An Experimental Evaluation of Value-Splitting in Transactional Memory

Rui Ribeiro, José Pereira, Nuno Faria

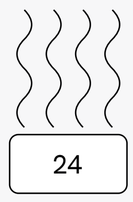
In databases, contention on popular items (*hot spots*) is undesirable, as it can severely hinder performance.

Previous research on the topic found that using *value-splitting* techniques successfully alleviated the problem.

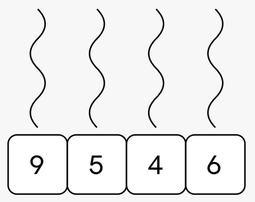
As transactional memory (TM) borrows several of the concepts behind database transactions, our objective is simple: **analyse the viability of value-splitting techniques on TM systems.**

Value-Splitting

The process of dividing a value into multiple slices, to promote parallel work.



Single value: All threads try to access/modify the same value, generating contention.

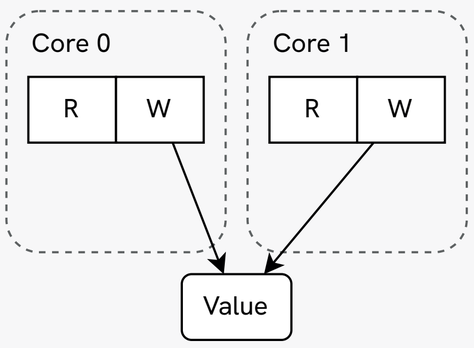


Split value: The threads are spread out through the slices.

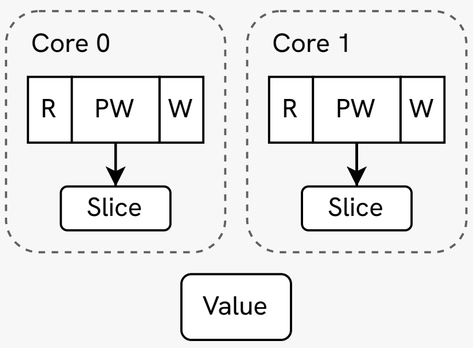
Phase Reconciliation (PR)

The value transitions between several phases, depending on the level of contention. The phases are:

- **Joined:** Single value.
- **Split:** Each core has its own slice of the value.
- **Reconciliation:** Merging all slices of *split* to return to *joined*.



In the *joined* phase, updates are issued to the value itself.



In the *split* phase, updates are issued to a private per-core slice.

It has some limitations!

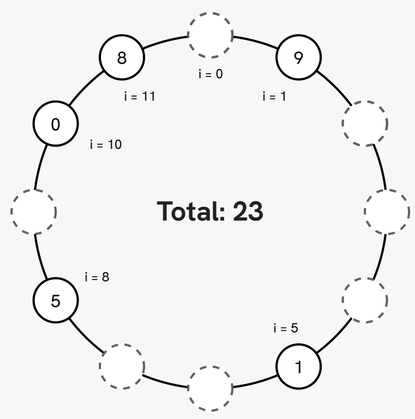
- The number of slices is static (equal to the number of cores on the system).
- The *split* phase is restricted to one type of operation (the one with highest contention).
- Threads that try to run other operations are blocked until *reconciliation*.

Multi-Record Value (MRV)

The value has only one single state, regardless of the number of slices; all operations work the same way.

Main features:

- The slices (records) of the value are stored in a ring-like fashion, with access indexes determined at random; there is no assignment.
- No operations block, as all threads can access all slices.



Implementation

- C++ 20 library
- Targets the *Wyatt-STM* system
- Uses *immer* for internal immutable data structures
- Provides *read*, *add*, and *sub* operations

Selected Results

Microbenchmark with long-running transactions, each performing one operation *(read, write increment)* on the value-splitting object.

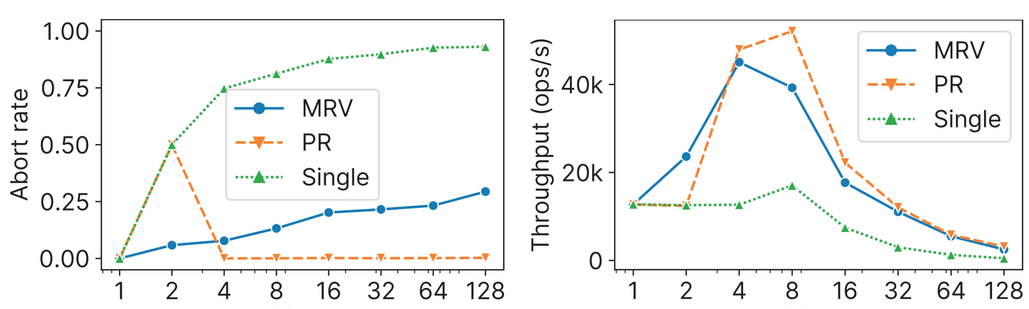


Fig. 1: Results for a write-only workload with a variable number of clients.

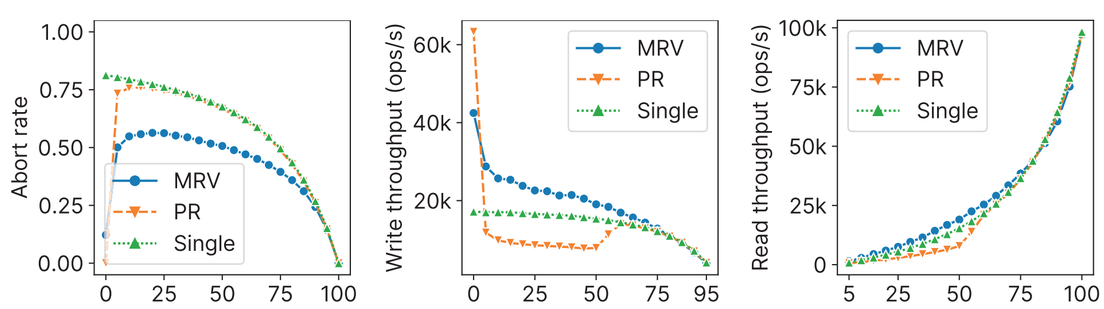


Fig. 2: Results for a mixed workload with eight clients.

- Fig. 1 demonstrates how splitting techniques offer a performance benefit over using a single value in write-only workloads, both in terms of abort rate and throughput.
- Fig. 2 demonstrates how MRV handles well both reads and writes, showing an improvement over the single value; PR falls short due to its need to transition phases frequently.

Machine specifications

- 2x HiSilicon Kunpeng 920 (ARM); 128 cores
- 8x 32GB DDR4 2666MT/s
- Rocky Linux 8.4 w/ Linux 4.18.0
- gcc 12.2.1, with flags `-O3` and `-march=native`

Conclusion

Preliminary results show that value-splitting is worth exploring in the context of transactional memory systems, with MRVs specifically offering an all-round performance improvement over single values.

INESCTEC

fct Fundação para a Ciência e a Tecnologia

This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020.