CS 575 -- Spring Quarter 2022

Paper Analysis Project

Threads Cannot Be Implemented as a Library

Bhanu Prasanth Konda

934403560

kondab@oregonstate.edu

Threads Cannot Be Implemented As a Library

1. **What is the general theme of the paper you read? What does the title mean?**

   The theme of this paper is to show that multi-threading can't be implemented using a library-based approach. The paper proceeds in a way that first an approach was developed which nearly proves that Threading is possible using the library approach but later there was some situation made up to explain how it falls short in expressing an efficient parallel algorithm.

   The Title "Threads Cannot Be Implemented as a Library" is completely apt for the research paper as it talks about exactly what it's going to prove.

2. **Who are the authors? Where are they from? What positions do they hold? Can you find out something about their backgrounds?**

   The Author Hans-J. Boehm is a Software engineer. Currently, working as Software Engineer at Google since 2014. During this paper's publishing (2005), he was working as a Researcher in the Research Manager role at HP Labs.

   He completed his under graduation at the University of Washington in 1978, then pursued their master's from Cornell University in 1983. During the final year of his master's, started working as an assistant professor at the University of Washington in 1982. Later, joined as an Assistant and Associate Professor at Rice University in 1984. In 1989, started working as a Researcher for Xerox PARC then shifted to SGI as a software engineer in 1996. Posted as a researcher at HP in 1999 and worked will 2014.

   During his professional career, He has published more than 60 papers. He is currently working on the most concurrent programming issues at Google, generally focused on Android. He has served as Chair of ACM SIGPLAN from

2001 to 2003. Then was posted as the Chair of ISO C++ Concurrency Study Group (WG21/SG1) till 2017. He has contributed a lot to Android's ART Java language runtime and athematic equation evaluator for the default Android calculator application. His recent works contributed to compiler optimization consequences and non-volatile byte-addressable memory.

3. **What experiments did the paper present?**

Every traditional programming language supports concurrency which means, that when one thread is a = 2; x1 = b; and another thread is executing b = 2; x2 = a; In which scenario either a or b should be executed first then either x1 or x2 should have a value 2 in them after execution. This is the simplest memory model, but it isn't always the most practical because, in practice on traditional architectures, it looks doubtful that such a restricted memory model can be implemented with adequate performance. If computed normally, both x1 and x2 should contain a zero so the following are the 2 reasons for this to happen.

1. Compilers must reorder the code/instructions for the memory operations to overcome this scenario. This program will slightly perform better because of the better instruction scheduling.
2. The hardware also may reorder the memory to increase the performance of the program. It was mentioned that X86 processors may reorder a store followed by a load and when the data is loaded into the cache then all the threads will be able to view the data.

To avoid this PThread approach was considered, as Pthreads standards are to intentionally avoid specific formal semantics for concurrency.

PThread approach is implemented using C and C++ as in Java, this is designed to limit this kind of possibility of malicious coding. The following are the C/C++ implementations that support PThread

1. pthread_mutex_lock () and pthread_mutex_unlock(): These function guarantees the memory synchronization by using hardware instructions that prevent the re-ordering done by the hardware component.

2. To prevent the compiler from re-ordering the memory, pthread_mutex_lock() will be called immediately after the function call is done, which makes the compiler assume that this function is going to modify the values and won't let any other thread access those values and wait till it is done.

This method clearly works most of the time but unfortunately, we'll see that it's too unclear to allow a programmer to make a persuasive argument regarding program correctness, or to provide the compiler implementor with explicit instructions. The following are the three issues that needed to be corrected during the implementation:

1. Concurrent modification: The PThread avoids racing by stopping the threads access some part of the memory which is shared among the threads.

2. Rewriting of Adjacent Data: When storing a variable in memory, the compiler may write to nearby memory regions. For instance, there is a struct defined with 2 variables in it. Pthread prohibits writing the values in a single memory space and reading them at the same time but here there is nothing blocking the compiler from doing it.

3. Register promotion: This promotes the code optimization which will increase the read and write of the particular memory location to speed up the process which is against the PThread standards.

Due to PThread, we may expect that in some cases we may get a single thread performance despite having numerous threads.

4. **What conclusions did the paper draw from them?**

   All the above problems can be solved by adopting a different memory model like Java Memory Model. The following can be the goal while adapting to a new data model:

   1. If type-safety is not guaranteed then it appears not necessary to completely define the semantics for all data types. This may be allowed to prevent the data racing that is happening by preventing data sharing done through a particular library. This will preserve the PThreads approach.
   2. Few topics, such as type-safety and security motivated, received little attention since they were not considered important to examine because such a large amount of work is not expected in a type-safe language.
   3. The trade-off of adopting the Java Memory Model is the performance for a simpler memory model which won't allow reordering of the memory.
   4. In C++ bit-fields, the compiler must introduce memory stores that will prevent the race among the threads, which can be illustrated as modern architecture which prohibits adjacent bit-field reading and writing.

5. **What insights did you get from the paper that you didn't already know?**

   The paper covers the topic related to the operating system, java and C/C++ which are a bit familiar, but the new thing I learned from this paper is Java Memory Model and PThreads, their advantages and disadvantages. Some of the topics I have done my research on are the locking mechanism from Operating systems and race conditions. Parallel Sieve of Eratosthenes algorithm was a new topic for me and took a while to analyze how was it included in the experiment. I've also learned that enabling concurrent data races makes it hard to get the most out of a multi-processor without atomic actions on shared variables, which can avoid damage. In library-based threads, where synchronization is required for data change, this scenario is completely unachievable.

6. **Did you see any flaws or short-sightedness in the paper's methods or conclusions? (It's OK if you didn't.)**

   I haven’t seen any flaws in the paper as it was clearly explained with the help of examples like functions and code lines of different threads. The Conclusion part of the paper has not provided a clear problem statement for each result or the possible solution. There can also be a section that explains how Java Memory Model differs from the current data model that is been used. This addition could have helped me view the paper from the author's perspective of views the memory model of the experiment.

7. **If you were these researchers, what would you do next in this line of research?**

   If I was a researcher, I would have extended the research on how these behave in different data models and find which will be the best way to use PThread or normal theirs to maximize my performance and reduce the overhead calls for the library. Because threads and PThreads are such a vast topic that have a wide range of applications, we will be able to expand our research by incorporating some more functionalities that might help them work independently despite the data model.

Reference:

Hans-J. Boehm. 2005. Threads cannot be implemented as a library. SIGPLAN Not. 40, 6 (June 2005), 261–268. https://doi.org/10.1145/1064978.1065042