USER-GUIDED SPECULATIVE LOCKS FOR ALGEBRAIC MULTIGRID SMOOTHERS

Barna L. Bihari[1], *James Cownie*[2], *Hansang Bae*[2], *Ulrike M. Yang*[1] and *Bronis R. de Supinski*[1]

[1]*Lawrence Livermore National Laboratory*, Livermore, California
[2]*Intel Corporation*, Santa Clara, California

OpenMPCon Developers Conference
Nara, Japan, October 3-4, 2016

Acknowledgements (LLNL)

Technical:

- Trent D'Hooge (LLNL)
- Tzanio Kolev (LLNL)

Supervisory:

- Lori Diachin (LLNL)

Borrowed example and mesh figures:

- A.H. Baker, R.D. Falgout, Tz.V. Kolev, and U.M. Yang, Multigrid Smoothers for Ultraparallel Computing, *SIAM J. Sci. Comput.*, **33** (2011), pp. 2864-2887. LLNL-JRNL-473191.

Financial support:

- Dept. of Energy (DOE), Office of Science, ASCR
- DOE, under contract DE-AC52-07NA27344

Legal 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.

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.

Intel, Xeon, and the Intel logo 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

- Introduction and history
- Prior/standard approaches
- Review of TSX on Intel
- Review of AMG
- Review of TM for iterative methods, such as AMG
- Experimental results
  - Problem, mesh, code and experimental setup
  - Convergence studies
  - Performance
- Conclusions

Introduction and History

- Motivation: Increased number of cores per node for the foreseeable future – shared memory parallel programming or threading
- In 2011: IBM Blue Gene/Q: first production-quality HTM system
- In 2013: Intel Haswell: HTM system called TSX with two flavors
- Predecessor to HTM: STM (Software Transactional Memory) by IBM, Intel and Sun
- Current work is related to our IWOMP 2010, 2012, 2014, and 2015 papers
- Direct Methods: Multiple threads in a shared-memory setting can lead to race conditions, memory conflicts and incorrect execution
- Iterative Methods: Multiple threads do lead to race conditions, but incorrect execution is hard to quantify; result: inefficiency instead
- Current work: Study and compare the effects of:
  - (i) Restricted Transactional Memory (RTM – part of TSX),
  - (ii) Hardware Lock Elision (HLE – part of TSX), and
  - (iii) OpenMP critical

Can TSX benefit iterative methods ?

Possible Alternatives to TM

- Mutual exclusion: One thread at a time – OMP critical
  - very safe, but not very scalable
  - potential deadlocks
  - convoying of execution: poor performance
- OpenMP atomics: Non-blocking protection – OMP atomic
  - compare-and-swap (CAS)
  - load-linked store conditional (LL/SS)
  - limited to single instruction
  - several atomics are not equivalent to one large transaction
- Lock elision: Speculative technique - similar to TM (on Haswell/Broadwell)
  - optimistic execution of a critical section
  - elides or bypasses acquiring the lock
  - uses cache coherency mechanism to track reads/writes
  - transaction size limited to cache size – not as general as TM
  - if there is no hardware support, defaults to acquiring the lock

Only OMP critical is available/applicable as comparison

TM as part of OpenMP?

- TM solves the concurrency control problem
- TM is not new: goes back to Herlihy and Moss (1993)
- TM raises the level of abstraction
- It can coexist with current OpenMP concurrency mechanisms
- TM is deadlock-free and expected to be scalable
- Ease of use is a key consideration
- Simple syntax, e.g. IBM's current OpenMP extension:

```
#pragma tm_atomic [(safe_mode)]
{
   < code >
}
```

Need high level support for TM

Intel TSX Overview

- Intel TSX = Transactional Synchronization Extensions
  - cache coherency protocol detects memory access conflicts
  - this is transactional memory (TM) with restricted working set
  - this allows coarse-locked codes to behave as if implemented with fine-grain reader-writer locks
- Hardware Lock Elision (HLE): extension for existing locks
  - processor speculates critical section, but preserves all lock semantics
  - in case of a conflict the lock is taken "for real"
- Restricted Transactional Memory (RTM): new transaction instructions
  - explicit begin and commit transaction operations, no visible lock
  - there has to be a non-speculative back-off path in case of conflict

Two types of TM with hardware support

User-Guided Speculative Locks

- Fundamental requirement: do not break any existing code
  - new functionality is introduced as hints
- Three options were considered
  - pragmas to prefix existing lock routines with the desired hint
  - complete set of new locking routines and lock types
  - new lock initialization routines to use with the existing lock API
    - minimal code modification, allows for incremental code adoption
- OpenMP lock review
  - variable of type **`omp_lock_t`** or **`omp_nest_lock_t`**
  - must be initialized before first use with **`omp_init[_nest]_lock()`**
  - routines to *initialize*, *set*, *unset*, and *test* a lock and finally to *destroy* it

Software was needed for convenient usage of Intel's TSX

User-Guided Speculative Locks (Cont'd)

- Two new lock init functions provide hints to the runtime system
  - `void kmp_init[_nest]_lock_hinted( omp[_nest]_lock_t*, omp_lock_hint )`
- The `kmp_lock_hint` type lists high-level optimization criterions:
  - **`kmp_lock_hint_none`**
  - **`kmp_lock_hint_uncontended`** optimize for an uncontended lock
  - **`kmp_lock_hint_contended`** optimize for a contended lock
  - **`kmp_lock_hint_nonspeculative`** do not use hardware speculation
  - **`kmp_lock_hint_speculative`** use HLE hardware speculation
  - **`kmp_lock_hint_adaptive`** adaptively use RTM speculation
  - … plus room for vendor-specific extensions
- Fundamental requirement: do not break any existing code
- Open source OpenMP runtime – part of LLVM as well

Software was provided for convenient usage of Intel's TSX

TM for Scientific Computing

- Few/no benchmarks exist for experimentation with TM
- Numerical methods can be divided into two large classes:
  - Direct methods (exactly serializable)
  - Iterative methods (not easily serializable)
- For iterative methods: what is the "wrong" answer?
- Most thread synchronizations will converge -- eventually
- Best solution: the one that is fastest to converge
- Synchronization may not be crucial or necessary for convergence
- However: *unsynchronized* code is incorrect code (and unpredictable)
- Therefore: threaded code should be synchronized

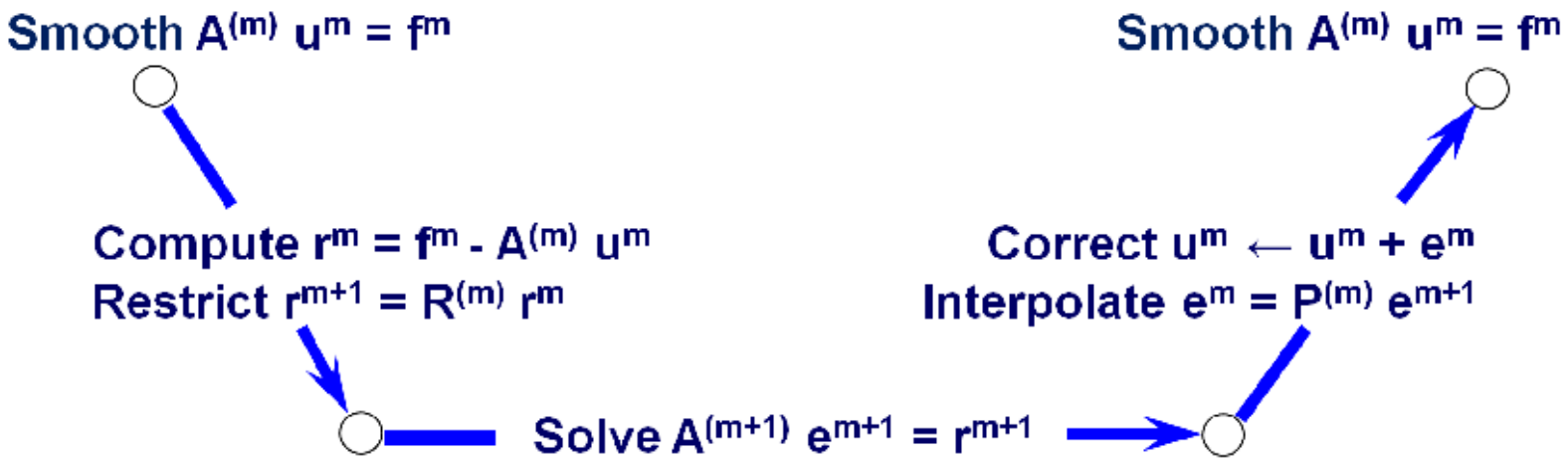
Need to study the effects of synchronization mechanisms

Brief Review of AMG

Setup Phase

- Select coarse "grids"
- Define interpolation, $P^{(m)}$, m=1,2,…
- Define restriction, $R^{(m)} = (P^{(m)})^T$
- Define coarse-grid operators, $A^{(m+1)} = R^{(m)} A^{(m)} P^{(m)}$

Solve Phase (level m)



Brief Review of AMG (Cont.'d)

- Smoothers are a critical part of AMG
- Reduce erors in the direction of eigenvectors
- Simple point-wise smoothers like Jacobi or Gauss-Seidel (G-S) reduce errors associcated with large eigenvectors rapidly
- It can be symbolically represented by the equation:

$$u_i^{(n+1)} = \frac{1}{A_{ii}}(f_i - \sum_{j=1}^{N} A_{ij} u_j^{(l)})$$

where

- u_i is the approximation itself
- f_i is the right hand side
- A_{ij} represents the *j*-th component of row *i* in matrix *A*
- *l* can be either *n* or *n+1*, depending on 'age'

- This is parallelized by partitioning *A* row-wise
- Hybrid Gauss-Seidel has Jacobi-like update on node or thread boundaries

The Connection to TM

- The same TM-assisted method used for mesh smoothing can also be used for Hybrid G-S
- Indeed, the formulae look similar:

for AMG:

$$u_i^{(n+1)} = \frac{1}{A_{ii}}(f_i - \sum_{j=1}^{N} A_{ij} u_j^{(l)})$$

for mesh smoothing:

$$\mathbf{x}_i^{(n+1)} = \frac{1}{N_i} \sum_{j=1}^{N_i} \mathbf{x}_j^{(m)}$$

- Nontrivial to parallelize because of the dependencies in u_i:
- Race conditions exist, where probability of conflicts is low, but nonzero
- Transactional memory will synchronize differently than OpenMP critical
- We have a write-after-read (WAR) conflict where the average itself might change during the averaging process

The Relevant Code Section

```
  omp_lock_t lock;
  kmp_init_lock_hinted(&lock, kmp_lock_hint_speculative);
.....
#pragma omp parallel for private(i,ii,jj,res) HYPRE_SMP_SCHEDULE
  for (i = 0; i < n; i++)
  {             // start of for-loop threaded over rows
    if (cf_marker[i] == relax_points &&
        A_diag_data[A_diag_i[i]]  != zero)
    {           // start of if-statement
      res = f_data[i];
      for (jj = A_offd_i[i]; jj < A_offd_i[i+1]; jj++)
      {
        ii = A_offd_j[jj];
        res -= A_offd_data[jj] * Vext_data[ii];
      }
      omp_set_lock(&lock);
      {         // start of critical region
// Step 1:  Take weighted-average.
      for (jj = A_diag_i[i]+1; jj < A_diag_i[i+1]; jj++)
        {       // start of averaging for-loop
          ii = A_diag_j[jj];
          res -= A_diag_data[jj] * u_data[ii];
        }       // end of averaging for-loop
// Step 2:  Update current u.
        u_data[i] = res / A_diag_data[A_diag_i[i]];
      }         // end of critical region
      omp_unset_lock(&lock);
    }           // end of if-statement
  }             // end of for-loop threaded over rows
```

The Role of TSX

- Within the transaction we have a WAR (write-after-read) type update
- The potential memory conflict comes from a variable being updated after it had been read (for another variable's update by a different thread)
- TSX rollbacks will then "refresh" the stale variables
- TSX will typically yield "fresher", more up-to-date info
- With **HLE**: becomes OMP critical upon conflict
- With **RTM**: multiple retries are possible depending on KMP_ADAPTIVE_LOCK_PROPS = M, N: *adaptive*-lock
- The only comparable alternative to TSX is omp critical
- TSX has an effect on how the solution converges
- Tradeoff: RTM is more expensive but more "accurate"

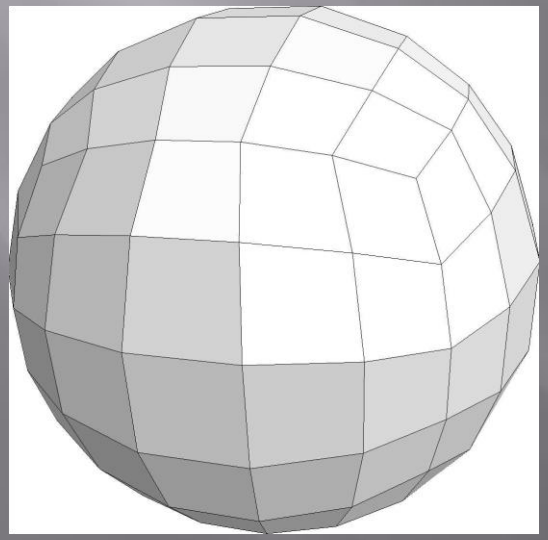
TSX adds algorithmic aspects of its own.

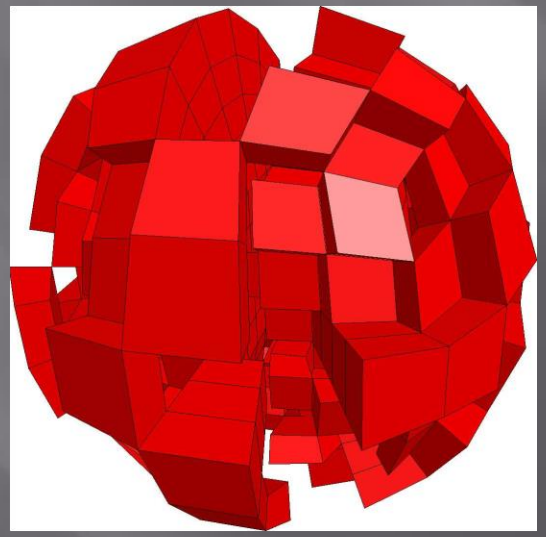
Experimental Results

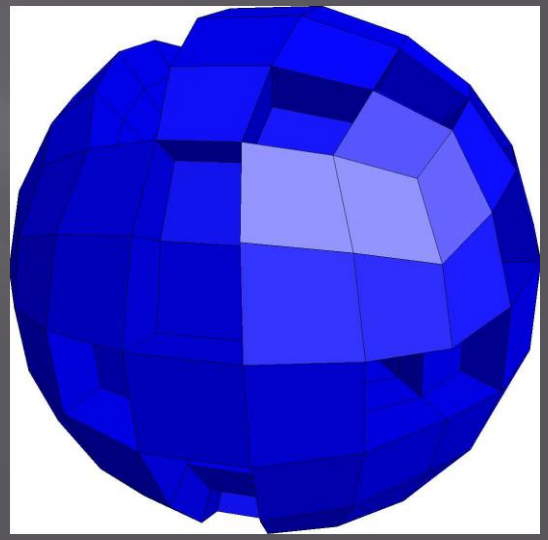
Solving Scalar diffusion:

$$-\nabla \cdot (a(x,y,z)\nabla u) = f$$

- 3-D sphere mesh with hexahedral finite elements
- Two arbitrarily-placed material subdomains
- Material coefficients *a(x,y,z)* are 1 and 1000





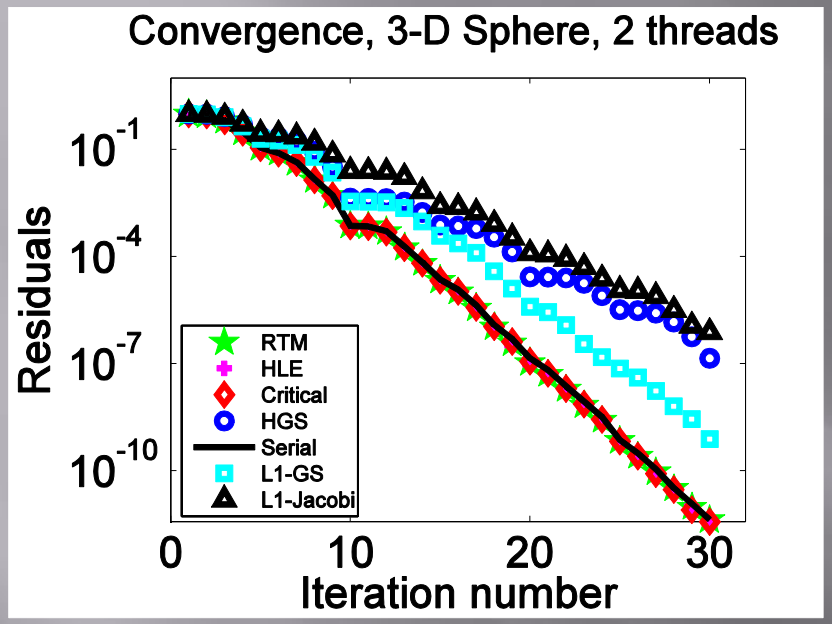


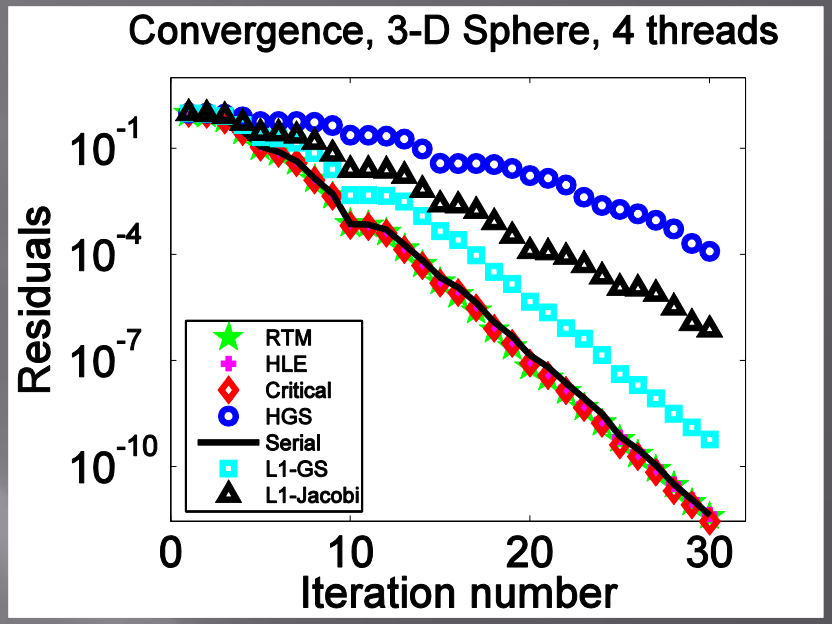
Experimental Results (Cont.'d)

- Ran on 1, 2, 4, 8, 16, 32 and 64 threads
- Three modes of running with threads:
  - HLE
  - RTM
  - OpenMP critical
- Modification of BoomerAMG branch of *hypre*
- Compared against HGS, l_1 GS and l_1 Jacobi
- HMIS coarsening with extended+i interpolation
- AMG-preconditioned GMRES as solver
- Run on Intel Xeon E5-2695v4 (Broadwell), 64GB mem.
- Turbo Boost, hyperthreading and TSX enabled
- Convergence measured in terms of residuals

Goal: study convergence of AMG smoother

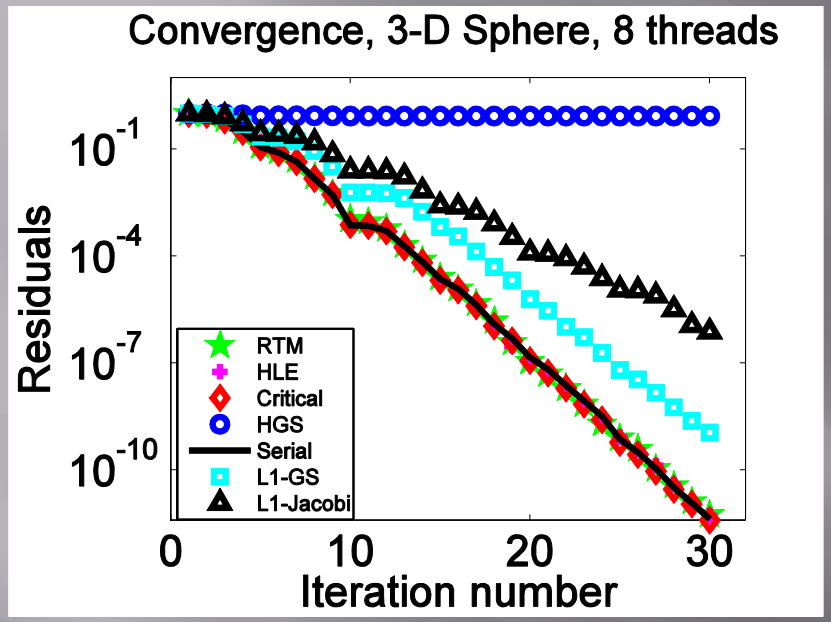
The Convergence

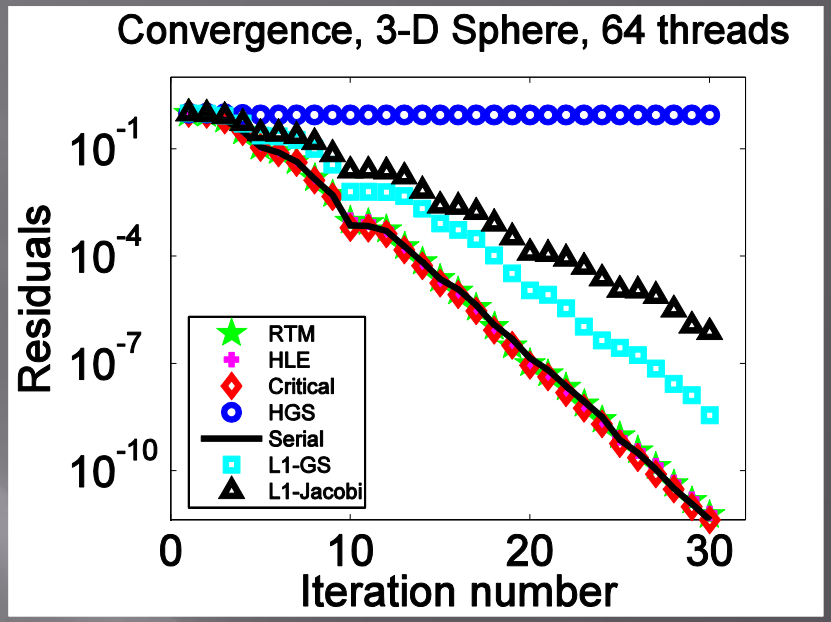




- On 1 thread: *RTM, HLE*, *critical* and *HGS* identical to *serial* (not shown)
- On 2 threads: substantial difference between *HGS* and all other options; *HGS* approaches *L1-Jacobi*
- On 4 threads: *RTM, HLE, critical, serial* are very close to each other; *HGS* deteriorates considerably, *L1-GS* does so slightly
- On all thread counts: no change in *L1-Jacobi* (thread invariant)

The Convergence

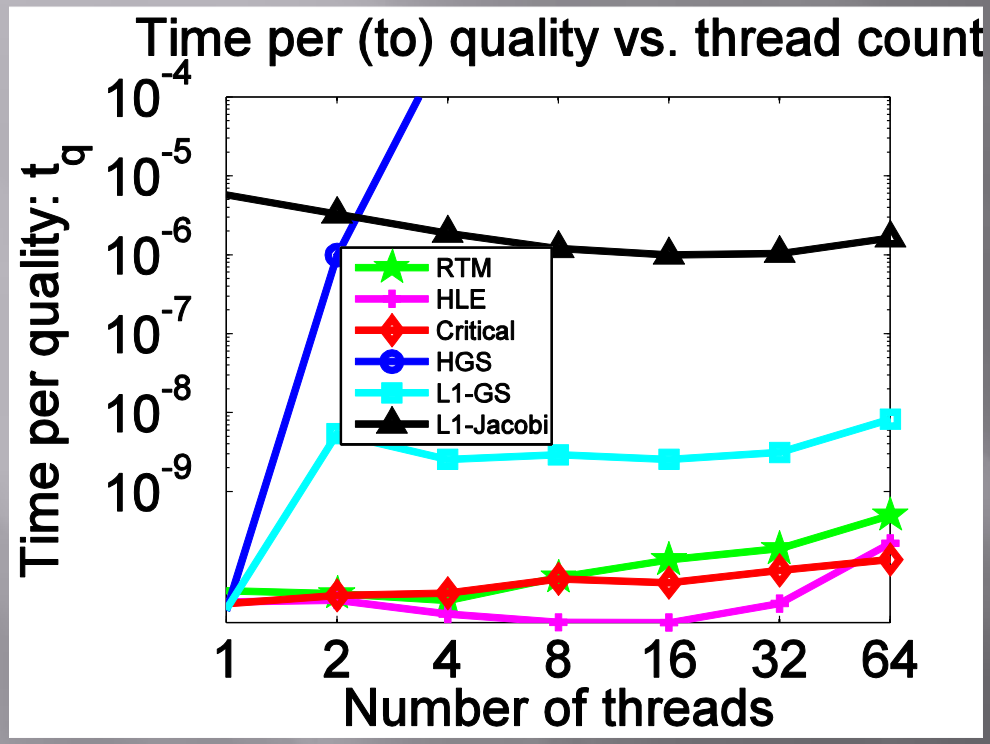




- On 8 threads: *RTM, HLE*, and *critical* remain close to *serial*; *HGS* stopped converging altogether
- On 64 threads: *RTM, HLE*, and *critical* remain close to *serial*; *HGS* does not converge; *L1-GS* deteriorates more
- On all thread counts: no change in *L1-Jacobi* (thread invariant)

Adding synchronization to *HGS* resulted in convergence

Measuring Perfomance



•New way to measure performance (introduced at IWOMP 2014):

$$t_q^{(n)} = \frac{t^{(n)}}{q^{(n)}} = t^{(n)} r^{(n)}$$

where *r* is now residual (not actual error or "distance to exact solution")

• *HLE, RTM* and *critical* outperform *all others* by orders of magnitude (except on 1 thread)

- Orders of magnitude difference in performance between the various smoothers
- On 2 through 32 threads: HLE is better than *unsync*
- On 2, 4 and 8 threads: *RTM* is competitive
- On 4 through 32 threads: *HLE* is better than all others
- For this problem *HLE* is overall "best" on 16 threads

HLE is the overall best performer for this problem

Related Talk at IWOMP 2016

- Stay tuned for a closely related presentation at IWOMP this Friday, at 3:50pm:

"Transactional Memory for Algebraid Multigrid Smoothers"

by Barna Bihari, Ulrike Yang, Michael Wong, and Bronis R. de Supinski

Summary and Future Work

- For select algorithms, transactional memory promises thread-safety, easier programming, and performance simultaneously
- Intel Xeon (formerly "Broadwell") has TSX with two options
- TM/TSX pay-off is expected to be highly code- and problem-dependent - as shown by our previous work
- New study of Algebraic Multigrid smoothers used in *hypre*
- First time TSX was applied to an industrial-grade package
- Synchronization made a non-convergent scheme converge
- AMG smoothers appear to be a good use case for TSX
- HLE outperforms other options in "time-to-quality"
- Hope to apply TSX to other GS-flavored methods, like CG
- Potential new lock type: "pure RTM""
- Always looking for collaborators and new candidates for TM