Transactional Memory Coherence and Consistency

Presented to CS258 on 4/28/08
by David McGrogan

Review: Multiproc configurations

- Message passing
  - Simple hardware
  - Tricky to program
- Shared memory
  - Complex hardware
  - No implicit synchronization
  - Sometimes easier to program

Something new: TCC

Transactional memory Coherence and Consistency model

- Provides transactional shared memory
- Requires specialty hardware
- Relies on broadcast and snooping
- Not infinitely scalable

Something new: TCC

Transactional memory Coherence and Consistency model

- Implements CCR (Conditional Critical Regions):

```
atomic (condition) {
     statements;
}
```

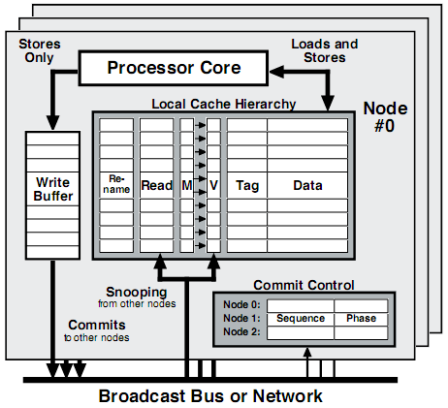
TCC protocol

- Writes in a transaction stored locally
- System-wide commit arbitration
- All writes broadcasted as a packet
- Different commits cannot overlap
- Transactions can be ordered via hardware-managed 'phase numbers'

TCC hardware

- Extra cache bits
- Write buffer
- Commit control

- Optional:
  Shadow registers
  Victim buffer



TCC hardware

- Cache bits
  - Read (can be multiple per line)
  - Modified
  - Renamed (multiple) – entire associated word/byte has been written, so future reads can't cause violations

TCC hardware

- Write buffer
  - Hardware-based packet buffer
  - Lets processor do more important things
- Commit control
  - Stores phase numbers for all other nodes
  - Reduces arbitration request traffic greatly

Optional TCC hardware

- Shadow registers
  - Copy core registers upon transaction start
  - Permits rollback
  - Can be done in software
- Victim buffer
  - Holds flushed read/modified cache lines
  - If absent, must wait for and hold commit permission until transaction commits

Errata

- I/O
  - Input can only be accepted if rollback is impossible
  - Commit permission requested immediately
- Hardware optimizations
  - Break up large transactions
  - Merge small transactions
  - Force forward progress by increasing phase

Cache state size

- Most benchmarks operate within 12 kB of read state and 8 kB of write state
- Easily fits in the smallest of modern caches

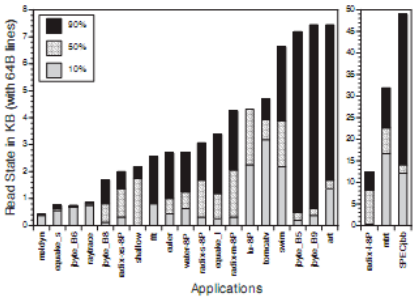


Figure 6: State read by individual transactions with store buffer granularity of 64-byte cache lines. We show state required by the smallest 10%, 50%, and 90% of iterations.

Performance (ideal)

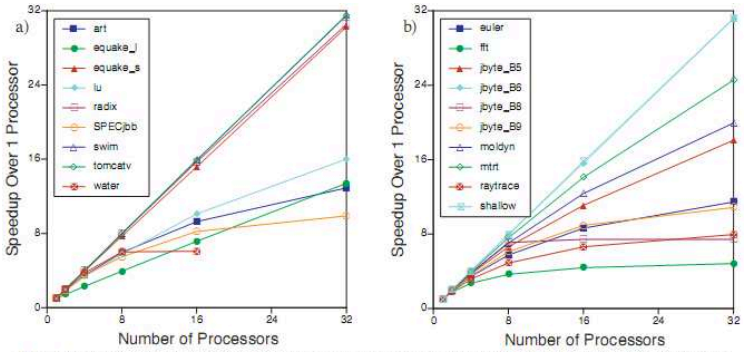


Figure 4: Speedups for varying numbers of processors with our manually parallelized benchmarks (a) and Java benchmarks with automated parallelism (b) on a perfect TCC system with 1 IPC processors, no memory delays, and ∞ commit bandwidth.

- Limited by sequential code, load imbalance, inter-transaction dependencies

Performance Errata

- More realistic simulations
  - Viability retained in most cases
  - Sometimes crippled by very high commit arbitration times (200 cycles)
- Various optimizations had little effect
  - Double-buffering write buffer
  - Extra read bits (sub-line read records)