STM in the Small

Trading Generality for Performance in Software Transactional Memory

Aleksandar Dragojević[1] Tim Harris[2]

[1]I&C, EPFL
Lausanne, Switzerland

[2]Microsoft Research
Cambridge

presented by Thomas Herzog
Concurrency and Memory Management Seminar
Prof. Christoph Kirsch, University of Salzburg

UNIVERSITÄT SALZBURG

1. Motivation
   - CAS vs STM
   - Why STM is slow
   - SpecTM
2. SpecTM
   - Short Transactions
   - Explicit Transactional Variables
   - Combined Metadata with Value-Based Validation
3. Evaluation
   - Skiplist
   - Performance

UNIVERSITÄT SALZBURG

CAS vs STM

CAS ⟷ STM

CAS	STM
high speed	low speed
harder to use	easier to use

UNIVERSITÄT SALZBURG

Why STM is slow

- Book-keeping required when starting a transaction
  - Taking a snapshot of processor state
- Managing the transaction record on each read and write
- Visiting meta-data locations for concurrency control

UNIVERSITÄT SALZBURG

CAS ⟷ STM

SpecTM

CAS	STM
high speed	low speed
harder to use	easier to use

SpecTM

- Provides a special API
  - Improved performance
  - Less generality
- Can be mixed with normal transactions
  - Use normal transactions in the general case
  - Use SpecTM API in performance-critical sections

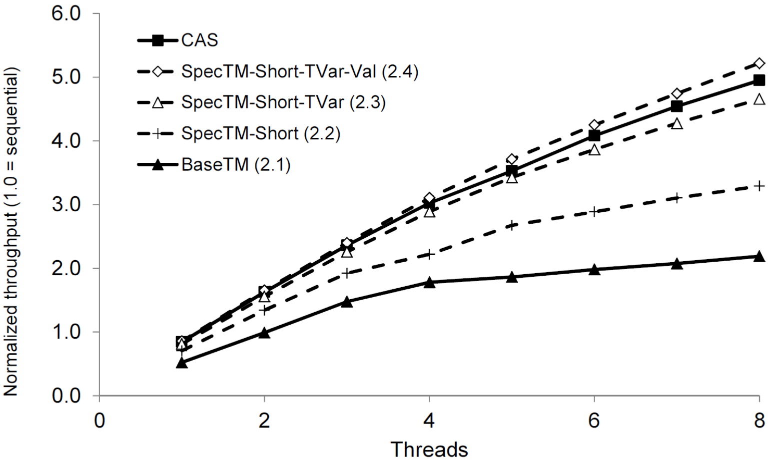

Figure 1: Throughput of operations on a hash table (90% lookups)

Traditional STM

```
void *Items[QUEUE_SIZE] = { NULL };
int LeftIdx = 0;
int RightIdx = 0;
void *PopLeft(void) {
  void *result = NULL;
  TX_RECORD t;
  do {
    Tx_Start(&t);
    int li = Tx_Read(&t, &LeftIdx);
    void *result = Tx_Read(&t, &Items[li]);
    if (result != NULL) {
      Tx_Write(&t, &(Items[li]), NULL);
      Tx_Write(&t, &LeftIdx, (li+1)%QUEUE_SIZE);
    }
  } while (!Tx_Commit(&t));
  return result;
}
```

Specializations

- Short transactions
- Explicit transactional variables
- Combined metadata with value-based validation

Short Transactions: Basic Idea

- Access only a small number of locations
- Indicate the sequence of operations
- Avoid write-to-read dependencies

Short Transactions: Code Example

```
void *Items[QUEUE_SIZE] = { NULL };
int LeftIdx = 0;
int RightIdx = 0;
void *PopLeft(void) {
  void *result = NULL;
  TX_RECORD t;
restart:
  int li = Tx_RW_R1(&t, &LeftIdx);
  void *result = Tx_RW_R2(&t, &Items[li]);
  if (!Tx_RW_2_Is_Valid(&t)) goto restart;
  if (result != NULL) {
    Tx_RW_2_Commit(&t, (li+1) % QUEUE_SIZE, NULL);
  } else {
    Tx_RW_2_Abort(&t);
  }
  return result;
}
```

Short Transactions: Continued

- Each access must be to a distinct memory location
- Processor state is not saved
- Writes are deferred until commit-time

UNIVERSITÄT SALZBURG

Short Transactions: API

```
typedef void *Ptr;

// Single read/write/CAS transactions:
Ptr Tx_Single_Read(Ptr *addr);
void Tx_Single_Write(Ptr *addr, Ptr newVal);
Ptr Tx_Single_CAS(Ptr *addr, Ptr oldVal, Ptr newVal);
```

Short Transactions: API

```
// Read-write short transactions:
Ptr Tx_RW_R1(TX_RECORD *t, Ptr *addr_1);
Ptr Tx_RW_R2(TX_RECORD *t, Ptr *addr_2);
...
bool Tx_RW_1_Is_Valid(TX_RECORD *t);
bool Tx_RW_2_Is_Valid(TX_RECORD *t);
...
void Tx_RW_1_Commit(TX_RECORD *t, Ptr val1);
void Tx_RW_2_Commit(TX_RECORD *t, Ptr val_1, Ptr val_2);
...
void Tx_RW_1_Abort(TX_RECORD *t);
void Tx_RW_2_Abort(TX_RECORD *t);
...
```

Short Transactions: API

```
// Read-only short transactions:
Ptr Tx_RO_R1(TX_RECORD *t, Ptr *addr_1);
Ptr Tx_RO_R2(TX_RECORD *t, Ptr *addr_2);
...
bool Tx_RO_1_Is_Valid(TX_RECORD *t);
bool Tx_RO_2_Is_Valid(TX_RECORD *t);
...
```

Short Transactions: API

```
// Commit combined read-only & read-write transactions:
bool Tx_RO_1_RW_1_Commit(TX_RECORD *t, Ptr val1);
bool Tx_RO_1_RW_2_Commit(TX_RECORD *t, Ptr val_1, Ptr val_2);
...
```

Short Transactions: API

```
// Read-only short transactions:
// Upgrade a location from RO to RW:
bool Tx_Upgrade_RO_1_To_RW_2(TX_RECORD *t);
...
```

Short Transactions: RO -> RW Upgrade

```
bool DCSS(void **a1, void **a2,
          void *o1, void *o2,
          void *n1) {
  TX_RECORD t;
restart:
  if (Tx_RO_R1(&t, a1) == o1 &&
      Tx_RO_R2(&t, a2) == o2 &&
      Tx_Upgrade_RO_1_To_RW_1(&t)) {
    if (Tx_RO_2_RW_1_Commit(&t, n1)) return true;
  } else if (Tx_RO_2_Is_Valid(&t)) return false;
  goto restart;
}
```

Short Transactions: Advantages

- No need for an update log
  - Values written are provided at commit-time
- Read-after-write checks are no longer necessary
- Accessed locations can be held in a fixed-size inline array

Table of Ownership Records

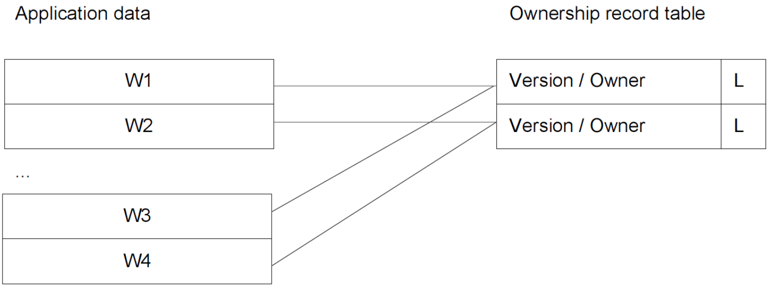

Figure 2: Meta-data held in a table of ownership records, indexed by a hash function

Explicit Transactional Variables

Figure 3: Meta-data co-located with application data in TVars

Combined Metadata with Value-Based Validation

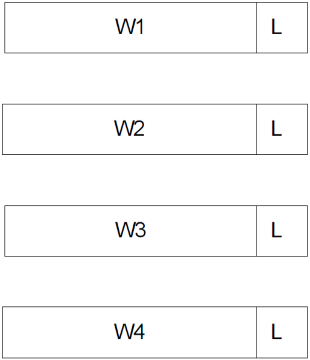

Figure 4: One lock-bit of meta-data held in each data item

UNIVERSITÄT SALZBURG

Value-Based Validation: Caution!

- Incorrect for the general case
- Special cases:
  - Read-Modify-Write transactions lock orecs before update
    - No version numbers needed
  - Mostly-read-write transactions (one read-only location)
    - RW locations are locked, RO location's value is compared
  - Locations satisfy a "non-re-use" property
    - The values are taking the place of version numbers

Skiplist

- Uses short transactions for levels 1 and 2
- Uses normal transactions for higher levels

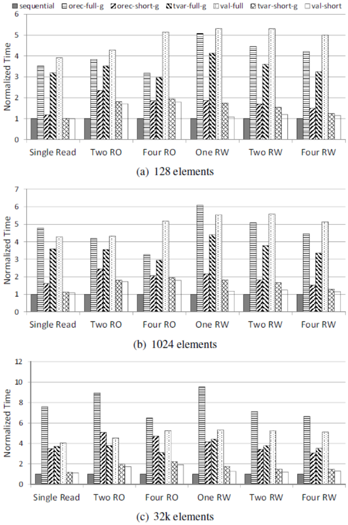

Figure 5: Single thread performance of SpecTM

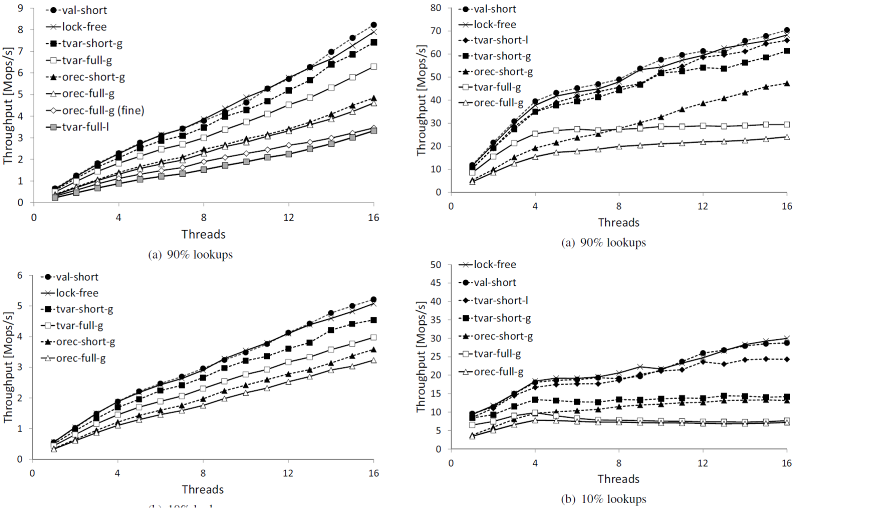

Figure 6: 16 cores

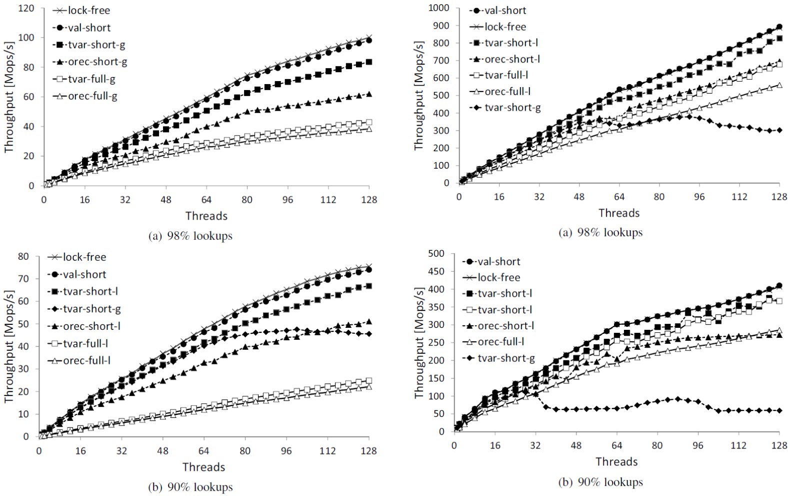

Figure 7: 128 Hardware Threads

UNIVERSITÄT SALZBURG

References I

Aleksandar Dragojević, Tim Harris
STM in the Small
Trading Generality for Performance in Software Transactional Memory
Proceedings of the 7th ACM european conference on Computer Systems (EuroSys '12). ACM, New York, NY, USA, 1–14., 2012.