On Relaxing Memory Consistency for Transactional Memory

Annette Bieniusa, Arie Middelkoop, Peter Thiemann
Counting transactions

```c
volatile int counter;

void threadWorker() {
    stm_begin();
    // complex computation

    // update counter
    int position =  stm_read(counter) + 1;
    stm_write(counter, position);

    printf("%i at %i",tid,position);

    stm_end();
}
```

0 at position 1.
1 at position 1.
4 at position 2.
3 at position 2.
9 at position 2.
2 at position 2.
5 at position 2.
8 at position 2.
7 at position 2.
...

Challenge

Semantics of memory access
  • inflexible isolation levels
  • serializability, opacity, etc.

Real-world requirements
  • performance and throughput
  • interaction with legacy code
  • integration of I/O
  • simple model
Twilight STM

- Separation of declarative transactional and imperative irrevocable phase
- Extension of the standard STM API
  - Inspection and repair of inconsistencies
  - Tags for grouping variables
  - Safe embedding of I/O and syscalls
Counter with Twilight

```c
void threadWorker() {
    stm_begin();
    // complex computation
    // update counter
    stm_enter_region(X);
    int pos = stm_read(counter) + 1;
    stm_write(counter, pos);
    stm_exit_region(X);

    bool success = stm_prepare();
    if (!success)
        stm_reload();
    if (stm_only_incons(X))
        pos = stm_reread(counter) + 1;
        stm_rewrite(counter,pos);
        else stm_restart();
    printf("%i finished at %i",tid,pos);
    stm_end();
}
```
Twilight STM

- Speculative execution
- Lock the write set
- Analyze the read set
- Twilight Zone
- Finalize writes and release the locks

Conflict

User abort
Features of Twilight STM

- Fine-grained conflict handling
  - “Repair” inconsistencies
  - Group variables into regions
- Concurrent irrevocable twilight zones
  - Transaction can always commit successfully
  - Transactions executing the twilight zone concurrently have disjoint write sets
- Rescue of “doomed” transactions
  - Prevent starvation!
- Twilight API is dead-lock free
Semantics of Twilight

- Formalization in terms of a monadic lambda calculus
  - Non-deterministic scheduling
  - Interleaving of transactions
  - Always consistent snapshots

- Analysis of effect traces
  - Only application-specific semantics provable in Twilight
Relaxing memory consistency

- Level of consistency should be adjustable
  - to resolve contention points
  - to cater to application-specific needs
- Semantics should still be simple
  - to verify correctness (formally and informally)
- Flexible API
  - stm_read and stm_write is not enough
  - Real-world scenarios require system calls
Alternatives

- Commit/Abort hooks
  - compositional
  - fixed order
  - only invoked after transaction's fate is determined
- Irrevocable transactions
  - reduces concurrency dramatically
- __tm_waiver
  - complex semantics
- View Transactions (→ next talk)
Our position

- “Abort on inconsistency” is too rigid
  - We need more flexibility and expressiveness,
  - but still require safety and progress guarantees
- Twilight API is rather simple and powerful
  - should be usable by advanced TM programmers