LogTM: Log-Based Transactional Memory

Kevin E. Moore, Jayaram Bobba, Michelle J. Moravan, Mark D. Hill, & David A. Wood

Presented by Colleen Lewis

Credits

- Animations from the original LogTM HPCA presentation
- Original graphs modified for readability

Big Picture

- Hardware transaction motivation
- Per thread log
- Optimize commits (Hardware)

Design Decisions

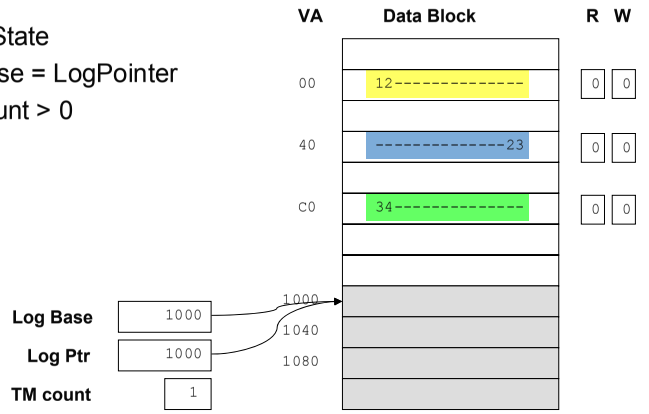
- Version Management
  - Eager – write in place
  - Lazy – write on commit
- Conflict Detection
  - Eager – detect at read/write time
  - Lazy – detect at commit time

Transaction Logs

- Pointer to the beginning of the log
- Pointer to the end of the log
- Read and Write bits for each cache line

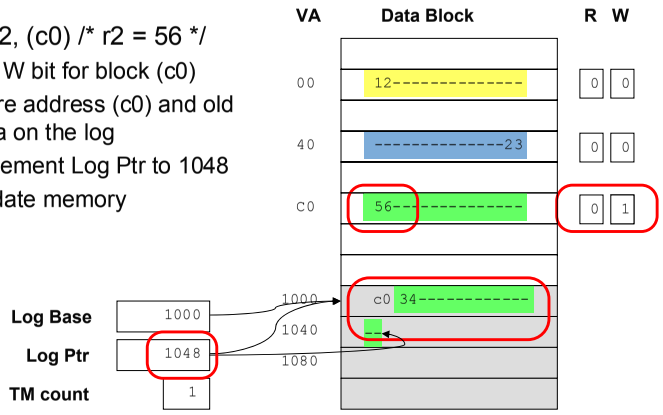
Transaction Log Example

- Initial State
- LogBase = LogPointer
- TM count > 0



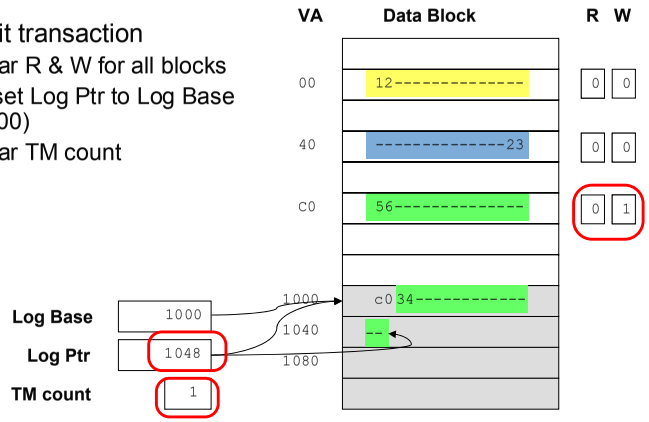
Transaction Log Example

- Store r2, (c0) /* r2 = 56 */
  - Set W bit for block (c0)
  - Store address (c0) and old data on the log
  - Increment Log Ptr to 1048
  - Update memory



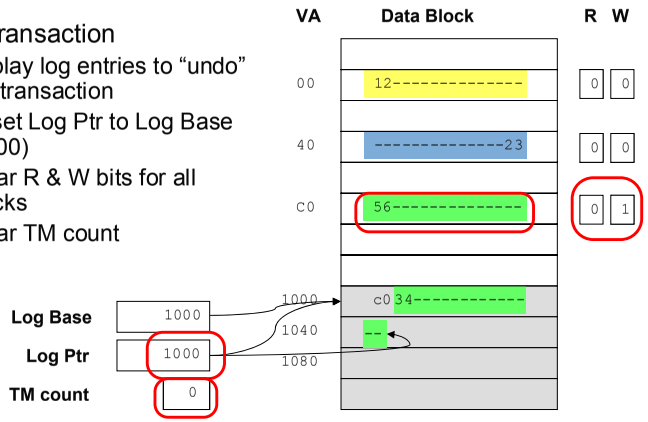
Transaction Log Example

- Commit transaction
  - Clear R & W for all blocks
  - Reset Log Ptr to Log Base (1000)
  - Clear TM count



Transaction Log Example

- Abort transaction
  - Replay log entries to "undo" the transaction
  - Reset Log Ptr to Log Base (1000)
  - Clear R & W bits for all blocks
  - Clear TM count

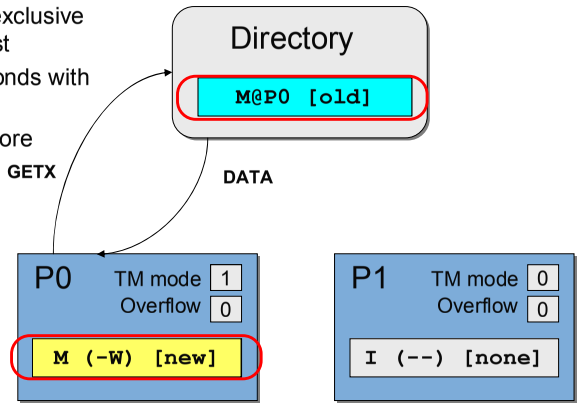


Conflict Detection

- Checked at every read/write
- Directory forwards read requests
- Directory can have "sticky" data
- Individual nodes responsible for detecting conflicts
- Needs
  - Transaction mode bit
  - Overflow bit

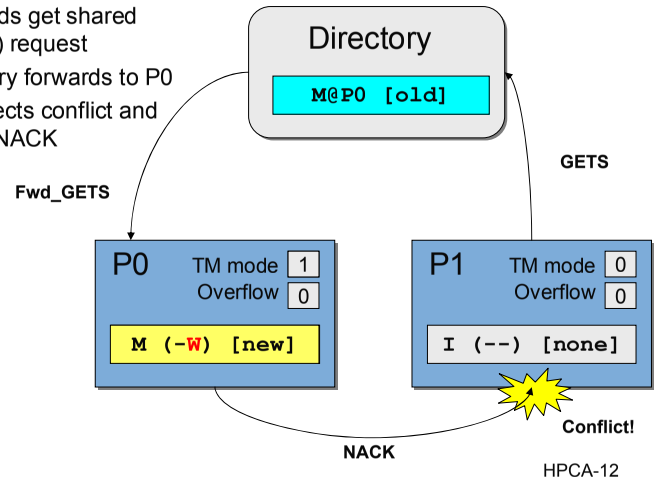
Conflict Detection (example)

- P0 store
  - P0 sends get exclusive (GETX) request
  - Directory responds with data (old)
  - P0 executes store



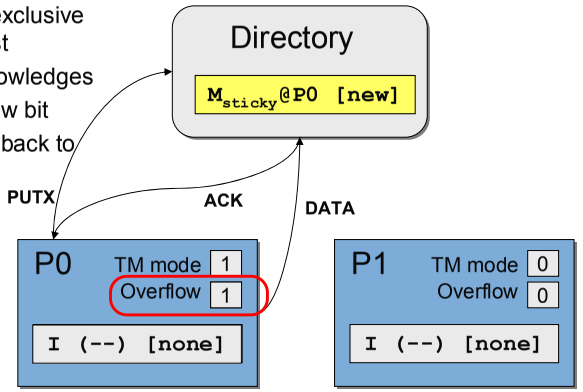
Conflict Detection (example)

- In-cache transaction conflict
  - P1 sends get shared (GETS) request
  - Directory forwards to P0
  - P0 detects conflict and sends NACK



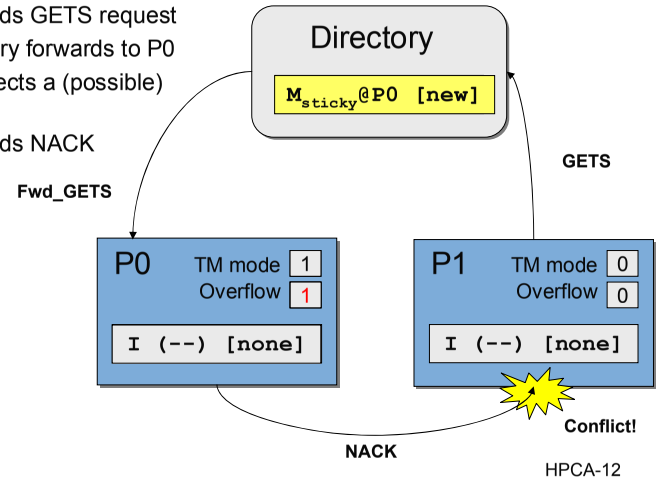
Conflict Detection (example)

- Cache overflow
  - P0 sends put exclusive (PUTX) request
  - Directory acknowledges
  - P0 sets overflow bit
  - P0 writes data back to memory



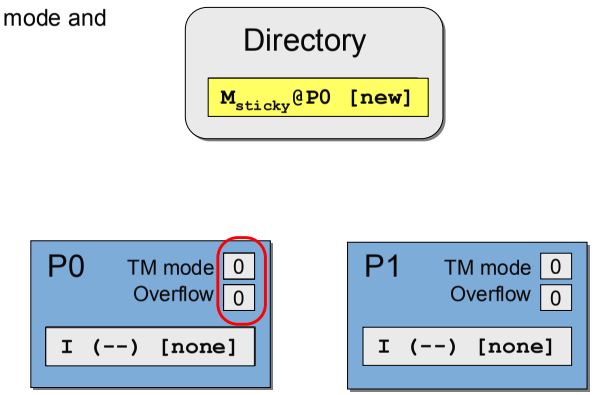
Conflict Detection (example)

- Out-of-cache conflict
  - P1 sends GETS request
  - Directory forwards to P0
  - P0 detects a (possible) conflict
  - P0 sends NACK



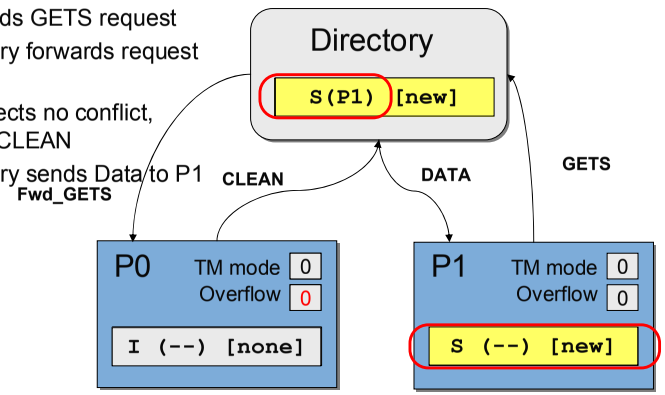
Conflict Detection (example)

- Commit
  - P0 clears TM mode and Overflow bits



Conflict Detection (example)

- Lazy cleanup
  - P1 sends GETS request
  - Directory forwards request to P0
  - P0 detects no conflict, sends CLEAN
  - Directory sends Data to P1

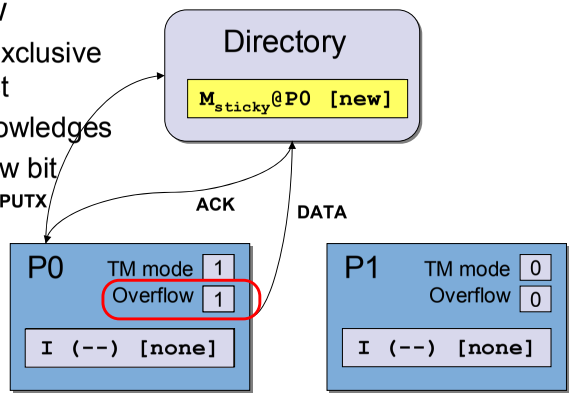


False Positives?

- What if P0 has started a new transaction without cleaning the sticky data?

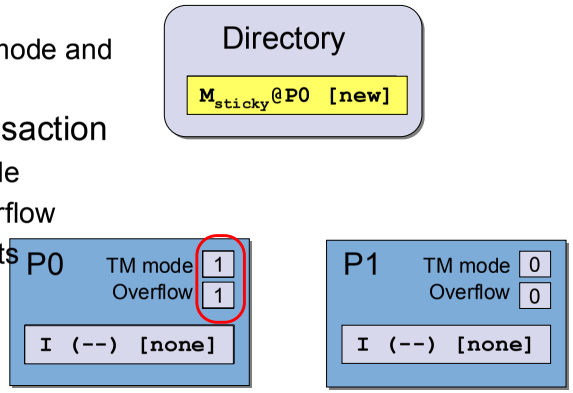
False Positive Example

- Cache overflow
  - P0 sends put exclusive (PUTX) request
  - Directory acknowledges
  - P0 sets overflow bit
  - P0 writes data back to memory



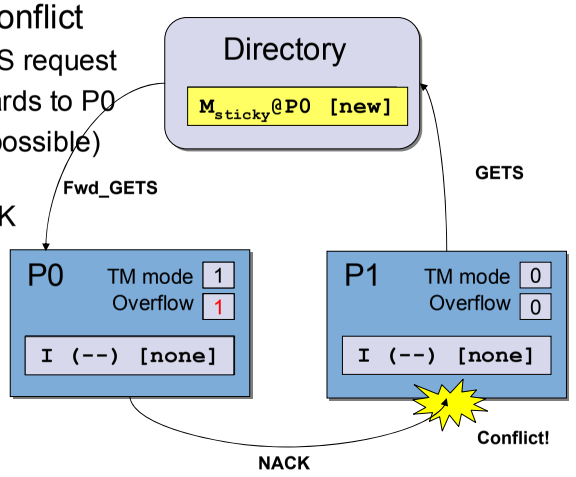
False Positive Example

- Commit
  - P0 clears TM mode and Overflow bits
- Start New Transaction
  - P0 set TM mode
  - Eventually overflow
  - Set overflow bits



Conflict Detection (example)

- Out-of-cache conflict
  - P1 sends GETS request
  - Directory forwards to P0
  - P0 detects a (possible) conflict
  - P0 sends NACK



Conflict Resolution and Deadlock Avoidance

- Options
  - Wait – risk deadlock?
  - Abort – risk livelock?
- Current Behavior
  - Wait
  - Abort if waiting on a logically younger process
- Future Behavior?
  - Software contention manager

Evaluation

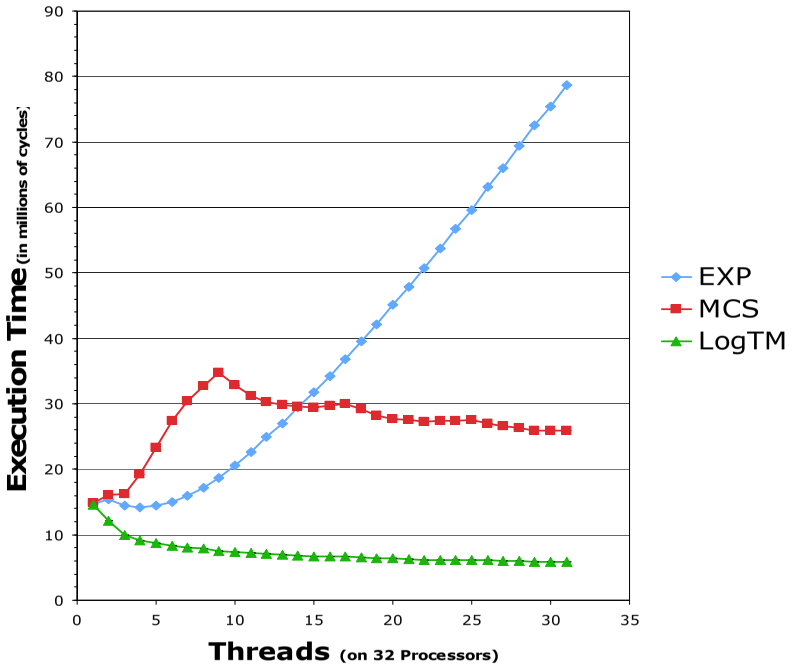
- 32 SPARC processors
- Solaris 9 OS
- SIMICS – full system simulator
  - Magic no-ops
- Tests
  - Micro-benchmarks
  - SPLASH suite

Microbenchmarks

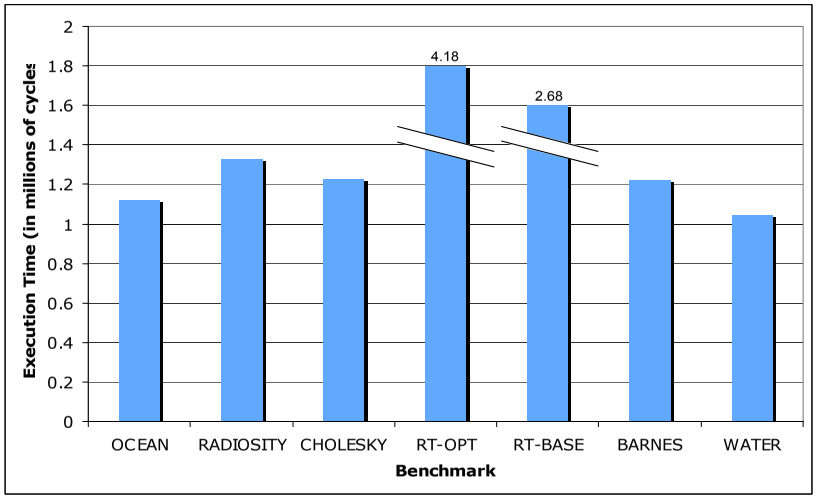
- High Contention / Short Transactions

```
BEGIN_TRANSACTION();

 new_total = total.count + 1;
 private_data[id].count++;
 total.count = new_total;

COMMIT_TRANSACTION();
```

- Comparing:
  - EXP - TTS locks with exponential backoff
  - MCS – SW Queue based locks



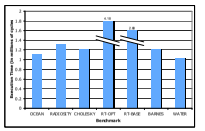
SPLASH2 Benchmark Results



SPLASH2 Benchmark Results

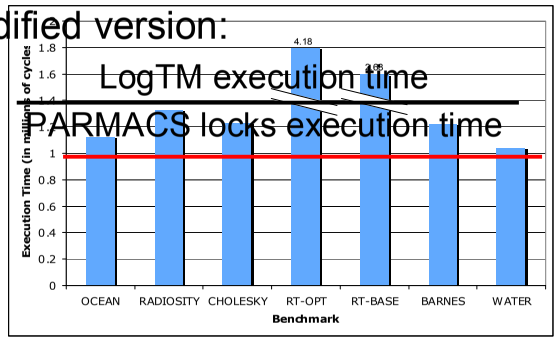
- Data presented as:

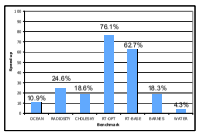
$$\frac{\text{PARMACS locks execution time}}{\text{LogTM execution time}}$$



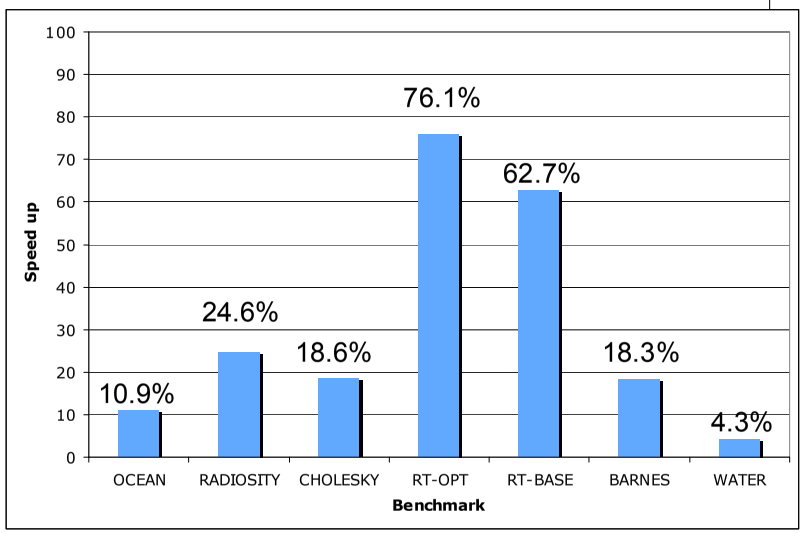
- Modified version:

$$1 - \frac{\text{LogTM execution time}}{\text{PARMACS locks execution time}}$$





SPLASH2 Benchmark Results



Conclusions

- Optimize commits
- Aborts handled by software
- Stall to avoid wasting work
- Allow sticky data because overflow is rare
- Good performance on microbenchmark
- False sharing has a big impacts on LogTM

Questions?