate that the level of difficulty of the stimuli (accuracy rate of 80.88%) was appropriate and that participants were from our target population of experts. The results also provide insight into the type of information included in expert parallel programmers' mental models and suggest that the data structures aspect of the situation model (the identification of data structures) was not present, however there is evidence that the data structures aspect of the execution model (the behaviour of data structures) was present. Further investigation into this topic is needed, and this study provides a stimulus set that would be useful for those wanting to expand the research on mental model representations to include the parallel programming paradigm.

1. Introduction

Programmers are frequently tasked with modifying, enhancing, extending, and debugging applications. To perform these tasks, programmers must understand existing code by forming mental representations. The understanding of programmers' mental representations formed during parallel program comprehension is important for developing instructional practices as well as tools and visualizations that are effective in assisting programmers with tasks that are uniquely challenging in parallel programming such as debugging and verifying the correctness of parallel code. For example, data races are a type of bug that can occur only in parallel programming. Data races occur when multiple threads of execution access the same memory location without controlling the order of the accesses and at least one of the memory accesses is a write (Liao, Lin, Asplund, Schordan, & Karlin, 2017). Depending on the order of the accesses, some threads may read the memory location before the write and others may read the memory location after the write, which can lead to unpredictable results and incorrect program execution. Data races are difficult to detect and verify, requiring close consideration of the code by programmers as they will not appear every time that the program is executed. Since there are no applications that can do this work reliably, programmers need to be taught strategies for identifying data races and they need tools that help them better understand the parallel execution of code to assist them with this task.

Empirical research on mental representations formed by programmers during program comprehension has been predominately conducted using sequential code (Bidlake, Aubanel, & Voyer, 2022). Because of the considerable differences between parallel and sequential programming, it is impossible to determine if the findings of the empirical research using sequential code would resemble the mental representations formed during program comprehension using parallel code. The comprehension of parallel code requires

programmers to mentally execute multiple timelines that are occurring in parallel at the machine level. Therefore, parallel program comprehension may require additional dimensions to construct a mental representation (Aubanel, 2020). As a first step to uncover the mental models formed during program comprehension we conducted a pilot study (Bidlake et al., 2022) followed by the main study presented here, that examines the components of shared-memory parallel programs that are critical to programmers' understanding of the execution of the code to detect data races. This study also investigates the types of mental models formed during program comprehension of parallel code when detecting a data race. Another goal of this study was to validate the programs we developed that were written in C and used OpenMP directives and for parallelization, referred to here as stimuli or stimulus set. OpenMP is a collection of compiler directives, library routines, and environment variables for shared-memory parallelism in C, C++ and Fortran programs in multi-core architectures (OpenMP-ARB, 2013). Given the continued popularity of OpenMP (Gonçalves, Amaris, Okada, Bruel, & Goldman, 2016), the stimulus set would be relevant for replication studies and incremental research that builds on previous work in the psychology of programming field, as well as expanding research on mental representations to include the parallel programming paradigm.

- RQ1: Does the stimulus set produce a reasonable level of difficulty?
- RQ2: What type of information is included in programmers' mental representations of parallel programs?

2. Background

Research in program comprehension encompasses both the study of the cognitive processes used by programmers to understand code and how programming languages and tools support these cognitive processes (Storey, 2006). During the comprehension process, programmers form mental representations of the code they are working with (Détienne, 2001). The mental model approach to program comprehension using sequential code involves the construction of the program model and the situation model (Détienne, 2001). The program model is formed by programmers when applying structural knowledge to the code resulting in a surface level representation (Pennington, 1987). The situation model, also referred to as the domain model, is formed during program comprehension when prior knowledge is used to form an understanding of the main goals, meaning, or real world situation represented by the program (Pennington, 1987). The definition of the situation model has also been extended by Aubanel (2020) to include the identification of data structures since they are used to infer the program's meaning. In addition to the program and situation model, an execution model has been proposed that includes the behaviour of the program in terms of data flow and data structures (Aubanel, 2020). The behaviour of data structures is considered an important part of parallel program comprehension. Specifically, how the data structures are accessed and changed by one or more threads and the relationship between data structures, is particularly important for determining if a data race exists.

3. Method

This study was preregistered with Open Science Framework (OSF) prior to data collection and analysis, and the stimuli, data, and R code are publicly available (Bidlake, 2022).

3.1. Participants

Participants had to have experience programming in C and using OpenMP 4.0 directives to implement parallelization. To recruit participants an advertisement with a link to the study was placed in the March 2022 OpenMP newsletter, emailed to the OpenMP Language Committee mailing list, shared during the International Workshop on OpenMP 2022, posted in the Psychology of Programming Interest Group discussion board, and posted on social media. In the end, a final sample of 20 participants completed the experiment online. Participants could choose to receive a $10 e-gift card as an incentive. Participation was voluntary and the protocol was reviewed by the research ethics board at the University of New Brunswick. Five participants were excluded from analysis because their average response rates were less than three seconds which is not a reasonable amount of time to examine the code, leaving a final

sample of 15. Eleven participants were professionals and four participants were students. The mean age of participants was 39 years. Their mean amount of programming experience was 20 years and their mean amount of work experience was 12.53 years. The mean self-perceived level of expertise of participants, from 0 (novice) to 100 (expert), was 82.8 for programming expertise and 80.4 for parallel programming expertise. The mean self-perceived level of expertise of participants compared to their peers, from 0 (much worse) to 100 (much better), was 66.73 for programming and 70.07 for parallel programming.

3.2. Materials

The stimulus set used in this study was taken from our pilot study (Bidlake et al., 2022). They are programs written in C using OpenMP 4.0 directives and clauses with no comments or documentation. The selection of OpenMP directives and clauses for the stimuli was based largely on the set of directives and clauses referred to as The Common Core (Mattson, 2019).

In the pilot study (Bidlake et al., 2022), participants were asked what cues or program components they used to determine whether or not there was a data race for 20 of the stimuli. The responses varied greatly ranging in level of detail and also in the number responses they provided making it challenging to analyze these data. We speculate that the reason for the variation in responses is due to the open-ended nature of this question. In the main study we used an approach similar to Burkhardt, Détienne, and Wiedenbeck (2002) and included questions about specific components of the code to determine the type of information that is included in the mental representations formed by participants. In adding questions for the main study, we also considered that, in addition to the program and situation model, an execution model has been proposed that includes the behaviour of data structures (see section 2). The importance of data structures was also evident by the responses in the pilot study (Bidlake et al., 2022). Therefore, in the main study, 12 of these 20 stimuli, six with a data race and six without, were followed by more specific questions regarding the data structures in the programs (see Table 2). For eight of the stimuli, four with a data race and four without, participants were asked what cues or program components they used to determine whether or not there was a data race. The stimuli were presented to participants in a random order, therefore randomizing the order of the questions. The stimuli was not available for participants to look at while answering the questions and for each question participants were provided a text box to type their answer.

3.3. Procedure

Participants completed the experiment online. The experiment was developed using PsychoPy 3 (Peirce et al., 2019), an open source software package, and Pavlovia was used to host the experiment online. Qualtrics was used to administer the consent form at the beginning of the experiment, the questionnaire at the end of the experiment, and to collect participants' emails if they chose to receive an incentive.

The link to the online experiment was included in the advertisement for the study. Participants were instructed to determine as quickly and accurately as possible if each program contained a data race and respond by pressing the 'y' or 'n' key on their keyboard. The 76 experimental stimuli were presented to the participants in random order. For each stimulus, participants were given a time limit of 60 seconds to view the stimulus and respond to the data race task. If participants exceeded the time limit, exposure to the stimulus ended and they were asked to decide if the stimulus contained a data race or not. Participants were asked after each data race question to rate their level of confidence in their answer using a visual analogue scale that ranged from "Not Confident" to "Very Confident". Twenty of the stimuli were followed by a question about the code (see section 3.2). There was no time limit for answering the questions about the code. After completing the experiment portion, participants were redirected to the questionnaire documenting their level of education, age, programming experience, and their perceived level of programming expertise (Feigenspan, Kastner, Liebig, Apel, & Hanenberg, 2012). Participants were then asked if they would like to receive the e-gift card, and lastly, were redirected to the debriefing.

4. Results

For each trial, the accuracy (1 = correct, 0 = incorrect), level of confidence (0 = not confident to 100 = very confident), response time, and whether each trial had a race condition (y) or no race condition (n) were recorded and analyzed for 15 participants using the statistical software program R (R Core Team, 2021). The responses to questions following select stimuli were also collected and analyzed for 15 participants. The results of the experiment were analyzed in two contexts: identification of data races and responses to questions about the code.

4.1. Identifying Data Races

The mean accuracy for correctly determining whether or not a program contained a data race was 80.88% (SD = 39.34), the mean level of confidence in their responses was 83.09 (SD = 21.04), and the mean response time was 28.54 seconds (SD = 15.97). A one sample t-test was performed to test the null hypothesis that the sample accuracy was equal to chance (mu = .50) with a 95% confidence interval. The results indicated that the accuracy of participants was significantly higher than chance, $t(14) = 12.52$, $p < .001$.

Spearman rank correlations were computed to examine the relationship between accuracy and confidence, and the measures of self-perceived expertise, to determine if participants were from our target population. The strength of the correlations was determined using the scale for r values: r = .1 is weak, r = .3 is moderate, r $\geq$ .5 is strong (Cohen, 1988). The correlation coefficients are listed in Table 1. There were moderate positive correlations between accuracy and self-estimation of expertise in programming compared to their peers and between accuracy and self-estimation of expertise in parallel programming compared to their peers. When correlating their years of experience, a common measure of expertise, and accuracy there was no significant correlation.

	Accuracy	Confidence
	r	r
Number of years of programming experience	-.11	.61
Number of years of work experience	-.09	.47
Self-estimation of expertise in programming compared to their peers	.37	.12
Self-estimation of expertise in parallel programming compared to their peers	.40	.58
Self-estimation of level of expertise in programming (novice to expert)	-.20	.63
Self-estimation of level of expertise in parallel programming (novice to expert)	.03	.78

Table 1 – Spearman correlation coefficients.

The data were also analyzed using a mixed linear model that was fitted with the **lme4** package (Bates, Mächler, Bolker, & Walker, 2015) in R. The design used confidence as a continuous predictor, race condition as a repeated measures factor, and accuracy as the dependent variable. Using generalized linear mixed model with the **glmer** procedure from the **lme4** package, we first determined the best fitting model. Likelihood ratio values were then obtained with the **Anova** procedure from the **car** package (Fox & Weisberg, 2019), using the best fitting model with accuracy as the dependent variable for confidence and race condition. Results showed a significant effect of confidence, $LR = 38.55$, $p < .001$ and race condition, $LR = 30.63$, $p < .001$. The main effect of confidence in the data race task was significant, indicating that as confidence increased, accuracy on the data race task also increased (slope = .019, SE = .01, z = 3.80, $p < .001$).

4.2. Responses to Questions

The responses to the specific questions about the stimuli were also analyzed to determine the types of mental models formed during program comprehension (see Table 2). Questions that participants did not provide a response to were marked as incorrect since there was no time limit. Question 7, that asked "What is the value of 'z' at the end of the program?", was omitted from the analysis as some participants

stated that the value was undefined since there was a data race whereas some participants answered according to what the value would be if there was no data race.

The 11 questions included in the analysis were six questions pertaining to the behaviour or mutation of data structures by one or more threads (execution model) and five questions related to the contents and size of the data structures (situation model). For five of the six questions pertaining to the relationship between data structures and how data structures are accessed and changed by one or more threads, participants were able to answer correctly with accuracy significantly greater than zero ($t(72) = 4.33$, $p < .001$) (see Table 2). However, no participants were able to correctly answer any of the five questions pertaining to the specific contents of the data structures and the size of data structures. There was no significant correlation between the accuracy of the specific questions related to the execution model and the accuracy of detecting a data race ($r = -0.06$, $p = 0.463$).

		Data	Question Accuracy		Data Race Accuracy	
Question	Model	Race	Mean	SD	Mean	SD
1. How is array ‘a’ being updated?	E	y	40.00	50.71	80.00	41.40
2. How are array ‘a’ and array ‘b’ related?	E	y	33.33	48.80	73.33	45.78
3. How are array ‘a’ and array ‘b’ related?	E	y	13.33	35.19	73.33	45.78
4. How are the elements for array ‘a’ determined?	E	n	6.67	25.82	60.00	50.71
5. How is array ‘a’ updated after values were initially assigned?	E	n	6.67	25.82	93.33	25.82
6. What value is the last element in each row assigned?	E	n	0	–	80.00	41.40
8. What is the value of a[0]?	S	y	0	–	80.00	41.40
9. What are the dimensions of matrix ‘a’?	S	y	0	–	73.33	45.77
10. What is the length of array ‘a’?	S	n	0	–	100.00	0
11. What are the contents of array ‘a’?	S	n	0	–	80.00	41.40
12. What are the contents in the first column of matrix ‘a’?	S	n	0	–	80.00	41.40

Table 2 – Mean accuracy (percentage of correctness), when answering specific questions and determining the presence of a data race, as a function of specific question, model (E = execution, S = situation), and data race condition (y = data race, n = no data race).

Question 1, “How is array ‘a’ being updated?”, followed a stimulus where the data race occurred due to how array ‘a’ was updated. This may have been why the most participants were able to answer this question correctly. For this question, six participants accurately recalled how the array was updated with varying levels of detail. Two participants recalled the exact assignment statement (a[i+1] = a[i];). Another participant recalled the exact if condition that the update was made under (if a[i+1] > a[i]) although they only recalled the assignment statement in general terms as “…assigned to another value in the array”. Similarly, another participant recalled it was updated “according to valor (value) of other iteration”. This implies that observing that the update to the array was from another part of the array or used a different index value than the current position was important in determining the presence of the data race.

For question 2, “How are array ‘a’ and array ‘b’ related?”, followed a stimulus where the data race only involved array ‘a’. Five participants were able to give accurate responses describing the relationship between the arrays. Four participants responded directly that they were unrelated and one participant stated that they were both updated in a nowait clause and thus could happen concurrently, which was also correct. Another participant stated they could not recall if they were related but did state correctly

that the data race was with array ‘a’ and therefore they did not look any further into the code.

Question 3, “How are array ‘a’ and array ‘b’ related?”, was repeated following a stimulus that also contained a data race, however, only two participants answered correctly. The data race in this stimulus did not involve array ‘a’ or ‘b’ which may explain why so few participants were able to answer correctly. Two participants who were not able to recall the answer did include in their responses that the race involved a scalar variable or the value‘ z’ which was correct. Another participant responded that they were both used in loops which was accurate but did not answer the question posed.

For question 4, “How are the elements for array ‘a’ determined?”, the stimulus it followed did not contain a data race. Only one participant was able to answer this question correctly. However, another participant’s answer was correct for the stimulus they saw before this one which did contain a data race.

Question 5, “How is array ‘a’ updated after values were initially assigned?”, followed a stimulus that did not contain a data race. Only one participant was able to correctly answer this question. Another participant did correctly identify that the update occurred within a nowait clause, however, their response was marked as incorrect because they said that it was updated by another array, when in fact it was updated using the original values from array ‘a’.

The responses to the specific questions were also analyzed to determine if the presence of a race condition affected the accuracy of the specific questions about the behaviour of data structures in the code (questions 1-6). The design used accuracy as the dependent variable and race condition as predictor. Using linear mixed model with the **lmer** procedure from the **lme4** package, we first determined the best fitting model was the fixed slope model given that the random slope model had a singular fit. Likelihood ratios were then obtained with the **Anova** procedure using the best fitting model with accuracy as the dependent variable for race condition. Results showed the effect of race condition on accuracy of the specific questions was significant, $LR = 11.01$, $p < .001$. The mean accuracy of the specific questions about the behaviour of data structures for stimuli without a data race was 4.44% (SD = 20.84) and the mean accuracy of the specific questions for stimuli with a data race was 28.89% (SD = 45.84).

The responses to the question asking what cues or program components participants used to determine whether or not there was a data race were also analyzed. There were 116 responses to this question collected. Based on the responses, categories were created that correspond to the components of the code that were included (see Fig. 1). If responses included more than one component of the code they were counted under each of the categories referred to in the response. The most common component of the code that was used to determine the presence of data races was the reading from and/or writing to memory with 64 responses referring to reading, writing, modification, accessing, or updating of variables or memory.

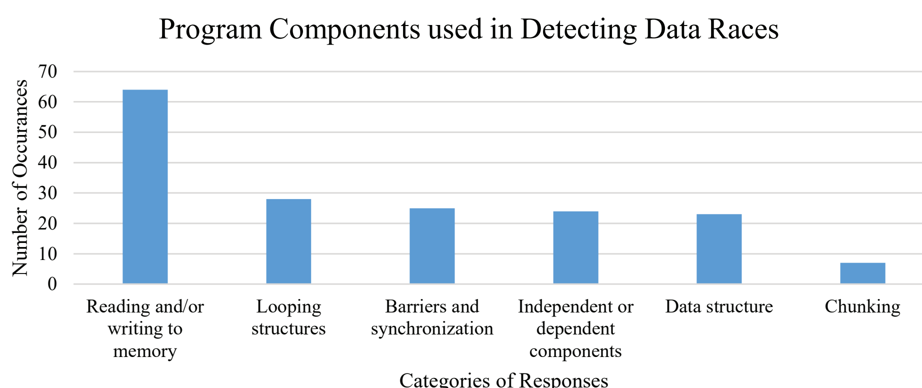

Figure 1 – Responses to question asking what cues or program components were used to determine whether or not there was a data race.

Twenty-eight responses referred to looping structures or iterations of a loop. These responses varied in detail with some referring generally to the "iteration space" and "access pattern of the loops", others specified they were looking for "overlap between loop bounds" and "independent loop iterations". Other responses used the presence of "barrier(s) between the loops" to verify the correctness of the code.

Barriers, both implicit and explicit, and synchronization, were referred to in 25 of the responses. In some responses where the barrier was implicit, participants made statements that implied they visually saw a barrier:

- "I saw that there was a barier between the loops and that in the second loop a thread didn't work with adjacent i indices."
- "I looked for the barrier between writing x and reading x in the second loop. Since that barrier was there, I determined there was no race."

Participants indicated in 24 responses that they were examining the code to determine if there were dependent or independent components (e.g.: tasks, sections, portions of arrays, data, loops, iterations). Responses that used dependencies to describe how they determined whether or not there was a data race tended to be less descriptive and included fewer details. For example:

- "Backwards dependency on a loop"
- "dependencies between sections"
- "Independent loop iterations"

There were 23 responses that referred to the data structure or array in the code. These included general references such as "array", referring to the indices of the array, and specifically using the variable name. In 19 responses, the indices of the data structure were used when determining the presence of a data race. The responses again varied in detail with some generally referring to the "index" or "indices" while others gave examples using variables for the indices: "out-of-bounds access to array a[i][j-1] which causes actual access at a[i-1][...]" and "the unshared j direction rather than i". In other responses, participants recalled specific index values that were used in verifying the correctness of the code:

- "...the first assignment (a[0]) was not synchronized..."
- "The last task reads a(0) while the first one writes the same data."
- "The barrier ensured that a element 0 was set before the loop started. Thus a0 was set before interation i started accessing element 0."

Seven of the responses described breaking the code into chunks described as blocks, regions, or sections[1]. Participants indicated the use of barriers, parallel regions, or loops when chunking. Experiments conducted using sequential code have also found that programmers use chunking to develop a mental representation of a program (Shneiderman, 1976; Barfield, 1986, 1997). In program comprehension, chunking is a cognitive process where groups of related lines of code are recalled together as a unit rather than each line individually (Barfield, 1997).

- "Break programm into implicitly 'barriered' blocks, then find Read/Write dependecy"
- "... Only check inside each "region" determined by barriers/synchronization."
- "Identify parallel blocks, look for reads overlapping with writes between blocks"
- "Divide programm into to blocks, analyze parallel loop iterations"
- "Checking if follow-up sections have to wait in a barrier for the previous one to finish."

[1]The sections directive was not used in the stimuli these responses were given for.

5. Discussion

This section discusses the results of our study with respect to our two research questions presented in the introduction (see section 1).

5.1. Level of Difficulty

Participants' self-estimation of expertise in programming and parallel programming compared to their peers had the strongest correlations with their accuracy on the data race task (see Table 1). This implies that their self-estimation of expertise was more accurate when comparing themselves to their peers as a point of reference. Given the mean level of expertise of participants compared to their peers for parallel programming was 70.07 (0 = much worse, 100 = much better), we believe that our participants can be classified as expert parallel programmers. We suspect, given the weak correlations between self-estimations of expertise and accuracy, participants may have found it difficult to measure their expertise (novice to expert) without a point of reference, i.e.: who is a novice and who is an expert.

The result of the t-test showing that the accuracy of participants is significantly higher than chance suggests that the level of difficulty of the stimuli is appropriate. The mean response times for all participants were within the 60 second exposure limit implying that the time exposure limit was appropriate. The results of the study show that as confidence increased, accuracy on the data race task also increased. Confidence based assessment which combines accuracy and confidence levels provides four regions of knowledge: uninformed (wrong answer with low confidence), doubt (correct answer with low confidence), misinformed (wrong answer with high confidence), and, mastery (correct answer with high confidence) (Maqsood & Ceravolo, 2018). In addition to the finding that the accuracy of participants was significantly higher than chance, the significant effect of confidence in the data race task indicates that higher performing participants had greater confidence in their answers and therefore had higher levels of mastery of the programming language and parallelization directives, both suggesting that it is unlikely participants were guessing.

5.2. Mental Representations

The ability of participants to correctly answer specific questions pertaining to the behaviour of the data structures, suggests the presence of an execution model. However, since no participants were able to answer any questions related to the contents and size of the data structures, this implies that the data structure aspect of the situation model was not formed during the data race task. The results also showed that when the specific questions pertaining to the behaviour of data structures were about stimuli that had a data race, participants performed significantly better than when the stimuli did not have a data race. One possible explanation is because to conclude that a data race exists programmers would need to devise an ordering of thread execution that would result in a race and then be able to convince themselves that ordering is possible. By developing this execution ordering and testing it to confirm the race is possible, programmers may have developed a stronger execution model and therefore were able to more accurately answer specific questions about the behaviour of the data structures when a data race was present, especially when the data race involved the data structure.

The responses to the question of what cues or program components participants used in determining the presence of a data race can be used to inform instructional practices in parallel programming education. Reading from and writing to memory was the most common consideration when detecting data races. Although this may seem obvious, to detect all of these instances and then determine if at least one write may occur at the same time as a read is not trivial. Reading and writing may occur in obvious places in code such as assignment statements but they can also occur within condition statements and function calls, for example. Even though assignment statements are easily identified, it has been shown that novice programmers often possess non-viable mental models of assignment statements (Ma, Ferguson, Roper, & Wood, 2007). Consequently, learning to identify data races may be even more challenging if programmers do not completely understand what is occurring in memory during a basic assignment statement. Information pertaining to looping structures including the iteration space, overlap between loop bounds, and access patterns of the loops were also important in determining the presence of data races. Being able to observe dependencies in the code either between or within looping structures or

between different sections of the code was also essential to data race detection. Given the importance of barriers in identifying data races, identifying implicit barriers in particular is critical. In OpenMP, for example, programmers must be able to recall for each directive if an implicit barrier follows or not. OpenMP uses braces for implicit barriers and in cases where there is only a single line of code in a region braces are not required. The braces used for implicit barriers can easily be missed and also confused with braces used for other structures such as loops. Data structures, in this case arrays, were also frequently mentioned in the responses as program components used to identify data races. Unfortunately, arrays are among the topics that novice programmers feel they have the most difficulty with (Piteira & Costa, 2012), potentially compounding the challenges of teaching programmers how to identify data races.

One interesting observation came from looking at the stimuli that participants had the most difficulty in finding the data race. The stimulus with the lowest accuracy (mean = 26.67%, SD = 45.77) that used only the most basic OpenMP directives and clauses was one that involved shared memory and two-dimensional arrays (see Fig. 2). This stimulus was followed by the question asking what program components or cues were used in determining whether or not there was a data race. Nine participants that responded no to the data race task stated there were no loop carried dependencies. These participants may not have looked closely enough at the code; they would have been correct if the inner for loop had started at one instead of zero in line 18 of Fig. 2. Only one participant correctly indicated there was a data race and identified that it was from a loop carried dependency. Four participants stated in their answer that the program was incorrect and three of them specified it was incorrect due to an out of bounds error (see line 19 of Fig. 2). In programming languages such as Java this statement would be correct, however, this is not the case in C. The typical visualization of a two-dimensional array is a matrix in the form of rows and columns where the elements in the left and rightmost columns have no adjacent elements to their left or right (Leopold & Ambler, 1996). This visualization matches the limitations in programming languages like Java where indices are constrained but does not accurately reflect how these structures are stored in memory. In memory the rows are contiguous, meaning that the last element of the first row and the first element of the second row are adjacent. Despite the confusion the matrix visualization of two-dimensional arrays may cause as demonstrated here, these visualizations are commonly used in the context of the C programming language (Mattson, 2019; Srivastava, 2020). Given that the matrix visualization of two-dimensional arrays does not easily transfer from one programming language to another, and in fact may contribute to misunderstandings, we should also be questioning how likely it is that our visualizations and models of sequential programming will transfer to parallel programming.

6. Threats to Validity

The sitmulus set used in this study was taken from our pilot study (Bidlake et al., 2022), and although precautions were taken to reduce bias in the selection of OpenMP directives, the types of errors that were introduced to create data races may have been biased towards the background knowledge and experiences of the authors'. Bias that may have been introduced by participants because of their prior experiences with data races that may have caused them to look for mistakes they commonly make and having more familiarity with some directives than others. Another threat to validity is our lack of control over the experiment environment. Because the study was conducted online, participants may have been in a distracting environment.

Another bias that may have been introduced was the specific questions about the code and the selection of which stimuli they would be asked for. The level of difficulty of the questions may not have been equitable depending on what data structures the question referred to and if there was a data race. For example, if the question asked about the data structures that were directly involved in the data race, this may have been an easier question to answer than if the data structure was not involved directly with the data race or if there was no data race.

The small sample size we ended up with was likely the biggest limitation of our work as it greatly reduced statistical power. Prior to conducting our main study, we performed a power analysis on the pilot study (Bidlake et al., 2022) and also considered the minimum recommendation proposed by Brysbaert

```
1   #include <stdio.h>
2   #include <omp.h>
3
4   int main(int argc, char* argv[]){
5
6       int n = 5, m = 10;
7       int a[n][m], i, j;
8
9       #pragma omp parallel
10      {
11          #pragma omp for
12          for(j = 0; j < m; j++){
13              a[0][j] = 2 + j;
14          }
15
16          #pragma omp for private(j)
17          for (i = 1; i < n; i++){
18              for (j = 0; j < m; j++){
19                  a[i][j] = a[i][j-1] * 2;
20              }
21          }
22      }
23
24      printf ("%d\n", a[n-1][m-1]);
25
26      return 0;
27  }
```

Figure 2 – Stimulus containing a data race.

and Stevens (2018) of 1600 observations per condition. The target for the main study was to recruit 60 participants (2280 observations) to ensure adequate power. Despite our efforts to recruit participants, we were not able to meet our target. In the end we had 570 observations (38 observations per condition x 15 participants). We suspect this was due to the specific programming language and API knowledge required to complete the study.

7. Conclusion

Parallel architectures, such as multi/many-core processors, are now widely available, however, to exploit this processing power we need more programmers capable of working with parallel applications and applying parallel computing models (Gonçalves et al., 2016). The lack of programmers with this specialization may be due to its reputation of being more challenging than sequential programming (McKenney, 2017). Despite having found no empirical evidence that parallel programming is harder to learn, one possible explanation as to why may it may be perceived as such could be that the teaching strategies, models, and visualization techniques that worked for sequential programming are not transferable to the parallel programming paradigm.

The results of this study lead us to believe that expert parallel programmers' mental models formed during program comprehension given a data race task consist of the data structure components of the execution model. Therefore, visualizations that allow programmers to better understand the relationships between data structures how they are accessed and changed by one or more threads of execution could assist them in detecting data races. Visualizations of data structures should also accurately match the indexing capabilities of the programming language.

Further research is needed on the mental representations formed by programmers when interacting with parallel code to determine how best to model this for learners and to inform the development of visualizations and tools that are effective in assisting with tasks such as detecting data races. We have developed a stimuli set and experimental design that produces a reasonable level of difficulty and would be relevant for replication studies and for expanding the psychology of programming research to include parallel programmers.

8. References

Aubanel, E. (2020). Parallel program comprehension. In *31st Annual Workshop of the Psychology of Programming Interest Group (PPIG 2020)* (pp. 8–16).

Barfield, W. (1986, 1). Expert–novice differences for software: Implications for problem-solving and knowledge acquisition. *Behaviour & Information Technology*, *5*(1), 15–29. doi: 10.1080/01449298608914495

Barfield, W. (1997, 12). Skilled performance on software as a function of domain expertise and program organization. *Perceptual and Motor Skills*, *85*(3, Pt 2), 1471–1480. doi: 10.2466/pms.1997.85.3f.1471

Bates, D., Mächler, M., Bolker, B., & Walker, S. (2015). Fitting linear mixed-effects models using lme4. *Journal of Statistical Software*, *67*(1), 1–48. doi: 10.18637/jss.v067.i01

Bidlake, L. (2022, Feb). *Validation of stimuli for studying mental representations formed by parallel programmers during parallel program comprehension.* OSF. Retrieved from `https://osf.io/fcnyx/?view_only=d8b98b812f164de28dd12eb29d35fd14` doi: 10.17605/OSF.IO/FCNYX

Bidlake, L., Aubanel, E., & Voyer, D. (2022). Pilot study: Validation of stimuli for studying mental representations formed by parallel programmers during parallel program comprehension. In *33rd Annual Workshop of the Psychology of Programming Interest Group (PPIG 2022)* (pp. 47–56).

Brysbaert, M., & Stevens, M. (2018). Power analysis and effect size in mixed effects models: A tutorial. *Journal of Cognition*, *1*(1). doi: 10.5334/joc.10

Burkhardt, J.-M., Détienne, F., & Wiedenbeck, S. (2002). Object-oriented program comprehension: Effect of expertise, task and phase. *Empirical Software Engineering*, *7*, 115–156. doi: 10.1023/A:1015297914742

Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed ed.). Hillsdale, N.J.: L. Erlbaum Associates. Retrieved from `http://www.gbv.de/dms/bowker/toc/9780805802832.pdf`

Détienne, F. (2001). *Software design-cognitive aspect.* Springer Science & Business Media.

Feigenspan, J., Kastner, C., Liebig, J., Apel, S., & Hanenberg, S. (2012, Jun). Measuring programming experience. In *20th IEEE International Conference on Program Comprehension (ICPC)* (pp. 73–82). IEEE. doi: 10.1109/ICPC.2012.6240511

Fox, J., & Weisberg, S. (2019). *An R companion to applied regression* (Third ed.). Thousand Oaks CA: Sage. Retrieved from `https://socialsciences.mcmaster.ca/jfox/Books/Companion/`

Gonçalves, R., Amaris, M., Okada, T., Bruel, P., & Goldman, A. (2016, Jan). OpenMP is not as easy as it appears. In *2016 49th Hawaii International Conference on System Sciences (HICSS)* (p. 5742–5751). doi: 10.1109/HICSS.2016.710

Leopold, J., & Ambler, A. (1996, Sep). A user interface for the visualization and manipulation of arrays. In *Proceedings 1996 IEEE Symposium on Visual Languages* (p. 54–55). doi: 10.1109/VL.1996.545267

Liao, C., Lin, P.-H., Asplund, J., Schordan, M., & Karlin, I. (2017). Dataracebench: A benchmark suite for systematic evaluation of data race detection tools. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis* (pp. 11:1–11:14). ACM. doi: 10.1145/3126908.3126958

Ma, L., Ferguson, J., Roper, M., & Wood, M. (2007, Mar). Investigating the viability of mental models held by novice programmers. In *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education* (p. 499–503). New York, NY, USA: Association for Computing Machinery. Retrieved from `https://doi.org/10.1145/1227310.1227481` doi: 10.1145/1227310.1227481

Maqsood, R., & Ceravolo, P. (2018, Jul). Modeling behavioral dynamics in confidence-based assessment. In *2018 IEEE 18th International Conference on Advanced Learning Technologies (ICALT)* (p. 452—454). doi: 10.1109/ICALT.2018.00112

Mattson, T. (2019). *The OpenMP Common Core: A hands on exploration.* Argonne Training Program on Extreme-Scale Computing (ATPESC).

McKenney, P. E. (2017). Is parallel programming hard, and, if so, what can you do about it? (v2017.01.02a). *CoRR*, *abs/1701.00854*. Retrieved from http://arxiv.org/abs/1701.00854

OpenMP-ARB. (2013). *OpenMP Application Program Interface Version 4.0* (Tech. Rep.). OpenMP Architecture Review Board (ARB). Retrieved from http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf

Peirce, J., Gray, J. R., Simpson, S., MacAskill, M., Höchenberger, R., Sogo, H., ... Lindeløv, J. K. (2019, Feb). Psychopy2: Experiments in behavior made easy. *Behavior Research Methods*, *51*(1), 195–203. doi: 10.3758/s13428-018-01193-y

Pennington, N. (1987, July). Stimulus structures and mental representations in expert comprehension of computer programs. *Cognitive Psychology*, *19*(3), 295–341. doi: 10.1016/0010-0285(87)90007-7

Piteira, M., & Costa, C. (2012, Jun). Computer programming and novice programmers. In *Proceedings of the Workshop on Information Systems and Design of Communication* (p. 51–53). New York, NY, USA: Association for Computing Machinery. Retrieved from https://doi.org/10.1145/2311917.2311927 doi: 10.1145/2311917.2311927

R Core Team. (2021). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria. Retrieved from https://www.R-project.org/

Shneiderman, B. (1976, 6). Exploratory experiments in programmer behavior. *International Journal of Computer & Information Sciences*, *5*(2), 123–143. doi: 10.1007/BF00975629

Srivastava, A. K. (2020). *A practical approach to data structure and algorithm with programming in c*. Oakville, ON: Arcler Press.

Storey, M.-A. (2006, 9 01). Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, *14*(3), 187–208. Retrieved from https://doi.org/10.1007/s11219-006-9216-4 doi: 10.1007/s11219-006-9216-4