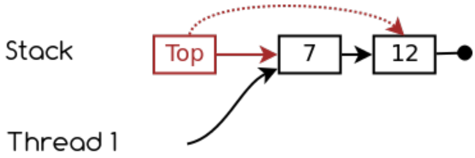
Memory Management for Concurrent Data Structures on Hardware Transactional Memory

Peter Pirkelbauer
Amalee Wilson
Hadia Ahmed
Reed Milewicz

Computer and Information Sciences
University of Alabama at Birmingham

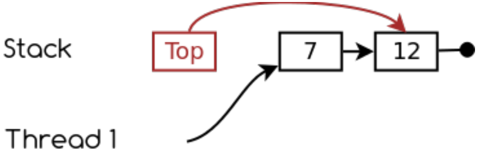
Non-blocking Stack (pop)

```
pair<int, bool> pop(Stack::Top& top) {
  Stack* nexttop = nullptr;
  Stack* currtop = top.load();

  do {
    if (currtop == nullptr) return make_pair(0, false);

    nexttop = currtop->next;
  } while (!top.compare_exchange_strong(currtop, nexttop));

  int res = currtop->val;
  delete currtop; // OK?
  return make_pair(res, true);
}
```

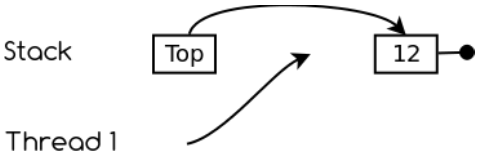


Non-blocking Stack (pop)

```
pair<int, bool> pop(Stack::Top& top) {
  Stack* nexttop = nullptr;
  Stack* currtop = top.load();

  do {
    if (currtop == nullptr) return make_pair(0, false);

    nexttop = currtop->next;
  } while (!top.compare_exchange_strong(currtop, nexttop));

  int res = currtop->val;
  delete currtop; // OK?
  return make_pair(res, true);
}
```



```
pair<int, bool> pop(Stack::Top& top) {
  Stack* nexttop = nullptr;
  Stack* currtop = top.load();
  do {
    if (currtop == nullptr) return make_pair(0, false);
    nexttop = currtop->next;
  } while (!top.compare_exchange_strong(currtop, nexttop));
  int res = currtop->val;
  delete currtop; // OK?
  return make_pair(res, true);
}
```



Solutions

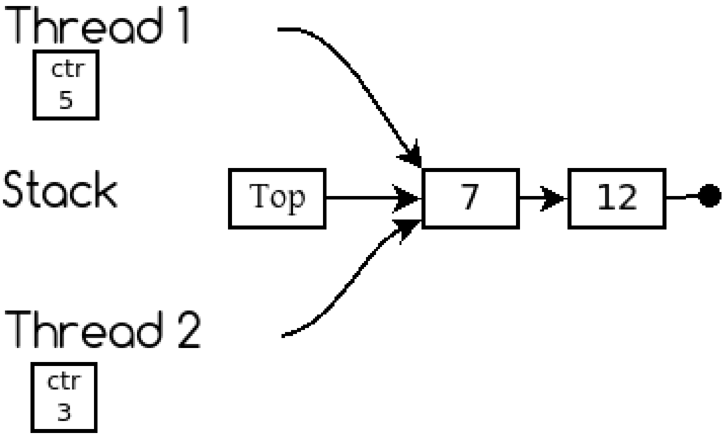
- Garbage collection
- Epochs, Harris (2001)
- Hazard Pointers, Michael (2004)
- Repeated Offender, Herlihy et al. (2004)
- Reference Counting, Gidenstam et al. (2007)
- Optimistic Access, Cohen and Petrank (2015)
- Debra+, Brown (2015)
- QSense, Balmau et al. (2016)

Epochs

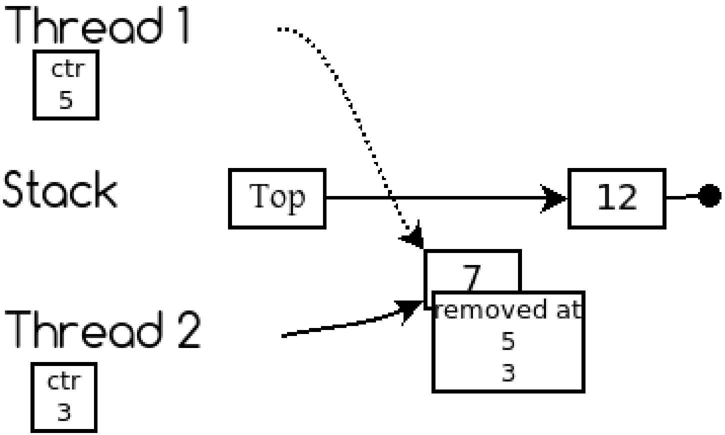
Idea

- Record when a thread starts and finishes an operation.
- Record when an object gets removed.
- Delay deletion until threads are past the deletion time or inactive

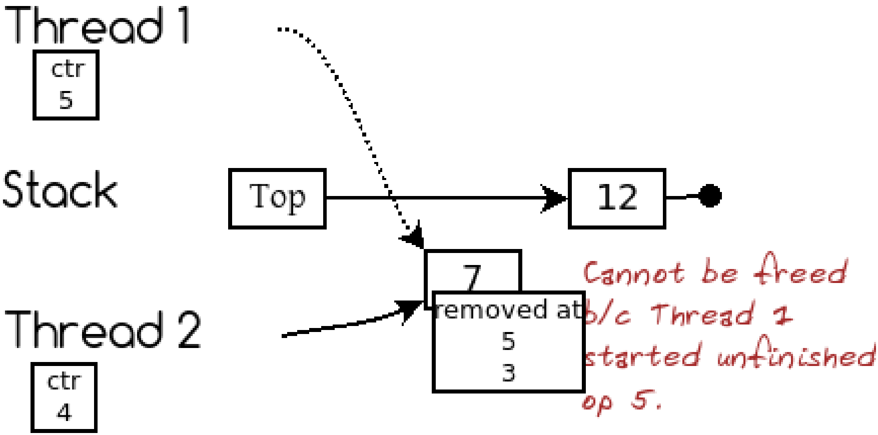
Epochs



Epochs

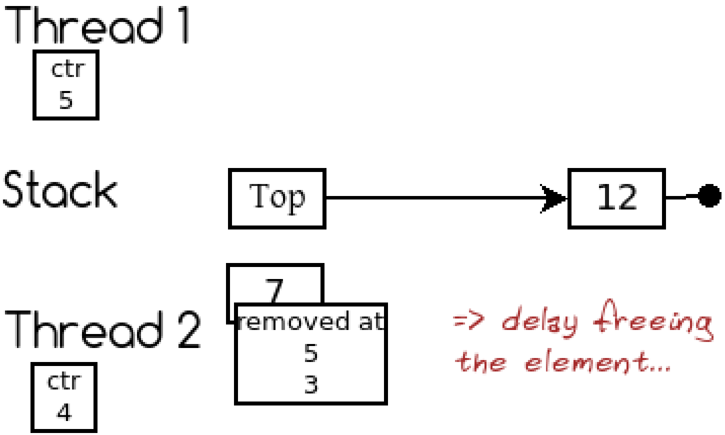


Epochs



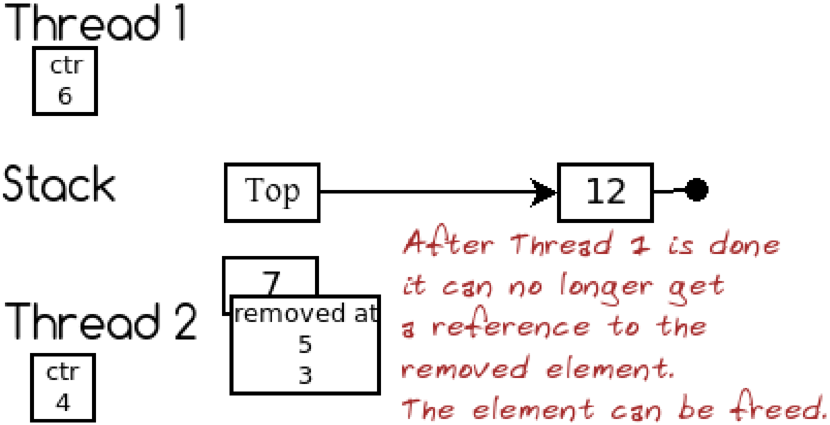
Thread 1
ctr
5
Stack
Top
12
7
removed at
5
3
Thread 2
ctr
4
Cannot be freed
b/c Thread 1
started unfinished
op 5.

Epochs



Thread 1
ctr
5
Stack
Top
12
7
removed at
5
3
Thread 2
ctr
4
=> delay freeing
the element...

Epochs


Thread 1
ctr
6
Stack
Top
12
7
removed at
5
3
Thread 2
ctr
4
After Thread 1 is done
it can no longer get
a reference to the
removed element.
The element can be freed.

Epochs

Benefits

- Low overhead (coarse grain)
- Can bulk free at most every n times, $n > |threads|$

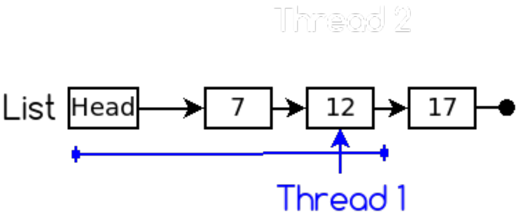
Problems

- What if a thread fails in the middle of an operation?

Hardware Transaction

In principle

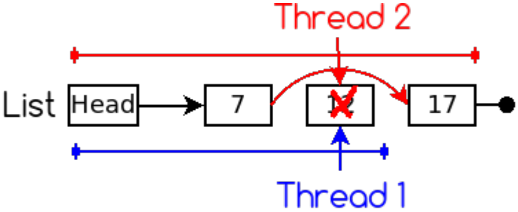
HTM makes memory management easy.



Hardware Transaction and Memory Management

In principle

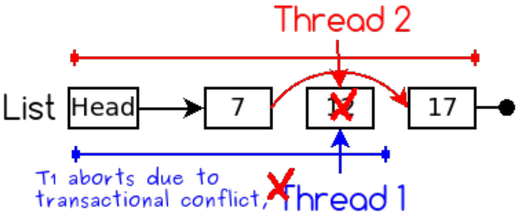
HTM makes memory management easy.



Hardware Transaction and Memory Management

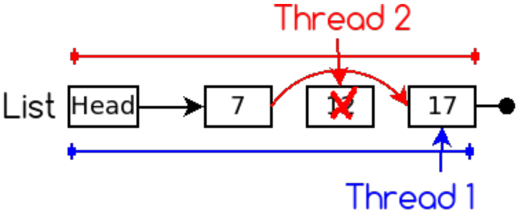
In principle

HTM makes memory management easy.



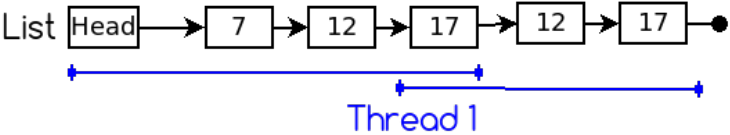
In practice

- Hardware limits transaction size
- It may be undesirable to keep entire data structure in transaction



Partition Access

- Access data structures in partitions.
- Nodes at partition boundaries need to be safe.
- Cost of memory management is reduced greatly.



Partition Size

Benefits of longer transactions

- Memory management is cheap
- Relaxed memory operations
  (Threads sync at begin and end of transaction)
- Amortize cost of transactional overhead over more operations

Drawbacks of longer transactions

- Increases chance of transactional aborts
  - conflict aborts
  - capacity aborts
  - OS aborts

Partition Size

Solution

Compute TX size over rolling average of *n* successful operations.
Dragojević et al., 2011.

Epochs / HTM

Epochs for HTM

Problem

A delayed or failed thread prevents memory from being reclaimed.

Solution

Abort operation executed by delayed thread. (Debra+, Brown (2015)).

HTM

- HTM offers hardware support for aborting operation.
- increment other thread's epoch counter.

Transaction and Epochs

- Start epoch
- Validate epoch in transaction
  - Abort and restart operation if counter is even

Use of Epochs

```
start_epoch();
// transaction partition
if (tx_begin()) {
   ...
   if (!validate_epoch())
      tx_abort();
   ...
   tx_end();
}
end_epoch();
```

Transaction and Epochs

- Start epoch
- Validate epoch in transaction
  - Abort and restart operation if counter is even

Use of Epochs

```
start_epoch();
// transaction partition
if (tx_begin()) {
  ...
  if (!validate_epoch())
    tx_abort();
  ...
  tx_end();
}
end_epoch();
```

Transaction and Epochs

- Start epoch
- Validate epoch in transaction
  - Abort and restart operation if counter is even

Use of Epochs

```
start_epoch();
// transaction partition
if (tx_begin()) {
  ...
  if (!validate_epoch())
    tx_abort();
  ...
  tx_end();
}
end_epoch();
```

Epochs and Reclamation

Removed Object

- Address to be freed
- Epoch vector when object was removed
- Survival count

Thread Aborts

Abort other threads that prevent object reclamation for *n* times.

Epochs - Implementation Details

- A transaction is tried *n* times before partition size is reduced
- Lock-based fall back path
- HTM and C++ atomics

Abort Delayed Thread

```
void cancel_transaction(size_t threadid, size_t currepoch) {
  epoch_t* epochptr = lookup_thread(threadid);
  epochptr->epoch.compare_exchange_strong(currepoch, currepoch+1);
}
```

Epochs Summary

- Small memory and time overhead
- Operation on a data structure requires transaction

Evaluation

Hazard Pointers for HTM

- Publish needed nodes at end of partition
- Scan other threads for nodes that are referenced
- Storing hazard pointers reduces effective transaction size

  Dragojević et al., 2011.

StackTrack

- Publish beginning and end of pointer array on stack
- \+ No overhead for storing hazard pointers
- Collecting other threads pointers requires TX

Alistarh et al. 2014.

Reference Counting

- Each object has a reference counter
- Increment counter / decrement counter if object is no longer needed

\+ No need to scan other thread's pointers

\- frequent conflicts on counter accesses

\- counters reduce effective transaction size significantly

Dragojević et al., 2011.

Systems

Intel Haswell

- 1 socket
- 4 cores
- 8 threads
- Cache line: 64 bytes
- Level 1: 32K, 8-way
- Level 2: 256K, 8-way
- Max TX (load): 4M
- Max TX (store): 32K
- no progress guarantee

IBM Power 8

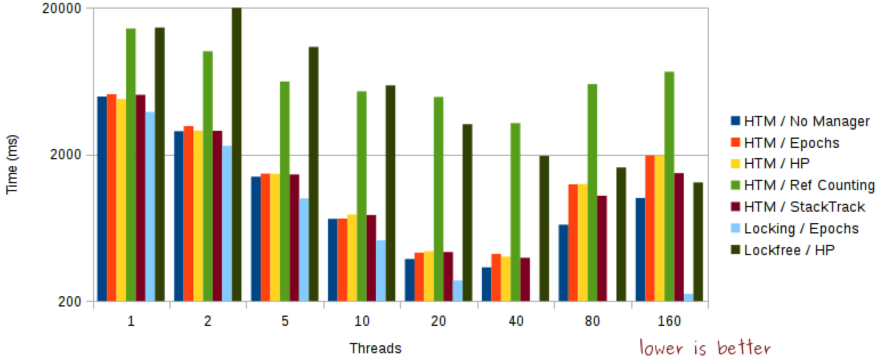
- 2 socket
- 20 cores
- 160 threads
- Cache line: 128 bytes
- Level 1: 64K, 8-way
- Level 2: 512K, 8-way
- Max TX (load): 8K
- Max TX (store): 8K
- no progress guarantee

Evaluation

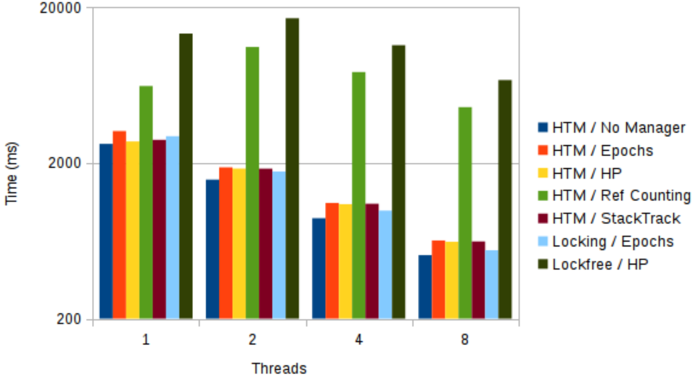
Linked List

- 100K operations (total)
- 10% untimed initial inserts
- alternating insert and erase
- threads access disjoint regions

Evaluation - IBM Power 8



Evaluation - Intel Haswell

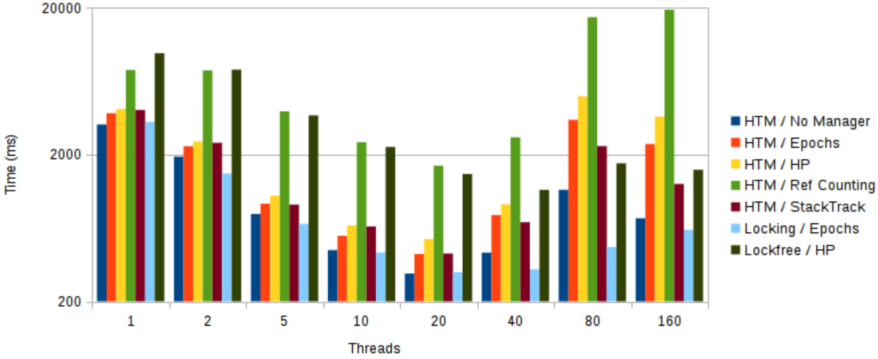


Evaluation

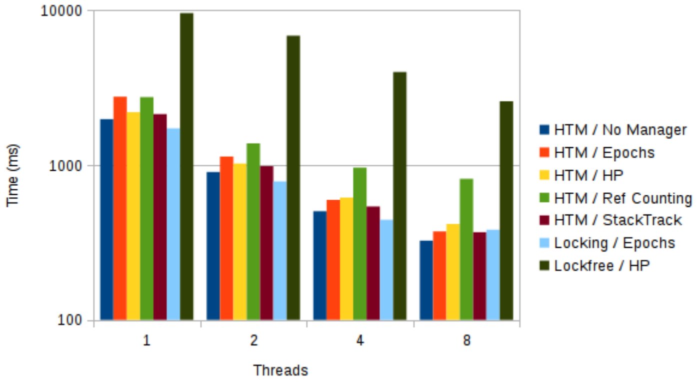
Skip list

- 4M operations (total)
- 10% untimed initial inserts
- alternating insert and erase
- threads access disjoint regions
- Intel: 32 levels, elements in next layer $\frac{1}{2}$
- IBM: 16 levels, elements in next layer $\frac{1}{8}$

Evaluation - IBM Power 8



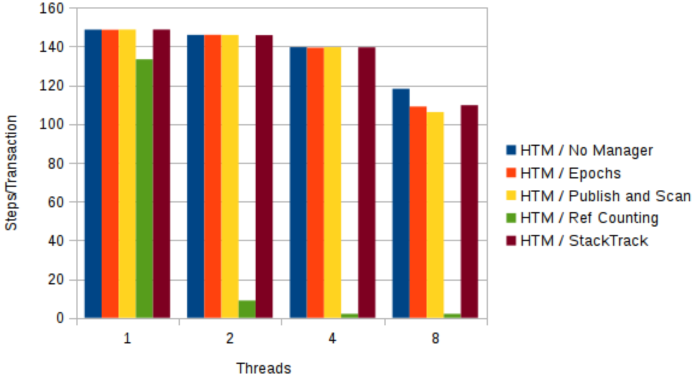
Evaluation - Intel Haswell



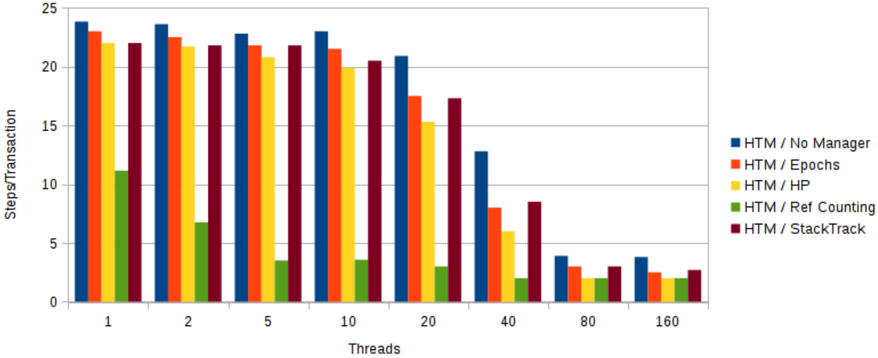
Evaluation

How does the memory management technique impact TX size?

Evaluation - Intel Haswell / Linked list



Evaluation - IBM Power 8 / Skip list



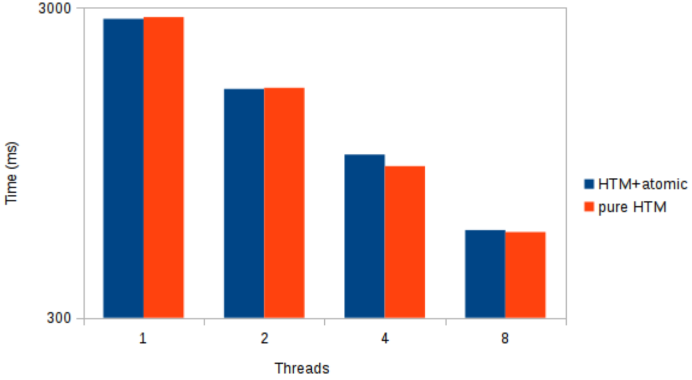
Summary

- Introduced Epochs / HTM
- HTM is a powerful mechanism to speed up common case
- Most memory management techniques perform similarly

- Future Work: Evaluation with other data structures and access patterns

Thank you!

Evaluation - Intel Haswell / Linked list



Evaluation - Intel Haswell / Skip list