

Laws of Order: Expensive Synchronization in Concurrent Algorithms Cannot be Eliminated

hagit A, Rachid G, Danny H, Petr K, Maged M, Martin V

Presented by
Akshay Gopalakrishnan

November 2, 2021

Introduction

- Designing Concurrent Algorithms with good performance is a non-trivial tasks.
- Major part of slowdown is due to synchronization.
- Hence efforts are made to remove such costly synchronizations.
- This paper shows that it is impossible to remove completely all expensive synchronization primitives for a class of concurrent implementations.
- Such a result helps designers to assert when they can stop improving an algorithm this way.

Summary of Results

The results are based on the usage of atomic Read-after-Write and Write-after-Read operations. The authors show that without using the above primitives, it is:

- Impossible to build a linearizable implementation that is non-commutative and respects deterministic sequential specification.
- Impossible to build an algorithm that respects mutual exclusion and is deadlock-free.

Mutual Exclusion

Given N processes, each of which accesses a critical section by acquiring a lock, mutual exclusion requires

No more than one process can be in the critical section at the same time

Proof intuition

Proof by contradiction. Part 1:

- Assume that we do not use a shared write updating the lock.
- Then this would imply more than one thread can be in the critical section.

Part 2:

- Assume that we do not use ARAW or AWAR as part of the lock implementation.
- Then this would imply more than one thread can read concurrently the same lock and be ready to acquire the lock (update the lock variable).
- Then one thread can acquire the lock (by updating the lock variable).
- After which another thread can also acquire the lock (by updating lock variable too!).
- Thu, violating mutual exclusion.

An algorithm is linearizable w.r.t a sequential specification if

Each execution of an algorithm is equivalent to some sequential execution of that specification, where the order between non-overlapping methods is preserved.

Note: Not all linearizable algorithms need to use RAW/WAR. Only some, which have two properties.

Two Properties of Linearizable Algorithms that must use RAW/WAR

- Deterministic Sequential Specification.
- Strongly non-commutative methods.

Proof Intuition

Examples of strongly non-commutative methods

Formal proof elements

- Formal language with transition semantics.
- Formal definition for executions.
- Notion of histories as sub-executions (traces).
- Proving both above claims in the same flow using these formal elements.

Conclusion

Thank you

Questions?