A Qualitative study on challenges faced in multithreading in Julia

by

Harshita

Bachelors of technology, PTU, 2020

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

Master of Computer Science

In the Graduate Academic Unit of Computer Science

Supervisor(s):	Eric Aubanel, Ph.D., Faculty of Computer Science
Examining Board:	Michael Fleming, Ph.D., Faculty of Computer Science
	Daniel Rea, Ph.D., Faculty of Computer Science
	Luc Theriault, Ph.D., Department of Sociology

This thesis is accepted by the
Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

June, 2023

Abstract

Multithreading is defined as a technique that allows a single processor to execute multiple simultaneously. Developers working on high performance or resource-intensive applications may be required to write multithreaded code to take advantage of these performance benefits. Writing multithreaded code is more complex and difficult than writing single-threaded code.

To help developers, it is important to understand their interests and difficulties they face. This thesis examines the challenges faced by developers in multithreading in Julia using qualitative content analysis. A total of 211 posts were collected from online discussion forums called Stack Overflow and Julia Discourse. The data were collected using inductive qualitative content analysis.

The study found that developers face a lot of performance related issues for various reasons, one of the major ones being memory allocation. They also ask many questions about usage of macros, threads, etc. The results obtained are presented in the form of themes.

Dedication

I dedicate this thesis to my mother, Mrs Manju Bala, who has always been my source of inspiration, encouragement, and support. She has been my rock throughout my academic journey, and I am forever grateful for her unwavering love and support. This thesis is a testament to her strength, wisdom, and love. I love you, Mom.

Also, I would like to thank my sister, Loveleen for her continuous support and encouragement throughout the process of doing this research.

Acknowledgements

I would like to thank Dr. Eric Aubanel for being my supervisor, guiding me throughout the path of this research and introducing me with the concept of parallel programming. Also I would like to thank Leah Bidlake for her support throughout the process of researching and writing the thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Objective

Multithreading is a powerful technique that is used to achieve efficiency and better performance. Software developers are increasingly required to write multithreaded code. However, multithreaded code presents challenges including thread initialization, data decomposition, race condition detection and memory allocation [27]. Code is multithreaded when multiple threads are executed concurrently but independently; otherwise, it is sequential. Most developers think sequentially and they find writing multithreaded code difficult[6].

Julia is a general purpose, high-level dynamic language used for numerical analysis and computational science[21]. It is an open source language developed in 2009 by Jeff Bezanson, Stefan Karpinski and Adel Edelman and was first released in 2012 [21]. Multithreading in Julia was introduced in version 1.3, which was released in November 2019 [9]. Since then, developers have been using multithreading in Julia to parallelize their code, to achieve efficiency and speed [9].

To help developers, it is imperative to understand and decode challenges faced by them while working on multithreaded code. This study focuses on evaluating the

discussions on multithreading by developers on Stack Overflow and Julia Discourse. This study involves qualitative content analysis of 211 posts extracted from Julia Discourse and Stack Overflow. By using inductive content analysis, this study reports challenges faced by developers in multithreading in Julia with the help of some statistical analysis as well.

This study focuses on helping developers to fine tune their code and Julia developers to improve their platform for multithreading developers. This can also help multithreading researchers, teachers, and development communities to better decide how to direct their efforts.

1.2 Thesis Organization

The remainder of this thesis is organized into following sections:

- Chapter 2 (Background) provides an overview of Julia, its architecture, multithreading, multithreading in Julia and content analysis. It also describes three approaches used in qualitative content analysis and also focuses on key differences between them.

- Chapter 3 (Methodology) focuses on the research methodology used to conduct this study.

- Chapter 4 (Qualitative content analysis) describes the results obtained from qualitative content analysis conducted on the data extracted.

- Chapter 5 (Conclusion) discusses implications, threats to validity and future work of the study.

Chapter 2

Background

2.1 Julia

Julia was designed to be an open source language as fast as C, as dynamic and convenient for statistical calculations as R, as usable for general programming as Python, as natural as Perl, and as powerful as Matlab. Julia is a general-purpose, managed, strongly and dynamically typed programming language with emphasis on high performance scientific computing [26]. It is a multi-paradigm language, containing features of imperative, functional and object-oriented programming. Julia has been under development since 2009 by Jeff Bezanson, Stefan Karpinski, Viral B.Shah and Adel Edelman and was first released in 2012. Julia takes an innovative approach to numerical computing which combines productivity and performance. It is an open-source project with contribution from over 500 developers and is licensed by MIT (Massachusetts Institute of Technology). Julia is faster than other scripting languages, providing development as rapid as R, Matlab and Python, with the fast code production of C and FORTRAN[9]. Julia is fast because of key design features. The core design feature called type-stability through specialization via multiple dispatch makes Julia easy to compile, making the code more efficient and concise. Type sta-

bility means that there is only one possible type that can be returned from a method. For example, the expected type to return from *(::Int64,::Int64) is Int64. No matter what is provided, it will return Int64. The * operator calls a different method depending on the types that it sees. When it notices Int64, it will return Int64. If a function is type stable, then the compiler can know what the type will be at all points in the function and provide the same smart optimization as C and FORTRAN. If the given function is not type stable, Julia has to add expensive "boxing" to ensure types are known before execution. Julia offers a number of advantages such as being an open source, fast and compact language that supports distributed programming, multithreading [2].

Julia provides optional typing, multiple dispatch and good performance, achieved using type inference and just-in-time compilation that has been achieved using LLVM (Lower level virtual machine). High-level numerical computing is easy and expressive as Julia includes in its lineage such mathematical and dynamic programming languages as Lisp, Perl, Lua and Ruby. Julia differs from all the other dynamic languages in terms of providing efficient code, performance and the fact that Julia is implemented in Julia itself .

Julia v1.8.2 was released on August 17, 2022, featuring new language capabilities, improved multithreading support, and updates to the compiler and runtime. [2]

2.2 Julia Architecture

Julia is a dynamic language with enhanced execution features [33]. It has inherited both the performance qualities of statically compiled languages and the advantages of high productivity languages such as Python and R. The key ingredients of the architecture of Julia are "rich type information provided naturally by multiple dispatch, aggressive code specialization against run time types and JIT (Just in Time)

compilation using the LLVM compiler framework"[23].

Some core Julia language components have played a vital role in giving Julia's architecture its special features [11] such as implementation of generic functions and dynamic multiple dispatch based on these types. Julia provides some symbolic and corresponding data structures for representing certain types, and implementation of lattice operators like meet, join, etc. Julia supports symbolic computing, which is defined as manipulation of mathematical expression as symbols rather than numeric values. Symbolic data structures allow you the creation and manipulation of mathematical expressions. It also contains a syntax layer that translates surface syntax to a suitable intermediate representation (IR) with a mechanism for binding top-level names. This makes it easy to write code using standard mathematical notation, while still allowing code to be executed efficiently.

Overall, Julia's architecture is designed to provide high performance while still being easy to use and flexible enough to accommodate a wide range of programming styles and use cases.

2.3 Julia Macros

Julia macros are a powerful and interesting feature of Julia programming.Macros are a generated code in the final body that can be reused. That generated code can perform a specific action which can be re-used in the code multiple times [2]. A macro maps the generated code to arguments which is compiled directly rather than runtime evaluation [2]. In short, it can change existing code or can generate entirely new code [2]. It is important for the user to understand the transformation that the macro is doing to the code, which often goes hand-in-hand with non-standard syntax. Macro arguments may include expressions, literal values and symbols [2]. Macros are represented using the at @ sign followed by the unique name declared in a macro.

Julia helps in building user-defined macros and also provides some useful built-in macros. Macros not only generate some pre-defined code but rather take the code they are applied to and transform it in the required useful way. Built-in macros are used for various purposes, such as @ threads, @ time, @ eval, @ inline etc. Julia supports parallel loops using @ threads macro. This macro is used with a "for" loop to indicate to Julia that the given for loop is in a multi-thread region [2]. @ time is used to execute an expression, print the time that it took to execute, the number of allocations and the total number of bytes its execution caused to be allocated, before returning the value of the expression. @ eval is used to evaluate an expression with values interpolated in it.

2.4 Multithreading

With the growing gap between processor and memory speed, memory latency has become a major bottleneck in achieving high processor utilization. Multithreading has emerged as one of the most promising techniques used to compensate for memory latency by utilizing thread-level parallelism and speed up the computation [9]. In programming, threads are defined as the smallest unit of processing within a process. Each thread contains information about how it relates to the overall program. Each thread has its own private stack and an instruction pointer which points to the next instruction to be executed. A thread's stack is used to store the location of a return address. This stack is used to store variables of fixed length which are in the local scope of an active function. Multithreading is defined as processing of various threads simultaneously to carry out a single task. The main reason behind using multithreading is to improve the performance of the code. As well as improving code performance, multithreading works best for efficient use of resources. It is less resource intensive as compared to management of processes. Even though it is

faster for an operating system to switch between threads for an active CPU task than switching between different processes, multithreading requires careful programming or various issues may arise. Issues like deadlocks, slow performance, race conditions and optimization are common while performing multithreading. There are various advantages of multithreading [1] such as improved throughput, useful utilization of resources, better performance, better communication, better optimized code, time saving and improved responsiveness.

However, its disadvantages include more difficulties with writing, testing, debugging and porting code, than are found in sequential programming. There are three steps that can be followed to achieve parallelization in the code. The first step involves the process of figuring out the decomposition of tasks that can be processed concurrently. The second step involves determining how many tasks can be appropriate for the parallel computer. The last step is to figure out how to map tasks to the particular thread [3].

Multithreading requires programmers to pay careful attention to prevent race conditions and deadlock.

2.5 Multithreading in Julia

Multithreading is a recent feature in Julia. From the very beginning, prior to even the release of version 0.1, Julia has had the task type providing symmetric coroutines which was used for event based I/O. The developers always had a unit of concurrency. Although parallelism was present in Julia from the very beginning, the multithreading version was released as experimental for some years. It was finally released as a stable version with the release of Julia version 1.3. Multithreading in Julia provides a powerful mechanism to create threads dynamically and a mechanism for loop parallelism. Being a recent feature, there are only limited studies related to multithreading

in Julia. In the case of loop parallelism, the mapping of parallel loop iterations to the threads is performed by Julia's internal scheduler, which makes the programmer unable to choose the scheduling of loops or the work within the loops [21]. So, it can be stated that scheduling of loops is entirely dependent upon the strategy of the internal scheduler's scheduling, which can further impact performance. In Julia, multithreading can be performed in two ways- using **@spawn** followed by wait and fetch or using **@threads**. The **@spawn** macro is used in creating a task and scheduling it to run on any available thread after it becomes available. The **wait** function is used to wait for the task to get finished and the **fetch** function to return its return value. The following illustrates an example of usage of **@spawn** to parallelize code:

Listing 2.1: Multithreaded code

```
function parallel_fibonacci(n::Int)
        if n < 2
                return 1
        end

        p1 = @spawn parallel_fibonacci(n-1)
        p2 = parallel_fibonacci(n-2)

        return fetch(p1) + p2
end
```

The Listing 2.1 function takes an integer n as input and returns the nth Fibonacci number using a parallelized algorithm. If n is less than 2, the function simply returns 1. It generates one parallel task with the current task using the @spawn macro, to compute the (n-1)th Fibonacci number and to compute the (n-2)th Fibonacci number. The fetch function is used to retrieve the results from the Future objects returned by the @spawn macro. Finally, the results are added together and returned.

The **@threads** macro executes a for loop in parallel. The execution of the loop

waits for the evaluation of all iterations. It runs the loop in an unspecified order and potentially concurrently. The following illustrates an incorrect example of usage of **@threads** in multithreaded code:

Listing 2.2: Multithreaded code

```
function parallel_fibonacci(n::Int)
        if n < 2
                return 1
        end

        a = 1
        b = 1

        @threads for i = 3:n
                c = a + b
                a = b
                b = c
        end

        return b
end
```

This listing 2.2 takes an integer n as input and returns the nth Fibonacci number using a parallelized algorithm with @threads. If n is less than 2, the function simply returns 1. Otherwise, it initializes a and b to 1, and uses @threads to parallelize the loop that computes the (n-2)th and (n-1)th Fibonacci numbers, and then sums them to compute the nth Fibonacci number.

It looks like a sequential code with usage of @threads and could give the same output as the sequential program. There are two main concerns to address: performance and correctness. It's important to note that race conditions can impact correctness, while memory allocation can affect performance. While writing the code, one could

have gone through a lot of issues such as race condition, slow performance, memory allocation and garbage collection. However, this implementation has a race condition because multiple threads may access and modify the same variables a and b concurrently, resulting in unexpected and incorrect results. To fix the race condition, you can use a Threads.Atomic object to ensure atomic access to the shared variables. A race condition occurs when two or more threads access the same shared resource simultaneously, leading to an unpredictable outcome. This typically occurs when one thread is trying to access the shared resource and the other thread is in the process of modifying it. In some cases, race conditions also occur due to non-atomic operations on shared resources. For example, one variable is being modified in various steps, one thread is reading its value, incrementing it, writing it back and another thread is trying to access in middle of the operation. Programmers can avoid race conditions by using various synchronization tools such as locks, semaphores, etc.

When we talk about slow performance of multithreaded code, we are referring to the reduced speed of execution when compared to its sequential or single-threaded counterpart. Memory allocation refers to the process of reserving a portion of memory to store variables, data structures, function call stacks, and other program-related information. In Julia, memory allocation is managed automatically by the garbage collector. When a new object is created, the garbage collector finds a suitable space in the heap and allocates memory for the object. If the object is no longer needed, the garbage collector will free up the memory. Memory allocation impacts the performance of multithreaded code by causing performance overhead.

2.6 Content analysis

In recent years, content analysis has been a growing area of qualitative research [18]. This approach is sometimes referred to as quantitative analysis of qualitative data [18]. Qualitative content analysis is used to analyze verbal, visual and textual data [24]. Qualitative content analysis is defined as a systematic method of classifying data to interpret the content of data. There are three approaches used in qualitative content analysis : conventional (inductive), directed (deductive) and summative content analysis. It is helpful as a hypothesis-generating approach, because the researchers gather rich information that can inspire future research.

2.6.1 Inductive content Analysis

Inductive content analysis is defined as a qualitative research method that researchers use to develop a theory or create a category by studying the data without using any preconceived categories. Researchers obtain new insights that emerge from the data. This type of method is usually appropriate when existing theory or research literature is limited. Inductive content analysis focuses on producing an overall summary of the content of different individual texts in data sets [35]. The key characteristic of inductive content analysis is that it is developed inductively by closely examining the text, rather than searching for predetermined data within the texts. One of the primary objectives of inductive content analysis is to establish a transparent connection between the research data and the summary derived from the data, ensuring clarity and coherence between them. The other purpose is to convert extensive and varied raw data into well summarized data. It helps the researcher to develop a theory about the underlying structure of experiences or processes which are evident in raw data. There are some underlying assumptions which have encouraged the use of inductive approaches [16]. Analysis is done using both the researcher's objective (deductive

approach) and multiple readings and evaluation of raw data (induced data). Thus the findings are derived from both the research objective and summary conceived from raw data. The initial mode of analysis is the development of categories from raw data into a theory which is in accordance with the main objective of research. To make the research more relevant, it is important for the researcher to decide whether the data are important or not. The researcher will likely produce findings that are not identical to previous findings and have non-overlapping components. There are various techniques that can be used to verify the findings obtained, such as replication of research and reevaluation of research to check the new findings with previous findings.

While conceiving categories from raw data, it is important to evaluate the trustworthiness of categories developed. Concepts like quality, authenticity and truthfulness of the findings together work towards the trustworthiness in inductive content analysis. Four primary criteria of assessing trustworthiness are credibility, authenticity, criticality and integrity. There are strategies like the researcher's responsiveness, methodological coherence, theoretical sampling, sampling adequacy, etc., that should be included in the qualitative research process if the researcher wants to be careful while conducting a research. Rigor is defined as being extremely thorough and careful. In content analysis, trustworthiness is the main parameter for checking whether the qualitative research is rigor or not.

2.6.2 Deductive content analysis

This method is used in cases where the researcher wishes to retest the existing data of a study with a new idea. This method can be used for testing concepts, categories, models or hypotheses. Existing theory or prior research exists about a particular topic which is incomplete. The goal of deductive content analysis is to validate or extend conceptually an existing research or theory. This may help in focusing and

refining research questions. It can help in providing predictions about variables of interest or about relationships among variables. Deductive content analysis is more structured than inductive content analysis. Deductive content analysis applies pre-determined/pre-conceived categories to the data [18]. By using these pre-determined theories or prior research, researchers begin by identifying the key concepts or variables as the initial coding categories [18]. The next step is to provide descriptions of the categories developed. The results from a directed content analysis provide supporting and non-supporting evidence for a theory [18]. These types of evidence can be verified by matching the codes with the respective description. The prior research can help in discussing the new findings. There may be a situation where the new findings may contradict the previous findings of prior research which may help in refining or extending the research [16]. This is a strength of deduced content analysis. Some advantages of using deductive content analysis include helping to sort the data into various organizational categories such as data type, participant or time period, etc., organizing data into categories to maintain alignment with research questions and allowing the application of theoretical/conceptual frameworks.

2.6.3 Summative content analysis

Summative content analysis starts with identifying and quantifying certain words or some content in the text with the aim of understanding the content of those words in the theory. This type of content analysis is done to explore the usage of those words but not infer their meaning. It is also known as manifest content analysis as it analyzes the appearance of some specific words in textual material. In contrast, quantitative content analysis focuses on counting the occurrences of words. The summative content analysis approach offers several benefits. It is a nuanced and non-reactive method for studying a phenomenon, employing a straightforward approach. It provides straightforward insights by examining the usage of simple words, con-

tributing to improved understanding. But there is a disadvantage to this approach as it is limited to the extensive meanings within the data, since the researchers try to explore the word usage or the range of these words in their normal usage [18].

2.6.4 Key difference among these three approaches

The primary distinction among the conventional (inductive), directed (deductive), and summative approaches lies in how the categories are developed. In the case of the inductive (conventional) approach, categories are formed by closely reading the text and deriving them directly from the data. This means that the analysis begins with observation, without any preconceived categories. On the other hand, the deductive (directed) approach involves developing categories based on preconceived findings derived from prior research or theory. In summative content analysis, the categories are developed both before and during the data analysis process. Rather than analyzing the data as a whole, the text is analyzed word by word or in relation to particular content [18].

2.6.5 Difference between Grounded Theory and Qualitative content analysis

Grounded theory is defined as an inductive methodology which is a set of rigorous research, procedures leading to systematic generation of theory from systematic research whereas qualitative content analysis is defined as the method of classifying data(textual, visual, etc.) into categories with some meaning. Although both appear similar, they differ as well. The research goal of grounded theory is to generate theory through systematic research whereas developing categories from research is the research goal of qualitative content analysis. They both differ in terms of the research outcomes. Grounded theory leads to substantive theory whereas qualitative content

analysis leads to lists of categories acquired from the research [14].

2.6.6 What did we choose for this study?

This study was conducted using the inductive content analysis approach. One of the reasons to choose this study was absence of any previous research in the same area. The other reason behind choosing this approach is that it allows us to have an in-depth understanding of the challenges faced, due to the absence of any previous studies. The data set chosen was well evaluated and refined by this approach. This approach didn't require any cost since the data was collected from free open-source and was analyzed by a human.

2.7 Research on Concurrency

This is a complex topic and there are various areas of research in concurrency including teaching concurrency, detecting bugs in concurrency, designing concurrent systems and understanding and overcoming parallelism bottlenecks and parallel program comprehension. In this section, we discuss what is being taught in concurrency, the way students learn concurrency, parallel program comprehension. At the end, we discuss other qualitative studies that have been analyzed and surveyed during this study.

2.7.1 Teaching Concurrency

Teaching concurrency is an iterative process [13]. It is an essential part of a computer scientist's education where each set of features presents a model for thinking concurrently and a path for learning how to design concurrent algorithms [37]. When instructing students about concurrency, it is important to provide comprehensive education on various subjects, including data decomposition, synchronization techniques,

and thread management. Various researchers have studied how concurrency should be taught and different techniques to teach concurrency, worked on designing systems to teach concurrency and described of the process of learning concurrency. Introductory topics such as mutual exclusion, absence of deadlocks, synchronization techniques and communication mechanisms are challenging for all the teaching strategies. M.Carro et.al have developed a methodology that divides teaching concurrency into six stages - Process identification, identification of interprocess interaction, defining the control flow of processes, process interaction definition, refinement of process interaction and analysis of the solution's properties [13].

Aceto, Ingolfsdotter, Larsen and Srba have emphasized the correctness problem, which is the problem of ascertaining whether the given computer program meets the specification derived [5]. It is a fundamental problem in computer science. They have worked on providing courses to learn concurrency by focusing on the correctness problem and algorithm verification. Leslie Lamport has talked about how concurrency should be taught to students [25]. He has highlighted the common misconception and difficulties that students face when learning about concurrency. It is well understood that reasoning about concurrent systems require a shift in thinking as compared to sequential programming. Students often struggle to understand and reason about non-deterministic behavior, race conditions and other concurrent phenomena. While teaching concurrency, it is important to focus on the concept of correctness, safety and liveliness, how to specify and verify properties of concurrent programs.

2.7.2 Learning Concurrency

Concurrency is a difficult subject matter to learn because of the non-determinism of execution of a concurrent program conducted by the multiple threads of control. Students must demonstrate an understanding of the various approaches to and implications of concurrency through implicit structures in programming languages or

explicit structures in design [33]. Mental models play an important role in constructivism. Constructivism claims that previously acquired knowledge is constructed by the learner rather than being transferred from teacher to learner [22]. This theory states that the learner constructs a mental model while learning anything, whether a subject or a skill. While implementing concurrent programming, students try to use the same mental model as sequential programming. A promising path to improved concurrency education is studying errors made by developers and students [33]. It has been found that students are generally able to identify concurrency issues, but correctly solving those issues is not as common as all aspects of problem need to be known. Various studies have pointed this out, noting that sequential programming could be solved by trial and error, but concurrency requires a more formal approach and deeper understanding of concepts to solve concurrency related problems [34]. Learning concurrency involves understanding problem decomposition, synchronization and object oriented bugs [29]. Various studies have focused on how the course should be designed for students to make learning concurrency easy and effective.

2.7.3 Parallel Program Comprehension

Program comprehension is defined as the study of how programmers understand a program. It is the ability to understand the structure, behavior and connection to the application domain [12]. It is relevant to many tasks in programming. For example, when working on a design, developers need to read and process what they have thought and what they have implemented. There are other tasks such as testing, maintenance and modification that involve a good understanding of program comprehension. There has hardly been any research on parallel programming comprehension. Existing theoretical models of program comprehension include knowledge stored in long-term memory (syntax of languages and semantics, programming schemas) and the development of mental models of programs in working memory

[8]. Different types of mental models are formed while comprehending a program. The propositional model is referred to as the program model and the meaning is represented by the situation model. The program model represents the surface of the program, i.e., the syntax of the program, whereas the situation model represents the program's data flow and goals. These two models have been successfully used for comprehension of procedural and object-oriented programs [8]. Comprehending parallel code also requires thorough understanding of the data access pattern of the threads, i.e., data flow of parallel code. A point to be noted that these previous models didn't consider data structures and there were no previous models for parallel programming. Aubanel(2020) has focused on adding execution model that focuses on data structures as well as data flow in the program and computer [8]. The execution model can be divided into three layers of abstraction in which at the top, it represents the behavior of the operations on the data structures. In the middle, the data flow between variables in the program text. At the bottom, the data flow in the processing elements of the computer [8].

2.7.4 Bottlenecks of parallel programming

Since parallel programming is hard to learn and tech, it also has various bugs that are hard to detect and understand. A novice programmer needs to concentrate on exploring parallel programming in depth to understand the process of detecting bugs in multithreaded code. To create a corps of well-trained parallel programmers who know the concepts to transfer sequential/serial code to optimized parallel code, there must be an understanding of what makes a person an expert. There must be proper progression from novice to an expert status. Although parallel programming is difficult, programmers can still master it by using the right approach.

There has been research on identifying the common issues faced by novice programmers. Programmers, especially students, are the center of research in many cases

because firstly they are inexperienced, which means they need more help as they face more difficulties. Secondly, helping them understand their mistakes helps them to learn and get their programs to work[32]. Ben Ari and Kolikant (1999) describe how students face issues while performing operations permitted by the concurrency model according to their assumptions but not following the formal rules [22]. The students tend to use the same development approaches for concurrent programming that they use for sequential programming. Approaching a problem from the wrong perspective may lead to erroneous conclusions. The greatest challenge in writing concurrent programs is getting concurrent threads to reliably interact properly in the face of this non-determinism.

Even though there are tools such as OpenAcc and AMD HSA that make it easier to design and implement efficient and correct parallel programs, programmers still need to know the details about parallel computing architecture and platforms [17]. These implicit parallel programming models aim to abstract away some of the details of parallel programming, but to different extents and in different ways. While these abstractions can simplify the programming process, optimizing performance for specific hardware platforms may still require some understanding of the underlying hardware and programming techniques. Learning the basic concepts about parallel programming is, and will continue to be, required in order to write correct and high-performance code. Programmers who are unaware of fundamental concepts often write parallel code that is slower than the sequential versions. It is also easy to write code that produces incorrect answers under specific conditions, which are hard to detect and correct [17].

Correctness and performance are two issues that students tend to confuse (i.e they fail to distinguish properly between them). Kolikant explains that students define a "correct program" as a program that exhibits "reasonable output for many legal inputs". They interpret receiving the desired output as proof of the correctness of

their program. This must be correct in the case of sequential but not in the case of concurrent programs. While concurrent programs can also yield correct output even while having errors or race conditions, it can behave incorrectly under certain conditions. Concurrency involves synchronization, and the developer must know the synchronization goal and its pattern of execution in the code [22]. Verification is a method that can be used to make sure a program is correct according to correctness measures. The common way to do verification is to test the program. Finding test cases for a sequential program can be difficult but exposing concurrency defects is harder.

It has been noticed that students are unable to focus on the division of a problem; i.e., students feel it is difficult to structure sub-problems to allow solving them concurrently [22]. Moreover, research has found that students are unaware of how concurrency is achieved in a particular system. Some students use a mental model of concurrency with too strong guarantees compared to what is typically provided by language and hardware. Students have difficulty understanding the co-operative aspect of concurrency as well; i.e., they find it difficult to focus on the complexity associated with running multiple, largely unrelated threads simultaneously and how using concurrency affects the working of such programs. Also, it is difficult for students to work on the communication aspect of concurrency as they are unable to eliminate issues present due to communication between threads.

2.7.5 Qualitative analysis of discussion groups

In this section, we discuss a selection of notable and complementary research that addresses the same methodology used in this study. Ahmed and Bagherzadh(2018) conducted a large scale study to learn about the interests and difficulties faced by concurrency developers using Stack Overflow [6]. Their method involved developing a set of concurrency tags to extract concurrency questions they ask. They used LDA

(Latent Dirichlet Allocation) topic modeling and open card sort to manually determine the topics. They created a topic hierarchy by repeatedly grouping similar topics into categories and lower level categories into higher level categories. They measured the popularity and difficulty of concurrency topics and analyzed their correlation. The findings of their study were that developers were concerned with topics ranging from multithreading to parallel computing, mobile concurrency to web concurrency and memory consistency to speedup. The questions were grouped into categories such as concurrency models, programming paradigms, correctness, debugging, basic concepts, persistence, performance and GUI. Developers are more concerned about the correctness of their program rather than focusing on the efficiency of the code. The most popular topics about which most of the questions are asked were thread safety and database management. Difficulty and popularity of concurrency are negatively correlated to each other.

A similar study was conducted to understand what mobile developers ask using Stack overflow. Rosen and Shihab employed the same model called Latent Drichlet allocation (LDA)-based topic models to summarize the mobile related questions [**?**]. They found that questions related to app distribution, user interface, and input were amongst the popular of all of them. They have downloaded the data set from stack overflow and identified mobile related posts by composing a set of keywords by considering mobile platforms, mobile hardware, and mobile development environments. The posts were aggregated using LDA based models to discover what is being asked in the mobile Stack Overflow posts.

Another study was conducted to study the discussions done by IOT developers on Stack Overflow and also used LDA to generate the topics using MALLET library [36]. They found that questions are primarily asked in four different categories- Software, Network, Hardware and Tutorial. This study was performed in three steps- Obtaining a latest SO dataset, identifying IoT Tagset in the dataset and identifying IoT-related

posts from the dataset using LDA.

Chapter 3

Methodology

3.1 Introduction

The materials for this study were extracted from online discussion forums. Two platforms, Stack Overflow and Julia Discourse, were examined in this study. With over 14 billion users and 21 million questions and 31 million answers, Stack Overflow has globally replaced programming books for day-to-day reference and plays an important role in computer programming [4]. It was launched on September 15, 2008 by Jeff Atwood and Joel Spolsky. It accepts questions related to programming, covering everything related to computers.

Julia Discourse is one of the platforms used by the Julia community to ask questions and get help when struggling with Julia. From requesting a new feature to contributing towards a new feature, it covers all the help needed for Julia. Both platforms are repositories for developers to post questions, receive answers and learn about a range of topics. Julia Discourse was released at the end of 2016 to replace the Julia mailing lists, the largest of which were Julia-users and Julia-dev [20]. Julia Discourse helps developers keep up-to date on important Julia announcements like language releases and the JuliaCon conference. The community poses questions and responds

to the conversations related to any aspect of using or developing Julia or its associated packages and tools. Also, the community provides feedback on the new packages announced and can also advertise Julia-related jobs and events.

To report these findings, this research has been conducted on the textual content of Stack Overflow and Julia Discourse to answer these research questions:

RQ1: What questions do developers ask about multithreading?

RQ2: What categories do these questions belong to?

RQ3: What specific challenges do developers encounter when programming in a multithreaded environment specifically in Julia?

To answer these questions, this study involves five steps:

1. Reading posts related to multithreading in Julia Discourse and Stack Overflow.
2. Extracting the relevant questions manually from the acquired dataset.
3. Constructing the categories from the relevant dataset required.
4. Determining the challenges and difficulties faced by developer.
5. Looking for a hierarchy from the issues determined .

3.2 Research Design

Qualitative research deals with human understanding of different phenomena related to a perceived situation. It is an iterative process that continuously works on reducing the data into concepts. Qualitative content analysis focuses on the themes and topics of the categories developed from the data and inter-relationship between them within the data under analysis. Although content analysis has been criticized for its subjectivity, still it is one of the most used methods for investigating discussions [28]. It is suitable for various kinds of data such as interview and observational studies. There is no defined way to conduct qualitative content analysis but one can form the

structure by clearly understanding the objective of the study. It may seem mysterious until the data starts making sense to the researcher. This research study involves steps such as planning, data collection, data analysis and presentation of results.

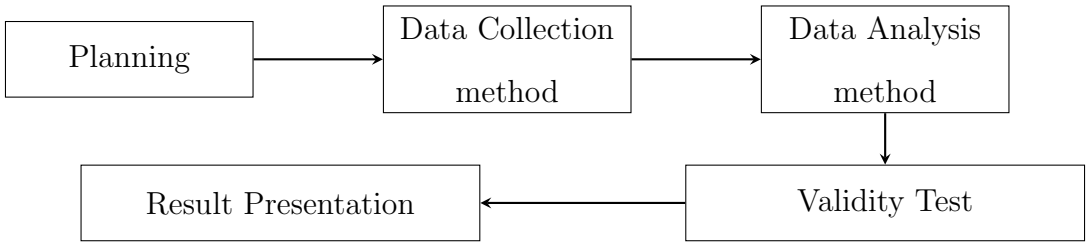

Figure 3.1: Steps involved in Qualitative content analysis

3.3 Planning

In qualitative research, it is important to become familiar with the way the study should be conducted to illuminate what the researcher wants to find out, how and from whom. Planning helps the researcher to save time and avoid the problem of ambiguity in future. The researcher must know whether the research should be inductive or deductive. To assist in the planning process, one must work on these five aspects: clarification of the aim of the study, sample and unit of analysis, method of data collection,method of data analysis and practical implications [10]. The design must be established prior to launching the study. This study follows these five aspects of planning (Figure 3.2):

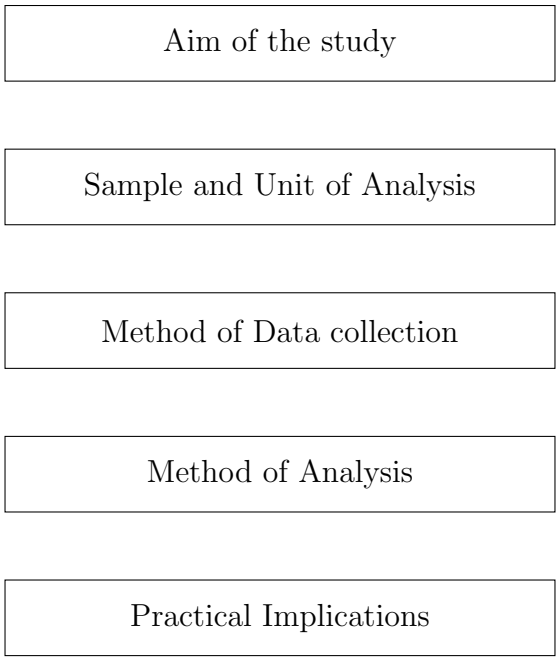

Figure 3.2: Five aspects of Planning

3.3.1 Aim of the study

In qualitative research it is important to set the aim of the study and establish the structure of the study through it. Going through an entirely new concept, then making an assumption from it is a difficult task. The aim can ease it and can help in defining the steps needed to obtain the aim. This would also help the researcher to formulate the research questions and focus on them to obtain findings. If the aim of the study is too broad, the study can be difficult to complete. This can be avoided by setting the boundaries of the study so that the study does not yield an inconclusive result [10]. The aim of this study is to measure the challenges faced by developers while working with multi-threaded code in Julia. The aim of this study helped in developing three research questions which were answered through the qualitative research performed.

3.3.2 Sample and unit of Analysis

In qualitative content analysis, the researcher must determine the sample size of data required to answer the research questions. The researcher can answer these questions with full confidence on the basis of sample size of data collected and analyzed. The units of analysis are decided by the researcher to determine whether the data to be analyzed should be divided into smaller units or should be considered as a whole unit. There are no established criteria for the size of unit of analysis. In this study, 211 posts were collected from Julia Discourse and Stack Overflow through the data collection method described below. Also, 25 posts were rejected due to various reasons explained below.

3.3.3 Method of data collection

Data collection in qualitative research plays a vital role as it involves gathering information through interviews, observations, focus groups, etc. The data collection is non-numerical, subjective and contextual, and is used to understand the meaning, experience and perspective of the research participants. In this study, the posts involved discussion between developers on topics related to multithreading. The main aim of data collection was to collect posts that are concerned with challenges faced while writing multithreaded code. The first major step was to follow the main keyword "multithreading", then further keywords were scooped from it. The method of data collection is briefly explained in the next step.

3.3.4 Method of Analysis

This study uses the inductive approach to analyze data with the purpose of obtaining categories flowing from the posts. With chances of ambiguity and overlapping, this approach can be difficult to work with. To avoid this, it is important to make sure that

the description of categories obtained is properly documented and well understood.

3.3.5 Practical Implications

While working with human subjects or human participants, it is required to consider ethical aspects to carry out a study involving human participation. To preserve their confidentiality, they must be informed about their rights as volunteers in the study. They can withdraw their data from the study any time they want. When the design is established, the blueprint of the study is sent to the ethics committee with the purpose of achieving smooth working between informants and the researcher. Since this study does not have any direct involvement of humans, it does not require any kind of approval from an ethics committee.

3.4 Data Collection process

There are various ways of collecting data such as verbal interviews, document study, surveys and questionnaires. To obtain hypotheses from scratch, it is very important to get familiar with data collected for qualitative research. In this study, questions asked by authors in the form of posts are considered as data. The data collection process for this qualitative research involves collection of posts from Stack Overflow and Julia Discourse. The posts that are tagged under the tag "Multithreading" are taken into consideration.

3.4.1 Scan Dataset related to multithreading

In the first step of our analysis, we searched for the required datasets from Stack Overflow and Julia Discourse. To search for the required posts, several search keywords were used. To carry out this search, different search strategies were used:

Table 3.1: Keywords used to scan dataset

Keywords used in Julia Discourse	**Keywords used in Stack Overflow**
Multithreading	Threads Julia
Race condition **AND** multithreading	multithreading Julia
Deadlocks **OR** multithreading	Deadlocks in Julia
Performance in multithreading	Kill threads in Julia
Performance, multithreading	Atomic in julia
Threading	Processes in julia
Threads	Macros performance threads julia
Parallelization	Speed multithreading Julia
Optimization in multithreading	performance multithreading Julia
Thread safety in multithreading	
Multithreading performance	
Multithreading Performance in Julia	
Julia threads	
Parallelization	
Macros performance	
Slow code	
Memory allocation multithreads	
Macros performance multithreading	
Threads performance	
Multithreading and performance	
Memory allocation and multithreading	
Multithreading Julia performance	
Performance threads	
Parallel,julia	
Race condition **AND** threads	
Threading	
speed multithreading	
performance threads	

- **Keyword Searching-** Keyword Searching is an effective method for finding information in a system. Keyword search includes choosing the most important words that relate to the topic one is working on. It is a broad and flexible search which is useful to know what's out there. One advantage of using this technique is that even if the way one phrases a concept isn't the subject term, there is still a good chance of finding what one is looking for. Selecting a keyword is a multi-step process that involves identifying the main concepts of the topic, brainstorming synonyms and antonyms that could also be used to describe the topic and spell out abbreviations.

- **Using Boolean logic to combine terms-** There are three basic Boolean

search commands: AND, OR and NOT. Rather than just using a single word to search, one can group the words together. AND search will find all items that contain both terms in the string. For example- search keywords like *multithreading and performance* would yield all the posts that involve both the keywords. OR searches find one term or the other. For example, *Multithreading* or *threads* would return all the posts that contain either of the specified terms. For example, *multithreading* NOT *distributed* will return the posts that contain the keyword multithreading and not the keyword distributed.

- **Using adjacency searching for more accurate results -** An adjacency search allows one to search for two keywords or phrases that appear within a set number of words of each other. This search is less precise than exact phrase search but is simpler than an AND search. For example, posts related to multithreading and speed could be searched using keywords speed multithreading so there could be chances that the keyword **speed** and **multithreading** could appear in the same sentence. But sometimes adjacency is also used to mean that the words must appear in the same order the way they are typed in the search, with the condition that they must be within a specified number of words apart from each other. There is also the chance that there are no words between the terms.

- **Searching for exact phrases-** An exact phrase search retrieves only those items in which the exact phrase appears. One of the precise ways of searching for exact phrases is done by putting the phrase in between quotation marks. Sometimes it happens that it is unclear whether quotation marks are needed and sometimes it limits the search too much. It is useful to conduct the search with quotation marks.

- **Snowball Searching-** The snowball searching method is a technique used to

track down related posts by using references from one post to another. Searching for relevant posts with appropriate keywords to your research plays a vital role. The advantage of the snowball searching method is that one can find a lot of keywords quickly and relatively easily.

Table 3.1 shows all the keywords used to search for the relevant posts. Different keywords were used for both the platforms because Stack Overflow is a larger platform that accepts question for all the languages and contexts related to computer science. The fact that there would be less questions related to Julia on Stack Overflow as compared to Julia Discourse. We found that these limited keywords yielded enough posts from Stack Overflow.

3.4.2 Extract relevant questions manually

In the second step of extracting the relevant questions manually, it is important to decide whether the given post belongs to the criterion of your selection or not. For this study, the first selection criteria states that the given post is tagged under the main tag "multithreading" or at least one relevant multithreading tag. The keyword developed in the previous step helps in developing more questions and it is possible that some of the posts may not contain the main tag "multithreading". All the other posts which are not related to multithreading or are concerned with some libraries are added to the rejection list. From this, we got 240 posts for our analysis. In order to know what possible questions developers can ask, it was important to know what basic challenges one could face while working with multithreaded code such as slow performance, race condition, etc. After searching for the keyword "multithreading", other criteria may be applied, such as including only posts that do not involve any other libraries, packages, or external tools.

3.5 Time Horizon

All the posts have been gathered from the year 2014 until the version 1.8.0 release date August17, 2022.

3.6 Data Analysis Process

After the data collection process, it is essential to focus on the analysis phase. The researcher must have a clear understanding of the research question and the objectives of the study in order to effectively analyze the data. The process of analyzing data can be challenging, but with careful planning and attention to detail, it can yield valuable insights. It is also important to keep in mind that data analysis is an iterative process and that the researcher may need to revisit and re-evaluate the data multiple times in order to gain a deeper understanding of the results. This process may include reviewing literature, testing different hypotheses, or conducting further analyses. Ultimately, the researcher should aim to present clear and concise conclusions that are supported by the data. This may involve creating tables, graphs, and charts to clearly communicate the findings and make it easy for the audience to understand the results.

In summary, the data analysis process is a crucial step in any research project, and requires careful planning, attention to detail, and a thorough understanding of the data and the research question. With these steps, the researcher can perform a useful analysis. In order to gain a comprehensive understanding of the challenges faced by developers when writing multithreaded code in Julia, this study employs a detailed procedure of qualitative content analysis. Despite the various approaches presented in literature reviews on qualitative content analysis, there are commonalities in the way the process is conducted. The chosen approach for data interpretation does not affect the outcome of the analysis; however, there are two approaches used

for interpreting the analysis: latent and manifest analysis. Latent analysis involves interpreting the data by delving deeper into the text to uncover the implicit meaning of the participants' experiences, while manifest analysis focuses on interpreting the text at face value and remaining close to the original data. This study uses the latent analysis as method to interpret data used in the research. As this research is based on human-generated data, there is a potential for error, such as fatigue, misinterpretation, and personal bias. Therefore, it is crucial for the researcher to ensure the reliability and validity of the study through rigorous data analysis and quality control measures.

- **Get to know your data**-To yield valuable insights, getting to know your data is a very important step in the data analysis process. The researcher must consider the quality of data and should proceed accordingly. This research involved various steps to understand the data, including reviewing the data collection process. This study involved posts from two different platforms, to understand the context of the posts. To attain that, it was important to conduct a thorough reading of the posts and highlight some specified keywords that exhibit the topic of question asked. Reading the posts included the question asked by the author, the resulting discussion done by other participants and the time that post was posted. Highlighting keywords gives an idea about what the author is expecting and how the question has been written. It was crucial to understand how the data were collected, what type of data were collected, and any limitations or biases that may have occurred during the data collection process. The next step was to explore the data, to explore the data's overall structure and patterns and determine approaches to their visualization. To synthesize insights from this research, bar graphs, pie charts and histograms were used. It was also made sure that the data do not contain any missing elements such as incorrectly omitted posts. It was important to understand

the context in which the data was collected and how it related to the research question.

- **Focus the analysis**- Focusing the analysis means reviewing the purpose of research and understanding what you want to find out. The methodology is appropriate if it can answer the questions well. This study contains three well developed research questions that play an important part in this process of focusing. We ensured that the research methodology we followed effectively addressed parts of the research question. For example, while working on our data collection process, one of the steps involved scanning the required datasets, which was all dependent upon the first research question. The first research question was all about the questions asked about multithreading, so the step of scanning the dataset answers this question to some extent. The analysis in this study was designed to evaluate the questions asked in the posts, with a focus on the time constraints, the method of questioning, and the expertise of the participants. The objective of the analysis was to determine the questions asked before the release of the latest version. The posts were collected within a time constraint to achieve this aim. The questions asked were analyzed to provide valuable insights into the study. The participants were grouped based on their level of expertise, namely novice, intermediate, and expert. The findings showed that the questions asked and answered differed based on the level of expertise, providing further insight into the study.

- **Construction of categories**-The third step in the data analysis involves construction of categories. In this crucial step, the categories are extracted from the data. This step contains further substeps such as pretesting the categories definition and rules and assessing the reliability and validity. The coding system should define the critical features of the specific categories and tags which

should properly distinguish between the differences and similarities between them. Some sub-categories should have some sub-categories to have a hierarchy between them. To avoid the problem of ambiguity, it is important to include features that are exhaustive: having a particular tag defining an attribute that does not include the possibility of another feature of a different tag. This can help in deepening one's understanding of the content under investigation through meaningful insights developed.

In pretesting the category definitions and rules step, a small sample of categories was carefully pretested against the description of categories (tags) already created to determine whether the classification is consistent or not. This was done to make sure that the already classified data and corresponding categories (tags) match each other. There could be chances of ambiguity in this case as the same post could go by two different categories as well. The outcome of this step could be both positive and negative. It could be positive if the descriptions of categories fit the posts extracted. The other possibility could be that some of the posts may fall outside the pre-established categories.

The categories were assessed to verify their reliability and validity. Reliability is defined as accuracy of categories (tags) established in relation to the extracted questions. It is the degree to which the result of the classification is accurate. It is important to gather information without losing the reliability of data. To access the reliability, we apply the same method to the same set of data under the same condition to obtain the same result. If we do not obtain the same result, that means the method of measurement may be unreliable or biased, which may invalidate the research. There are four types of reliability tests that can be performed: test-retest, interrater, parallel forms and internal consistency. We used test-retest reliability to measure the consistency of tags over the same set of data to verify whether the tags designated do not change with time.

- **Determine the issues faced**-In this step, all the issues faced by the programmers are determined using questions asked by them on Julia Discourse and Stack Overflow. In order to determine that, the data collected from the posts are saved in an excel sheet. The Excel sheet saves all the data such as the tag name provided using the steps above, cause behind the issue, date the post was found, date the post was posted on the website, URL of the post, title of the post, context of the post and whether the query in the post was solved or unsolved. The data analysis done provides the issues faced so far.

3.7 Validity Test

To evaluate a qualitative study, it is important to check the validity of its findings. Validity in qualitative research refers to the extent to which the findings of the study are credible and trustworthy. There are various ways of conducting a validity test for qualitative research such as the triangulation method, member checking and maintaining a clean audit trail [15]. Triangulation involves using multiple data sources, researchers, or methods to study the same phenomenon, which ensures the validity of the findings of the study. The aim of this method is to cross check the findings and compare the results obtained from different sources, methods or researchers. Member checking involves sharing the research findings with the participants to ensure that their perspective and experience have been accurately represented. Maintaining a clean audit trail refers to the process of recording every single task performed, including all the decisions made, changes in the research design and any unexpected findings or challenges. In this study, we have used the triangulation method to validate the findings of the study by cross checking the findings obtained by a secondary researcher. In this method, my supervisor Dr. Eric Aubanel was involved as secondary researcher. He was provided with correct descriptions of tags developed and

20 posts to compare the data. Later, the tags assigned by me were compared with the tags assigned by Dr. Aubanel.

3.8 Saturation

Saturation is defined as the criteria for assessing a point to discontinue data collection/and or data analysis [30]. There is no proper answer to how saturation can be achieved [31]. There are four different forms (theoretical, inductive, a priori thematic and data saturation) of saturation that relate to the research process in different forms [30] [31]. Theoretical saturation relates to grounded theory methodology where new theories emerges from scratch [30] [31]. Inductive saturation relates to inductive analysis where new code emerges [30] [31]. A priori thematic saturation refers to the extent to which the identified codes or themes are demonstrated in the data [30] [31]. Data saturation relates to which newly collected data replicate or reinforce what was previously expressed in the data [30] [31]. Researcher's response to the theories obtained from the study is a point to be considered while deciding a point of saturation. Saturation feature helps to have a clarity about the research objective. In case of the inductive approach, the sampling size relates to theory generation which might differ from how the objective has been achieved. In this study, saturation was obtained when after searching with all the keywords yielded the same posts again and again. All these posts have already been considered for the analysis.

Chapter 4

Results

This study uses the latent content analysis approach to interpret results achieved from the analysis process. Before presenting the results, it is crucial to understand the data interpretation strategy to be used to present results. The data interpretation strategy used in this study involves three key points : using themes and connections to explain findings, interpreting the data with its significant meaning to the analysis and developing a list of key points or important findings. Using these three methods, the results of this study were represented using five key themes. These themes were collectively named based on the key points they represented from the sample of 211 posts retrieved, out of which 138 posts were abstracted from Julia Discourse and the remaining 73 from Stack Overflow. Figure 4.1 shows issues faced in multithreading. As can be seen from the graph, the majority of the posts were concerned with queries about various topics such as threads, macros, etc., tagged under the tag "Need-to-know". However, developers struggling with slow speed of the multithreaded code takes the second spot after queries. Out of 211 posts, 21 posts were concerned with how to parallelize the code, which points towards the fact that developers have difficulty writing parallel code and are eager to ask for help from other developers. With this we found that even if they write parallelized code, 18 posts

were concerned with a way to achieve good performance from the multithreaded code. We found that they feel it is difficult to understand the memory allocation pattern; this was quite evident from 15 posts obtained. There were other issues as well, such as high CPU allocation, garbage collection, exception, type inference, system crash, error, type instability, race condition and what If I do this. Table 4.1 explains all the tags perceived(with descriptions) from the study which represents challenges faced by developers in multithreading in Julia. Table 4.2 represents descriptions of categories for all the need-to-know categories formed.

4.1 Theme I- Where should I tag this?

This theme represents confusion about tagging the questions under the right tag, which is common in developers. Developers working on threads often tend to perceive that anything related to threads is multithreading. This was seen in the rejected lists of posts which represented all the posts that were tagged under multithreading but didn't contain anything related to multithreading. These posts were mostly related to other programming languages or other programming models. Some of the posts included in the rejected list were tagged under Julia but not related to it. Developers often mistake multithreading with other programming models such as distributed programming, meta-programming, etc. Another reason behind wrong tagging was unclear vision of their code. Because they were familiar with the concept of multithreading that they were trying to use in their code, they attributed the issue to it, despite not knowing which specific part of their code was causing the problem. Some of the posts were concerned with the external library used, with some other programming language, the system design and some unofficial bug not being reported, etc. It is important to note that writing the question under the wrong tag can result in the question being directed to the wrong audience, which can confuse the

Table 4.1: Descriptions of Tags extracted

Name of the tag	**Description of the tag**
Slow multithreading	This tag refers to all the posts in which the authors are reporting slow speed or performance of multithreaded code (parallel code) as compared to sequential code (single threaded).
Need-to-know	This tag is all about developers inquiring about concepts in multithreading like asking about difference between macros, initialization of threads, etc.
How to speed up this code	In this tag, developers explicitly ask about a way to speed up this code.
How to parallelize this code	This tag refers to all the posts where developers want to know the way to parallelize the sequential code they provided. In some cases, they want the discussion forum to parallelize their code.
Novice coding	This tag refers to all the posts where the developer has reported being new to Julia, being new to coding, novice in multithreading, etc.
Incorrect output	This tag refers to all the posts where the developer has reported incorrect output from their code i.e. they didn't get what was expected from the code.
Race condition	This tag refers to all the posts where the code suffers race condition, but the developer doesn't know the reason behind it.
Error	This tag refers to all the posts where the code throws an error at runtime such as bounds error, nested task error, etc.
Exception	This tag refers to all the posts where an exception was reported at run time. Exceptions such as – TaskFailed, Task Interrupt, Deadlock error, concurrency violation, etc.
Memory allocation	This tag refers to all the posts where the developers have reported a lot of memory allocation in the code.
Type inference issue	This tag refers to all posts where the code suffers type inference issue which means the process of inducing output values from the type of input values.
Type instability	This tag refers to all the posts where the code reports type instability I.e., the compiler is unable to infer the type of variable.
Macro mistaken	This tag refers to all the posts where the developers are mistaken about the usage of macros which means they are using macros in a wrong way as intended.
Garbage collection	This tag refers to all the posts in which code suffers from garbage collection issue. Garbage collection refers to the process of how the allocated memory is recovered.
System crash	This tag refers to all the posts in which the developers have reported system not working properly after running the code i.e., system crashed.
High CPU utilization	code reported high CPU utilization.
What If I do this?	This tag is a hypothetical tag where the author has not actually written any code but wants to know the opinion of discussion forum about what would happen if they do something.

participant and the discussion forum as well. It can lengthen the process of getting proper guidance from the forum.

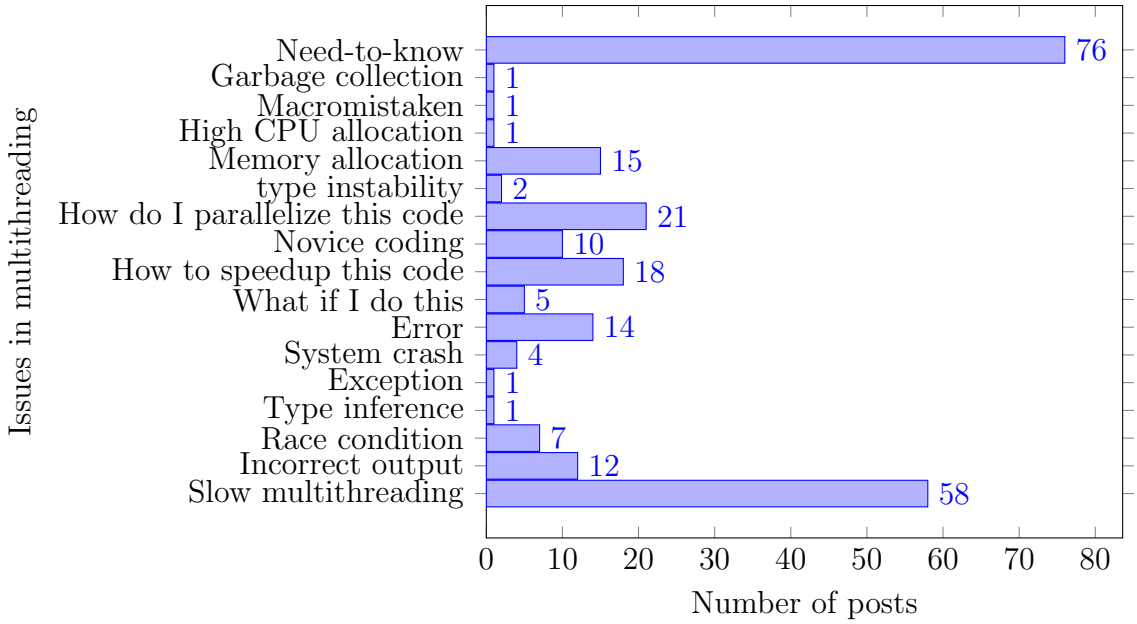

Figure 4.1: Issues faced in multithreading

4.2 Theme II- How do I do this?

Developers, especially novice programmers, write code according to their existing knowledge. We found that developers ask a lot of questions related to the performance ,i.e., speed and efficiency of the multithreaded code. This was evident from the statistical data (Figure 4.1) obtained from the study. In 60 out of 212 posts, developers reported facing issues with speed of the code ("slow multithreading") as compared to the the sequential code. There were various causes such as memory allocation, garbage collection, bugs, false sharing and type instability, etc. The major cause behind slow speed was memory allocations happening in the code. Developers tend to write the code using prior knowledge that leads to various issues such as type inference, race condition, etc. This can be avoided by going through the official documentation of the language supporting the technique. But with Julia, even with the presence of official documentation, developers tend to inquire about the concept with a minimal working example to have proper understanding of the concept.

> "I read documentation and many topics in the discourse still I'm not sure I'm doing it right."
>
> "I am sure the elegant answer is in the documentation, but I could not find it."

The official documentation of Julia explains multithreading and its concepts in detail and focuses on providing some performance tips that can help developers to write efficient code. Throughout this study, it was found that developers do not take this information under consideration while writing code. In one of the posts, one of the answers stated:

> "So the important morals of the story: Read Julia's performance tips, the performance of your code, and fix any bottleneck"

With a proper reading of documentation and abiding by the tips provided, developers could have also understood the concept of multithreading in depth. It was apparent that many developers lack a proper understanding of how @threads work. There is a prevalent misconception among them that simply putting @threads in front of a FOR loop constitutes multithreading. This misunderstanding is more common among novice programmers, and their multithreaded code often consists mainly of FOR loops and the belief that using @threads is equivalent to implementing multithreading.

4.3 Theme III-Expectation vs Reality

Throughout the data collection phase of this study, there was evidence that didn't support some hypothesis. Various researchers have made statements regarding the difficulty of using multithreading. Jeff Huang and Charles Zhang (2016) mentioned that "debugging concurrent programs is a challenging task because of complexity in reasoning about concurrency" [19]. Also, Al-ladan (2001) mentioned that "testing and

debugging parallel and distributed software are much more difficult than testing and debugging sequential programs" [7]. This is due to the fact that errors are usually reproducible in sequential programs as compared to parallel programs [7]. These statements led to a notion that developers must be facing challenges with debugging, which led us to think that we would be going through a lot of posts concerning debugging and testing of multithreaded programs. Surprisingly, only 3 out of 212 posts were concerned with debugging of multithreaded programs (Figure 4.2) tagged as "How to debug" (See Table 4.2 for reference). Developers have asked about the method to debug their code in these three posts. In the case of testing, this study encountered only one post where a developer asked about testing of multithreaded code. This suggests that developers focus less on looking for the error by themselves and a possible reason could be lack of knowledge on how to do that.

Multithreaded code can be complex, and it can be difficult to identify performance issues and bottlenecks that may arise due to the interactions between threads. Profiled code refers to code that has been analyzed using a profiling tool to gather information about its performance characteristics, such as its execution time, memory usage, CPU usage, and method calls. Profiling is a process of measuring and analyzing the behavior of a program or system, and it helps developers to identify performance bottlenecks and optimize their code. Profiling multithreaded code can help identify issues such as thread contention, where multiple threads are competing for the same resources, or deadlocks, where threads are waiting for each other to complete but are unable to do so. Profiling can also help identify which parts of the code are taking the most time to execute and which threads are spending the most time waiting for resources. Experienced developers expect to have a profiled code so as to solve the error, but novice developers find it difficult to profile code as they are not aware of how to do this, possibly due to incomplete knowledge about multithreading and Julia. Developers are always required to work on different technologies. While working on

multithreading, this study found that developers working on Julia want it to provide the same multithreading platform as their other programming language they have worked with. There were several posts encountered where they compared the speed of their multithreaded code written in other language with Julia. In one of the posts, the developer mentioned that

> "I guess my point is that we can atomics in OPENMP and get better performance."

This study found that developers do continuous comparisons between the multithreading platform of their primary working language and Julia. Although Julia's official documentation provides information about its memory model, some developers don't assume that Julia provides sequential consistency for data race free programs. Clearly not all developers are aware of this which leads them to think no matter what the behavior of the program is, it will be unknown and irrational. In one of the posts, a developer mentioned that

> "I think this is expected. Although Julia does not specify its memory model, I guess it's safe to assume it's aiming to provide sequential consistency for data race-free programs(SC for DRF), like C,C++ and Java do. In that case, as soon as you write a data race, your program's behavior is undefined and you cannot expect any sane behavior."

This also validates that developers do not go through the official documentation thoroughly. They prefer to work according to their existing knowledge.

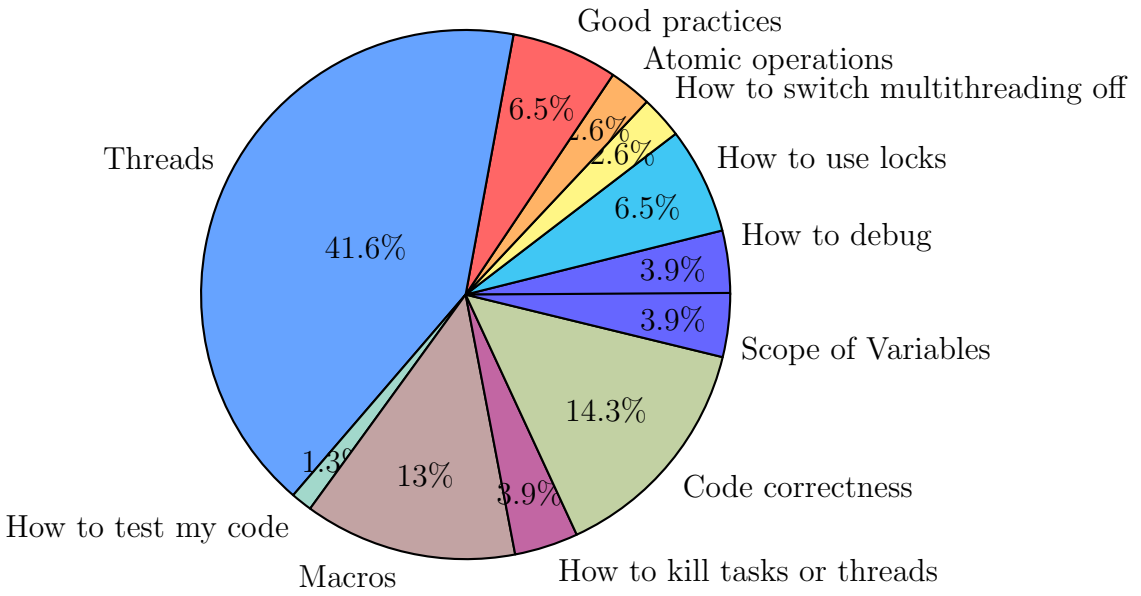

Figure 4.2: Queries asked under Need-to-know category

We also found that atomic operations have been referred to be used for avoiding race conditions since Julia supports accessing and modifying values atomically. But in some posts we found authors saying that

> "Just to be clear, If you want performance, you should not use atomics for computations. Atomics are used for thread synchronization and if you simply replace += with atomic add!. you are guaranteed a slow down on the order of 100x. "

This suggests that expert developers in Julia do not recommend atomics to be used as they impact performance of the code in a bad way. It is quite evident that this is due to lack of presence of more information in the official documentation.

4.4 Theme IV- Difference of Usage of Platforms

While working on collecting data, it was noticed that both platforms are used in different ways by developers. There are significant differences in the way developers

Table 4.2: Need-to-know categories

Name of the tag	**Description of the tag**
Macros	In this sub-tag, developers ask questions about macros. For example- difference between two macros, usage of macros with an example, how macros effect the code, etc.
Code correctness	Here the developers ask questions about whether their code is correct or not, what is the best way to write this code, what is wrong with the code, etc.
How to debug	Developers ask about the process of debugging multithreaded code.
Good practices/Thread safety	Developers asking about best practices to write a thread safe multithreaded code.
How to use locks	Developers asking about how to use locks in the code, if the usage of locks will benefit the code or not etc.
How to test my code	Developers asking about how to test their code.
How to kill tasks/threads	
How to switch multithreading on and off	Developers want to know if they can turn multithreading on and off in single code.
Atomic operations	Developers inquiring about usage of atomic operations in the code.
Scope of variables	Developers asking about scope of variables in the code.
Threads	This sub tag refers to all the inquiries where developers ask about threads. For example- How to set number of threads, how to pin tasks, how to run threads in order, how to add threads after initializing, how to control number of threads etc.

ask and answer questions, and the way in which posts are maintained. We found that in Julia Discourse, although many posts were solved through discussion, these were not marked solved by the author posing the question or by the platform's maintainers. Out of 74 posts, only 1 post related to multithreading in Stack Overflow was solved but was not marked as solved as compared to Julia Discourse. In case of Julia Discourse, 63 out of 137 posts were marked unsolved but initially there were solved but were not reported as solved (Figure 4.3). We observed that developers tend to be more formal and organized in their inquiries on Stack Overflow, as opposed to Julia Discourse. They demonstrate a serious intent to find solutions by providing well-crafted questions with minimal working examples. Overall, it was noted that

developers lack a clear understanding of how to effectively pose their inquiries on both platforms. Although they have a clear idea of the information they seek, they struggle to articulate their questions with the necessary code and terminology. In contrast, experienced developers pose their inquiries with properly formulated terminologies and minimal working examples.

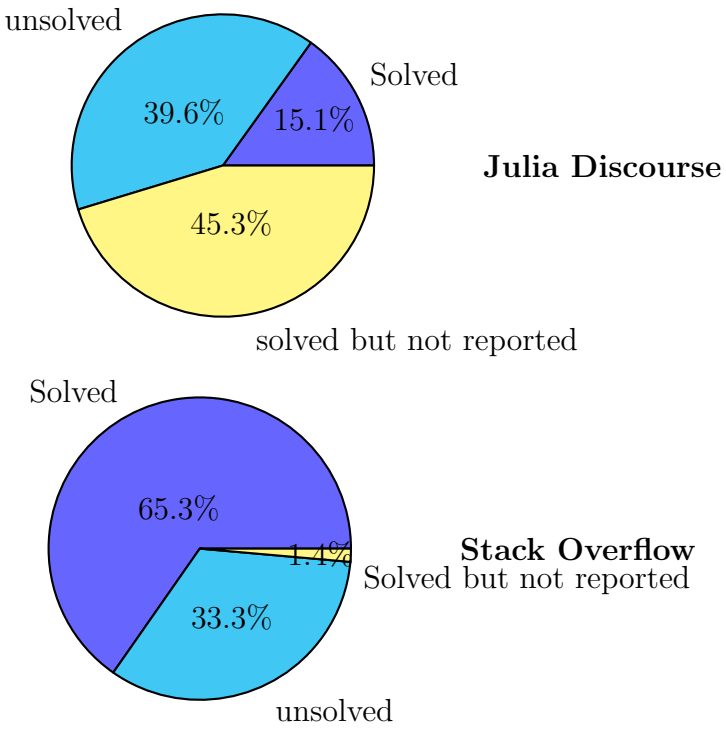

Figure 4.3: Posts marked as solved/unsolved

4.5 Theme V- Difficulty

This theme represents the difficulties developers have faced while writing multithreaded code. Various issues like race condition, memory allocation, garbage collection and type instability are common in multithreading. In this study, we found that developers find it difficult to understand the pattern of memory allocation (Figure 4.4). Developers are still confused about how their code is allocating memory. This confusion has also led to slow performance of multithreaded code. Memory alloca-

tion is defined as the process of reserving memory to store data. In Julia, memory allocation is allocated dynamically as needed during program execution. Julia uses a garbage collector to automatically manage memory allocation. Though Julia provides a number of functions and tools to manage memory allocation such as the "sizeof()" function to determine the size of a data type in bytes and a "pointer" function can be used to obtain a pointer to a memory allocation, developers still find it difficult to know where the memory is being allocated.

Race conditions are an issue that developers face. The interesting fact is that even though they are aware of it, they write their parallel code without keeping race conditions in mind. They know that it can occur but are not able to understand at what point and when it will occur. Race conditions occur when two or more different threads try to access the same shared resource at the same time and at least one of them is a write, leading to unexpected behavior. Race conditions can be difficult to detect and debug, because timing of threads can be unpredictable and may also depend upon other external factors such as speed of the CPU.

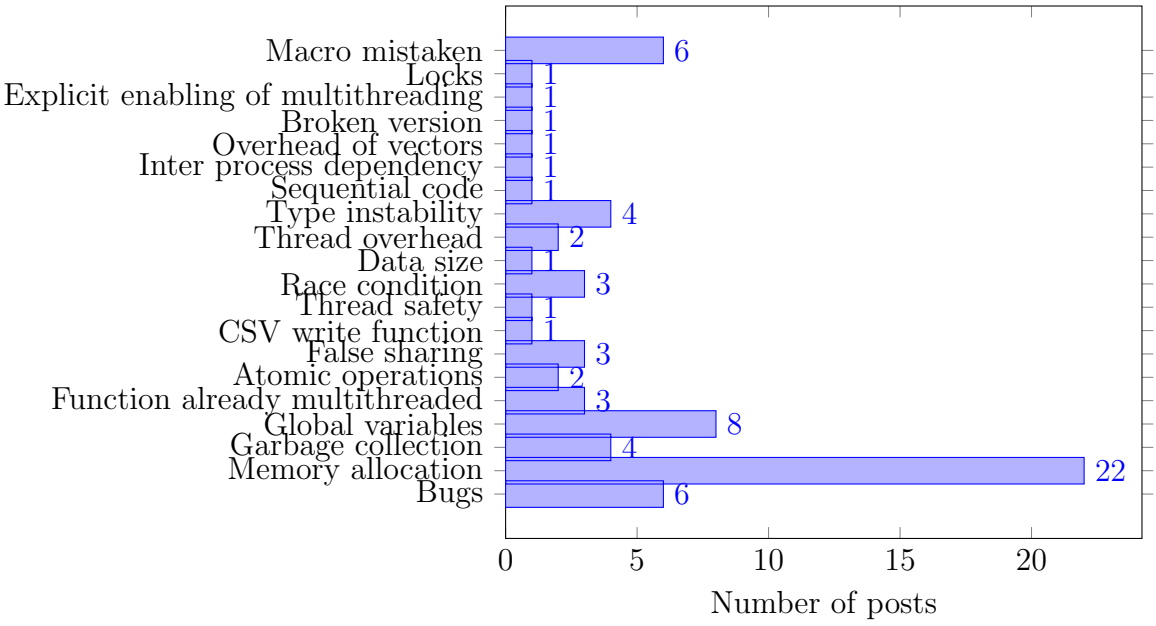

Figure 4.4: Causes behind slow multithreading

One of the other reasons behind the occurrence of race conditions is lack of knowledge about scope of variables. Scope of variables is defined as block of code in the entire program where the variable can be created and modified without any error. Julia's official documentation contains information about the memory model of Julia. A memory model is defined as the way of organizing and defining the memory, which means it describes how threads interact through memory and how they share their data. In many posts, we found that developers are curious to know about the memory model as they want to know how the compiler accesses threads. As discussed earlier, Julia has provided its official documentation but some features/usage factors of various macros have not been clearly documented. For example, it was not clear whether some of the macros can be multithreaded or not, i.e. whether we can parallelize them or not. In one of the posts, a developer mentioned that

> "It's not clearly documented, but @allocated cannot be used concurrently."

Developers find it difficult to know about these macros as they prefer to rely on the official documentation. But it is well observed that the official documentation lacks in information.

Chapter 5

Conclusion

Multithreading is a powerful tool that allows developers to create high level performance and responsive applications. Writing multithreaded code is a difficult task for developers. To help them, it is important to understand their interests and the challenges they faced. Julia is a programming language that is designed for high performance computing, scientific computing and data science. Julia has a sophisticated multithreading platform that allows users to take advantage of the power of modern CPUs with multiple cores and threads. However, as with any multithreaded application, users face various challenges such as slow performance, race condition and memory allocations. This study was conducted using a qualitative content analysis approach. The objective of this study was to answer these three research questions through inductive content analysis.

- **RQ1:** What questions do developers ask about multithreading?
- **RQ2:** What categories do these questions belong to?
- **RQ3:** What specific challenges do developers encounter when programming in a multithreaded environment specifically in Julia?

We found that developers ask questions about topics involving the correctness of

their code, inquiries about how they can write parallelized code and how they can increase the speed of the code, challenges they have been facing with their code and inquires about various concepts related to multithreading such as threads, macros and atomic operations, etc. To understand these questions and to obtain the statistics, they have been categorized into various tags (Figure 4.1). This study found that developers face issues such as race conditions, slow performance: garbage collection, memory allocations and type inference. There were various causes reported behind slow performance memory allocation in the code, including garbage collection, bugs reported, global variables in the code and atomic operations. The results of this study were presented in the form of themes. The five themes were Where should I tag this? (tagging confusion), How do I do this ? (Self Knowledge), Expectation vs Reality, Difference of Platform Usage, and Difficulty. The first theme represents the confusion of tagging the posts under the right tag. Many posts were incorrectly labeled with the "multithreading" tag simply because they contained information about threads. As a result, a significant number of posts that actually discussed other programming models, such as distributed programming or meta-programming, were also labeled under "multithreading". The second theme, "How do I do this?" in the study represents the developers' pre-existing knowledge and their attempts to work on multithreaded code based on that knowledge. The study's statistical data revealed that developers often ask performance-related questions, specifically regarding their difficulties with slow speed compared to sequential code. Despite Julia having its official documentation, developers often seek clarification on concepts through minimal working examples. This is supported by the observation that posts that included a minimal working example were more easily and seriously resolved compared to those without one. The Julia documentation provided some performance tips that can be helpful in writing efficient code but it was found that developers tend to ignore them and write their code without considering them. The third theme,

"Expectation vs. Reality," highlights instances where certain facts were expected to be true but were found to be untrue in reality. For example, we were expecting to have more posts on testing and debugging but in reality we got very few posts.The fourth theme represents "Difference of platform usage" that represents how developers approached Stack Overflow and Julia Discourse differently. The last theme represents "Difficulty" that represented areas that developers felt were challenging. Overall, this study made a significant contribution towards understanding the interests of developers working on multithreading. This could help them in focusing their efforts on the required tasks. Also, it would help Julia language maintainers to improve their documentation.

5.1 Implications

The results of this study can not only help concurrency developers, but also researchers, educators, developers and maintainers of Julia to better decide where to focus their efforts. Figure 4.1 represents challenges faced by developers while writing multithreaded code. Challenges like slow performance have causes which are well shown in Figure 4.4.

Developers By referring to Figure 4.1 and Theme II, a beginning developer can concentrate on their learning and gain knowledge on how to prevent the occurrence of certain issues. Moreover, they can acquire the skill of asking appropriate questions on online discussion forums, which can help them receive helpful responses from the community. Experienced developers can gain insights into how they can assist novice developers, as well as expand their understanding of the multithreading platform through exploration.

Researchers Since this is one of the first studies focused on the multithreading platform in Julia using qualitative content analysis, researchers may decide to work on

multithreading and qualitative content analysis approach in more detail in the hope of making greater contributions towards multithreading. This can make multithreading and Julia accessible to a wider range of programmers. Software Engineering researchers could be encouraged to use a qualitative content analysis approach.

Educators Using the findings of this study, educators can redesign their instructional material and can decide what aspect to focus on. They can decide which topics in multithreading need more learning time and can tell students what mistakes they can avoid.

Maintainers of Julia This study can help maintainers of Julia improve their multithreading platform for developers by indicating what features they should provide and what features need some improvement. Also, it shows the need to focus on providing a clear documentation of Julia. They can work on improving certain features such as macros, atomic operations, compiler's optimization for scope of variables etc.

5.2 Threats to Validity

In this section, we will discuss the threats to the validity of the study.

Internal Threat- Creation of tags to identify the challenges faced by developers is an internal threat to validity. There could be chances that the tags perceived by the researcher may be different from the perspective of other researchers. To overcome this threat, we conducted a validity test as described in the methodology section above. It was found that 5 out of 20 posts were tagged differently by both the researchers. After that, the primary researcher and secondary researcher managed to agree on all the posts by discussing their perspective on how they think about them.

External Threat- Challenges obtained from this study are relevant to Julia only, the results might differ for other languages. This can open door for exploring challenges faced in multithreading in other languages which could be a promising future work.

The limitations of the study include the following:

- This study was concerned with posts from two different online discussion forums, which can lead to external validity concerns or selection bias.
- This study was conducted without the direct involvement of developers, and therefore there was no opportunity to include their perspective with the findings obtained from the study.
- Results were affected by how the posts in the online discussion forums were maintained.

5.3 Future Work

The aim of this study was to understand challenges faced by developers in multithreading in Julia. Future work should therefore focus on solving the issues identified. For example, Julia developers can focus on improving the official documentaion, improve atomic operations, compiler's optimization for scope of variables etc. These areas could be a good point of improvement for Julia's multithreading platform. In this study, results have been represented in the form of themes and statistical representation. One avenue for future work is to check the trustworthiness of the findings obtained. One can replicate the study and check if the tags assigned in the study are reliable and can add more data to it. Also, one can use approaches to carry out the same study such as interviews, questionnaire to check the perspective of developers. To have broader insights from this study, one can extend the range of the study to other concepts related to parallel computing.

Bibliography

[1] Advantages and disadvantages of multithreading. "https://docs.oracle.com/cd/E13203_01/tuxedo/tux71/htmlpgthr5.htm", last accessed on 09/12/23.

[2] Julia 1.8 documentation. "https://docs.julialang.org/en/v1/", last accessed on 08/12/22.

[3] Notes on multithreading with julia. "http://www.cs.unb.ca/~aubanel/JuliaMultithreadingNotes.html", last accessed on 08/12/22.

[4] Where developers learn, share, build careers. "https://stackoverflow.com/", last accessed on 01/12/23.

[5] Luca Aceto, Anna Ingolfsdottir, Kim G Larsen, and Ji Srba. Teaching concurrency: theory in practice. In *Teaching Formal Methods: Second International Conference, TFM 2009, Eindhoven, The Netherlands, November 2-6, 2009. Proceedings 2*, pages 158–175. Springer, 2009.

[6] Syed Ahmed and Mehdi Bagherzadeh. What do concurrency developers ask about? a large-scale study using stack overflow. In *Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement*, pages 1–10, 2018.

[7] M Al-Iadan. A survey and a taxonomy of approaches for testing parallel and distributed programs. In *Proceedings ACS/IEEE International Conference on Computer Systems and Applications*, pages 273–279. IEEE, 2001.

[8] Eric Aubanel. Parallel program comprehension. PPIG, 2020.

[9] Diana A Barros and Cristiana Bentes. Analyzing the loop scheduling mechanisms on julia multithreading. In *2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 257–264. IEEE, 2020.

[10] Mariette Bengtsson. How to plan and perform a qualitative study using content analysis. *NursingPlus open*, 2:8–14, 2016.

[11] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59:65–98, 2017.

[12] Leah Bidlake, Eric Aubanel, and Daniel Voyer. Validation of stimuli for studying mental representations formed by parallel programmers during parallel program comprehension. PPIG, 2020.

[13] Manuel Carro, Julio Marino, Angel Herranz, and Juan José Moreno-Navarro. Teaching how to derive correct concurrent programs from state-based specifications and code patterns. In *TFM*, pages 85–106. Springer, 2004.

[14] Ji Young Cho and Eun-Hee Lee. Reducing confusion about grounded theory and qualitative content analysis: Similarities and differences. *Qualitative report*, 19(32), 2014.

[15] Kathryn A Davis. Validity and reliability in qualitative research on second language acquisition and teaching. another researcher comments. *Tesol Quarterly*, 26(3):605–608, 1992.

[16] Satu Elo and Helvi Kyngäs. The qualitative content analysis process. *Journal of advanced nursing*, 62(1):107–115, 2008.

[17] Rogerio Goncalves, Marcos Amaris, Thiago Okada, Pedro Bruel, and Alfredo Goldman. Openmp is not as easy as it appears. In *2016 49th Hawaii International Conference on System Sciences (HICSS)*, pages 5742–5751. IEEE, 2016.

[18] Hsiu-Fang Hsieh and Sarah E Shannon. Three approaches to qualitative content analysis. *Qualitative health research*, 15(9):1277–1288, 2005.

[19] Jeff Huang and Charles Zhang. Debugging concurrent software: Advances and challenges. *Journal of Computer Science and Technology*, 31:861–868, 2016.

[20] Stefan Karpinski Jeff Bezanson. Discourse. `"https://julialang.org/community/discourse/"`, last accessed on 01/12/23.

[21] Tobias Knopp. Experimental multi-threading support for the julia programming language. In *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 1–5. IEEE, 2014.

[22] Yifat Ben-David Kolikant. Learning concurrency: evolution of students' understanding of synchronization. *International Journal of Human-Computer Studies*, 60(2):243–268, 2004.

[23] Lukas Koschmiedar and Kai Neumann. Julia, Jul 2014. `"https://hpac.cs.umu.se/teaching/sem-accg-14/julia.pdf"`, last accessed on 01/12/23.

[24] Helvi Kyngas, Kristina Mikkonen, and Maria Kaariainen. *The application of content analysis in nursing science research*. Springer, 2019.

[25] Leslie Lamport. Teaching concurrency. *ACM SIGACT News*, 40:58–62, 2009.

[26] Wei-Chen Lin and Simon McIntosh-Smith. Comparing julia to performance portable parallel programming models for hpc. In *2021 International Workshop*

on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), pages 94–105. IEEE, 2021.

[27] Ziyue Lin and Haoran Ding. Research on multithread programming method based on java programming. In *2021 3rd International Conference on Artificial Intelligence and Advanced Manufacture*, pages 2405–2412, 2021.

[28] Ulrike Pfeil and Panayiotis Zaphiris. Applying qualitative content analysis to study online support communities. *Universal access in the information society*, 9(1):1–16, 2010.

[29] Mitchel Resnick. Multilogo: A study of children and concurrent programming. *Interactive learning environments*, 1(3):153–170, 1990.

[30] Benjamin Saunders, Julius Sim, Tom Kingstone, Shula Baker, Jackie Waterfield, Bernadette Bartlam, Heather Burroughs, and Clare Jinks. Saturation in qualitative research: exploring its conceptualization and operationalization. *Quality & quantity*, 52:1893–1907, 2018.

[31] FY Sebele-Mpofu and S Serpa. Saturation controversy in qualitative research: Complexities and underlying assumptions. a literature review. cogent social sciences. 2020; 6 (1).

[32] M-A Storey. Theories, methods and tools in program comprehension: past, present and future. In *13th International Workshop on Program Comprehension (IWPC'05)*, pages 181–191. IEEE, 2005.

[33] Filip Stromback, Linda Mannila, Mikael Asplund, and Mariam Kamkar. A student's view of concurrency-a study of common mistakes in introductory courses on concurrency. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, pages 229–237, 2019.

[34] Filip Stromback, Linda Mannila, and Mariam Kamkar. Exploring students' understanding of concurrency-a phenomenographic study. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, pages 940–946, 2020.

[35] David R Thomas. A general inductive approach for qualitative data analysis. 2003. "https://www.researchgate.net/profile/David-Thomas-57/publication/263769109_Thomas_2003_General_Inductive_Analysis_-_Original_web_version/links/0a85e53bdc04f64786000000/Thomas-2003-General-Inductive-Analysis-Original-web-version.pdf".

[36] Gias Uddin, Fatima Sabir, Yann-Gaël Guéhéneuc, Omar Alam, and Foutse Khomh. An empirical study of iot topics in iot developer discussions on stack overflow. *Empirical Software Engineering*, 26:1–45, 2021.

[37] Dorian P Yeager. Teaching concurrency in the programming languages course. In *Proceedings of the twenty-second SIGCSE technical symposium on Computer science education*, pages 155–161, 1991.

Appendix A

Keywords Used

Table A.1: Causes behind slow multithreading

Name of the tag	**Description of the tag**
Memory allocation	This sub-tag refers to all the posts where slow speed has happened due to unnecessary allocation of memory in the code. By definition- Memory allocation is defined as the process of allocating some portion of memory aside in a computer system for a specific purpose such as running an application or storing data. ***- Remember it is a sub tag under slow multithreading , don't confuse this tag with the tag memory allocation as it involves direct reporting of memory allocation.**
Garbage collection	This sub tag refers to all the posts where slow speed has happened due to garbage collection happened in the code. Since garbage collection is dynamic in Julia , this can hinder the performance. If garbage collection is taking a lot of time, this hints towards a lot of memory allocation happening in the code.
Bugs	This sub tag refers to all the posts where slow performance has happened due to bugs reported. Some of the bugs were reported during the discussions as well but some of them were already reported.
Global Variables	This sub tag refers to all the posts where slow multithreading is caused due to presence of global variables in the code.
Atomic operations	This sub tag refers to all the posts that involve presence of atomics causing slow performance as reported by the participants in the discussion.
False sharing	This sub tag refers to all the posts that involve false sharing of variables causing slow speed.
Race condition	This sub tag refers to the posts where slow speed is due to presence of race condition in the code. Race condition happens when two or more threads try to access the same variable. ***- Remember it is a sub tag under slow multithreading , don't confuse this tag with the tag race condition as it involves direct reporting of race condition by the author.**
Macromistaken	This sub tag refers to all the posts where slow speed is due to misunderstanding of macro usage in the code. ***- Remember it is a sub tag under slow multithreading , don't confuse this tag with the tag macro-mistaken as it involves direct reporting of macros being mistaken by the participants where performance wasn't involved.**
Function already multithreaded	This sub tag refers to all the posts in which the code involves usage of certain functions which are already multithreading but author is trying to apply multithreading on it again.
CSV write function	This sub tag refers to all the posts where slow speed is caused due to presence of CSV write function.
Thread safety	This sub tag refers to all posts in which code didn't follow the rules of thread safety that lead to slow performance.

Table A.2: Causes behind slow multithreading(continued...)

Name of the tag	**Description of the tag**
Data size	This sub tag refers to slow speed due to usage of small data size for an efficient program.
Type instability	This sub tag refers to all the posts in which the slow speed was caused due to type instability issue. Type instability refers to a situation where the compiler is unable to predict the data type.
Sequential code	This sub tag refers to slow speed due to presence of sequential code in the multithreaded code.
Thread Overhead	This sub tag refers to all the posts in which the slow speed is due to overhead of threads.
Inter-process dependency	This sub tag refers to slow speed caused due to processes being dependent on each other.
Overhead of vectors	This sub tag refers to the slow speed due to overhead of vectors.
Explicit enabling of multithreading	This sub tag refers to all the posts in which the author didn't enable multithreading in the code explicitly due to which the code was running sequentially.
Broken version	the earlier version in which the code was written wasn't having good multithreading due to which the performance was slow.
Locks	This sub tag refers to all the posts where slow speed happened due to presence of locks in the code.

Vita

Candidate's full name:Harshita
University attended (with dates and degrees obtained):Punjab Technical University (Jalandhar), 2016-2020, Bachelor of Technology in Information Technology

The data for this study is available online- `https://doi.org/10.25545/XR28QO`.

This study has been accepted as a Talk at JuliaCon2023 which is scheduled to held at MIT in the month of July, 2023.