MANCHESTER 1824

The University of Manchester

Chip Multiprocessors COMP35112

Lecture 12 - Hardware Support for Transactional Memory

Graham Riley

Motivation

- There is a lot of interest in Transactional Memory as it promises to provide a simplified programming model when parallel threads share updateable memory (i.e. to replace locks and barriers)
- There are quite a few software implementations available supporting both compiled (C) and managed (Java) languages
- However, performance is not good for programs with a significant amount of sharing

Hardware Support

- So, is it possible to provide support in hardware which will make it more efficient?
- Real processor manufacturers are exploring this
  - Sun Microsystems announced the 'Rock' processor in 2008 (unfortunately abandoned in 2009)
  - IBM BlueGene/Q and Power 8 have some support
  - AMD have published proposed TM extensions to x86 instruction set (ASF)
  - Intel announced TSX APIs for the Haswell x86 architecture (subsequently found to contain an implementation error)
    - In 2017, in some Skylake processors but disabled in others(!)
    - Also Intel have RTM (Restricted TM) available
  - ARM are rumoured to be working on it… but not clear

Basis of TM Operation

- Declare sections of code to be ‘atomic’ (a transaction)
- When in an atomic section:
  - Keep note of all addresses loaded from (readset)
  - Buffer all writes – keep local values – do not write them to main shared memory (writeset – lazy versioning)
- If it is detected that another thread writes to something in readset (conflict), abort transaction and restart
- If end of transaction is reached successfully, commit writeset to main memory

Load Linked – Store Conditional

- … is a transaction, that is why it appears atomic

```
ldl
.
stc
```
} atomic section

- Readset is the single variable being RMW accessed – address is kept in load linked register
- Bus snooping detects any write to this readset
- Load linked flag is cleared if conflict detected
  - commits (writes value) if no conflict
  - otherwise jump back and try again (abort and restart)

Cache-based TM

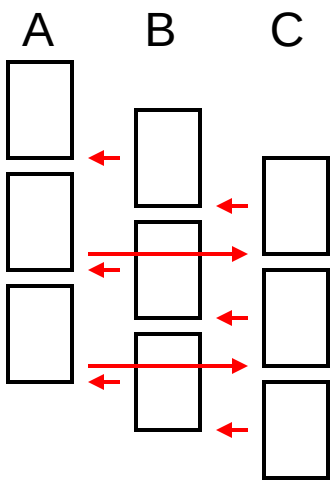
- Complex transactions can involve a significant number of variables read and written but we can extend the load linked/store conditional approach
- Cache can be regarded as the buffer to store both readset and writeset
- Need to add extra tags to cache entries to say this is data read/written inside an atomic block
- Modified writeback protocol – writes of transactional data are only made from cache to main memory when a transaction commits

Cache-based TM (continued)

- Snooping protocol is modified
- When writing a transactional variable, an invalidate is broadcast as usual
- Any core seeing an invalidate which matches one of its readset entries must invalidate all its transactional data, abandon the transaction and restart it
- When a transaction terminates, it can commit (flush) its writeset to main memory (may require some synchronisation – e.g. it cannot be aborted once it has started to commit)

Issues with Cache-based TM

- Limited cache size – limited readset/writeset
- Cache conflicts must not displace transactional data – it cannot be lost but neither can it be written to memory (until commit)
- Mechanism is naturally ‘eager conflict detection’ (eager validation) – risk of livelock
  - B causes A to abort – A restarts
  - C causes B to abort – B restarts
  - A causes C to abort – C restarts



Alternative Approaches

- Systems based on a snooping bus are likely to have similar problems (concerning extensibility) to those encountered in cache coherence
- More radical TM proposals advocate abandoning cache coherence entirely and making all shared data transactional (i.e. only accessible from within an atomic section)
- Memory mechanisms can then be modified significantly to (hopefully) overcome extensibility restrictions

TCC

- **T**ransactional **C**oherence and **C**onsistency
- Developed by Stanford University tcc.stanford.edu
  - From around 2004… quite mature by 2012-ish
- Original proposal is very simple
- Large number of optimisations possible
- Has been simulated with encouraging results
- As yet no practical implementations

TCC Principles

- Each core still has a local cache in which it stores its readset during a transaction – again with tags to differentiate from other cached data
- However, writes are stored in a separate buffer (can simply be a RAM-based queue)
- Writes to shared data only occur in transactions – they are therefore buffered locally and there is no need to broadcast invalidations

TCC Principles (continued)

- A transaction reaching its end successfully must commit its data to main memory
- However, before doing this, it broadcasts all the addresses in its write buffer to all other cores (as a single packet communication)
- If they are executing a transaction, they must compare all the addresses in the packet with their readset in cache
- If there are any matches, the associated comparing transaction must abort and restart

TCC Principles (continued)

- But to make this work, no two transactions must be trying to commit conflicting data at the same time
- Implies a central ‘permission to commit’ resource which could be a bottleneck
- If it can be determined that readsets/writesets do not overlap (e.g. that the transactions never accessed the same memory bank) then commits can occur concurrently

TCC Properties

- TCC is lazy conflict detection (validation)
- A transaction is never aborted until it is certain that the one causing conflict is about to commit – no livelock
- Inter-core communication only occurs at transaction commit – so less synchronisation
- Mechanisms do not require ‘instantaneous’ communication (like cache coherence does)
- Potentially allows looser communication between cores – hence extensibility

Hybrid TM

- Proposed hardware schemes still require caches and buffers
- These are inevitably of limited size
- If they overflow (readset and/or writeset too big) there is a problem
- General opinion (currently) seems to be that it is good to keep readsets/writesets relatively small
- Logic – large readsets/writesets mean higher conflict probability, hence more wasted work

Hybrid TM (continued)

- A popular suggestion is therefore that some hardware support is provided for small readsets/writesets but with a resort to software TM if this is insufficient
- e.g. the AMD ASF (Advanced Synchronisation Facility) only guarantees hardware support for four transactional cache lines
- There are known transactional applications for which this is almost certainly too small
- The jury is still out!

Next Lecture

- We have overlooked some potentially damaging effects of possible multi-core memory behaviours
- In particular, in certain circumstances the memory could appear to behave inconsistently
- We analyse this problem in the next lecture