LibTM with RTM support

TM Group

Main Goals

- Incorporate Intel’s Restricted TM (RTM) into LibTM.
- Maintain unified LibTM interface to provide ease of programming and correctness.
- Performance improvements with lower overhead of hardware TM.
- Adaptive TM system to optimize resource utilization and inter-mixing of HW/SW transactions.

Intel TSX/RTM

- Execute critical code sections as transactions
- Relies on underlying cache coherence protocol to maintain strong isolation on its readset.
- Tracks at cache line (64 bytes) granularity.
- Aborts due to memory conflicts, I/O or system calls, interrupts, and etc.
- Software fallback path through other synchronization means (locking, STM, etc.)

Fast but inflexible due to fixed hardware

LibTM

- Developed in house to provide a simple interface for parallelization of applications.
- Tracks transactional reads/writes by overloading operators.
- Provides variety of conflict detection and conflict resolution policy choices.

Flexible but slow due to software TM overhead

On-going work with LibTM

- Inspector for dynamically choosing most suitable policies through profiling.
- Incorporating additional metrics (abort rate, consistency granularity, data pattern, etc.) to fine tune application/LibTM performance.
- Predicated commits to enforce transaction ordering and dependencies.

LibTM + RTM

- Maintain LibTM interface to allow simple and correct parallelization of applications.
- Use inspector to determine most suitable usage of RTM.
- Split or merge parallel regions for optimal RTM transaction commit size.
- Use predicated commits to enforce ordering and atomicity of split/merged transactions.
- Use SW transactions as fall back.

LibTM Interface

RTM:

```
HW_TXN_BEGIN {
  if (sw_locked) abort;
  else { //critical section
      //read / write operations
  }
  HW_TXN_END
}
if (abort) {  // fall back path
  sw_lock(); // global lock
  // critical section
  //read / write operations
  sw_unlock();
}
```

LibTM:

```
BEGIN_TRANSACTION()
  // critical section
  //read / write operations
END_TRANSACTION()
```

Locking checking in HW transaction

- RTM / HW transactions have strong isolation due to cache coherence protocol.
- Transactionally accessed data is added to the readset/writeset.
- Cache invalidation cause immediate abort of HW transaction.
- Software critical sections (transaction or locked region) don't have strong isolation. Data races/inconsistencies can happen due to HW transactions committing while executing a SW critical section.

Solution:

- Check for global SW lock inside HW transaction, abort if SW is executing. (Status: Working)

Ideas:

- Check for fine grained SW locks
- Atomic updated flag for a group of locks
- Use predicated commits to check commit order
- Use a software transaction to ensure atomicity

RTM Performance Study

- Used array access micro-benchmark (large int array, multiple threads access different parts of the array with generated continuous/random access patterns)

Varied parameters:

- # of threads
- write ratio
- transaction size (# of reads & writes inside 1 transaction)
- RTM transaction retry limit.

Goals:

- Determine optimal transaction size
- Determine abort rate vs. throughput
- Determine retry profile/distribution

Results

Upcoming Work

- Automatic privatization of shared data.
  - Targeting applications with large number of updates (moldyn).
- Automatic merging/splitting of transactions
  - Improve on static/dynamic transaction merging experimented by Intel.
  - Create software based write buffers to pool commits
  - Use predicated commits to enforce atomicity/ordering if required
- History based retry/abort tuning.
- Integration with LibTM inspector to incorporate additional parameters and data pattern profiling.