Evaluating Intel Hardware Transactional Memory in the Monitor

Abhishek Kumar

Mentor: Jim Moore

Manager: Giritharan Rashiyamani

© 2011 VMware Inc. All rights reserved

Outline

- Background
- Motivation
- Design/Implementation
- Results
- Conclusion/Recommendations
- Challenges
- Future Works
- Questions

vmware®

Background

Problem:

- Multi threaded application take advantage of increasing number of cores to achieve high performance.
- Major issue: Data sharing among multiple threads, as access to shared data requires synchronization mechanisms.
- Programmers try to limit synchronization overheads by either minimizing the use of synchronization or through the use of fine-grain locks, which is difficult and error prone and a missed or incorrect synchronization can cause applications to fail.
- Use of coarser granularity lock simplifies correctness problem but limits performance due to excessive serialization.

Solution:

- Intel Transactional Synchronization Extension (TSX) aims to improve the performance of lock-protected critical sections while maintaining the lock-based programming model.

vmware

Motivation

- Intel TSX allow the processor to determine dynamically whether threads needs to be serialize through lock-protected critical sections, and to perform serialization only when required.
- With lock elision, the hardware executes the programmer-specified critical sections transactionally. The lock variable is only read within the transactional region; it is not written to and acquired, thus exposing concurrency.

Intel TSX provides two software interfaces to developers.

1. Hardware Lock Elision (HLE) is an interface into TSX using instruction prefixes that allows easy migration of existing mutex-based code to use transactional synchronization (comprising the XACQUIRE and XRELEASE prefix).
2. Restricted Transactional Memory (RTM) is new instruction set interface (comprising the XBEGIN and XEND instructions).

- MX_Lock and MX_Unlock methods now use HLE prefixes on vmcore-main.
  HLE was chosen because it is backward compatible on non-TSX hosts.The assumptions are:

    - There is no performance impact on non-TSX capable hosts.
    - HLE will transparently allow VMM/VMX concurrency in some cases.
    - HLE will at worst do no harm on TSX capable hosts.

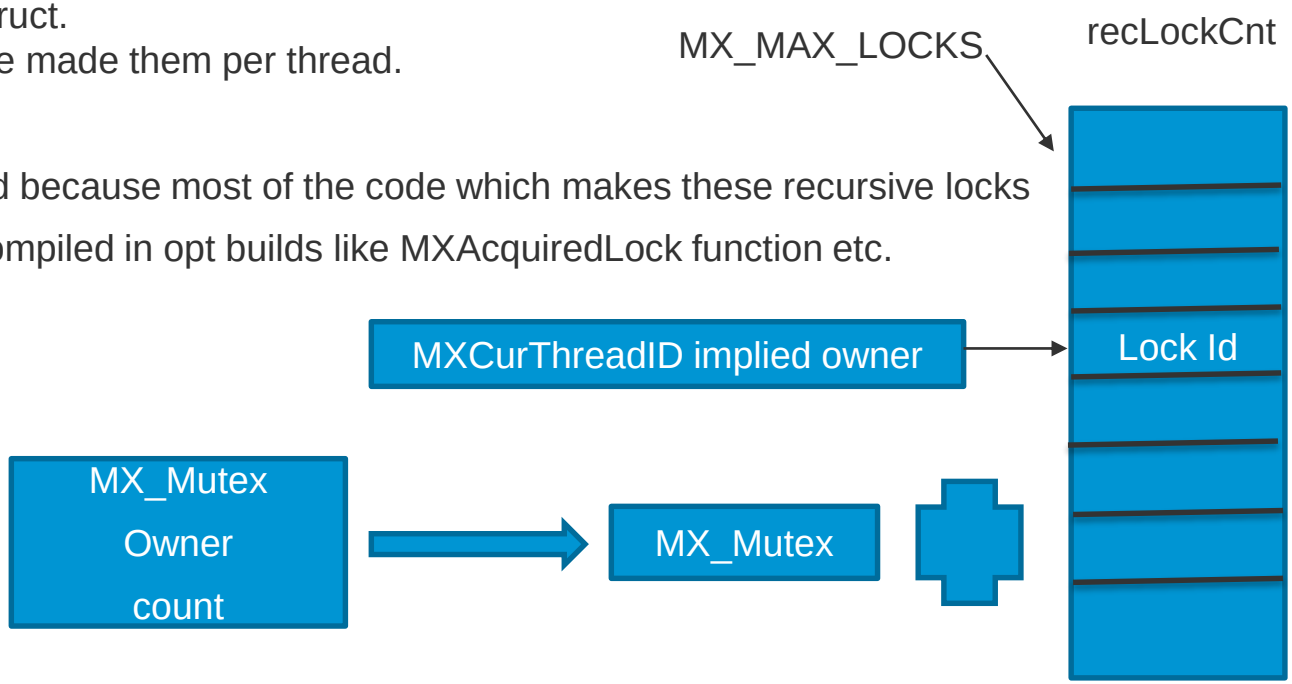
vmware

Design/Implementation

1. Assumption that HLE prefixes will transparently allow concurrency is not correct in many cases. The lock methods are not HLE friendly:
    1. Recursive locks never allow concurrency due to shared <owner, count> fields
    2. 29 Locks are recursive: busMemLock, physMemLock, vnptLock, userLevelLock, bigDeviceLock...
    3. Debug builds (FAT_LOCKS) adds owner field to non-recursive locks.

2. Now, as I mentioned recursive locks in monitor code are not HLE friendly, so for this project we have to make them HLE friendly, and for that we changed the way we used to keep the information of count and owner for MX_Mutexrec struct.
From a shared structure we made them per thread.

3. Lots of the work was saved because most of the code which makes these recursive locks HLE unfriendly was not compiled in opt builds like MXAcquiredLock function etc.



Design/Implementation

1. Changed 841842-tx to use new TSX pmc events.

2. Changed VMM and kstats.prl to supply lock data in opt build.

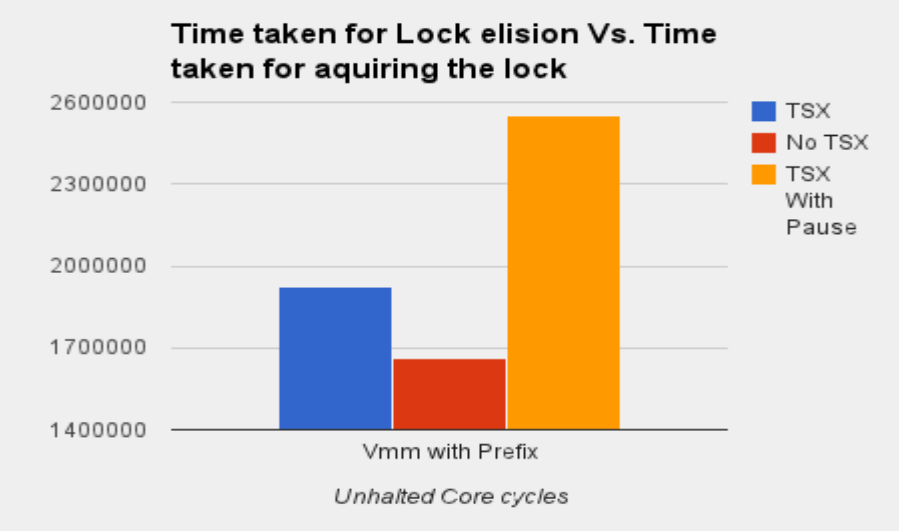
3. We also tried to see if two or more locks lie in the same cache line, which might cause aborts.
    1. vgaUserLock & vgaLock
    2. busMemLock & busMemLeafLock
    3. PhysMemLock & bigDeviceLock

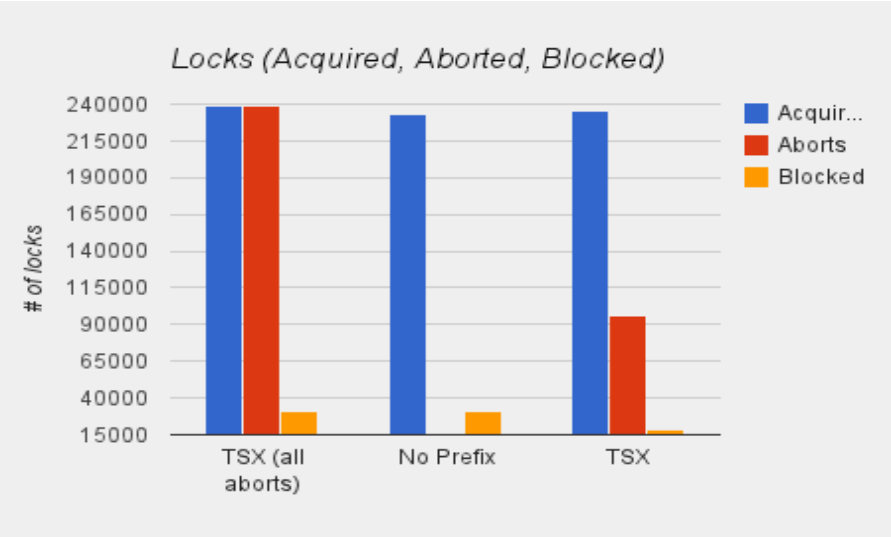
4. Made changes to existing Frobos test to put some load specifically on busmem lock and see the performance. Like: **1048957-toggleMMU, 514325-backdoor-hints.**

vmware

Results

- In this test we measure empty transactions Vs. Lock acquire & relase overhead **(841842-tx)** : Here we basically wanted to measure the penalty for performing lock elision.





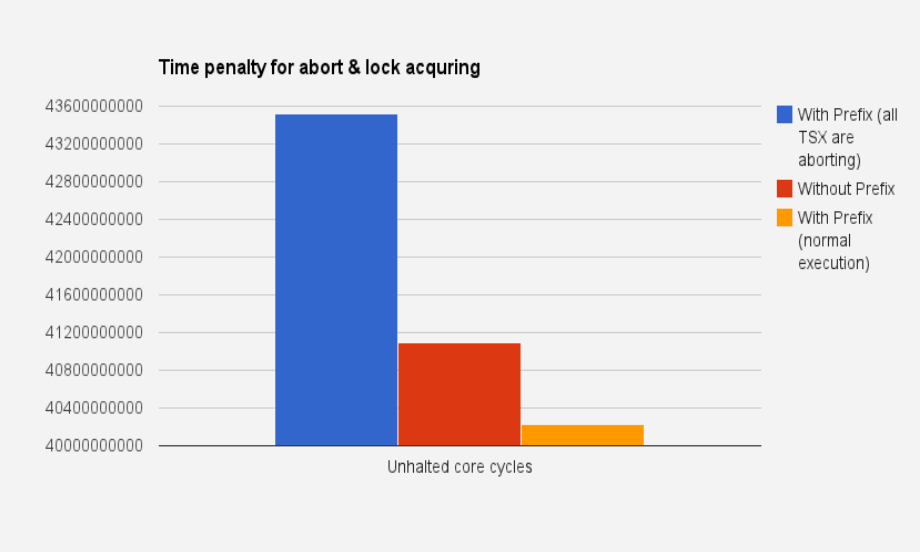
- **Frobos Test 1048957-toggleMMU:**
- In this test we wanted to see if we get any performance gain using HLE in presence of some concurrency

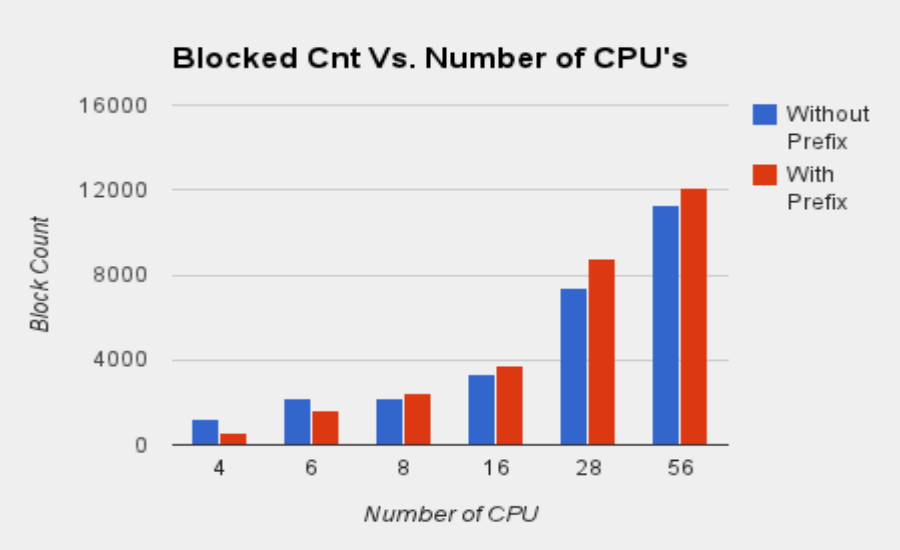
vmware

Results

- **Frobos Test 1048957-toggleMMU & 514325-backdoor-hints :**
- This is same as 2nd one. Here also we wanted to measure the performance gain using HLE.

NOTE: All these stats were collected by running frobos test on a 4 CPU machine.





- We can see that upto 8 cpu's we are getting some performance gain but after that we are seeing negative impact of HLE.
- On Running Boot and Halt test on workstation we did not observer any conclusive stats.

Conclusions/Recommendations

- **Conclusions:**
  - “No harm" on legacy/non TSX compatible CPUs.
  - Some penalty is incurred in using HLE. (so we have to gain that back from concurrency.)
  - Currently, most (at least half) of aborts in monitor code are due to unfriendly instructions.
  - Frequent aborts leads to performance degradation.
  - In case when we have concurrency, HLE does improve monitor performance. (Upto certain number of threads). But this is not very conclusive in lots of the cases whether its doing any good or not .
- **Recommendations:**
  - Without re-writing recursive lock methods, no benefit is possible.
  - Maybe we need opt-in or opt-out strategy on per-lock basis unless we figure out on per-lock basis whether its doing any good or not.
  - More control over lock placement to avoid false sharing..

vmware

Challenges

- Avoiding aborts due to locking and stat collection code changes.
- Measuring concurrency; balancing concurrency vs. abort costs
- Selecting Frobos test to put desired pressure on locks.
- Determine exact cause of aborts.
- Limited multi-threading as only 4 core hosts were available for most part of my internship.
- Now have access to 28 core HT host, no conclusive results as yet.

vmware®

Future work

- We can extend this experiment for RTM as well.
- And can do a comparative study of RTM Vs. HLE and can see which one suits better in our case.
- We can extend the current functionality to give more details about the aborted transactions like: getting instruction pointer where transaction failed. We can use PEBS framework for this.
- We can try Nesting. And see if some how we can leverage that.

vmware

Questions?

vmware®