Using Intel® Transactional Synchronization Extensions

Dr.-Ing. Michael Klemm
Software and Services Group
michael.Klemm@intel.com

OpenMP

Credits

“The Tutorial Gang“

Christian Terboven — IT Center | RWTH AACHEN UNIVERSITY

Michael Klemm — intel

Ruud van der Pas — ORACLE

Eric Stotzer — TEXAS INSTRUMENTS

Bronis R. de Supinski — LAWRENCE LIVERMORE NATIONAL LIBRARY

Members of the OpenMP Language Committee

Disclaimer & Optimization 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.

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, reference www.intel.com/software/products.

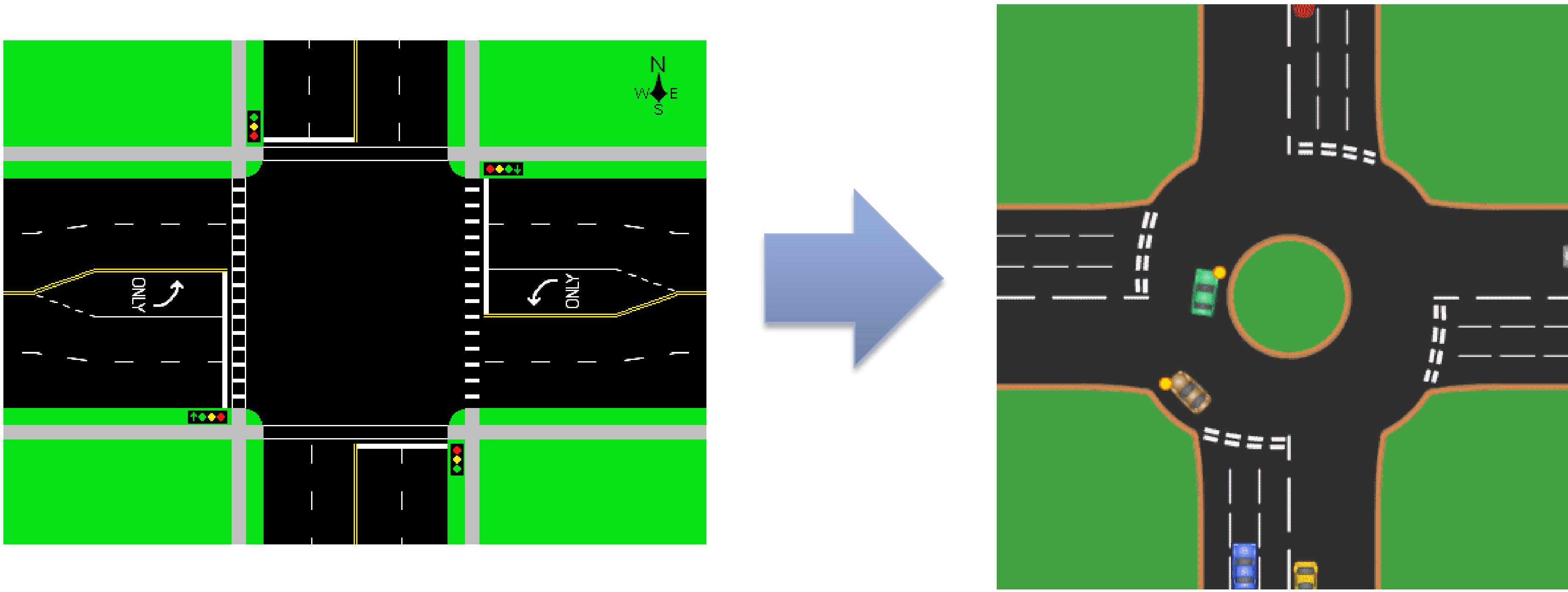
All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.
*Other names and brands may be claimed as the property of others.

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.

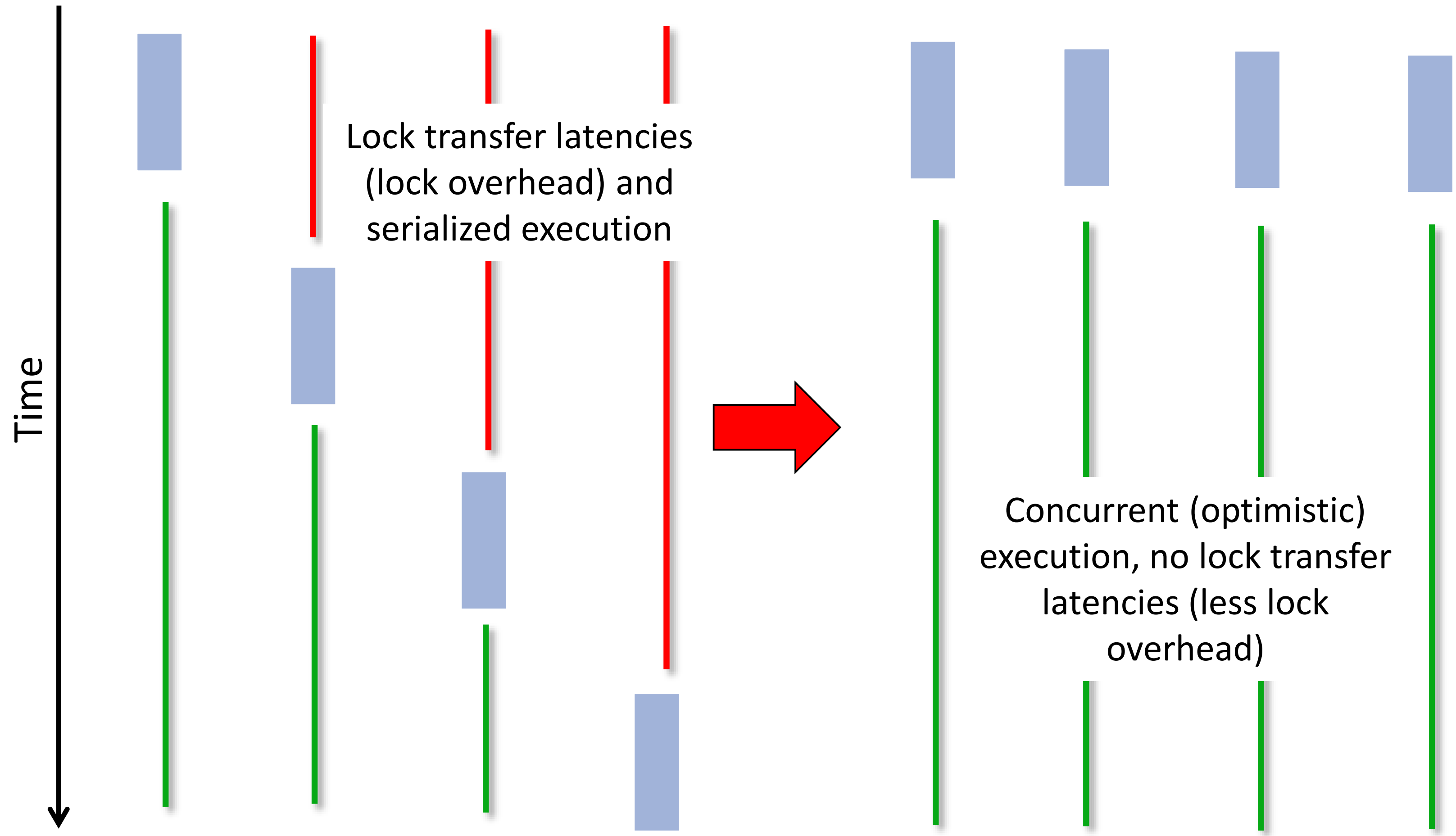
Notice revision #20110804

Optimistic Non-blocking Execution



Picture idea from Dave Boutcher

Lock Elision



Hardware Implementation (IA)

- 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 Implementation (IA)

- 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 Implementation (IA)

- 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)

Using TSX through pthreads

- GLIBC PThreads 2.18
  - → configure with --enable-lock-elision=yes
    - → SUSE SLES SP12 enables it by default
    - → Other distros may have this enabled, too.

- mutexes with PTHREAD_MUTEX_DEFAULT type are adaptively elided (RTM-based)

- PThreads 2.20: rwlocks can be elided too

Using TSX through TBB

- 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
- Also see
  - → 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

Per-lock Control

- Coarse-grained control does not help applications that have mixed locking requirements
  - → Some locks may be highly contended
  - → Some locks may be used to protect system calls (e.g., IO)
  - → Some locks may be just there for safety, but are almost never conflicting (e.g., hash map)

- Programmers need the ability to choose locks on a per-use basis

Two new API Routines

- `omp_init_lock(omp_lock_t *lock)`

- ```
  omp_init_lock_with_hint(omp_lock_t *lock,
                          omp_lock_hint_t hint)
  ```

- `omp_set_lock(omp_lock_t *lock)`

- `omp_unset_lock(omp_lock_t *lock)`

- `omp_destroy_lock(omp_lock_t *lock)`
  ```

Two new API Routines

- `omp_init_nest_lock(omp_nest_lock_t *lock)`
- ```
  omp_init_nest_lock_with_hint(
       omp_nest_lock_t *lock,
       omp_lock_hint_t hint)
  ```
- `omp_set_nest_lock(omp_nest_lock_t *lock)`
- `omp_unset_nest_lock(omp_nest_lock_t *lock)`
- `omp_destroy_nest_lock(omp_nest_lock_t *lock)`
  ```

Hints

- Hints are integer expressions
  - → C/C++: can be combined using the `|` operator
  - → Fortran: can be combined using the `+` operator

- Supported hints:
  - → `omp_lock_hint_none`
  - → `omp_lock_hint_uncontended`
  - → `omp_lock_hint_contended`
  - → `omp_lock_hint_nonspeculative`
  - → `omp_lock_hint_speculative`

New Clause for `critical`

- Specify a hint how to implement mutual exclusion
  - → If a `hint` clause is specified, the `critical` construct must be a named construct.
  - → All `critical` constructs with the same name must have the same `hint` clause.
  - → The expression of the `hint` clause must be a compile-time constant.

- Syntax (C/C++)

```
#pragma omp critical [(name) [hint(expression)]]
structured-block
```

- Syntax (Fortran)

```
!$omp critical [(name) [hint(expression)]]
structured-block
!$omp end critical [(name)]
```

Examples

```
void example_locks() {
    omp_lock_t lock;
    omp_init_lock_with_hint(&lock, omp_hint_speculative);
#pragma omp parallel
    {
        omp_set_lock(&lock);
        do_something_protected();
        omp_unset_lock(&lock);
}   }
```

```
void example_criticial() {
#pragma omp parallel for
    for (int i = 0; i < upper; ++i) {
        Data d = get_some_data(i);
#pragma omp critical (HASH) hint(omp_hint_speculative)
        hash.insert(d);
}   }
```

Monitoring TSX Execution

```
# 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+

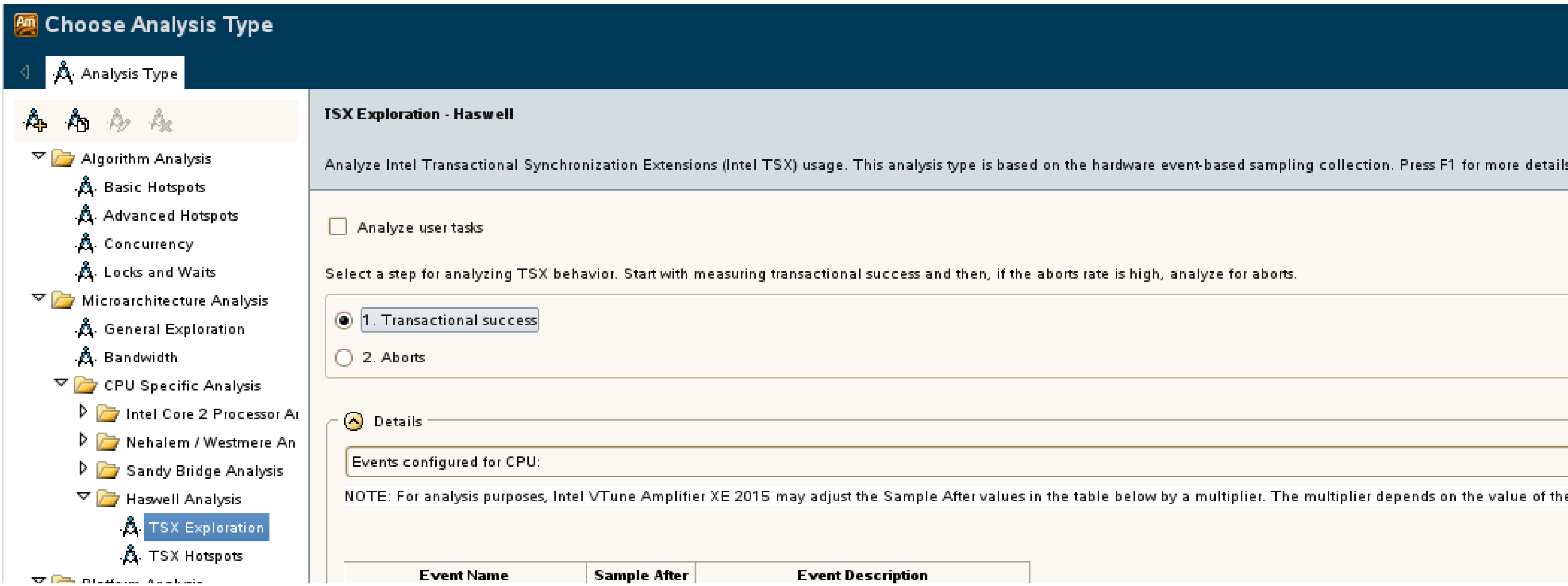
Windows/any Linux/
FreeBSD/OSX:
Intel PCM-TSX tool

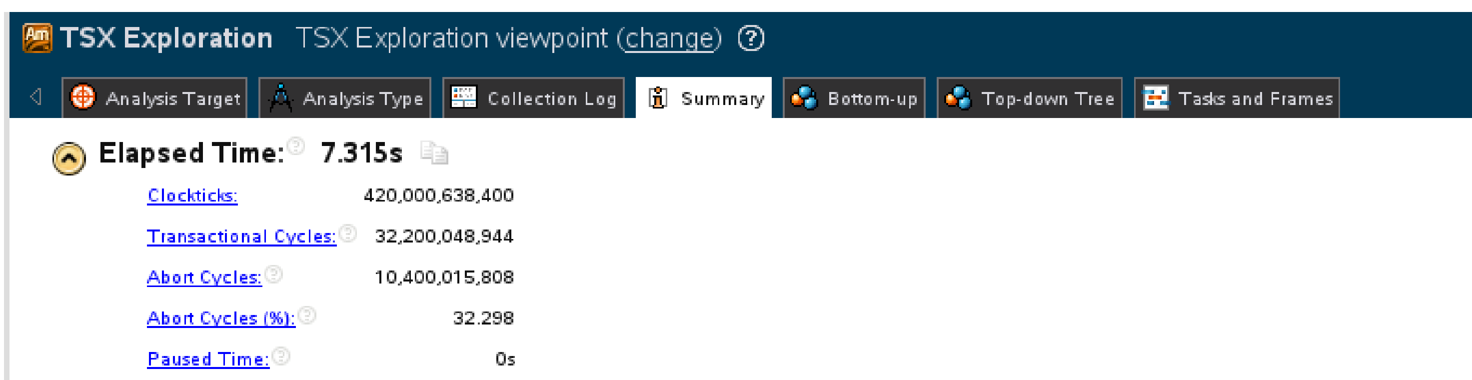
```
#./pcm-tsx.x ./program
 Intel(r) Performance Counter Monitor: Intel(r) Transactional Synchronization Extensions Monitoring Utility

 Executing "./program" command:

Time elapsed: 42 ms
Core | IPC  | Instructions | Cycles   | Transactional Cycles | Aborted Cycles  | #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
```

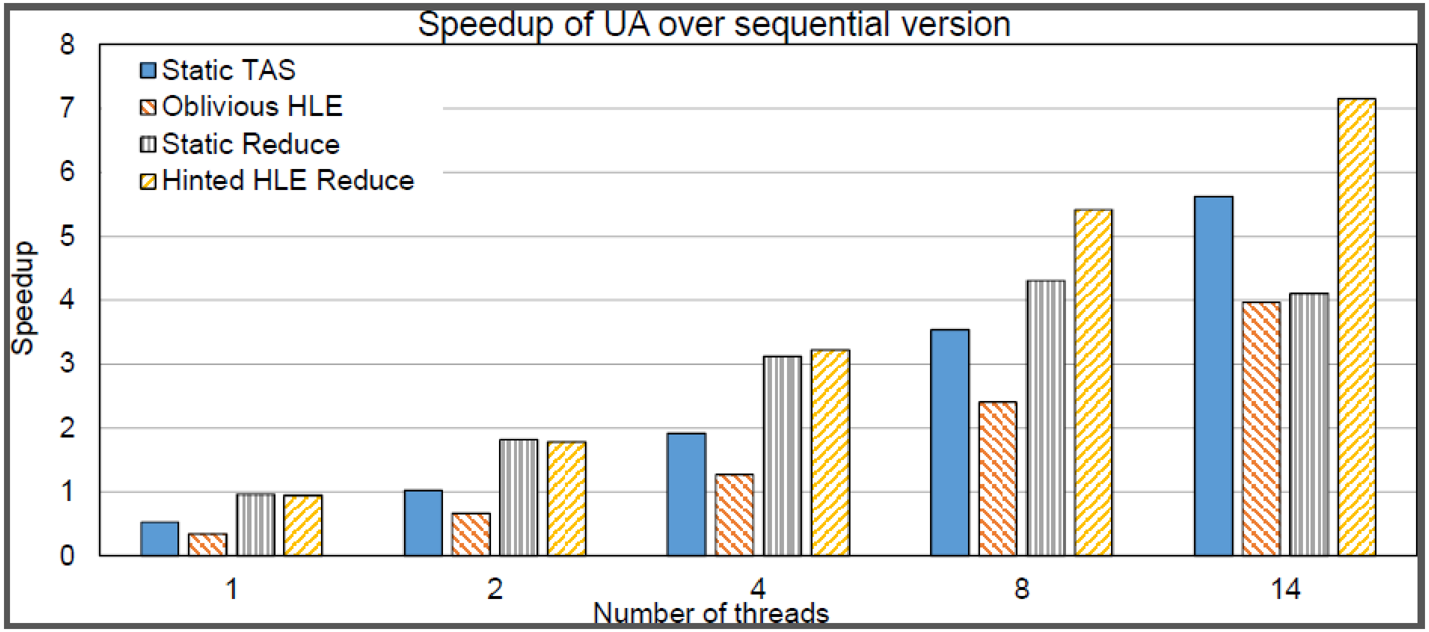
Intel® VTune™ Amplifier XE





Hints May Increase Performance

- Blindly using speculative locks does not help (KMP_LOCK_KIND=…)
- Speculative locks can benefit more with growing thread counts



H. Bae, J.H. Cownie, M. Klemm, and C. Terboven. A User-guided Locking API for the OpenMP Application Program Interface. In Luiz DeRose, Bronis R. de Supinski, Stephen L. Olivier, Barbara M. Chapman, and Matthias S. Müller, editors, Using and Improving OpenMP for Devices, Tasks, and More, pages 173-186, Salvador, Brazil, September 2014. LNCS 8766.