Exploring Garbage Collection
with
Haswell Transactional Memory Hardware
Intel’s Haswell

Transactional Memory Extensions (TSX-NI)
Transactional Synchronisation Extensions

(Restricted) Transactional Memory

... with limited processor support
XBEGIN ... XEND
Up to ~16KiB of read and writes
Complexities

Setup of transaction is expensive (3x CAS)
Fallback required if transaction fails
Aborted transactions expensive
Uses for TSX-NI

Simplify parallel collector activities
Facilitate concurrent bitmap marking
Accelerating concurrent copying collection
Concurrent Copying Collection

Sapphire on-the-fly concurrent copying collector

[Hudson and Moss, 2001 & 2003]
Mark Phase

<table>
<thead>
<tr>
<th>from-space</th>
<th>to-space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object 1</td>
<td>Object 1'</td>
</tr>
<tr>
<td>Object 2</td>
<td>Object 2'</td>
</tr>
<tr>
<td>Object 3</td>
<td>Object 3'</td>
</tr>
</tbody>
</table>
Copy Phase

From-space

Object 1

Object 2

Object 3

To-space

Object 1'

Object 2'

Object 3'
Copy Phase

from-space

Object 1 ➔ Object 1'
Object 2 ➔ Object 2'
Object 3 ➔ Object 3'

Mutator

to-space
Copy Phase

from-space

Object 1
Object 2
Object 3

Mutator

to-space

Object 1'
Object 2'
Object 3'
Copy Phase

from-space

Object 1

Object 2

Object 3

to-space

Object 1'

Object 2'

Object 3'

Mutator
Flip Phase

from-space

Object 1

Object 2

Object 3

to-space

Object 1'

Object 2'

Object 3'

Mutator
Finish

Mutator

Object 1'

Object 2'

Object 3'

to-space
Synchronised Replication (Copying Phase)

Mutator updates from-space and to-space

Collector copies data to to-space
u = load (to-space)
v = load (from-space)
if (u != v)
    compare-and-swap (to-space, u, v)
    restart
else
    break
v = load (from-space)
store (to-space, v)
XBEGIN
\[ v_1 = \text{load (from-space)} \]
\[ \text{store (to-space, } v_1) \]
\[ v_2 = \text{load (from-space)} \]
\[ \text{store (to-space, } v_2) \]
\[ \ldots \]
XEND
XBEGIN
copy object 1
copy object 2
...
copy object n
XEND
scan and record object 1 .. n
XBEGIN
copy object 1
copy object 2
...
copy object n
XEND
\( v_1 = \text{load (from–space)} \)
store (to–space, \( v_1 \))
\( v_2 = \text{load (from–space)} \)
store (to–space, \( v_2 \))
...
MFENCE
verify to–space with from–space
Test Setup

Intel Core i7-4770 3.4Ghz, 16GiB RAM
Ubuntu 12.04.3 LTS
DaCapo 2006 (sunflow from 2009)
Fixed 350m heap and GC trigger
Copy speed

bytes copied / time in copy phase
Transactional Penalty

Normal reads          = 11 bytes/ns
Normal writes         = 6 bytes/ns
Transactional reads   = 9 bytes/ns (~80%)
Transactional writes  = 6 bytes/ns (100%)
Peak Performance

Unsafe = 440 bytes/us

440 * 0.8 = 352 bytes/us (exactly as observed)
Scaling
Conclusions (Specific)

Speed up of 48-101% for concurrent copying collection
Weak consistency requirements make STM possible
No improvement for bitmap marking and parallel copying
Conclusions (General)

Transactional work must be *sufficient* and *concise*. Engineering this can add overheads which outweigh benefits.
Future Work

New collector designs which exploit strong consistency
Ground up batching of work for transactions
... mostly barrier free on-the-fly collection?
Questions?

Code: http://github.com/perlfu/sapphire