Sane Semantics of Best-effort HTM
A basis for discussion

Stephan Diestelhorst | September 16, 2010
Atomicity

- Great to reason with
  - All or nothing → not even foot-steps left
  - „Less“ (weaker) atomicity → harder to reason

- Why touch it?
  - We have to (HW constraints)
  - You want it (trust us ;-)  
    - Semi-visible side-effects
    - Mixed-mode accesses
Mixed-mode Accesses

- ASF (and Rock TM) support non-transactional memory accesses from transactions
- Useful:
  - Side-step capacity limitations
  - Export state
  - No conflict aborts
- Semantics tricky
Semantics of Mixed-mode Accesses

- Comments:
  - “One of the most promising features…”
  - “Burden for the programmer to be aware of behaviour”
  - “ASF breaks the memory consistency model”

- Note:
  - AMD64 memory consistency does not cover ASF
  - Canonical extension breaks
Semantics of Mixed-mode Accesses

- Canonical extension of AMD64 memory consistency
  - Stores globally ordered
Semantics of Mixed-mode Accesses

- Canonical extension of AMD64 memory consistency
  - Stores globally ordered

diagram:
- ST1 → ST2
- LOCK ST3 → ST4
Semantics of Mixed-mode Accesses

- Canonical extension of AMD64 memory consistency
  - Stores globally ordered
  - Transactional stores atomic

Diagram:
- ST1 → ST2
- LOCK ST3 → ST4
- LOCK ST1 → LOCK ST2
Semantics of Mixed-mode Accesses

- Canonical extension of AMD64 memory consistency
  - Stores globally ordered
  - Transactional stores atomic
- All stores inside transaction are atomic
Semantics of Mixed-mode Accesses

- Canonical extension of AMD64 memory consistency
  - Stores globally ordered
  - Transactional stores atomic
- All stores inside transaction are atomic
- Breaks external synchronisation
External Synchronisation Example

Shared: \( a = b = c = 0 \);

Thread 1:
```
tx_start();
...
  nontx_store(*a, 1);
  while(!nontx_load(*b))
    tx_store(*c, 1);
  tx_commit();
```

Thread 2:
```
tx_start();
...
  while(!nontx_load(*a))
  nontx_store(*b, 1);
  tx_commit();
```
External Synchronisation Example

Shared: $a = b = c = 0$

Thread 1:
```
tx_start();
...
nontx_store(*a, 1);
while(!nontx_load(*b))
tx_store(*c, 1);
```

Thread 2:
```
tx_start();
...
while(!nontx_load(*a))
nontx_store(*b, 1);
```

Diagram: shows the execution flow of Thread 1 and Thread 2 with shared variables $a$, $b$, and $c$. The diagram illustrates the synchronization points and the flow of execution.
Mixed-mode Accesses in ASF

- Two classes: transactional ↔ non-transactional
- Consistent inside class
- Merged at COMMIT instruction
  - Use of SFENCE to enforce ordering
Semi-visible Side-effects

- Applications do not care about some (architectural) side-effects → semi-visible
  - Page-faults
  - Lazy FPU switching
  - Software managed TLBs
- Expose on abort?
  - Difficult to fixup in app → ASF exposes
Semi-visible Side-effects

- Applications do not care about some (architectural) side-effects $\rightarrow$ semi-visible
  - Page-faults
  - Lazy FPU switching
  - Software managed TLBs
- Expose on abort?
  - Difficult to fixup in app $\rightarrow$ ASF exposes
Abort Timing, Opacity and Isolation

- Strong isolation & opacity are great
- Opacity vs. abort timeliness

```c
hytm_load(addr) {  
    ...
    asf_load(hash(addr));
    ...
    return load(addr);
}
```

```c
hytm_store(addr, val) {  
    ...
    asf_store(hash(addr));
    addr:=val;
    ...
}
```

- AMD64 memory consistency does not apply
- Conflict detection arming
Abort Timing, Opacity and Isolation

- Strong isolation & opacity are great
- Opacity vs. abort timeliness

```c
hytm_load(addr) {
    ...
    asf_load(hash(addr));
    ...
    return load(addr);
}
```

```c
hytm_store(addr, val) {
    ...
    asf_store(hash(addr));
    asf_store(addr:=val);
    ...
}
```

- AMD64 memory consistency does not apply
- Conflict detection arming
Abort Timing, Opacity and Isolation

- Strong isolation & opacity are great
- Opacity vs. abort timeliness

```c
hytm_load(addr) {
    ...
    asf_load(hash(addr));
    ...
    return load(addr);
}
```

```c
hytm_store(addr, val) {
    ...
    asf_store(addr:=val);
    asf_store(hash(addr));
    ...
}
```

- AMD64 memory consistency does not apply
- Conflict detection arming
Spurious Aborts and Progress

- **Abort**
  - When: unavoidable ↔ ... ↔ always
  - Whom: minimal set ↔ ... ↔ everybody
- **Requester wins**
- **Spurious aborts**
  - Capacity
  - Semi-architectural
  - Out-of-order speculation
- **Difficult / impossible to hide**
Spurious Aborts and Progress

- ASF ensures eventual forward progress
  - No contention
    - Grab the global lock if too crazy
  - Tight lower capacity (four cache lines)
    - Can have larger working sets
  - No exceptions, etc.
    - Semi-architectural ones visible to OS, vanish eventually
Trademark Attribution

AMD, the AMD Arrow logo and combinations thereof are trademarks of Advanced Micro Devices, Inc. in the United States and/or other jurisdictions. Other names used in this presentation are for identification purposes only and may be trademarks of their respective owners.

©2009 Advanced Micro Devices, Inc. All rights reserved.