



# Making the Most of Intel® Transactional Synchronisation Extensions

10 May 2016

Dr. Roman Dementiev

Software and Services Group, Intel

# Legal Disclaimer and Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

All products, platforms, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

Performance tests and ratings are measured using specific computer systems and/or components and reflect the approximate performance of Intel products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance. Buyers should consult other sources of information to evaluate the performance of systems or components they are considering purchasing. For more information on performance tests and on the performance of Intel products, visit Intel Performance Benchmark Limitations.

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. See [www.intel.com/products/processor\\_number](http://www.intel.com/products/processor_number) for details.

The Intel Core and Itanium processor families may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

The code names Arrandale, Bloomfield, Boazman, Boulder Creek, Calpella, Chief River, Clarkdale, Cliffside, Cougar Point, Gulftown, Huron River, Ivy Bridge, Kilmer Peak, King's Creek, Lewille, Lynnfield, Maho Bay, Montevina, Montevina Plus, Nehalem, Penryn, Puma Peak, Rainbow Peak, Sandy Bridge, Sugar Bay, Tylersburg, and Westmere presented in this document are only for use by Intel to identify a product, technology, or service in development, that has not been made commercially available to the public, i.e., announced, launched or shipped. It is not a "commercial" name for products or services and is not intended to function as a trademark.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or by visiting Intel's Web Site.

Intel, Intel Core, Core Inside, Itanium, and the Intel Logo are trademarks of Intel Corporation in the U.S. and other countries.

\*Other names and brands may be claimed as the property of others.

# Legal Disclaimer and Optimization Notice

## Pre-release Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright © , Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Core, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

## Optimization Notice

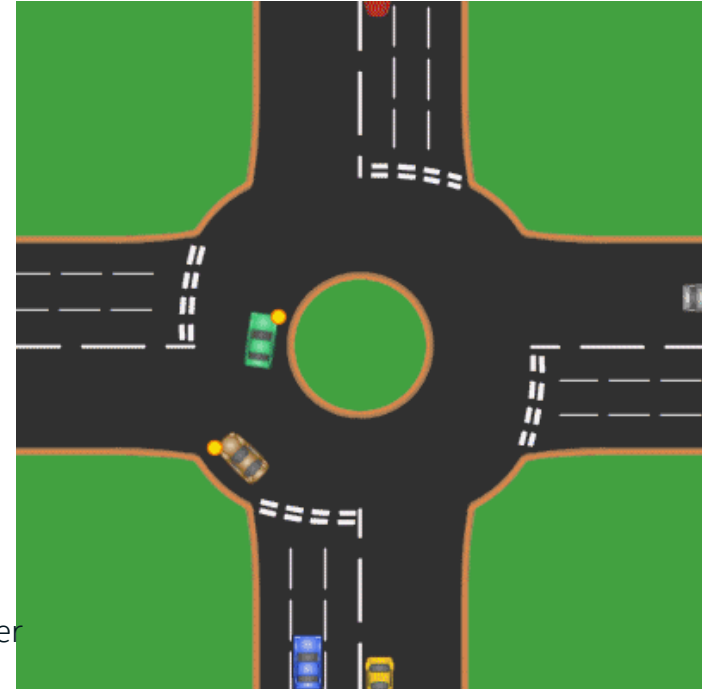
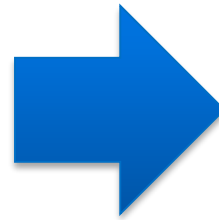
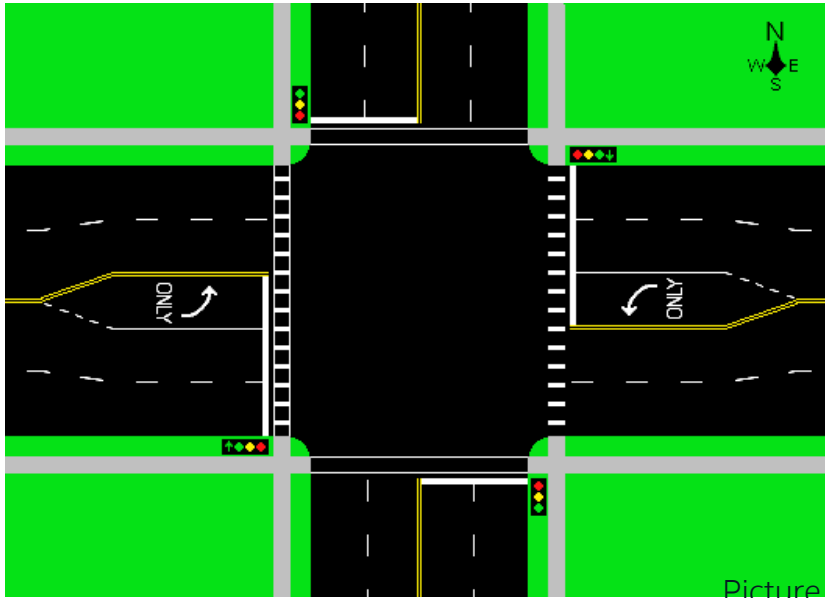
Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

# Agenda

- Intel® Transactional Synchronization Extensions (Intel® TSX)
- Enabling Intel TSX
- Intel TSX Optimization Workflow
- Monitoring and Profiling Intel TSX
- Root Causing Transaction Aborts
- Intel TSX Tuning Tips and Tricks
- Performance results
- Conclusion

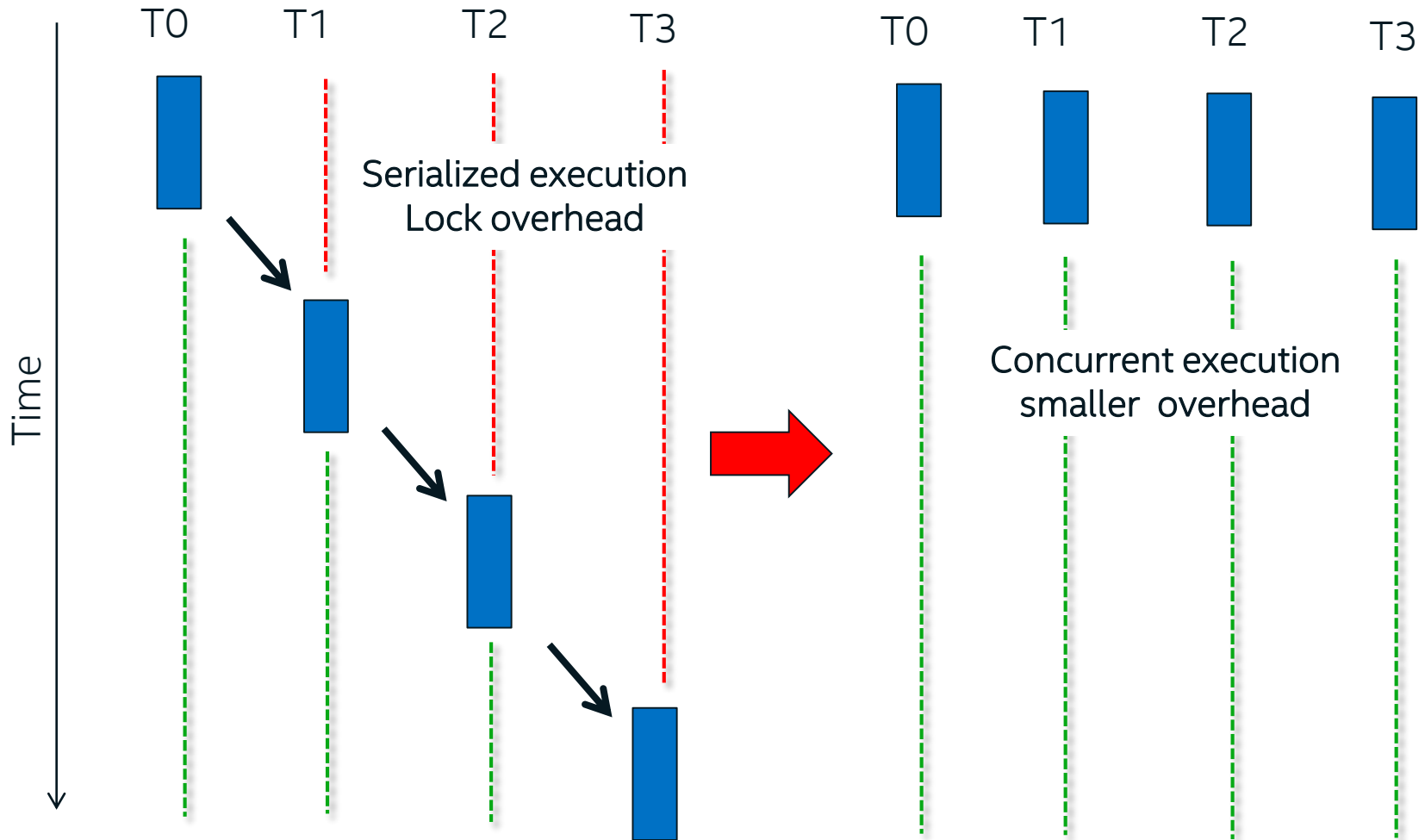
# Optimistic Non-blocking execution



Picture idea from Dave Butcher

*Serialize execution only when necessary*

# Transactional lock elision



*Expose Concurrency & Eliminate Unnecessary Overhead*

# Intel® Transactional Synchronization Extensions

Two extensions of instruction set

- Hardware Lock Elision (HLE) – elision hints for LOCK instructions
- Explicit „Restricted Transactional Memory (RTM) instructions

Usages:

- Lock libraries to implement elided mutexes/rwlocks (HLE/RTM)
- Acceleration of transactional memory programming model
  - gcc 4.8: `__transaction_atomic { c = a - b; }`

# RTM and HLE Hardware Implementation (Haswell, Broadwell, Skylake)

## Buffering memory writes

- Hardware uses L1 cache to buffer transactional writes
  - Writes not visible to other threads until after commit
  - Eviction of transactionally written line causes abort
- Buffering at cache line granularity

## Sufficient buffering for typical critical sections

- Cache associativity can occasionally be a limit
- Software (library) always provides fallback path in case of aborts

*Hardware Manages All Transactional Writes*



# RTM and HLE Hardware Implementation (Haswell, Broadwell, Skylake)

## Read and write addresses for conflict checking

- Tracked at cache line granularity using physical address
- L1 cache tracks addresses written to in transactional region
- L1 cache tracks addresses read from in transactional region
  - Additional implementation-specific probabilistic second level structure
  - Cache may evict address without loss of tracking

## Data conflicts

- Detected at cache line granularity
- Detected using cache coherence protocol (R/W snoops)
- Occurs if at least one request is doing a write (strong isolation)
- Abort when conflicting access detected (“eager” protocol)

*Hardware Automatically Detects Conflicting Accesses*

# RTM and HLE Hardware Implementation (Haswell, Broadwell, Skylake)

## Transactional abort

- Occurs when abort condition is detected
- Hardware discards all transactional updates

## Transactional commit

- Hardware makes transactional updates visible instantaneously
- No cross-thread/core/socket coordination required

More details in „Intel® 64 and IA-32 Architectures

Optimization Reference Manual” (chapter 12)

*No Global Communication for Commit and Abort*

# Agenda

- Intel TSX - Introduction
- Enabling Intel TSX
- Intel TSX Optimization Workflow
- Monitoring and Profiling Intel TSX
- Root Causing Transaction Aborts
- Intel TSX Tuning Tips and Tricks
- Conclusion

# Comparison to Fine-Grained Locking or Lock-Free

	Coarse grained lock elision	Fine-grained locking or lock-free
Development complexity	low	high (race conditions, dead-locks, limited data width of atomic instructions)
Maintainability	good (locking model well understood)	poor (experts required)
Overhead	low (only single Tx start/commit)	high (frequent long-latency atomic instructions)
Scalability	high	high

*Coarse grained lock elision: Scalable performance at coarse-grained development effort*

# Enabling TSX using a TSX Lock Library

\*

Intel® TBB 4.2 features *speculative\_spin\_mutex*

- HLE-based implementation of a speculative lock

RTM-based *speculative\_spin\_rw\_mutex*

- Allows both concurrent speculative reads and concurrent writes
- Allows non-speculative readers to proceed together with speculations

[http://www.threadingbuildingblocks.org/docs/help/reference/synchronization/mutexes/speculative\\_spin\\_rw\\_mutex\\_cls.htm](http://www.threadingbuildingblocks.org/docs/help/reference/synchronization/mutexes/speculative_spin_rw_mutex_cls.htm)

<http://software.intel.com/en-us/blogs/2014/03/07/transactional-memory-support-the-speculative-spin-rw-mutex-community-preview>

# Enabling TSX using a TSX Lock Library

Intel OpenMP (since Composer XE 2013 SP1)\*

- export KMP\_LOCK\_KIND=adaptive to enable RTM adaptive elision for all omp\_lock\_t
- [http://goparallel.sourceforge.net/wp-content/uploads/2014/03/TheParallelUniverse\\_Issue\\_17.pdf](http://goparallel.sourceforge.net/wp-content/uploads/2014/03/TheParallelUniverse_Issue_17.pdf)
- OpenMP 4.5 standard introduces (TSX) speculative hints

# Enabling TSX using a TSX Lock Library

## GLIBC PThreads 2.18

- configure with `--enable-lock-elision=yes`
  - SUSE SLES SP12 enables it by default
  - Other distros
- mutexes with `PTHREAD_MUTEX_DEFAULT` type are **adaptively** elided (RTM-based)
- PThreads 2.20: **rwlocks** can be elided too

# Enabling TSX in Java (JDK8u20 onwards )

-XX:+UseRTMLocking option on the Java command line

- JVM will automatically use RTM instructions for synchronized statements and methods
- Adaptive lock elision (disabled/re-enabled based on commit statistics)

Additional tunings:

<https://docs.oracle.com/javase/8/docs/technotes/tools/enhancements-8.html>

<https://bugs.openjdk.java.net/browse/JDK-8054376>

<https://bugs.openjdk.java.net/browse/JDK-8031320>

Performance evaluation: <http://transact2014.cse.lehigh.edu/yoo.pdf>



# Other lock libraries with TSX support

- Concurrency kit: spinlock elision with `ck_elide` wrappers
- Data plane development kit: „\_tm“ suffix to elide
  - Spin locks
  - Read-Write locks

# Agenda

- Intel TSX - Introduction
- Enabling Intel TSX
- Intel TSX Optimization Workflow
- Monitoring and Profiling Intel TSX
- Root Causing Transaction Aborts
- Intel TSX Tuning Tips and Tricks
- Conclusion

# When to optimize with TSX (lock elision)

Contention on lock but small contention on data protected

- Different cache lines modified (read-read is not a contention)

Not too large

- data must fit into Tx buffer
- not too long: timer/interrupts abort

Not too small

- Min 100s of cycles, avoid tight transactions (HSW client)

Don't do TSX-unfriendly operations that always abort

# Strategies to apply Intel TSX

**Elide all locks** -> analyze aborts -> optimize critical sections or disable elision for bad locks

*OR*

## Find good elision candidates

- Spin locks: Many cycles in lock functions
  - Use VTune hotspot analysis or Linux „perf record cycles“
- „Sleeping“ locks: wait&spin time from Locks&Waits VTune analysis

# The Workflow

1. Enable lock elision for (all/some) locks
2. Monitor TSX execution:
  - Too few cycles in transaction: check if right locks are elided: VTune locks&waits and mem\_uops\_retired.lock\_loads PEBS sampling. Goto 1.
  - Build abort reason distribution
3. Find sources of aborts (IP -> source code)
  - Synchronous aborts (unfriendly instructions, page fault, etc) : use a TSX profiler
  - Asynchronous aborts (conflicts, tx buffer overflow): only with a TSX emulator
4. Fix the aborts or disable elision. Goto 2

# Agenda

- Intel TSX - Introduction
- Enabling Intel TSX
- Intel TSX Optimization Workflow
- [Monitoring and Profiling Intel TSX](#)
- Root Causing Transaction Aborts
- Intel TSX Tuning Tips and Tricks
- Conclusion

# Monitoring Intel TSX Execution (transactional/aborted cycles)

```
# perf stat -T ./program
```

```
Performance counter stats for './program':
```

```
62.890098 task-clock           #    1.542 CPUs utilized
77874071 instructions          #    0.72  insns per cycle
108086139 cycles               #    1.719 GHz
68002201 raw 0x10000003c       #   62.91% transactional cycles
67779742 raw 0x30000003c       #    0.21% aborted cycles
 15050 raw 0x1c9               #    4518 cycles / transaction
     0 raw 0x1c8               #    0.000 K/sec
```

```
0.040780936 seconds time elapsed
```

Perf tool from  
linux kernel 3.13+

Windows/any Linux/  
FreeBSD/OSX:  
[Intel PCM-TSX](#) tool

```
# ./pcm-tsx.x ./program
```

```
Intel(r) Performance Counter Monitor: Intel(r) Transactional Synchronization Extensions Monitoring Utility
```

```
Executing "./program" command:
```

```
Time elapsed: 42 ms
```

Core	IPC	Instructions	Cycles	<u>Transactional Cycles</u>	<u>Aborted Cycles</u>	#RTM	#HLE	Cycles/Transaction
0	0.58	47 M	81 M	33 M (40.81%)	127 K ( 0.16%)	7239	0	4583
1	1.13	3278 K	2905 K	0 ( 0.00%)	0 ( 0.00%)	0	0	N/A
2	0.84	3831 K	4566 K	2659 K (58.24%)	1460 ( 0.03%)	576	0	4617
3	0.74	33 M	45 M	32 M (70.23%)	85 K ( 0.19%)	7233	0	4446
-----								
*	0.66	88 M	134 M	68 M (50.56%)	214 K ( 0.16%)	15 K	0	4519

# Success Metrics in Intel® VTune™ Amplifier XE

**Choose Analysis Type**

Analysis Type

- Algorithm Analysis
  - Basic Hotspots
  - Advanced Hotspots
  - Concurrency
  - Locks and Waits
- Microarchitecture Analysis
  - General Exploration
  - Bandwidth
- CPU Specific Analysis
  - Intel Core 2 Processor An
  - Nehalem / Westmere An
  - Sandy Bridge Analysis
  - Haswell Analysis
    - TSX Exploration**
    - TSX Hotspots

**TSX Exploration - Haswell**

Analyze Intel Transactional Synchronization Extensions (Intel TSX) usage. This analysis type is based on the hardware event-based sampling collection. Press F1 for more details.

Analyze user tasks

Select a step for analyzing TSX behavior. Start with measuring transactional success and then, if the aborts rate is high, analyze for aborts.

1. Transactional success

2. Aborts

Details

Events configured for CPU:

NOTE: For analysis purposes, Intel VTune Amplifier XE 2015 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the L

Event Name	Sample After	Event Description
------------	--------------	-------------------

**TSX Exploration** TSX Exploration viewpoint (change) ?

Analysis Target | Analysis Type | Collection Log | Summary | Bottom-up | Top-down Tree | Tasks and Frames

**Elapsed Time:** 7.315s

- [Clockticks:](#) 420,000,638,400
- [Transactional Cycles:](#) 32,200,048,944
- [Abort Cycles:](#) 10,400,015,808
- [Abort Cycles \(%\):](#) 32.298
- [Paused Time:](#) 0s



# Hardware events for abort (reason) statistics

# Intel TSX Perfmon Events (*why* transactions aborted?) (I)

Event name	Event code	Event Unit mask	Description
RTM/HLE_RETIRED.START	0xC9/0xC8	0x01	Number of times an RTM execution started.
RTM/HLE_RETIRED.COMMIT	0xC9/0xC8	0x02	Number of times an RTM execution successfully committed
RTM/HLE_RETIRED.ABORTED	0xC9/0xC8	0x04	Number of times an RTM execution aborted due to any reasons (multiple categories may count as one)
RTM/HLE_RETIRED.ABORTED_MISC1	0xC9/0xC8	0x08	Number of times an RTM execution aborted due to various memory events (conflict, overflow)
RTM/HLE_RETIRED.ABORTED_MISC2	0xC9/0xC8	0x10	Number of times an RTM execution aborted due to uncommon conditions (watchdog, etc)
RTM/HLE_RETIRED.ABORTED_MISC3	0xC9/0xC8	0x20	Number of times an RTM execution aborted due to RTM-unfriendly instructions
RTM/HLE_RETIRED.ABORTED_MISC4	0xC9/0xC8	0x40	Number of times an RTM execution aborted due to incompatible memory type (MMIO, page fault)
RTM/HLE_RETIRED.ABORTED_MISC5	0xC9/0xC8	0x80	Number of times an RTM execution aborted due to none of the previous 4 categories (e.g. interrupt)

# Intel TSX Perfmon Events (*why* transactions aborted?) (II)

Event name	Event code	Event Unit mask	Description
TX_MEM.ABORT_CONFLICT	0x54	0x01	Number of times a transactional abort was signaled due to a data conflict on a transactionally accessed address
TX_MEM.ABORT_CAPACITY_WRITE	0x54	0x02	Number of times a transactional abort was signaled due to limited resources for transactional stores
TX_MEM.ABORT_HLE_STORE_TO_ELIDED_LOCK	0x54	0x04	Number of times a HLE transactional region aborted due to a non XRELEASE prefixed instruction writing to an elided lock in the elision buffer
TX_MEM.ABORT_HLE_ELISION_BUFFER_NOT_EMPTY	0x54	0x08	Number of times an HLE transactional execution aborted due to NoAllocatedElisionBuffer being nonzero.
TX_MEM.ABORT_HLE_ELISION_BUFFER_MISMATCH	0x54	0x10	Number of times an HLE transactional execution aborted due to XRELEASE lock not satisfying the address and value requirements in the elision buffer.
TX_MEM.ABORT_HLE_ELISION_BUFFER_UNSUPPORTED_ALIGNMENT	0x54	0x20	Number of times an HLE transactional execution aborted due to an unsupported read alignment from the elision buffer.
TX_MEM.ABORT_HLE_ELISION_BUFFER_FULL	0x54	0x40	Number of times HLE lock could not be elided due to Elision Buffer Available being zero.

ABORT\_CAPACITY = RTM/HLE\_RETIRED.ABORTED\_MISC1 - TX\_MEM.ABORT\_CONFLICT

# Intel TSX Perfmon Events (*why* transactions aborted?) (III)

Event name	Event code	Event Unit mask	Description
TX_EXEC.MISC1	0x5D	0x01	Counts the number of times a class of instructions that may cause a transactional abort was executed. Since this is the count of execution it may not always cause a transactional abort.
TX_EXEC.MISC2	0x5D	0x02	Counts the number of times a class of instructions that may cause a transactional abort was executed inside a transactional region
TX_EXEC.MISC3	0x5D	0x04	Counts the number of times an instruction execution caused the nest count supported to be exceeded
TX_EXEC.MISC4	0x5D	0x08	Counts the number of times an HLE XACQUIRE instruction was executed inside an RTM transactional region

# Abort Reason Distribution

## Use PMU counting mode with TSX events

Linux Perf 3.13: <http://software.intel.com/en-us/blogs/2013/05/03/intel-transactional-synchronization-extensions-intel-tsx-profiling-with-linux-0>

### RTM:

```
perf stat -e '{tx-abort-count,r8c9,r154,r20c9}' program
pcm-tsx program -e RTM_RETIRED.ABORTED
    -e RTM_RETIRED.ABORTED_MISC1 -e TX_MEM.ABORT_CONFLICT
    -e RTM_RETIRED.ABORTED_MISC3
```

RTM -> HLE: **tx** -> **e1**; **c9**->**c8** ;

#conflicts: TX\_MEM.ABORT\_CONFLICT

#Tx buf overflow: \*\_RETIRED.ABORTED\_MISC1 - TX\_MEM.ABORT\_CONFLICT

#unfriendly instr: \*\_RETIRED.ABORTED\_MISC3

#other: \*\_RETIRED.ABORTED - \*\_RETIRED.ABORTED\_MISC1  
- \*\_RETIRED.ABORTED\_MISC3

Use additional TSX events to complete the picture

# Linux perf stat with TSX events

```
perf stat -e r4c9 -e r8c9 -e r154 -e r20c9 ./matrix_tsx
```

```
Performance counter stats for './matrix_tsx':
```

```
14184836 raw 0x4c9
```

```
14171948 raw 0x8c9
```

```
14395057 raw 0x154
```

```
0 raw 0x20c9
```

```
9.410623956 seconds time elapsed
```

Here most aborts are conflicts: 0x4c9  $\approx$  0x154 (-+ overcounting)

# PCM-TSX with TSX events

```
pcm-tsx.x ./matrix_tsx -e RTM_RETIRED.ABORTED -e RTM_RETIRED.ABORTED_MISC1 -e TX_MEM.ABORT_CONFLICT -  
e RTM_RETIRED.ABORTED_MISC3
```

Intel(r) Performance Counter Monitor: Intel(r) Transactional Synchronization Extensions Monitoring Utility

Executing "./matrix\_tsx" command:

Time elapsed: 9549 ms

**Event0: RTM\_RETIRED.ABORTED** Number of times an RTM execution aborted due to any reasons (multiple categories may count as one) (raw 0x4c9)

Event1: RTM\_RETIRED.ABORTED\_MISC1 Number of times an RTM execution aborted due to various memory events (raw 0x8c9)

**Event2: TX\_MEM.ABORT\_CONFLICT** Number of times a transactional abort was signalled due to a data conflict on a transactionally accessed address (raw 0x154)

Event3: RTM\_RETIRED.ABORTED\_MISC3 Number of times an RTM execution aborted due to HLE-unfriendly instructions (raw 0x20c9)

Core	Event0	Event1	Event2	Event3
0	8707 K	8701 K	8810 K	0
1	0	0	0	0
2	1	0	0	0
3	8247 K	8242 K	9231 K	0
-----				
*	16 M	16 M	18 M	0

# TSX Exploration in Intel® VTune™ Amplifier XE

**Choose Analysis Type**

Analysis Type

- Algorithm Analysis
  - Basic Hotspots
  - Advanced Hotspots
  - Concurrency
  - Locks and Waits
- Microarchitecture Analysis
  - General Exploration
  - Bandwidth
- CPU Specific Analysis
  - Intel Core 2 Processor An
  - Nehalem / Westmere An
  - Sandy Bridge Analysis
  - Haswell Analysis
    - TSX Exploration**
    - TSX Hotspots
- Platform Analysis
  - CPU/GPU Concurrency
  - Knights Corner Platform Analy

**TSX Exploration - Haswell**

Analyze Intel Transactional Synchronization Extensions (Intel TSX) usage. This analysis type is based on the hardware event-based sampling collection. Press F1 for more details.

Analyze user tasks

Select a step for analyzing TSX behavior. Start with measuring transactional success and then, if the aborts rate is high, analyze for aborts.

1. Transactional success

2. Aborts

Details

Events configured for CPU:

NOTE: For analysis purposes, Intel VTune Amplifier XE 2015 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on the value of the D...

Event Name	Sample After	Event Description
HLE_RETIRED.ABORTED_PS	10000	Number of times an HLE execution aborted due to any reasons (multiple categories may count as one).
RTM_RETIRED.ABORTED_PS	10000	Number of times an RTM execution aborted due to any reasons (multiple categories may count as one).

Note that the result will be empty if #aborts < „Sample After“

Decrease the „Sample After“ if the workload has only a few aborts



# TSX Exploration in Intel® VTune™ Amplifier XE

Instruction sync to ex

not precise

precise

Address	Sou.. Line	Assembly	TSX Aborts: Total				TSX Aborts: Self			
			Instruction	Data Conf...	Capacity	Other	Instruction	Data Conf...	Cap..	Oth..
0x401212		lea 0x602580(%rbp), %rdi	0.0%	0.0%	0.0%	0.0%				
0x401219		sub \$0x1, %eax	0.0%	0.0%	0.0%	0.0%				
0x40121c		test %eax, %eax	0.0%	0.0%	0.0%	0.0%				
0x40121e		movl %eax, 0x602580(%rbp)	0.0%	0.0%	0.0%	0.0%				
0x401224		<a href="#">jnz 0x4011a8 &lt;Block 34&gt;</a>	0.0%	0.0%	0.0%	0.0%				
<b>0x401226</b>		<b>Block 45:</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>				
0x401226		movl \$0x0, 0x201310(%rip)	0.0%	0.0%	0.0%	0.0%				
0x401230		movl \$0x0, 0x4(%rdi)	0.0%	0.0%	0.0%	0.0%				
0x401237		<a href="#">jmp 0x4011a8 &lt;Block 34&gt;</a>	0.0%	0.0%	0.0%	0.0%				
<b>0x40123c</b>		<b>Block 46:</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>				
0x40123c		movl 0x2012fe(%rip), %eax	0.0%	15.6%	0.0%	0.0%	0	110,000,165	0	0
0x401242		test %eax, %eax	0.0%	0.0%	0.0%	0.0%				
0x401244		<a href="#">jle 0x401189 &lt;Block 31&gt;</a>	0.0%	0.0%	0.0%	0.0%				
<b>0x40124a</b>		<b>Block 47:</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>				
0x40124a		xabort \$0x1	11.4%	0.0%	0.0%	0.0%	80,000,120	0	0	0
<b>0x40124d</b>		<b>Block 48:</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>	<b>0.0%</b>				
0x40124d		<a href="#">jmp 0x401189 &lt;Block 31&gt;</a>	0.0%	0.0%	0.0%	0.0%				
Selected 1 row(s):			11.4%	0.0%	0.0%	0.0%	80,000,120	0	0	0

# Profiling Cycles Inside Transactions

Traditional profiling associate all cycles inside transactions to transaction begin instruction

Haswell adds hardware capability to show cycles correctly

Use profilers that support this new capability

# Profiling Cycles Inside Transactions

Linux perf 3.13+: `perf record -e cycles:pp`

Intel® VTune™ Amplifier XE:

The screenshot shows the Intel VTune Amplifier XE interface. The left sidebar displays a tree view of analysis types, with 'TSX Hotspots' selected under 'Haswell Analysis'. The main panel is titled 'TSX Hotspots - Haswell' and contains the following text: 'Analyze hotspots inside transactions for systems with the Intel Transactional Synchronization Extensions (Intel TSX) feature enabled. Press F1 for more details.' Below this, there are two unchecked checkboxes: 'Collect stacks' and 'Analyze user tasks'. A 'Details' section is expanded, showing a text box with the text 'Events configured for CPU:'. Below this, a note states: 'NOTE: For analysis purposes, Intel VTune Amplifier XE 2015 may adjust the Sample After values in the table below by a multiplier. The multiplier depends on t'. At the bottom of the details section, there is a table header with three columns: 'Event Name', 'Sample After', and 'Event Description'.

Event Name	Sample After	Event Description
------------	--------------	-------------------

# Agenda

- Intel TSX - Introduction
- Enabling Intel TSX
- Intel TSX Optimization Workflow
- Monitoring Intel TSX
- [Root Causing Transaction Aborts](#)
- Intel TSX Tuning Tips and Tricks
- Conclusion

# Find Sources of Sync Aborts

## Need a TSX enabled profiler

- Linux kernel 3.13 perf
- or Intel® Vtune™ Amplifier XE 2013 (TSX Exploration)

Collect abort sources with PEBS abort sampling:

```
RTM: perf record -g --transaction --weight -e tx-aborts program
```

```
HLE: perf record -g --transaction --weight -e el-aborts program
```

Show abort sources (weight=aborted cycles):

```
perf report --sort symbol,dso,weight,transaction
```

# Find Sources of Async Aborts

Arrival of an abort „event“ is delayed/asynchronous

- tx buf overflow (loosing track of tx access) or conflict
- Perfmon/profiler can not tell killerIP

But: TSX emulation

- Big overhead, some aborts cannot be emulated (faults)
- But if many async aborts -> must show up in emulation too (reasoning statistically)

Intel Software Development Emulator

<http://software.intel.com/en-us/articles/intel-software-development-emulator>

<http://software.intel.com/en-us/blogs/2012/11/06/exploring-intel-transactional-synchronization-extensions-with-intel-software>

# Intel SDE TSX Emulator

Options: `-hsw -hle_enabled 1 -rtm-mode full -tsx_stats 1 -tsx_stats_call_stack 1`

- Tx killer {instruction, IP, source code file and line, stack}
- Top abort sources by type
- Many other abort statistics

# Agenda

- Intel TSX - Introduction
- Enabling Intel TSX
- Intel TSX Optimization Workflow
- Monitoring and Profiling Intel TSX
- Root Causing Transaction Aborts
- Intel TSX Tuning Tips and Tricks
- Conclusion



# Poor TSX lock Implementation

## Anti-patterns:

- Retrying forever
- Not putting lock into read-set
- „Lemming effect“ = Persistent non-speculative execution
- Avoid unsupported lock elision patterns (same value on lock word restore, lock word size/address must match)
- Avoid unaligned accesses or partially overlapping accesses to the lock word
- Avoid writes to the lock word without XRELEASE instruction

Consult „[TSX anti-patterns](#)“ article or use a proven TSX lock from a standard library (TBB, PThreads, OpenMP,etc)

# Reduce Generic Conflict Aborts

## Avoid false-sharing

- Padding, data structure re-org helps

## Avoid true-sharing

- Avoid global statistics and accounting: use sampling, per-thread stats, remove not critical stats, reduce windows of conflict (move bad accesses towards the tx end), use XTEST instruction

## Sharing in memory allocators

- Use thread-friendly allocators (TBB, Google tcmalloc)

## Silent stores: use conditional writes

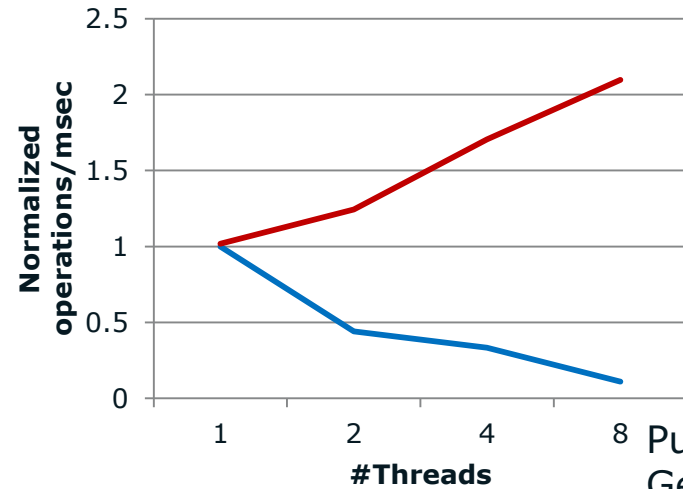
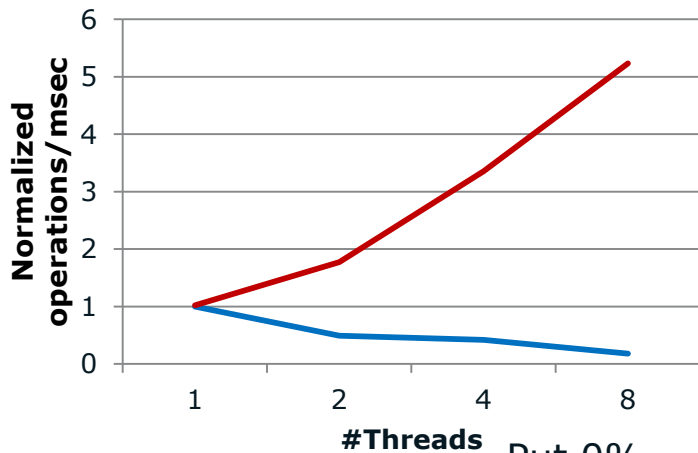
# Reduce Capacity/Overflow Aborts

- First time initialization: skip elision first time
- Check if particular cache sets are hot: use SDE emulator to log memory accesses inside transactions
- Change algorithm to touch less memory
- If cannot reduce the footprint: transit to fall-back early

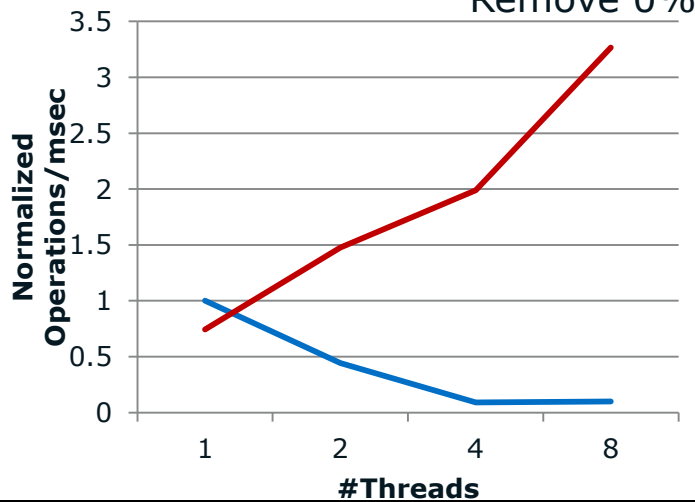
# Performance Studies

# Performance: Java Hashtable/HashMap

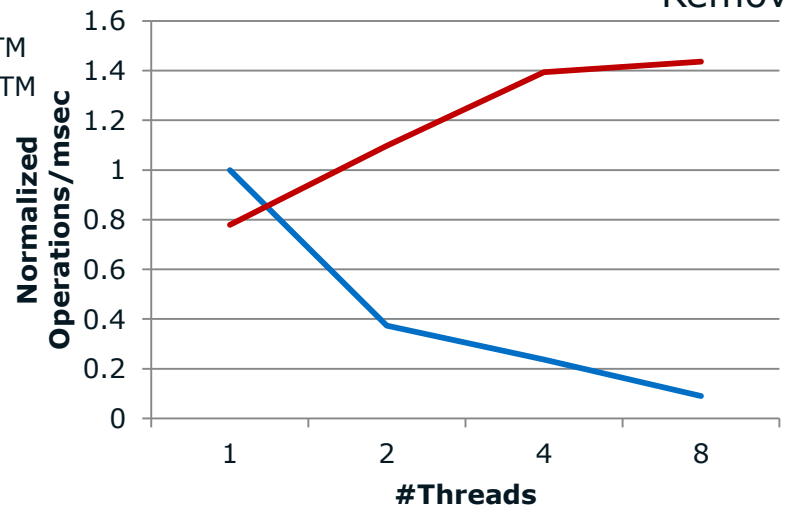
- Hashtable



- HashMap

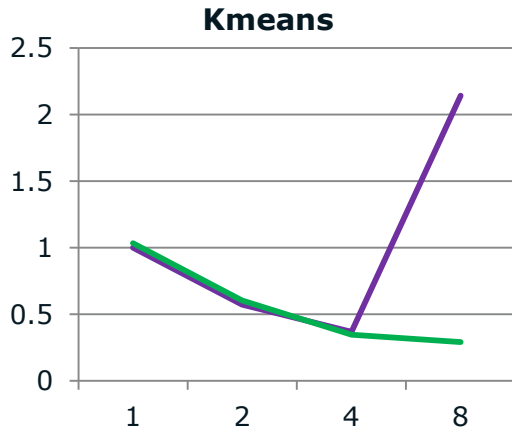
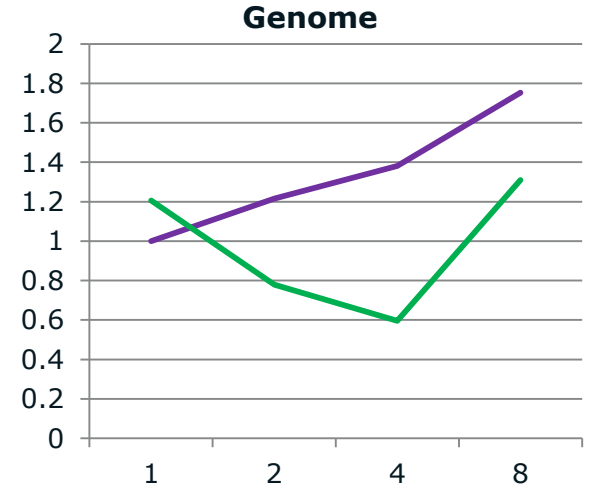
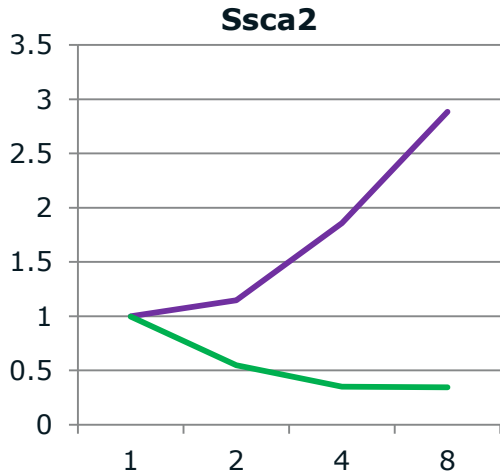


— :-UseRTM  
— :+UseRTM



# Performance: Java JSTAMP

<http://transact2014.cse.lehigh.edu/yoo.pdf>



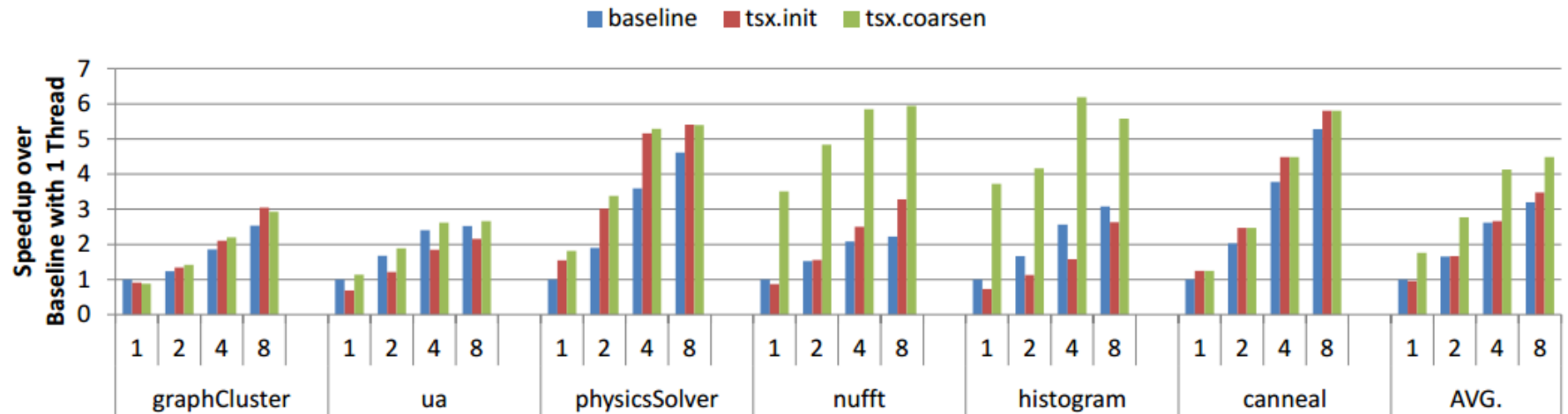
— -UseRTM  
— +UseRTM

X-Axis: #Threads  
Y-Axis: Normalized execution time

# Performance Evaluation of Intel R Transactional Synchronization Extensions for HPC

<http://pcl.intel-research.net/publications/SC13-TSX.pdf>

Workload	Description	Threading	Sync
graphCluster	Performs min-cut graph clustering. Kernel 4 of SSCA2 [1].	OpenMP	locks
ua	Unstructured Adaptive (UA) from NAS Parallel Benchmarks suite [9]. Solves a set of heat equations on an adaptive mesh.	OpenMP	atomics
physicsSolver	Uses PSOR to solve a set of 3-D force constraints on groups of blocks.	PThread	locks
nufft	Non-uniform FFT. Baseline reported in [15].	OpenMP	locks
histogram	Parallel image histogram construction.	PThread	atomics
canneal	VLSI router from PARSEC [2]. Performs simulated annealing.	PThread	lock-free



# TSX in Database studies

- HPCA 2014 **Improving In-Memory Database Index Performance with Intel® Transactional Synchronization Extensions** - Tomas Karnagel, Roman Dementiev, Ravi Rajwar, Konrad Lai, Thomas Legler, Benjamin Schlegel, Wolfgang Lehner (Intel, SAP AG and TU Dresden)
- EuroSys 2014 **Using Restricted Transactional Memory to Build a Scalable In-Memory Database.** - Zhaoguo Wang, Hao Qian, Jinyang Li, Haibo Chen (Fudan University, Shanghai Jiao Tong University, New York University)
- TDKE 2015 **Scaling HTM-Supported Database Transactions to Many Cores** - Viktor Leis, Alfons Kemper, Thomas Neumann (TU Munchen)



# Intel TSX in a Commercial Database Product

Up to 2x performance boost in transactional processing when running SAP HANA database with TSX enabled

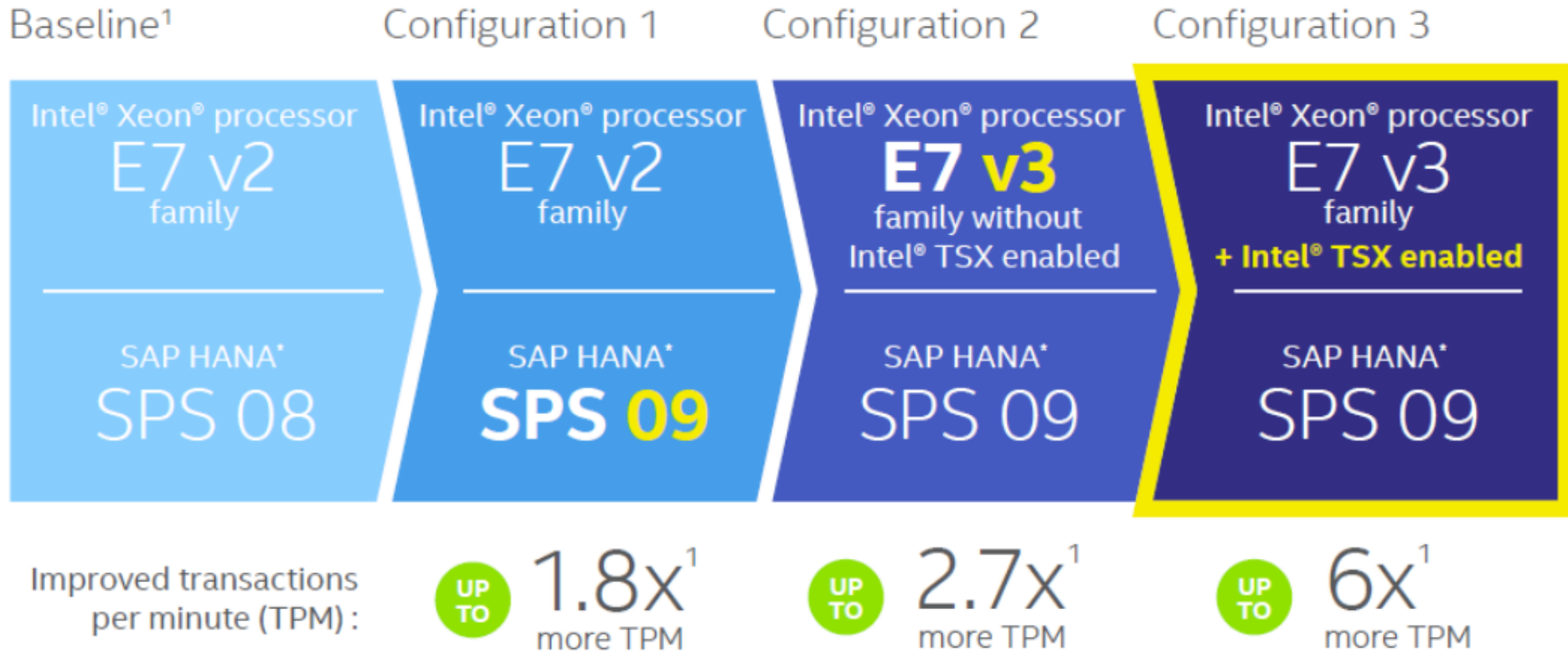
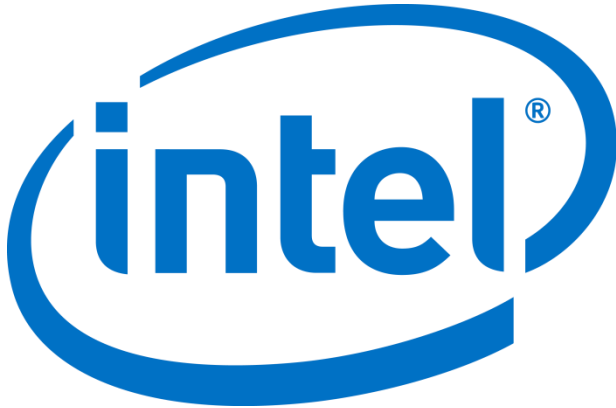


Figure 1. Upgrading to the Intel® Xeon® processor E7 v3 family and SAP HANA® SPS 09 (S-OLTP stress test lab results) provides incremental performance gains.<sup>1</sup>

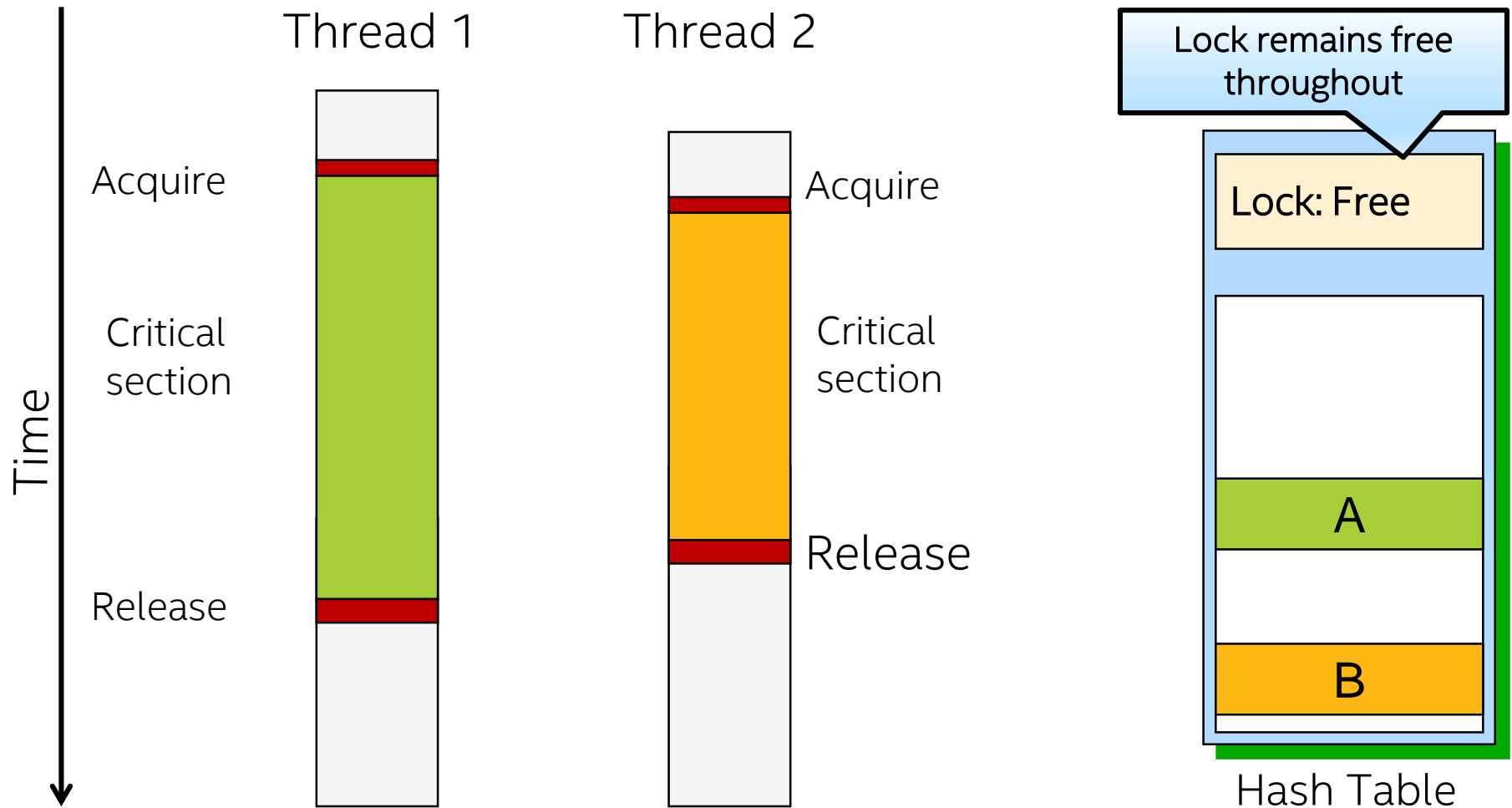
<http://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/sap-hana-real-time-analytics-solution-brief.pdf>

# Conclusion

- Intel TSX with lock elision is a simple technique to efficiently resolve locking and synchronization problems
- Tools for Intel TSX optimization are available
- Numerous studies show performance benefits
- Intel TSX tech resources + **performance studies:**  
[www.intel.com/software/tsx](http://www.intel.com/software/tsx)



# Transactional Lock Elision



*No Serialization and No Communication if No Data Conflicts*

# HLE/RTM Overview

**HLE is a hint inserted in front of a LOCK operation to indicate a region is a candidate for lock elision**

- XACQUIRE (0xF2) and XRELEASE (0xF3) prefixes
- Don't actually acquire lock, but execute region speculatively
- Hardware buffers loads and stores, checkpoints registers
- Hardware attempts to commit atomically without locks
- If cannot do lock-free, restart and execute non-speculatively

**RTM is three new instructions (XBEGIN, XEND, XABORT)**

- Similar operation as HLE (except no locks, new ISA)
- If cannot commit atomically, go to handler indicated by XBEGIN
- Provides software additional functionality over HLE

**XTEST is a new instruction to determine if in HLE or RTM**

- Can be used inside both HLE and RTM
- Allows SW to query execution status of HLE and RTM

# Enabling HLE in Custom locks

- Use intrinsics or direct machine code (compiler support needed) - see slides in backup for details
- Using HLE and RTM with older compilers with tsx-tools:  
<http://software.intel.com/en-us/blogs/2013/05/20/using-hle-and-rtm-with-older-compilers-with-tsx-tools> (BSD licensed)

# HLE Lock in Microsoft VC++

```
#include <intrin.h> /* For _mm_pause() */
#include <immintrin.h> /* For HLE intrinsics */
/* Lock initialized with 0 initially */
void acquire_lock(int *mutex)
{
    while (_InterlockedCompareExchange_HLEAcquire(&mutex, 1, 0) != 0) {
        /* Wait for lock to become free again before retrying speculation */
        do {
            _mm_pause(); /* Abort speculation */
            /* prevent compiler instruction reordering and wait-loop skipping,
               no additional fence instructions are generated on IA */
            _ReadWriteBarrier();
        } while (mutex == 1);
    }
}

void release_lock(int *mutex)
{
    _Store_HLERelease (mutex, 0); // was mutex = 0; in non-HLE version
}
```

# RTM Intrinsics

gcc 4.8 (-mrtm flag), Microsoft Visual Studio 2012, Intel® C++ Compiler XE 13.0

`_xbegin()`: attempt transaction. Returns status

- `_XBEGIN_STARTED` (-1): in transaction
- Otherwise abort code with bits:

<code>_XABORT_EXPLICIT</code>	Abort caused by <code>_xabort()</code> . <code>_XABORT_CODE(status)</code> contains the value passed to <code>_xabort()</code>
<code>_XABORT_RETRY</code>	When this bit is set retrying the transaction has a chance to commit. If not set retrying will likely not succeed.
<code>_XABORT_CONFLICT</code>	Another thread / processor conflicted with a memory address that was part of this thread's transaction.
<code>_XABORT_CAPACITY</code>	The abort is related to a capacity overflow
<code>_XABORT_DEBUG</code>	The abort happened due to a debug trap
<code>_XABORT_NESTED</code>	The abort happened in a nested transaction

`_xend()`: commit transaction

`_xtest()`: true if in transaction

`_xabort(constant)`: aborts current transaction



# Enabling RTM with a Lock Wrapper (basic, no retries, etc)

```
void elided_lock_wrapper(lock) {
    if ( __xbegin() == __XBEGIN_STARTED ) {          /* Start transaction */
        if (lock is free)          /* Check lock and put into read-set */
            return;                /* Execute lock region in transaction */

        __xabort(0xff);           /* Abort transaction as lock is busy */
    }                               /* Abort comes here */
    // abort handler: optionally analyze abort flags, retry with xbegin...
    take fallback lock
}

void elided_unlock_wrapper(lock) {
    if (lock is free)
        __xend();                /* Commit transaction */
    else
        unlock lock;
}
```

With retries, abort status bits analysis, etc: <http://software.intel.com/en-us/blogs/2012/11/06/exploring-intel-transactional-synchronization-extensions-with-intel-software>

# HLE in Linux (Intel® C++ Compiler and gcc)

## Inline asm:

```
#define XACQUIRE ".byte 0xf2; "  
#define XRELEASE ".byte 0xf3; "  
static inline int hle_acquire_xchg(int *lock, int val)  
{  
    asm volatile(XACQUIRE "xchg %0,%1" : "+r" (val), "+m" (*lock) :: "memory");  
    return val;  
}  
static void hle_release_store(int *lock, int val)  
{  
    asm volatile(XRELEASE "mov %0,%1" : "r" (val), "+m" (*lock) :: "memory");  
}
```

# HLE Intrinsics in gcc 4.8

Caution: needs `-O2` or higher opt (compiler bug)

`__ATOMIC_HLE_ACQUIRE` and `__ATOMIC_HLE_RELEASE`  
flags for

`__atomic_compare_exchange_n`,

`__atomic_store`,

`__atomic_clear`, etc

intrinsics

# Other Compilers with HLE support

**C++11 <atomic> in gcc 4.8:** extended memory models:

`__memory_order_hle_acquire` and  
`__memory_order_hle_release`

- i.e. for `atomic_flag::test_and_test(..)`, `clear(..)`, etc

## Windows: Intel and Microsoft compilers

- `_InterlockedCompareExchange{,64,Pointer}_HLE{Acquire, Release}`
- `_InterlockedExchangeAdd_HLE{Acquire, Release}`
- `_Store{,64,Pointer}_HLERelease`

# Reduce Conflict Aborts in Lock Elision

„lemming effect“ (persistent non-spec execution)

## Lock word is in the read-set

- Lock Elision specific (both with HLE and RTM)
- Non-spec lock acquisition -> conflict on lock word
  - Avoid „lemming effect“ (persistent non-spec execution)
- Solution: wait outside transaction (non-HLE path or abort handler) for fall-back lock to be free and only then respeculate/retry

See also <http://software.intel.com/en-us/articles/tsx-anti-patterns-in-lock-elision-code>

# (Single-Threaded) Transaction Overheads

The overhead is amortized in larger critical sections but will be exposed in very small critical sections.

One simple approach to reduce perceived overhead is to perform an access to the transactional cache lines early in the critical section

# Contention (conflicts) in SDE (I)

sde-tsx-stats.txt:

```
# COUNTERS OF TSX ABORTS PER ABORT REASON
```

```
#-----  
# REASON                                RTM ABORTS    HLE ABORTS  
  ABORT_CONTENTION                      6559         0
```

## Transaction killers:

```
# TOP 10 CONTENTION ABORTS
```

```
#-----  
#           IP          COUNT  INSTRUCTION DISASSEMBLY  
0x0000000000400ddf      3993  mov eax, dword ptr [rax+0x10]  
0x0000000000400ec2      2519  mov dword ptr [rax+0x10], 0x1  
0x00000000004008b2         31  mov eax, dword ptr [rax+0x4]  
0x000000000040104e         7  mov eax, dword ptr [rax+0x10]  
0x0000000000400ed0         3  mov dword ptr [rax+0x8], edx  
0x0000000000401117         3  mov dword ptr [rax+0x10], 0x1  
0x0000000000400e24         1  mov eax, dword ptr [rax+0x10]  
0x0000000000400eeb         1  mov dword ptr [rax+0xc], edx  
0x0000000000400fbb         1  mov eax, dword ptr [rax+0x10]
```

# Contention (conflicts) in SDE (II)

# STACK INFORMATION FOR CONTENTION ABORT KILLER IP: 0x000000000400ddf

```
#-----  
#           IP           FUNCTION NAME      FILE NAME           LINE    COLUMN  
0x00007fe4cf526520      start_thread                0         0  
0x00000000004015d6      worker                       /root/FP/double/matrix_tsx.c    56         0  
0x0000000000400d78      call_tsx_fasttrack_read     /root/FP/double/fasttrack_tsx.h 148         0  
0x0000000000400ddf      call_tsx_fasttrack_read     /root/FP/double/fasttrack_tsx.h 159         0
```

# STACK INFORMATION FOR CONTENTION ABORT KILLER IP: 0x000000000400ec2

```
#-----  
#           IP           FUNCTION NAME      FILE NAME           LINE    COLUMN  
0x00007fe4cf526520      start_thread                0         0  
0x00000000004015d6      worker                       /root/FP/double/matrix_tsx.c    56         0  
0x00000000004013fd      mm                           /root/FP/double/matrix_tsx.c    30         0  
0x0000000000400d78      call_tsx_fasttrack_read     /root/FP/double/fasttrack_tsx.h 148         0  
0x0000000000400ef0      call_tsx_fasttrack_read     /root/FP/double/fasttrack_tsx.h 188         0  
0x0000000000400ec2      call_tsx_fasttrack_read     /root/FP/double/fasttrack_tsx.h 182         0
```

#LIST OF TSX CONTENTION ABORT EVENTS

```
#-----  
# TID      TRANSACTION IP      KILLER TID      KILLER IP      KILLER DATA ADDRESS  INSIDE TSX  TSX TYPE  
1 0x000000000040138e      2 0x0000000000400ec2      0x00000000006067e0      YES      RTM  
2 0x000000000040138e      1 0x00000000004008b2      0x00000000006066e4      YES      RTM  
1 0x000000000040138e      2 0x0000000000400fbb      0x0000000000608408      YES      RTM  
2 0x000000000040138e      1 0x0000000000400ec2      0x00000000006065d8      YES      RTM  
<----- CUT ----->
```



# Instruction Specific Aborts

- Legacy X87 math (long double)/MMX -> use SSE/AVX math (compiler switch)
- X87 for function parameters -> inline
- SSE/XMM regs AVX instruction mix, VZEROUPPER -> avoid
- Avoid ring transitions: SYSENTER, SYSCALL, SYSEXIT, SYSRET
- Segment, control, debug regs (kernel) -> move outside Tx

# CPU Faults and Traps

- Instruction page faults, segm faults, divide errors, breakpoint, other exceptions are suppressed -> reexecute exactly the same path without elision to resolve the fault and/or debug the error

## Higher abort rate on program start up:

- Data page faults (memory init, program startup) -> skip elision for first time -> ok then
- Core sets Accessed and Dirty Bits in page table: skip elision for first time -> ok then

# Background (Noise-level) Aborts

Not a problem – should be just noise level

- NMI, SMI, INTR, IPI, PMI, timer ticks, etc
- L3 cache inclusion aborts
- Rare uarch conditions
- Handling corner cases with ucode (rare)

# Still Observing Aborts?

- One-time or temporal aborts: transition to fall-back early (XABORT or lock acquire commit), skip elision for some time (once)
- Disable TSX elision for selected lock

## Aborts/retries may be better than taking lock

- TSX with low commit/abort ratio may still outperform locks (better to retry than waste time in waiting for the lock)
- Taking lock is very expensive on servers (all 100s threads serialize): Tx retrying without serializing other threads is cheaper

# Further Reading

- Intel TSX tech resources + **performance studies**:  
[www.intel.com/software/tsx](http://www.intel.com/software/tsx)
- IA Optimization Guide (Chapter 12)  
<http://www.intel.com/content/www/us/en/architecture-and-technology/64-ia-32-architectures-optimization-manual.html>