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

Xuehai Qian, Benjamin Sahelices
Josep Torrellas, Depei Qian
University of Illinois

University of Illinois

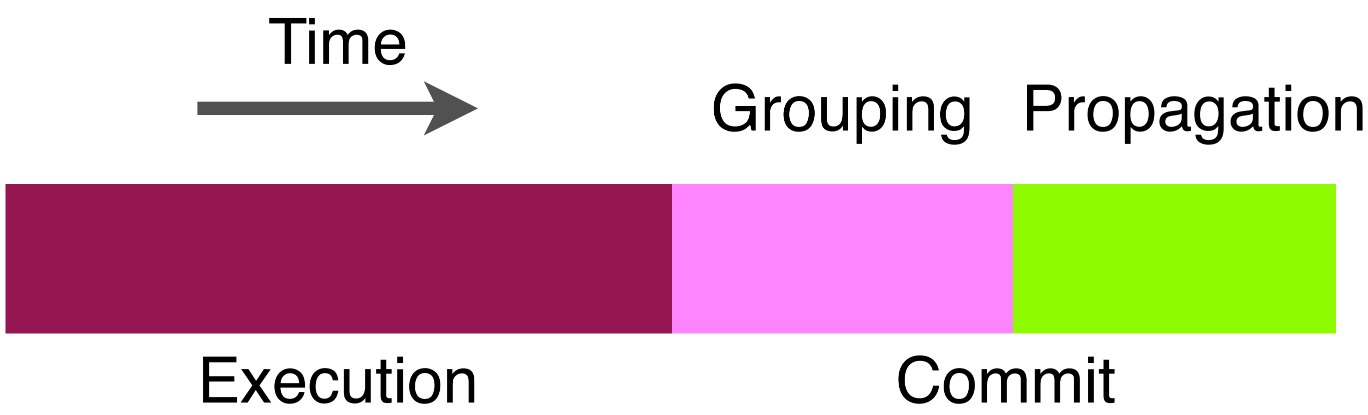
http://iacoma.cs.uiuc.edu/

i-acoma group

Motivation

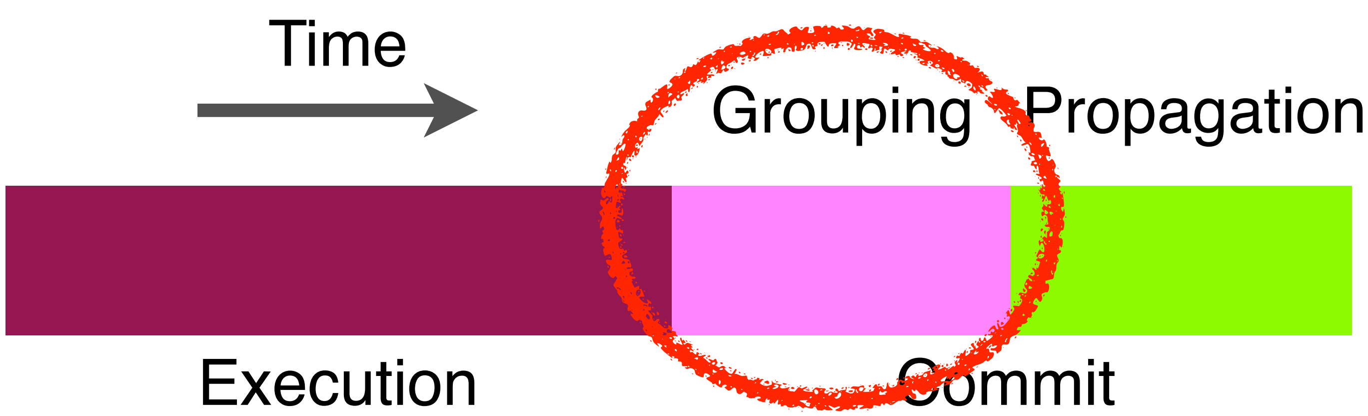
- Architectures that continuously execute Atomic Blocks or Chunks (e.g., TCC, BulkSC)
  - Chunk: a group of dynamically contiguous instructions executed atomically
  - Providing performance and programmability advantages [Hammond 04][Ahn 09]
  - Chunk commit is an important operation: making the state of a chunk visible atomically
- We focus on the designs with lazy detection of conflicts
  - Provides higher concurrency in codes with high conflicts
  - Parallelizing the commit is challenging
    - Requires the consistent conflict resolution decision over all the distributed directory modules
    - Therefore, most current schemes have some sequential steps in the commit
- In addition, the current lazy conflict resolutions are sub-optimal
  - Incur the squash when there is only Write-After-Write (WAW) conflict

Lifetime of a Chunk



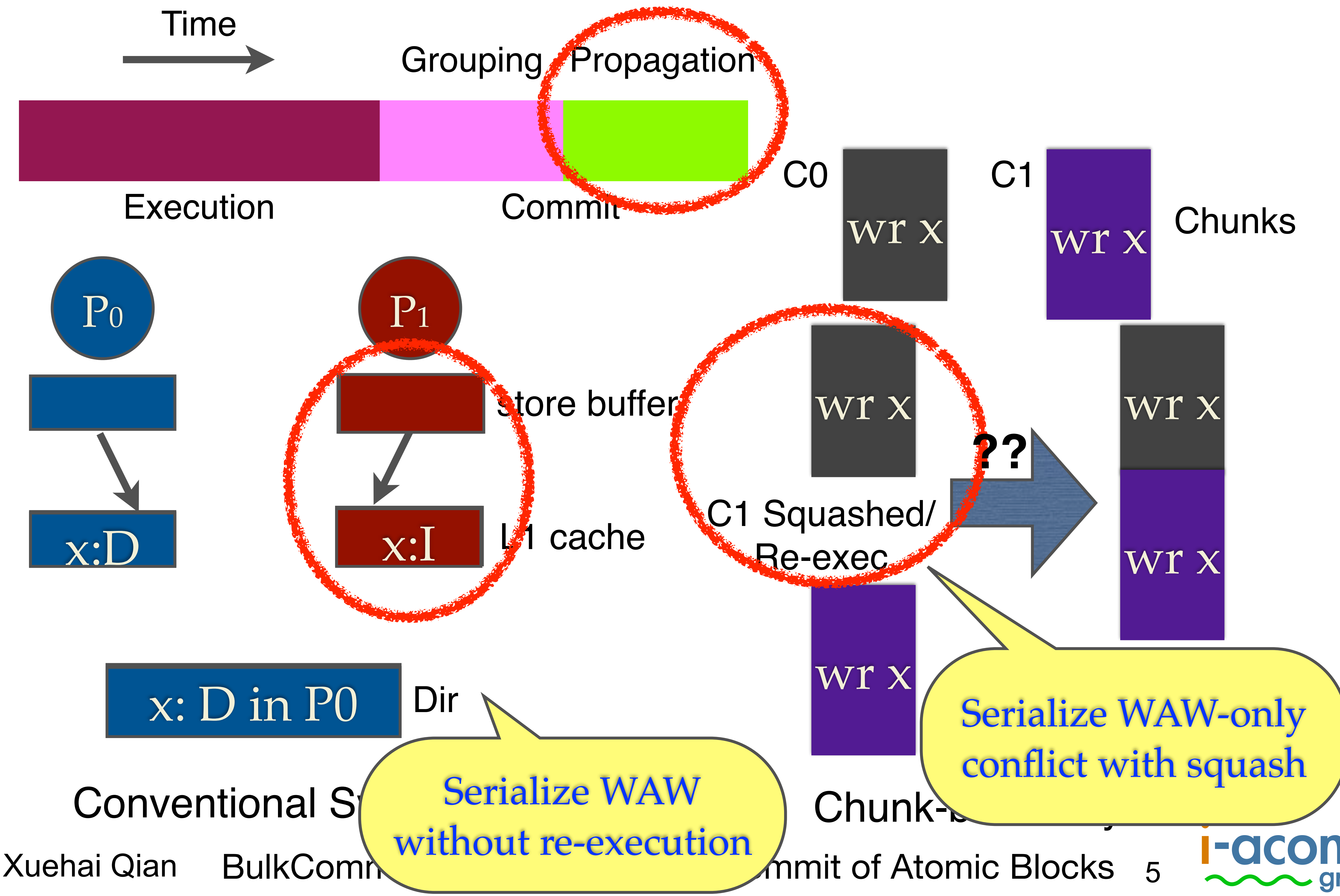
- Execution:
  - Reads and writes bring lines into the cache
  - No written line is made visible to other processors
  - Execution ends when the last instruction of the chunk completes
- Commit: make the chunk state visible atomically
  - Grouping: set the relative order of any two conflicting chunks
    - Grabbing the directory: locking the local memory lines and detecting the conflicts
    - After a commit grabs all the relevant directories, it is guaranteed to commit successfully
  - Propagation: making the stores in a chunk visible to the rest of the system
    - Involving sending invalidations and updating directory states
    - Atomicity is ensured since the relevant cache lines are logically locked by signatures during the process

Inefficiency 1: Sequential Grouping



	Grouping	Fine-grained Conflict	Broadcast?
? ? ?	**Parallel**	Yes	**No**

Inefficiency 2: Squash on WAW-Only



Contribution: BulkCommit

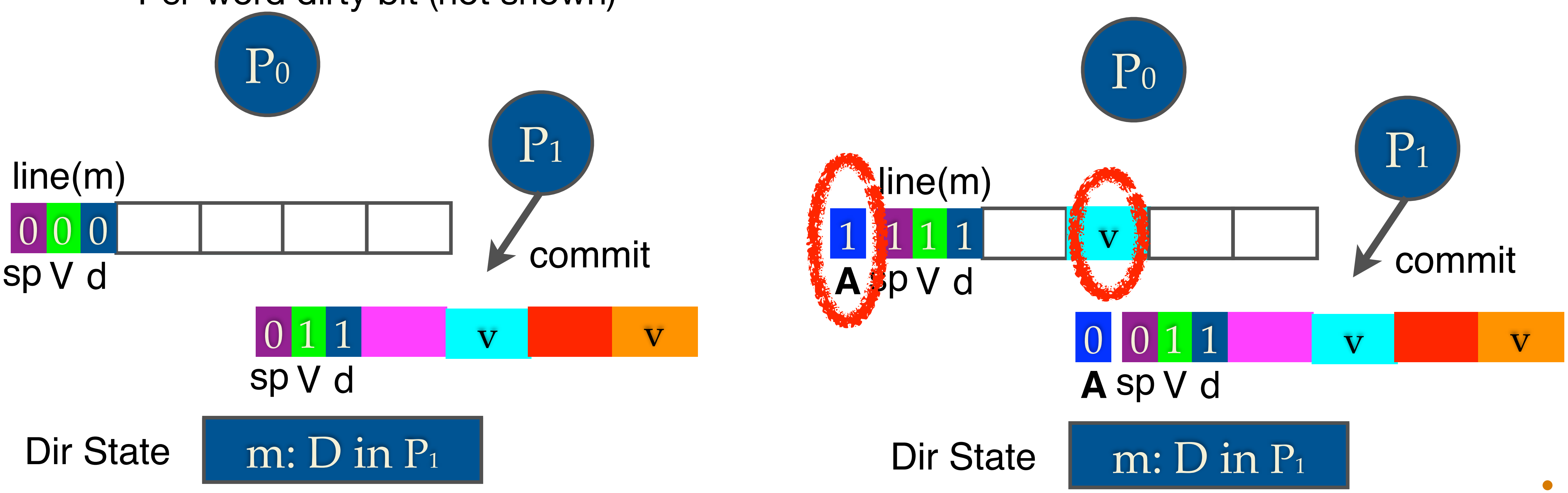
- BulkCommit: commit protocol with parallel grouping and squash-free serialization of WAW-only conflict
  - IntelliSquash: no squash on WAW
    - Insight: using L1 cache as the “store buffer” for the chunk
  - IntelliCommit: parallel grouping without broadcast
    - Insight: using preemption mechanism to ensure the consistent order of two conflicting chunks
- BulkCommit tries to achieve the optimal commit protocol design

Outline

- Motivation
- **IntelliSquash**
- IntelliCommit
- Evaluation

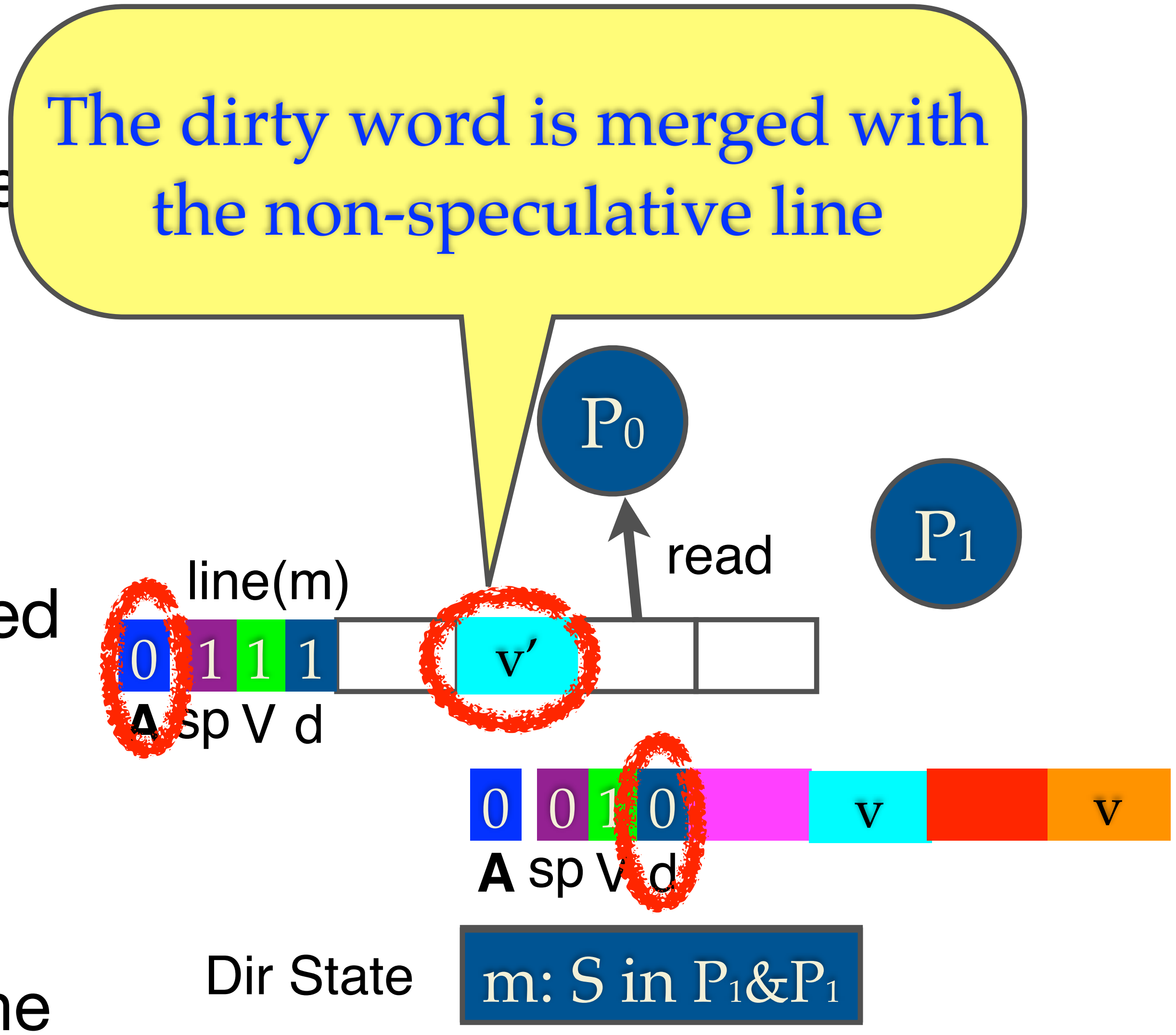
IntelliSquash: Insight

- Challenge: the speculative data produced by a chunk cannot be lost when the chunk is ready to commit
- Solution: use the L1 cache as the “store buffer” for a chunk
  - Similar to the store buffer in the conventional system
  - On receiving an invalidation, the speculative dirty words of a line are preserved
  - Absent bit: it is set when
    - The line is not presented
    - The line contains some speculative words
  - Per-word dirty bit (not shown)



IntelliSquash: Merge Operation

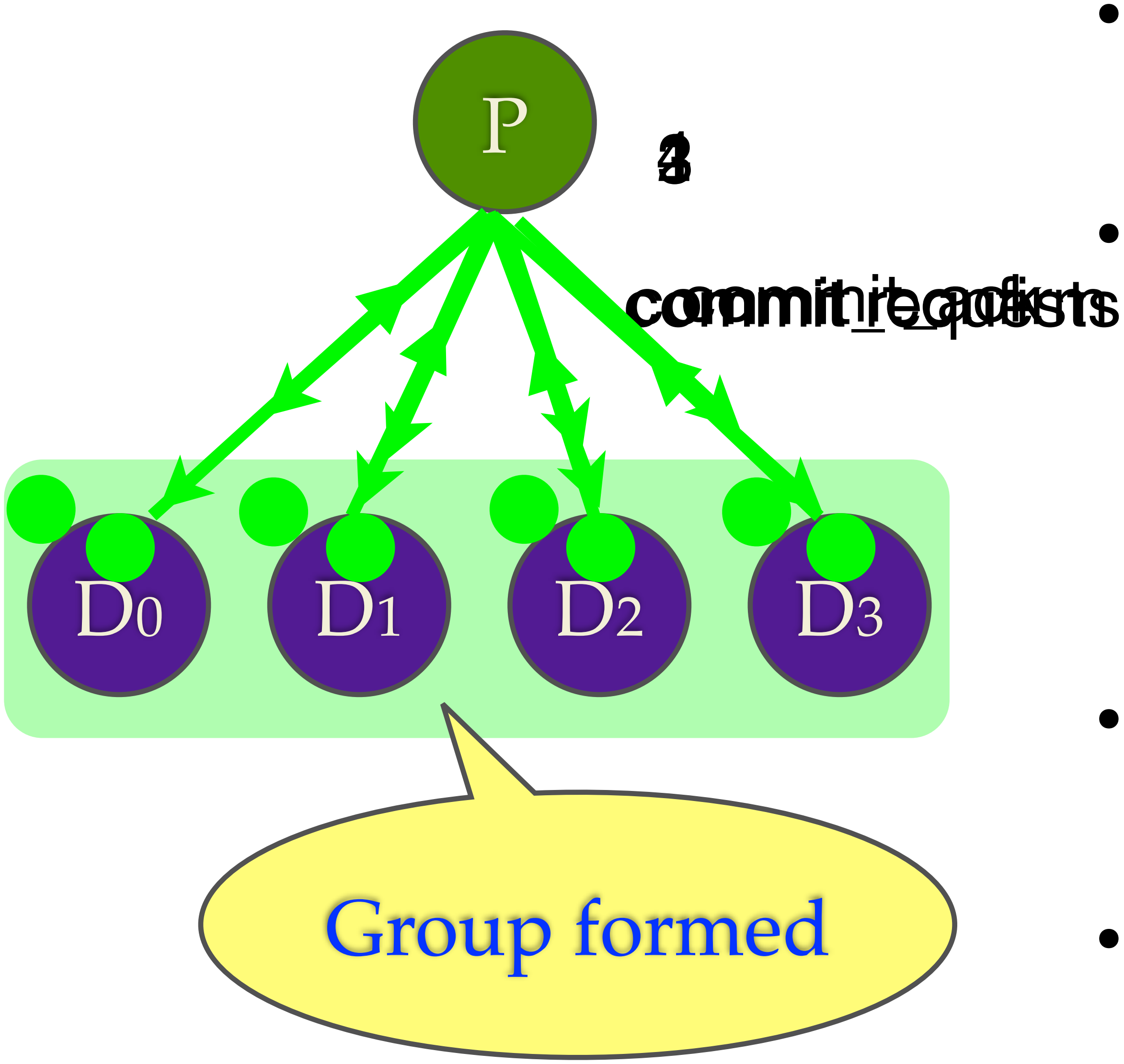
- Performed when the whole line with Absent bit set is brought to the cache
- Merge the remote non-speculative cache line with the local speculative words
  - On misses to a word not presented
  - On commit
    - The line is not accesses again
    - Therefore, need to bring the line to the cache as if there is a miss
  - Unset Absent (A) bit



Outline

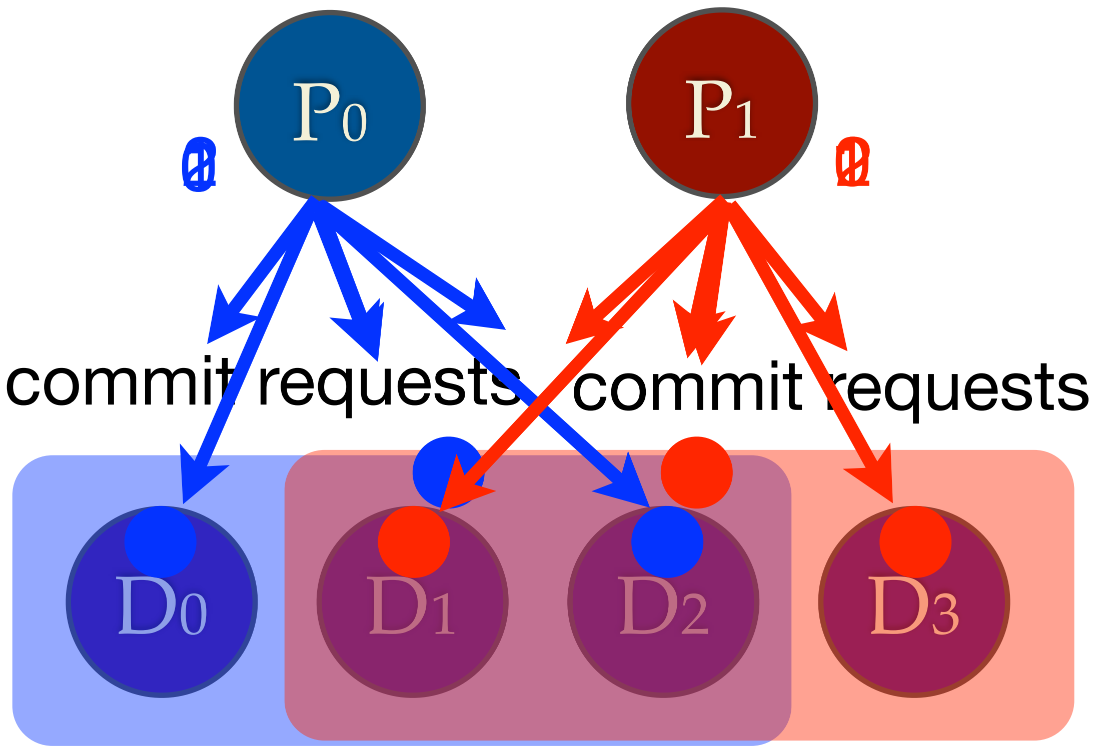
- Motivation
- IntelliSquash
- **IntelliCommit**
- Evaluation

IntelliCommit Protocol



- On chunk commit:
  - Processor sends commit requests to all the relevant directory modules
  - Directory module receives commit request:
    - Locks the memory lines
    - Responds with commit_ack
  - Processor counts the number of commit_ack received
  - Processor sends commit_confirm when it receives the expected number of commit_ack

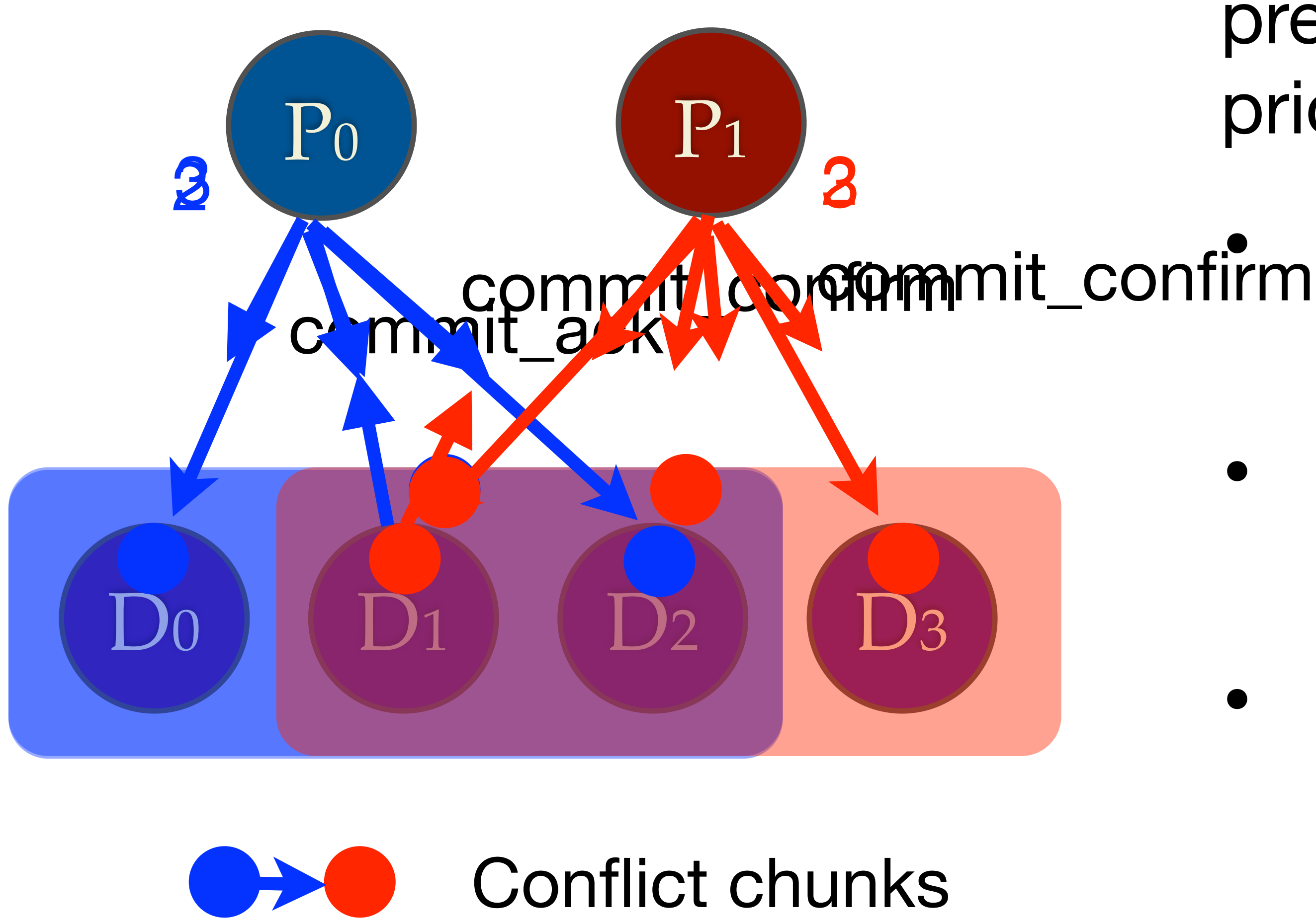
Conflicting Chunks Trying to Commit



- Different overlapped directory modules receive different commit requests in opposite order
- Need to avoid deadlock

IntelliCommit: Deadlock Resolution

- Basic idea: enforce a consistent order between two conflicting chunks
- Piggyback a hardware-generated random number with the commit request



- The chunk with higher priority preempts the chunk with lower priority
  - D1 requests permission from P1 to preempt its chunk
  - If P1 has not already formed the group, it allows it
  - After the first group commits, D1 will inform P1

Why Does IntelliCommit Work?

1. When the directory group of a chunk is already formed, the chunk cannot be preempted by another chunk

2. All the modules involved in a conflict reach the same decision on which chunk has the higher priority, locally

IntelliCommit Implementation

- Extra messages (P=Processor, D=Directory):
  - preempt_request (D→P)
  - preempt_ack (P→D)
  - preempt_nack (P→D)
  - preempt_finish (D→P)
- Commit Ack Counter (CAC): #(not received commit_ack)
- 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 directories
  - Increase PV[i]: about to send preempt_ack for Pi's chunk
  - Decrease PV[i]: received a preempt_finish for Pi's chunk
  - When to send commit_confirm?
    - **(CAC==0)&&(for each i, PV[i]==0)**
    - Received all commit_ack and the chunk is not preempted by any other chunks in any directory

Outline

- Motivation
- IntelliSquash
- IntelliCommit
- **Evaluation**

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)

SPLASH-2 Performance

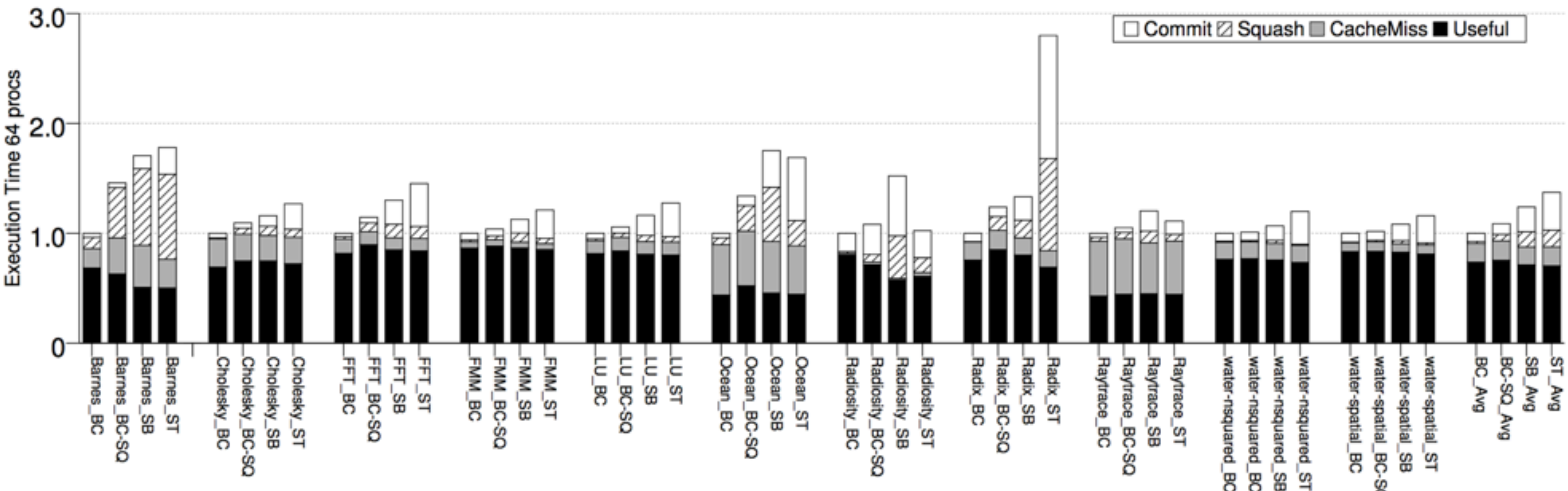


Figure 6: SPLASH-2 execution time with 64 processors.

- BulkCommit reduces both squash and commit time

PARSEC Performance

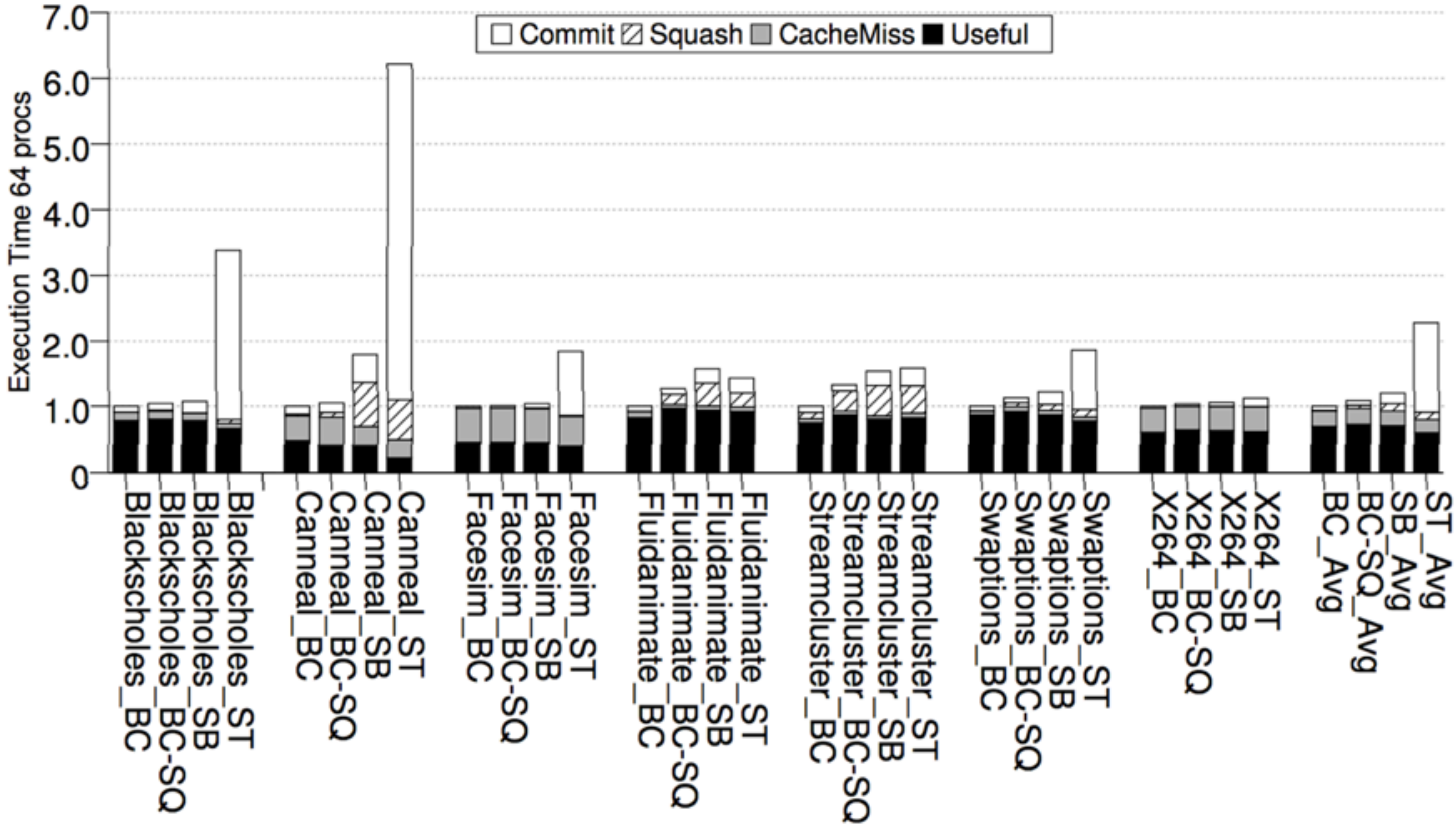
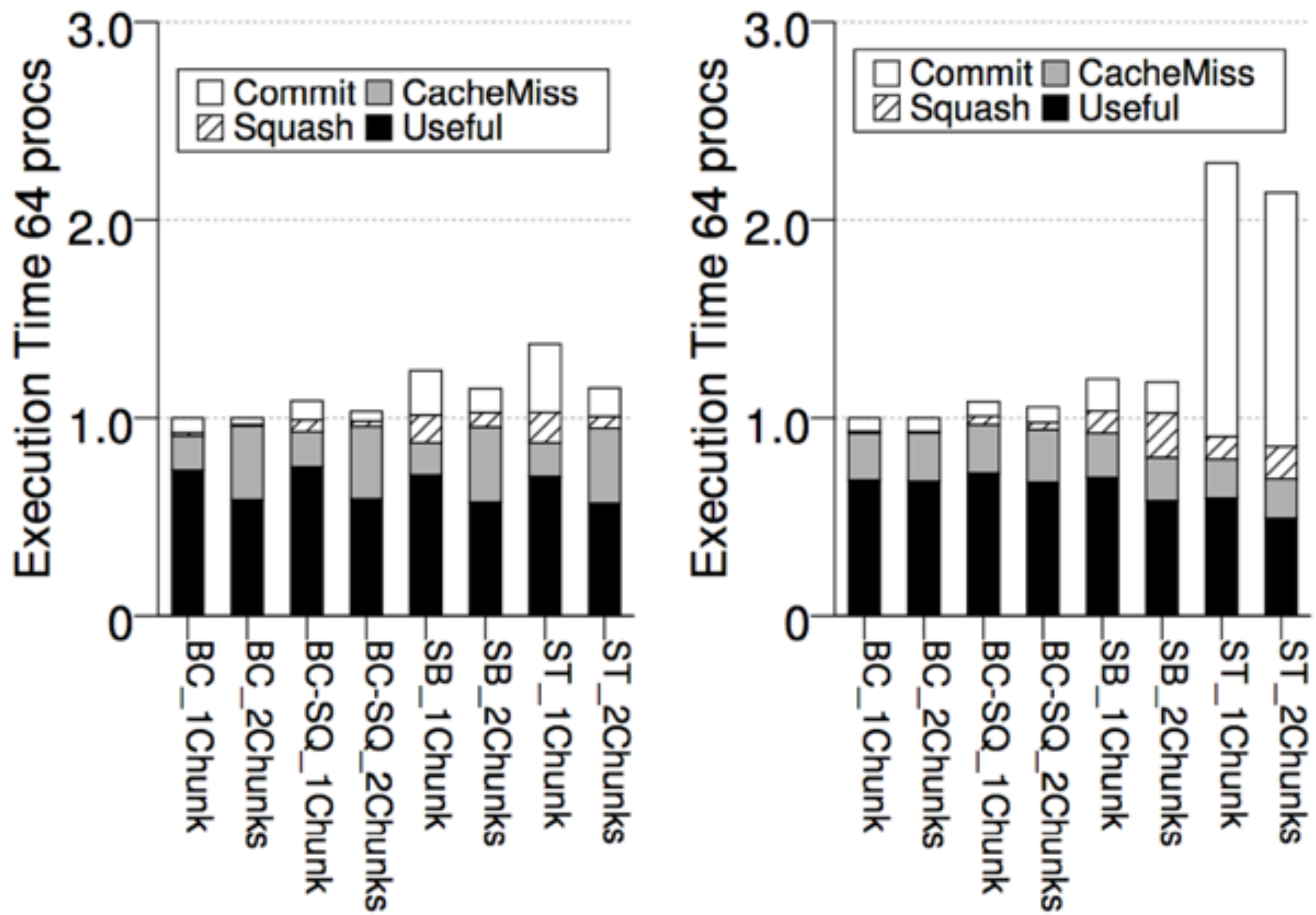


Figure 7: PARSEC execution time with 64 processors.

One and Two Outstanding Chunks



(a) SPLASH-2 (b) PARSEC

- Using two outstanding chunks is not always useful due to the set restriction
  - Two chunks from the same processor cannot write the same cache set

Also in the paper...

- IntelliSquash: Directory entry states with signature expansion
- IntelliCommit: Directory state diagram of a the committing chunk
- Discussion of correctness properties
- Discussion of complexity

Conclusion

- Proposed BulkCommit: commit protocol with parallel grouping and squash-free serialization of WAW-only conflict
- Key properties:
    - Serializing WAW between chunks without squashing
        - Exploiting the similarity of a chunk commit and an individual store
    - Parallel grouping
        - Using preemption mechanisms to order two conflicting chunks consistently
- Results:
    - Eliminate the commit bottleneck with even single outstanding chunk
    - Reduce the squash time for some applications
- We believe BulkCommit achieves the optimal design of the chunk commit protocol

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

Xuehai Qian, Benjamin Sahelices
Josep Torrellas, Depei Qian
University of Illinois

University of Illinois

http://iacoma.cs.uiuc.edu/