# Visualization approach for Expressing Synchronization Intentions

## Short Description

Reasoning about synchronization mechanisms in multithreaded software is hard. This work aims to create an interactive approach for the visualization of accesses to shared memory by different threads, on a higher abstraction level. Developers then will have an opportunity to visually express their synchronization intentions. During this work, it will be necessary to create an approach that will automatically transfer these visual intentions into synchronization mechanisms. It will be necessary to create a conceptual solution for this challenge, an architectural view visualizing behavior of threads, an algorithm for generation of synchronization mechanisms based on the visualization, and a prototype of a tool that demonstrates the benefits of this approach. Input for this work will be already identified memory locations between threads and their synchronization mechanisms, as well as an interface that enables automatic generation of new synchronization mechanisms.

## Abstract

In the era of multithreaded software, threads execute concurrently and independently. Input data and the interaction between concurrent threads determine the result of the concurrent software. In order for software to perform correctly, it is necessary to synchronize the interaction between threads. However, due to the nondeterministic nature of multicore hardware (e.g. shared cache, shared memory bus); it is hard to predict order of interactions between threads, as they do not progress uniformly. Developers find it very challenging to reason about necessary synchronization between the threads, as it is an np-hard problem. Eventually, developers either over synchronize their software (downgrading parallel programs by introducing many sequential sequences), or do not properly synchronize thread operations (leading to concurrency bugs). The particular concern is the issue of atomicity – if there are several accesses to shared memory within one thread, how are they related (i.e., do developers assume atomicity between them?)?

The main issue behind synchronization perils is transformation of developer's synchronization intentions into synchronization mechanisms. Developers, at all time, need to understand two things: i) which memory locations are shared among threads, and ii) how threads can access shared memory locations (different execution paths, different operations, atomicity). Finally, they need to make a decision, which parts of the code are one critical section and which parts can be different critical sections?

The idea of this work is to offer developers interfaces for visualizing dynamic behavior of software. Based on this view, it is necessary to offer developers means to express their synchronization intentions. Finally, it is necessary to create an algorithm that will transform these into appropriate synchronization mechanisms.

The main benefit of this work is the abstraction of synchronization from code to a conceptual, architectural level. Instead of thinking about possible execution paths and shared memory, developers only need to make explicit their intention. The underlying algorithms and frameworks should generate adequate synchronization according to the intentions that developers (in any

case) need to make. The visual approach brings two benefits: i) the synchronization intentions are now explicit, and ii) because of the visual representation are easy to manage. Because these are generated automatically, there will be no issue with the traceability between the design and the implementation.

List of actions:

- Understand different synchronization mechanism types.
- Understand concurrency bugs.
- Understand visualization and software architecture views for concurrent software.
- Create a view for visualization of concurrent threads and their accesses to shared memory (based on the existing analysis).
- Create a concept for expressing synchronization intentions.
- Create an algorithm for generating the synchronization intentions from high level visual synchronization intentions.
- Select and prepare benchmarks to test (e.g., remove existing synchronization mechanisms, introduce new synchronization through this approach).
- Apply the solution on the benchmarks.
- Use existing tools (e.g., Thread Sanitizer, Helgrind) in order to find concurrency bugs (compare original version with existing synchronization mechanisms, and the new one generated using this approach in terms of concurrency bugs).

Expected contributions:

- High level architecture view for representing threads and their accesses to shared memory.
- Interactive view for expressing synchronization intentions.
- An algorithm to convert visual synchronization intentions into concrete synchronization mechanism implementation.

Keywords: Multithreaded Software, Visualization, Software Architecture, Synchronization Intentions, Synchronization Mechanisms, Concurrency Bugs.