Carnegie Mellon University

Hardware Transactional Memory in Go (Go Library - SafetyFast!)

J. Craig Hesling

CH

Introduction

I set out to bring the world of Intel Hardware Transactional Memory(HTM) to Go.
To make this happen, I would need to present the following items:

- A library that made it easy to use HTM in Go
- Evidence that my primitives worked as intended and were special
- Some real-world gains in common data structures

Background

Intel has implemented transaction memory features under the name Intel TSX.
Intel TSX contains two different instruction extensions, HLE and RTM, which approach HTM from two different angles.
Simply put, **HLE** is designed as a drop-in replacement for current locks, whereas **RTM** allows low level control of transactions.

Results

I have implemented highly tuned Go primitives to make use of both HLE and RTM.
In the ideal HTM experiment, the new primitives are validated and exceed the expected performance. The hash-map experiment (and AVL tree experiment) stand to demonstrate the gains from using these primitives in real world applications.
A simple take away is that HTM can massively speed up reads or modifies, but often fails during insertions, due to global data structure tampering.

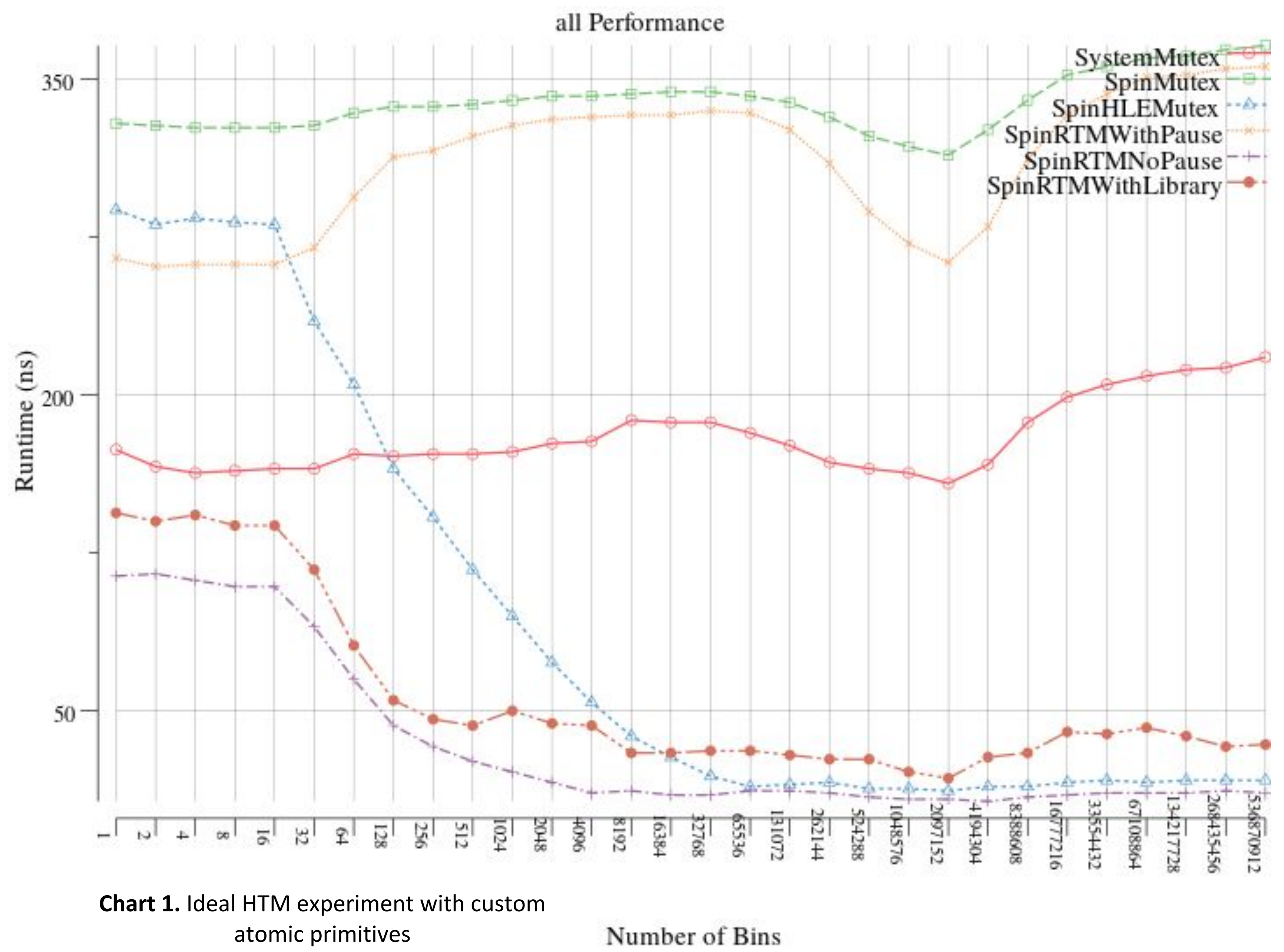

Chart 1. Ideal HTM experiment with custom atomic primitives

```go
var lock SpinHLEMutex
lock.Lock()
// Action to be done transactionally
count := m["word1"]
m["word1"] = count + 1
lock.Unlock()
```

Figure 1. Example of using the HLE lock replacement

```go
c := NewRTMContexDefault()
c.Atomic(func() {
    // Action to be done transactionally
    count := m["word1"]
    m["word1"] = count + 1
})
```

Figure 2. Example of using the RTM transaction context primitive.

```go
c := NewLockedContext(new(SpinHLEMutex))
c.Atomic(func() {
    // Action to be done transactionally
    count := m["word1"]
    m["word1"] = count + 1
})
```

Figure 3. Example of using the HLE transaction context primitive. This allows interoperability between RTM, HLE, and normal system locks.

```
// Note: Argument attempts must be greater than 0
// func HLESpinCountLock(val, attempts *int32)
TEXT ·HLESpinCountLock(SB),NOPTR|NOSPLIT,$0
    MOVQ val+0(FP), CX
    // Load attempt counter in DX
    MOVQ attempts+8(FP), R8
    MOVL (R8), DX
tryread:
    MOVL (CX), BX
    TESTL BX, BX
    JE tryacquire
    PAUSE
    DECL DX
    // If DX != 0, abort
    JNE tryread
    JMP abort
tryacquire:
    MOVL $1, AX
    XACQUIRE
    XCHGL AX, (CX)
    TESTL AX, AX
    JNE tryread
abort:
    // Write back attempt counter
    MOVL DX, (R8)
    RET
```

Figure 4. Implementation of the HLE spin lock in Go ASM

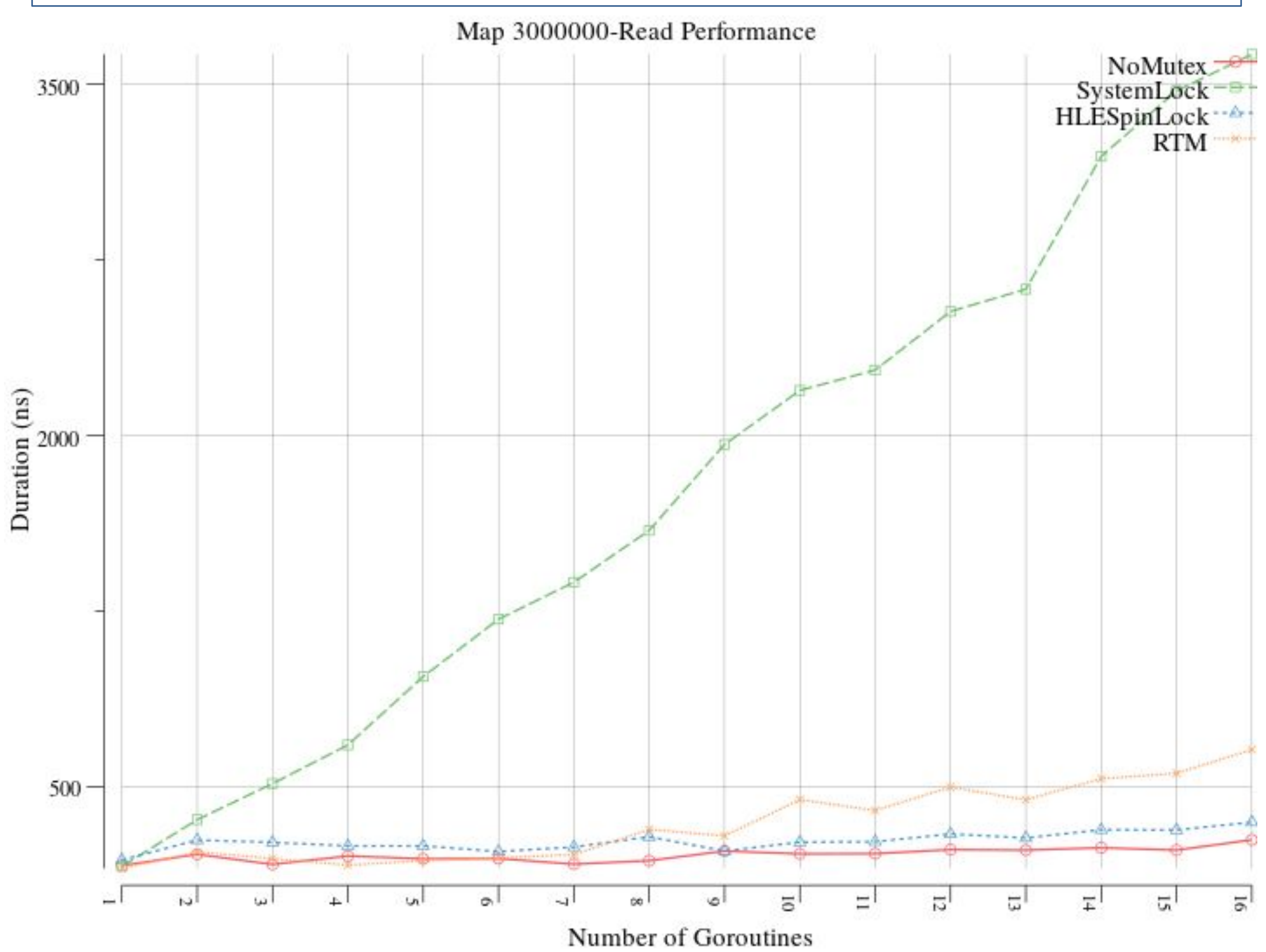

Chart 2. Test with Hash-Map Reads Only

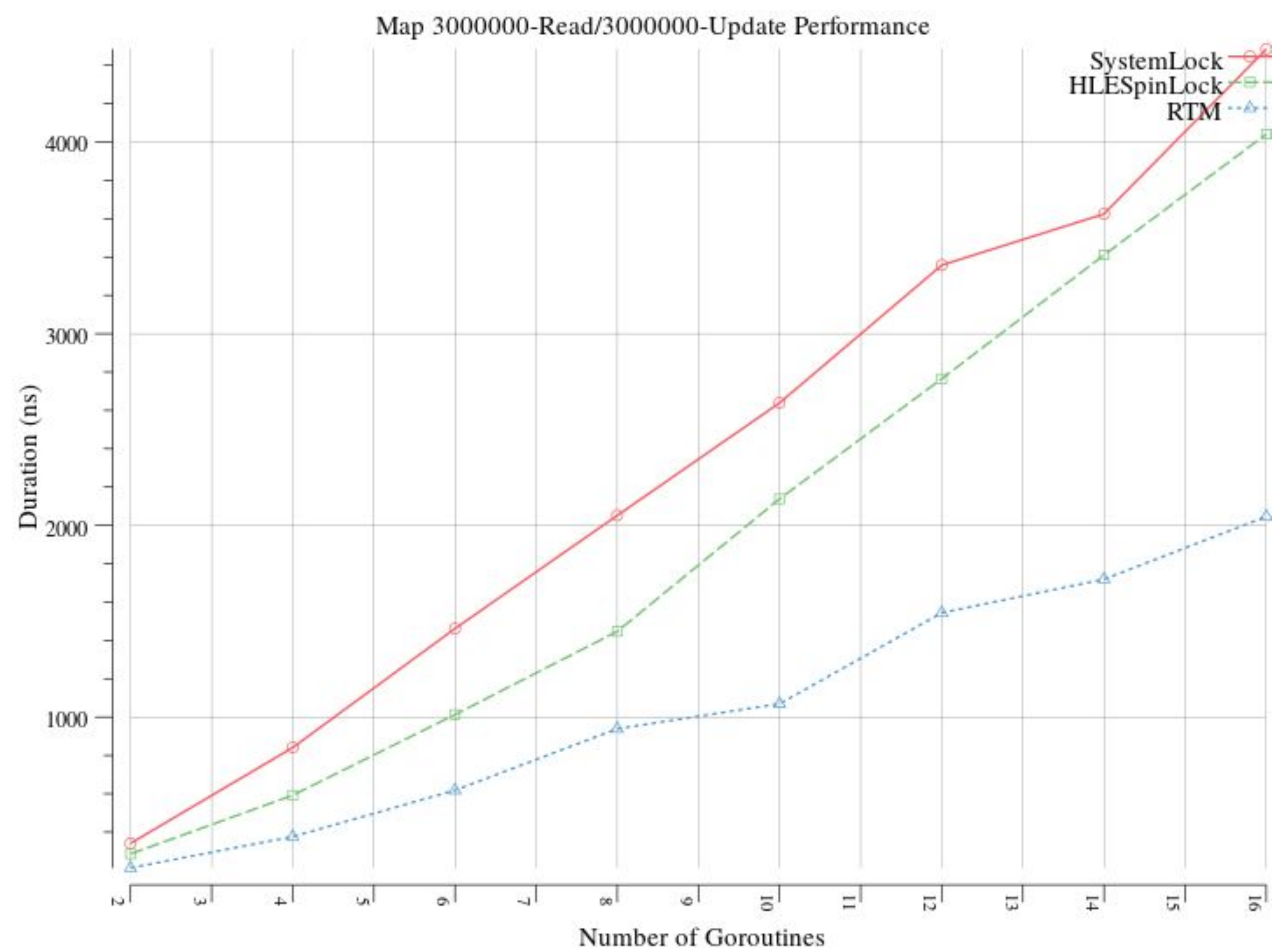

Chart 3. Test with Hash-Map Reads and Updates

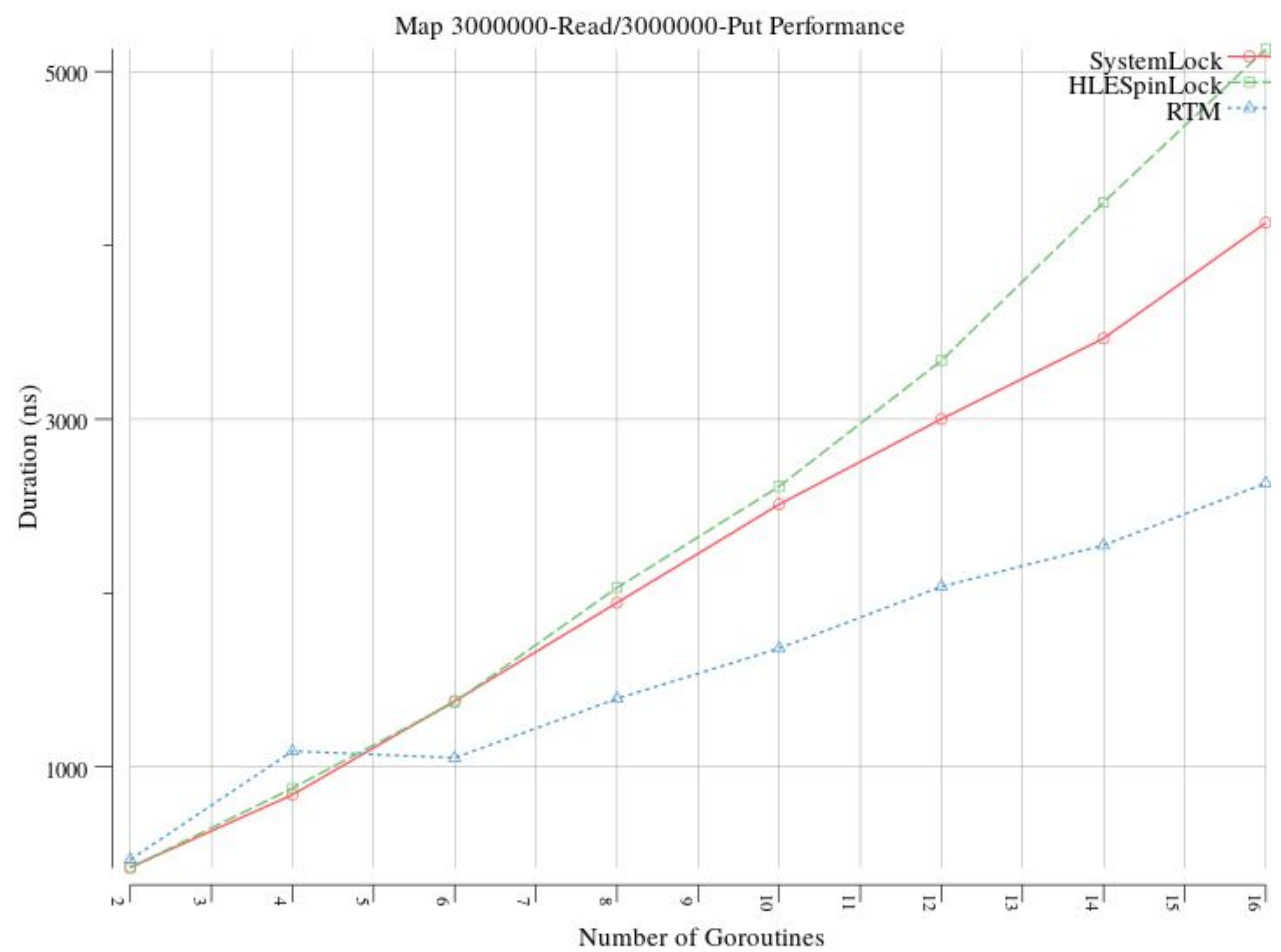

Chart 4. Test with Hash-Map Reads and Puts

Contact

J. Craig Hesling
Carnegie Mellon University
Email: craig@hesling.com
Website: http://craighesling.com

Project Page

https://github.com/linux4life798/safetyfast