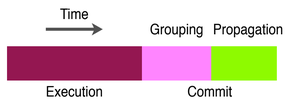
BulkCommit: Scalable and Fast Commit of Atomic Blocks in a Lazy Multiprocessor Environment

Xuehai Qian, Josep Torrellas（University of Illinois, Urbana-Champaign)
Benjamin Sahelices (Universidad de Valladolid), Depei Qian (Beihang University)

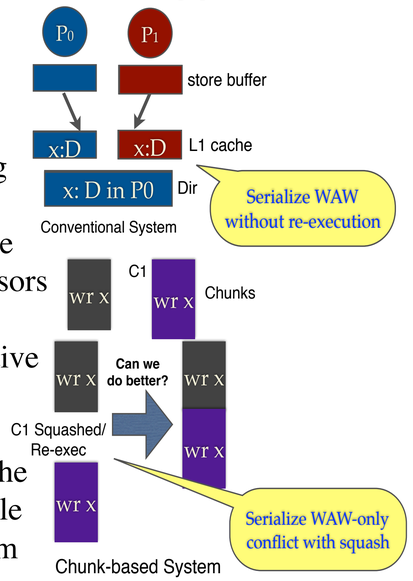
1. Motivation

- Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC [Ceze'07])
  - Chunk: a group of dynamically contiguous instructions executed atomically
  - Providing performance and programmability advantages [Hammond'04][Ahn'09]
  - Chunk commit is an important operation: making the state of a chunk visible atomically
- Lazy detection of conflicts provides higher concurrency in codes with more conflicts
- In a lazy directory-based cache coherent system, parallelizing the commit is challenging
  - Requires the consistent conflict resolution decision over all the distributed directory modules
  - Therefore, the current schemes use sequential commit operations
- In addition, the current lazy conflict resolution is sub-optimal
  - No squash is required if the conflict is only Write-After-Write (WAW)

2. Inefficiencies of Current Approaches



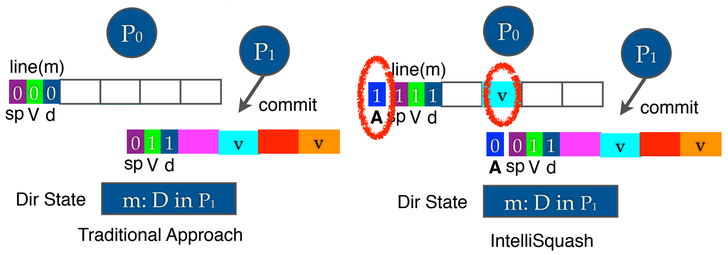
- Execution:
  - Reads and writes bring lines into cache
  - No written line is made visible to other processors
- Commit:
  - Grouping: set the relative order of any two conflicting chunks
  - Propagation: making the stores in a chunk visible to the rest of the system



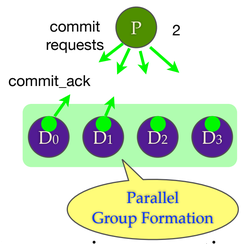
	Grouping	Fine-grained Conflict	Broadcast?
Scalable TCC [Chafi'07]	Parallel	No	Yes
SRC [Pugsley'08]	Sequential	No	No
Scalable Bulk [Qian'10]	Sequential	Yes	No
BulkCommit	Parallel	Yes	No

3. BulkCommit = IntelliSquash + IntelliCommit

- **IntelliSquash**: no squash on WAW-only conflicts
- Challenge: the speculative data produced by a chunk cannot be lost when the chunk is ready to commit
- Solution: L1 cache as the "store buffer" for a chunk
  - Similar to the store buffer in the conventional system
  - On receiving invalidation, the speculative dirty words of a line is preserved
- Extra hardware cost:
  - Absent bit: it is set when
    - The line is not presented
    - The line contains some speculative words
- Per-word dirty bit (not shown)



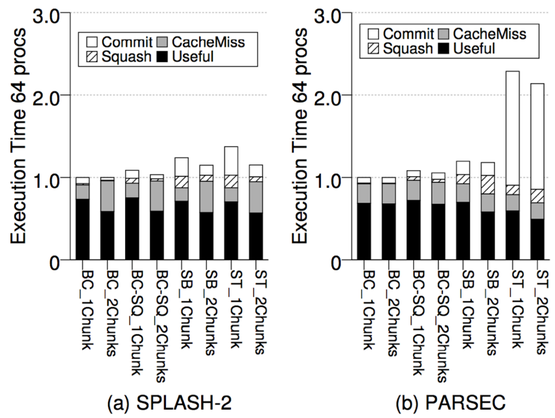
- **IntelliCommit**: parallel grouping
- On chunk commit:
  - Processor sends commit requests in parallel
  - Directory module receives commit request:
    - Lock the lines and responds with commit_ack
    - Processor counts #commit_acks received
  - Processor sends commit_confirm in parallel
- Challenge: resolving conflicts from two processors
- ChunkSort: ordering all the conflicting chunks in the same order in all relevant directories by preemption



- Messages in preemption:
  - preeemp_{request,finish} (D➔P)
  - preeemp_{ack,nack} (P➔D)
- Commit Ack Counter (CAC)
- Preemption Vector (PV) (N=#P=#D)
  - Each processor: N counters of size log(N)
  - PV[i] at Pj=k
    - Pj's chunk is preempted by Pi's chunk in k Dirs
  - Inc PV[i]: @send preempt_ack
  - Dec PV[i]: @recv preempt_finish
  - When to send commit_confirm?
    - (CAC==0)&&(for each i, PV[i]==0)

4. Experimental Evaluation

- Cycle accurate NOC simulation with processor and cache model
- Number of cores: 16 and 64
- 11 SPLASH-2 and 7 PARSEC applications
- One or two outstanding chunks
- Implemented most distributed commit protocols:
  - Scalable TCC (ST)
  - Scalable Bulk (SB)
  - BulkCommit without IntelliSquash (BC-SQ)
  - BulkCommit (BC)



5. Conclusion

- Proposed **BulkCommit**: commit protocol with parallel grouping and squash-free WAW-only conflict resolution
- Key properties:
  - Serializing WAW between chunks without squashing
    - Exploiting the similarity of the chunk commit and the individual store
  - Parallel grouping
    - Using preemption mechanisms to order two conflicting chunks consistently
- We hope that BulkCommit achieves the optimal design point

[Ahn 09] W. Ahn, S. Qi, J.-W. Lee, M. Nicolaides, X. Fang, J. Torrellas, D. Wong and S. Midkiff. BulkCompiler: High-Performance Sequential Consistency through Coperative Compiler and Hardware Support, MICRO'09.
[Ceze 07] L. Ceze, J. M. Tuck, P. Montesinos, and J. Torrellas. BulkSC: Bulk Enforcement of Sequential Consistency. ISCA'07
[Chafi 07] H. Chafi, J. Casper, B. D. Carlstrom, A. McDonald, C. Cao Minh, W. Baek, C. Kozyrakis, and K. Olukotun. A Scalable, Non-blocking Approach to Transactional Memory. HPCA'07.
[Hammond 04] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun. Transactional Memory Coherence and Consistency. ISCA'04
[Pugsley 08] S. Pugsley, M. Awasthi, N. Madan, N. Muralimanohar, and R. Balasubramonian. Scalable and Reliable Communication for Hardware Transactional Memory. PACT'08
[Qian 10] X. Qian, W. Ahn, and J. Torrellas. ScalableBulk: Scalable Cache Coherence for Atomic Blocks in a Lazy Environment. MICRO'10