CS-575 Parallel Programming

Spring 2022

Name: Akhil Sai Chintala

Email: chintala@oregonstate.edu

Project Name: Paper Analysis Project

Threads Cannot Be Implemented As a Library

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

   This paper "Threads cannot be implemented as a library" has explained the content and what the discussion is going to be in the paper by the name itself. The introduction of the paper refers that multi-threaded programs are rapidly growing and driven because many other programs need to carry on certain tasks which are needed to be done concurrently while the other one is under execution. Many multi-threaded programs that communicate to memory shared through threads are not based on add-on libraries but because of the compiler and language specification and they can't be addressed purely within the thread library or its specification. It also explains that without writing additional multi-threaded programs there is no other way to utilize the performance of the additional processor. This paper is going to revise the C++ language standard to for better accommodation of the threads and to justify why it is necessary for the other environments that are dependent on the library-based threads.

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

   The author of the paper is Hans-J. Boehm is from Palo Alto, CA who is currently working as a software engineer at Google since March 2014. His work is mostly concentrated on concurrent programming issues generally focused on android. He has done his under graduation from University of Washington and Graduation from Cornell University in the late 1970's and 80's. After his graduation, he worked as

   - an Assistant Professor in the same university from which he has done his under graduation till 1984.
   - Assistant and Associate Professor at the Rice University.
   - Researcher in Xerox PARC and HP labs.
   - Software Engineer in SGI and Google (currently working)

   Projects wise he has recently worked on Android's ART Java Language Runtime and implemented the arithmetic evaluation engine for the default android calculator. His

other works are on data race detection and understanding how to program systems with non-volatile byte addressable memory. In the past, he worked on,

- Conservative Garbage Collection using the implementation of Russell Programming when he started working at the Rice University
- Multiprocessor Synchronization Algorithms which are done using fast lock implementations for java virtual machines and fast multiprocessor synchronization.
- Constructive Real Arithmetic to explore the practical implementations of exact real computer arithmetic which was used to implement arithmetic in Google's calculator.
- Ropes or Scalable Strings using the Xerox cedar environment which heavily depends on a scalable implementation of strings which are also known as sequence of characters represented as trees.

Until Late 2017, he was chaired at the ISO C++ Concurrency Study Group which he was used to actively participate in it.

3. What experiments did the paper present?

In this paper, the experiments are done using Pthreads which are implemented using C or C++. Here Pthreads also known as POSIX threads are defined as an execution model that exists independently from a language and from the parallel execution model. The main of the experiments is to check the concurrency of the implementation of the threads and how the programming languages are specifying the semantics of multi-threaded execution. As all programming languages support the true concurrency because of two reasons:

- Re-ordering of operations in the program due to compiler which leads to intra-thread dependencies.
- Re-ordering of operations done by the hardware based on the familiar constraints which usually store the results in the write buffer entry.

To maintain the concurrency and to test the program with data races also known as concurrent reads and writes the support of Pthreads is implemented as functions known

as pthread_mutex_lock(); for memory synchronization by including hardware instructions that prevent the hardware re-ordering of memory operations. To prevent the compiler from the re-ordering of memory operations the function call to pthread_mutex_lock(); which is also known as opaque function call is implemented.

But, there's a problem with this approach because it excludes the best working algorithms in some cases which may result in poor performance. Because of this some of the issues were corrected and re-implemented. There are 3 issues which were sent for correction during the implementation. They are:

- Concurrent Modification: To prevent race condition i.e., by requesting the access of a thread which is currently under modification.
- Re-Writing of Adjacent Data: Race caused by the compiler due to assignment of variables which share a memory location by which adjacent read or multiple writes are performed.
- Register Promotion: Compliers must be aware of the threads and a language must address the semantics of the threads. Here, thread safety is expected which has a defined cost in single-threaded programming.

The only parallel programming style implemented by the pthreads using library are used to prevent concurrent modification of shared variables and this allows the above experiment to the proper implementation.

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

The problems which are raised in the experiment will be solved by using the approach of Java Memory Model but, adapted from different languages design goals are described in other ways by the absence of type-safety which leads to the definition of the data races and by restricting them. Whereas some type of safety and security issues have become less critical by expecting the less work. They can be mentioned as follows:

- It's not important to declare all the semantics of all data races when the data type-safety is not present. Whereas it is okay to restrict few things of data races

to volatile or shared memory accesses through specific library routines which preserves the Pthreads approach instead of libraries.

- Few issues like type-safety and security motivated didn't grab much attention as they were not that critical to investigate because that amount of work is not expected in the context of type-safe language.
- In this context, the Java Memory Model performance is traded for the importance of simplicity in very less cases in the volatile store followed by the volatile load.
- In the bit fields of C++, stores must be introduced by the compiler and due to this the probability of the races will no longer be present in the source and this can be limited to the adjacent bit fields.

This explains the process of steps and the cases of how the experiment is handled and how the expectations are met during and after the research work. In this case, Java memory model traded the performance for the simplicity in very few cases and the compiler must introduce the policy of stores by the possibility of the races that were not present in the source which can be led by the adjacent bits.

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

Most of the terms and the insights from the paper are familiar to me but the concept of pthreads and Java Memory model was a bit new for me. As I have learned java and threads execution in java for a basic level but didn't have much idea about these topics before. As I already know the race condition and C, C++ languages it was a bit easy for understanding the paper and extracting the information from it.

The process of expensive synchronization was a bit different as well as difficult for me to understand because of parallel Sieve of Eratosthenes algorithm, implemented on shared memory machines. I have also got to know that by allowing the concurrent data races it is impossible to gain the benefit from a multi-processor without using the atomic operations on shared variables which can prevent the harm. This case is totally un-achievable in library-based threads in which synchronization will be mandatory for data modification.

6. **Did you see any flaws or short-sightedness in the paper's methods or conclusions?**

   No, I haven't seen any flaws in the paper's methods because of the way it was implemented, and the way researchers explained the experiment. But, in my opinion I have felt that the conclusion could have been more better by explaining the results instead of concentrating on the performance heading in the paper. The conclusion should always explain the problems effects and how they are resolved by the performed experiment. We can also explain the process of how the expensive synchronization which was a part of the paper like how the Java Memory Model and its cases were explained. My view is not contradicting with the current conclusion but just to edit and revise the content written in it.

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

   If I was one of the researchers for this study, I would have extended this research by using threads that cannot be implemented as a library using some different approaches by not using Java Memory Model as a reference. As threads and pthreads are a vast topic which we can extract a lot of knowledge from which will also support in extending the research by using some more essential functionalities.

   I would have collaborated with the same people who have helped in the previous research and also by adding one or more new members who have a good idea of threads and their implementation in different programming languages using their semantic definitions.

References:

[1]: 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