Optimizing Hybrid Transactional Memory: The Importance of Nonspeculative Operations

Torvald Riegel (Red Hat)
Martin Nowack, Christof Fetzer (TU Dresden)
Patrick Marlier, Pascal Felber (Uni Neuchatel)
Challenges for hybrid SW/HW TM

- Synchronization: concurrent STM and HTM txns
- Performance: Slow STM vs. limited HTM

- First-generation HTM limitations:
  - Capacity for txnally accessed data is limited
  - Not all code allowed in txns, txns not virtualizable, ...

- AMD's Advanced Synchronization Facility (ASF):
  - HTM proposal for x86_64 (use near-cycle-accurate simulator)
  - Low runtime overhead, but limited capacity:
    - LLB8: max 8 cachelines
    - LLB256: max 256 cachelines
    - LLB8L1: L1 cache for reads (two-way associative!), max 8 cachelines for stores
Contribution: How to exploit nonspeculative accesses

- ASF supports nonspeculative accesses in txns
  - No HTM capacity required for nonspeculative accesses
  - Nonspeculative atomic instructions (e.g., CAS) allowed in txns
  - Speculative accesses monitor cache lines immediately for conflicts (monitoring happens in program order)

- General-purpose techniques, applied to HyTM:
  1) Monitor metadata but read data nonspeculatively
  2) Use nonspeculative atomic RMW operations to send sync msgs
  3) Nonspeculatively validate HW txns against SW sync msgs

- Benefits:
  - Require less HTM capacity → More txns in HW
  - Avoid false aborts due to sync with SW txns
  → Increases concurrency, more txns in HW despite SW txns
### Our HyTMs

<table>
<thead>
<tr>
<th>HyTM</th>
<th>HW/SW concurrency</th>
<th>HW load/store overheads</th>
<th>HW capacity used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyNOrec-2</td>
<td>Yes, SW commits stall other ops</td>
<td>Very small</td>
<td>Data</td>
</tr>
<tr>
<td>HyLSA</td>
<td>Yes</td>
<td>Small (load orec)</td>
<td>Orecs and data updates</td>
</tr>
</tbody>
</table>

1. Monitor metadata but read data nonspeculatively
2. Use nonspeculative atomic RMW operations to send sync msgs
3. Nonspeculatively validate HW txns against SW sync msgs
## Our HyTMs vs. previous HyTMs

<table>
<thead>
<tr>
<th>HyTM</th>
<th>HW/SW concurrency</th>
<th>HW load/store overheads</th>
<th>HW capacity used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>HyNOrec-2</td>
<td>Yes, SW commits stall other ops</td>
<td>Very small</td>
<td>Data</td>
</tr>
<tr>
<td>HyLSA</td>
<td>Yes</td>
<td>Small (load orec)</td>
<td>Orecs and data updates</td>
</tr>
<tr>
<td>Phased TM</td>
<td>No</td>
<td>None</td>
<td>Data</td>
</tr>
<tr>
<td>Damron et al.</td>
<td>Yes</td>
<td>Small (load orec)</td>
<td>Data and orecs</td>
</tr>
<tr>
<td>HASTM cautious</td>
<td>Yes</td>
<td>Medium (load+log orec)</td>
<td>Data</td>
</tr>
<tr>
<td>HASTM aggressive</td>
<td>Yes</td>
<td>Small (load orec)</td>
<td>Data and orecs</td>
</tr>
<tr>
<td>HyNOrec-DSS (PPoPP10)</td>
<td>Partial, <strong>SW commits abort HW txns</strong></td>
<td>None, but <strong>concurrent HW commits abort each other</strong></td>
<td>Data and 2 locks</td>
</tr>
<tr>
<td>HyNOrec-DSS-2 (concurrent work)</td>
<td>Yes, SW commits stall other ops</td>
<td>Very small, <strong>concurrent HW commits can still abort each other</strong></td>
<td>Data and 3 locks</td>
</tr>
</tbody>
</table>
STM Base Algorithm: NOrec
[Dalessandro et al. PPoPP10]

- Similar to single-orec LSA STM with redo/write-back on commit
- Value-based validation in addition to time-based validation

Global state:
gsl: volatile sequence lock/orrec:
locked (MSB), clock (other bits)

Per-txn state:
sl (copy of gsl), writeset, readset

stm-start():
do { sl = gsl; } while (sl.locked);
clear readset and writeset

stm-load(addr):
if (read after write)
return val from writeset;
val = *addr;
while (sl != gsl)
{ sl = validate(); val = *addr; }
return val;

stm-store(addr, val):
store {addr,val} in writeset;

stm-commit():
if (readonly txn) return;
while (!CAS(&gsl, sl, {true, sl.clock}))
  sl = validate();
write back values in writeset;
gsl = {false, sl.clock+1}

validate():
do {
  do { x = gsl} while (x.locked);
  for ({addr,val} in readset)
    if (*addr != val) abort();
} while (x != gsl);
STM Base Algorithm: NOrec

- Similar to single-orec LSA STM with redo/write-back on commit
- Value-based validation in addition to time-based validation

Global state:
- gsl: volatile sequence lock/orec: locked (MSB), clock (other bits)

Per-txn state:
- sl (copy of gsl), writeset, readset

stm-start():
  do { sl = gsl; } while (sl.locked);
  clear readset and writeset

stm-load(addr):
  if (read after write)
    return val from writeset;
  val = *addr;
  while (sl != gsl)
    { sl = validate(); val = *addr; }
  return val;

stm-store(addr, val):
  store {addr,val} in writeset;

stm-commit():
  if (readonly txn) return;
  while (!CAS(&gsl, sl, {true, sl.clock}))
    sl = validate();
  write back values in writeset;
  gsl = {false, sl.clock+1}

validate():
  do {
    do { x = gsl} while (x.locked);
    for ({addr,val} in readset)
      if (*addr != val) abort();
  } while (x != gsl);
STM Base Algorithm: NOrec

- Similar to single-orec LSA STM with redo/write-back on commit
- Value-based validation in addition to time-based validation

**Global state:**
- gsl: volatile sequence lock/orec: locked (MSB), clock (other bits)

**Per-txn state:**
- sl (copy of gsl), writeset, readset

**stm-start():**
- do { sl = gsl; } while (sl.locked);
- clear readset and writeset

**stm-load(addr):**
- if (read after write) return val from writeset;
- val = *addr;
- while (sl != gsl)
  { sl = validate(); val = *addr; }
- return val;

**stm-store(addr, val):**
- store {addr,val} in writeset;

**stm-commit():**
- if (readonly txn) return;
- while (!CAS(&gsl, sl, {true, sl.clock}))
  sl = validate();
- write back values in writeset;
- gsl = {false, sl.clock+1}

**validate():**
- do {
  do { x = gsl} while (x.locked);
  for ({addr,val} in readset)
    if (*addr != val) abort();
} while (x != gsl);
STM Base Algorithm: NOrec

- Similar to single-orec LSA STM with redo/write-back on commit
- Value-based validation in addition to time-based validation

Global state:
gsl: volatile sequence lock/orec:
    locked (MSB), clock (other bits)
Per-txn state:
sl (copy of gsl), writeset, readset

stm-start():
do { sl = gsl; } while (sl.locked);
clear readset and writeset

stm-load(addr):
if (read after write)
    return val from writeset;
val = *addr;
while (sl != gsl)
    { sl = validate(); val = *addr; }
return val;

stm-store(addr, val):
store {addr,val} in writeset;

stm-commit():
if (readonly txn) return;
while (!CAS(&gsl, sl, {true, sl.clock}))
    sl = validate();
write back values in writeset;
gsl = {false, sl.clock+1}

validate():
do {
    do { x = gsl} while (x.locked);
    for ({addr,val} in readset)
        if (*addr != val) abort();
} while (x != gsl);
STM Base Algorithm: NOrec

- Similar to single-orec LSA STM with redo/write-back on commit
- **Value-based validation** in addition to time-based validation

**Global state:**
- gsl: volatile sequence lock/orec:
  - locked (MSB), clock (other bits)

**Per-txn state:**
- sl (copy of gsl), writeset, readset

**stm-start():**
- do { sl = gsl; } while (sl.locked);
- clear readset and writeset

**stm-load(addr):**
- if (read after write)
  - return val from writeset;
- val = *addr;
- while (sl != gsl)
  - { sl = validate(); val = *addr; }
- return val;

**stm-store(addr, val):**
- store {addr, val} in writeset;

**stm-commit():**
- if (readonly txn) return;
- while (!CAS(&gsl, sl, {true, sl.clock}))
  - sl = validate();
- write back values in writeset;
- gsl = {false, sl.clock+1}

**validate():**
- do {
  - do { x = gsl} while (x.locked);
  - for ({addr, val} in readset)
    - if (*addr != val) abort();
  - } while (x != gsl);
HyNOrec-0: base HyTM
(like HyNOrec-DSS with 2 fixes)

Additional global state:
  esl: extra volatile seq lock, like gsl

Additional per-txn state:
  update: true iff HW update txn

stm-commit():
  if (readonly txn) return;
  while (!CAS(&gsl, sl, {true, sl.clock}))
    sl = validate();
  esl = {true, sl.clock}
  write back values in writeset;
  esl = {false, sl.clock+1}
  gsl = {false, sl.clock+1}

htm-start():
  sl = esl; // TXNAL
  If (sl.locked) abort();
  update = false;

htm-load(addr):
  return *addr; // TXNAL

htm-store(addr, val):
  *addr = val; // TXNAL
  update = true;

htm-commit():
  if (!update) {
    x = gsl; // TXNAL
    if (x.locked) abort();
    gsl = {false, x.clock+1} // TXNAL
  } COMMIT;
HyNOrec-0: base HyTM (like HyNOrec-DSS with 2 fixes)

Additional global state:
esl: extra volatile seq lock, like gsl

Additional per-txn state:
update: true iff HW update txn

stm-commit():
if (readonly txn) return;
while (!CAS(&gsl, sl, {true,sl.clock}))
  sl = validate();
esl = {true, sl.clock}
write back values in writeset;
esl = {false, sl.clock+1}
gsl = {false, sl.clock+1}

htm-start():
sl = esl; // TXNAL
If (sl.locked) abort();
update = false;

htm-load(addr):
return *addr; // TXNAL

htm-store(addr, val):
*addr = val; // TXNAL
update = true;

htm-commit():
if (!update) {
x = gsl; // TXNAL
  if (x.locked) abort();
  gsl = {false,x.clock+1} // TXNAL
} COMMIT;
HyNOrec-0: base HyTM (like HyNOrec-DSS with 2 fixes)

Additional global state:
  esl: extra volatile seq lock, like gsl

Additional per-txn state:
  update: true iff HW update txn

stm-commit():
  if (readonly txn) return;
  while (!CAS(&gsl, sl, {true, sl.clock}))
    sl = validate();
  esl = {true, sl.clock}
  write back values in writeset;
  esl = {false, sl.clock+1}
  gsl = {false, sl.clock+1}

htm-start():
  sl = esl; // TXNAL
  If (sl.locked) abort();
  update = false;

htm-load(addr):  
  return *addr; // TXNAL

htm-store(addr, val):
  *addr = val; // TXNAL
  update = true;

htm-commit():
  if (!update) {
    x = gsl; // TXNAL
    if (x.locked) abort();
    gsl = {false, x.clock+1} // TXNAL
  } COMMIT;
HyNORc-0: base HyTM (like HyNORc-DSS with 2 fixes)

Additional global state:
- esl: extra volatile seq lock, like gsl

Additional per-txn state:
- update: true iff HW update txn

stm-commit():
if (readonly txn) return;
while (!CAS(&gsl, sl, {true, sl.clock}))
  sl = validate();
esl = {true, sl.clock}
write back values in writeset;
esl = {false, sl.clock+1}
gsl = {false, sl.clock+1}

htm-start():
  sl = esl; // TXNAL
  If (sl.locked) abort();
  update = false;

htm-load(addr):
  return *addr; // TXNAL

htm-store(addr, val):
  *addr = val; // TXNAL
  update = true;

htm-commit():
if (!update) {
  x = gsl; // TXNAL
  if (x.locked) abort();
  gsl = {false, x.clock+1} // TXNAL
} COMMIT;
HyNOrec-0: base HyTM (like HyNOrec-DSS with 2 fixes)

**Additional global state:**
- esl: extra volatile seq lock, like gsl

**Additional per-txn state:**
- update: true iff HW update txn

**stm-commit():**
- if (readonly txn) return;
  - while (!CAS(&gsl, sl, {true, sl.clock}))
    - sl = validate();
    - esl = {true, sl.clock}
  - write back values in writeset,
  - esl = {false, sl.clock+1}
  - gsl = {false, sl.clock+1}

**stm-load(addr):**
- val = *addr;
  - while (sl != gsl)
    - { sl = validate(); val = *addr; }

**htm-start():**
- sl = esl; // TXNAL
  - If (sl.locked) abort();
  - update = false;

**htm-load(addr):**
- return *addr; // TXNAL

**htm-store(addr, val):**
- *addr = val; // TXNAL
  - update = true;

**htm-commit():**
- if (!update) {
  - x = gsl; // TXNAL
  - if (x.locked) abort();
  - gsl = {false, x.clock+1} // TXNAL
  - COMMIT;
HyNOrec-0: base HyTM (like HyNOrec-DSS with 2 fixes)

Additional global state:
- esl: extra volatile seq lock, like gsl

Additional per-txn state:
- update: true iff HW update txn

stm-commit():
  if (readonly txn) return;
  while (!CAS(&gsl, sl, {true,sl.clock}))
    sl = validate();
  esl = {true, sl.clock}
  write back values in writeset;
  esl = {false, sl.clock+1}
  gsl = {false, sl.clock+1}

Little STM/HTM concurrency

htm-start():
  sl = esl; // TXNAL
  if (sl.locked) abort();
  update = false;

htm-load(addr):
  return *addr; // TXNAL

htm-store(addr, val):
  *addr = val; // TXNAL
  update = true;

htm-commit():
  if (!update) {
    x = gsl; // TXNAL
    if (x.locked) abort();
    gsl = {false,x.clock+1} // TXNAL
  }
  COMMIT;

Little HTM/HTM concurrency
HyNOrec-1: Use nonspeculative RMW ops for sending sync msgs

- HTM can commit effectively after it has notified STM
  - Conflicting STM txn validates before HTM commit? → HTM aborts (=empty txn)
  - HTM commit successful? → Conflicting STM txn will abort on validation
  - gsl locked during increment? → Harmless: gsl.locked is MSB and gsl.clock not considered if locked

- HTM commits don't abort each other anymore :-)

```c
htm-start():
  while (esl.locked) spin();
  sl = esl; // TXNAL
  if (sl.locked) abort();
  update = false;

htm-load(addr):
  return *addr; // TXNAL

htm-store(addr, val):
  *addr = val; // TXNAL
  update = true;

htm-commit():
  if (!update) {
    x = atomicFetchAndInc(&gsl);
    if (x.locked) abort();
  }
  COMMIT;
```
• HW C. Abort Rate: due to contention, on HTM code path
• Only HTM txns, mostly update txns, frequent commits
  → HyNOrec-1 optzn is essential for HW/HW concurrency (see aborts)
HyNOrec-2: Use nonspeculative ops to process sync msgs

- HTM waits until concurrent STM commits finish
  - If not aborted, no conflict
  - Keep esl/gsl split to get less cache misses

- Nonconflicting STM commits don't abort HTM txns :-) (but keeps esl hot in cache)

```c
htm-start():
  // don't monitor esl
  update = false;

htm-load(addr):
  val = *addr; // TXNAL
  while (esl.locked) spin();
  return val;

htm-store(addr, val):
  *addr = val; // TXNAL
  update = true;

htm-commit():
  if (update) {
    atomicInc(&gsl);
    while (gsl.locked) spin();
  } COMMIT;
```
HyNOrec-2 performance gains

- There are SW update commits (HTM rate is about 80%)
- HyNOrec-2 optzn helps: STM commits do not abort HTM txns
5% STM is enough for HyNOrec-2 to scale better than HTM+serial.

With 100% HTM and medium-sized txns, HyNOrec-2 is as fast as HTM.
Conclusion

- Nonspeculative accesses very useful for HyTM and in general
- HyNORec-2:
  - Great performance
  - Uses HTM resources very efficiently
  - Suitable for systems software

- GCC’s transactional-memory branch
- libitm: GCC’s TM runtime library

Collaboration is welcome at Red Hat!

 triegel@redhat.com
Backup slides
HyNOrec-2 can suffer from contention on gsl (vs. HTM, but no aborts)

STM can win because of smaller conflict detection granularity
Performance: STAMP

![Graphs showing performance comparisons for different algorithms on various benchmarks.](image-url)
HyLSA: Monitor orecs but read data nonspeculatively

- **TODO**
  - Cache associativity limits capacity
  - Wasting HTM capacity decreases performance!

- **Linked List 8 thr. (LLB8)**
  - Ratio of HTM commits

- **Linked List 8 thr. (LLB8L1)**
  - Ratio of HTM commits

- **Range**
  - Wasting HTM capacity decreases performance!
  - Cache associativity limits capacity
# HTM ratio

- HTM ratio: HTM commits vs total HTM+STM commits
- Only switch to STM if capacity exceeded or if >100 retries

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>LLB8</th>
<th>LLB8L1</th>
<th>LLB256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Skiplist-Small</td>
<td>&lt;1%</td>
<td>HyNOrec: 95-100%</td>
<td>100%</td>
</tr>
<tr>
<td>RBTREE-Large</td>
<td>0-2%</td>
<td>HyNOrec: 95-100%</td>
<td>100%</td>
</tr>
<tr>
<td>RBTREE-Small</td>
<td>2-10%</td>
<td>HyNOrec: 100%</td>
<td>100%</td>
</tr>
<tr>
<td>Hashtable</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>STAMP Genome</td>
<td>HTM, HyNOrec-2: 65%  HyNOrec-1: 60% HyNOrec-0: 50%</td>
<td>HTM: 90-95% HyNOrec:85-90%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Advanced Synchronization Facility (ASF)

- ASF provides **Speculative Regions** (SRs)
  - Similar to transactions: SPECULATE, COMMIT
  - Speculative (LOCK MOV) and nonspeculative loads/stores allowed (selective annotation)
  - Speculative access → ASF monitors cacheline (R/W, W/W conflicts)
  - SR aborts on conflicts, exceeded capacity, far jumps, disallowed instructions
  - Simple guarantees:
    - Minimal capacity
    - SR will eventually commit (unless contention / exceeded capacity / far jumps /disallowed)

DCAS:

```
MOV R8, RAX
MOV R9, RBX
retry:
  SPECULATE
  JNZ retry
  MOV RCX, 1
  LOCK MOV R10, [mem1]
  LOCK MOV RBX, [mem2]
  CMP R8, R10
  JNZ out
  CMP R9, RBX
  JNZ out
  LOCK MOV [mem1], RDI
  LOCK MOV [mem2], RSI
  XOR RCX, RCX
  out:
  COMMIT
  MOV RAX, R10
```
HyNOrec-DSS
[Dalessandro et al.]

- STM base algorithm: NOrec
  - Synchronizes using single global sequence lock (gsl)
  - Write-back(/redo)-style writes, serialized commit phases
  - Time-based validation on gsl plus value-based validation

- Hybrid extension and its problems:
  - Use another global sequence lock (esl)
  - STM acquires/locks both gsl and esl using an HTM txn
    → HTM required on HTM fallback path
  - HTM monitors esl and aborts whenever it changes (STM commit)
    → Every STM commit will abort all concurrent HTM txns!
  - HTM increments gsl speculatively on commit
    → Committing HTM txns conflict with each other
  - Does not scale beyond 4-6 threads in our benchmarks
HyNOrec-2

- Our algorithm avoids all three problems of HyNOrec-DSS

- **esl and gsl can be acquired separately:** gsl protects esl
  \[ \rightarrow \text{No HTM required on STM code path} \]

- **HTM increments gsl with ordinary atomic-increment on commit**
  \[ \rightarrow \text{Nonconflicting HTM txns do not potentially abort each other anymore} \]

- **HTM waits for concurrent STM commits to finish instead of aborting**
  - Spin-wait until esl is not locked after each load
  - Spin-wait until gsl is not locked before committing
  \[ \rightarrow \text{Nonconflicting STM txns do not abort HTM txns anymore} \]