Drinking from Both Glasses: Adaptively Combining Pessimistic and Optimistic Synchronization for Efficient Parallel Runtime Support

Man Cao

Minjia Zhang

Michael D. Bond

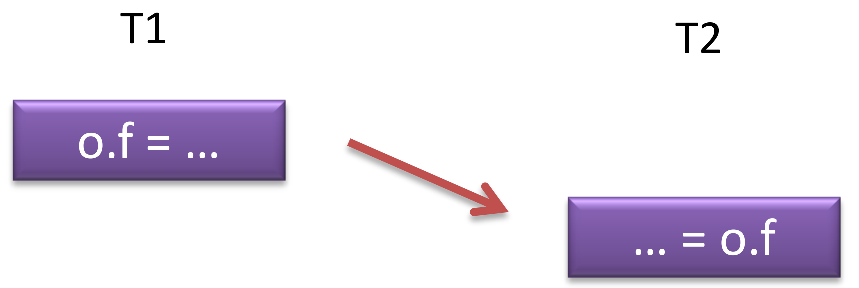
THE OHIO STATE UNIVERSITY

Dynamic Analyses for Parallel Programs

- Data Race Detector, Record & Replay, Transactional Memory, Deterministic Execution, etc.

- Performance is usually bad!
  - several times slower
- Fundamental difficulties?

Cross-thread dependences



- Crucial for dynamic analyses and systems
- Capturing cross-thread dependences
  - Detecting

    e.g. data race detector, dependence recorder
  - Controlling

    e.g. transactional memory, deterministic execution

Typical approach

- Per-object metadata (state)
  - E.g. last writer/reader thread
- At each object access:
  - Check current state
  - Analysis-specific action
  - Update state if needed
  - Perform the access

} Atomically

Typical approach

- Per-object metadata (state)
  - E.g. last writer/reader thread
- At each object access:
  - Check current state
  - Analysis-specific action
  - Update state if needed
  - Perform the access

Atomically

How to guarantee?

Pessimistic Synchronization

- Used by most existing work
  - Data Race Detector
    - [FastTrack, Flanagan & Freund, 2009]
  - Atomicity Violation Detector
    - [Velodrome, Flanagan et al., 2008]
  - Record & Replay
    - [Instant Replay, LeBlanc et al., 1987]
    - [Chimera, Lee et al., 2012]

Pessimistic Synchronization

LockMetadata()

Pessimistic Synchronization

LockMetadata()

Check and compute new metadata

Pessimistic Synchronization

LockMetadata()

Check and compute new metadata

Analysis-specific actions

Pessimistic Synchronization

LockMetadata()

Check and compute new metadata

Analysis-specific actions

Program access

Pessimistic Synchronization

LockMetadata()

Check and compute new metadata

Analysis-specific actions

Program access

UnlockAndUpdateMetadata()

Pessimistic Synchronization

LockMetadata()

Check and compute new metadata

Analysis-specific actions

Program access

UnlockAndUpdateMetadata()

- Synchronization on every access
- 6X slowdown on average

Optimistic Synchronization

- Used to improve performance
  - Biased Locking
    - [Lock Reservation, Kawachiya et al., 2002]
    - [Bulk Rebiasing , Russell & Detlefs, 2006]
  - Distributed Memory System
    - [Shasta, Scales et al. 1996]
  - Framework Support
    - [Octet, Bond et al. 2013]

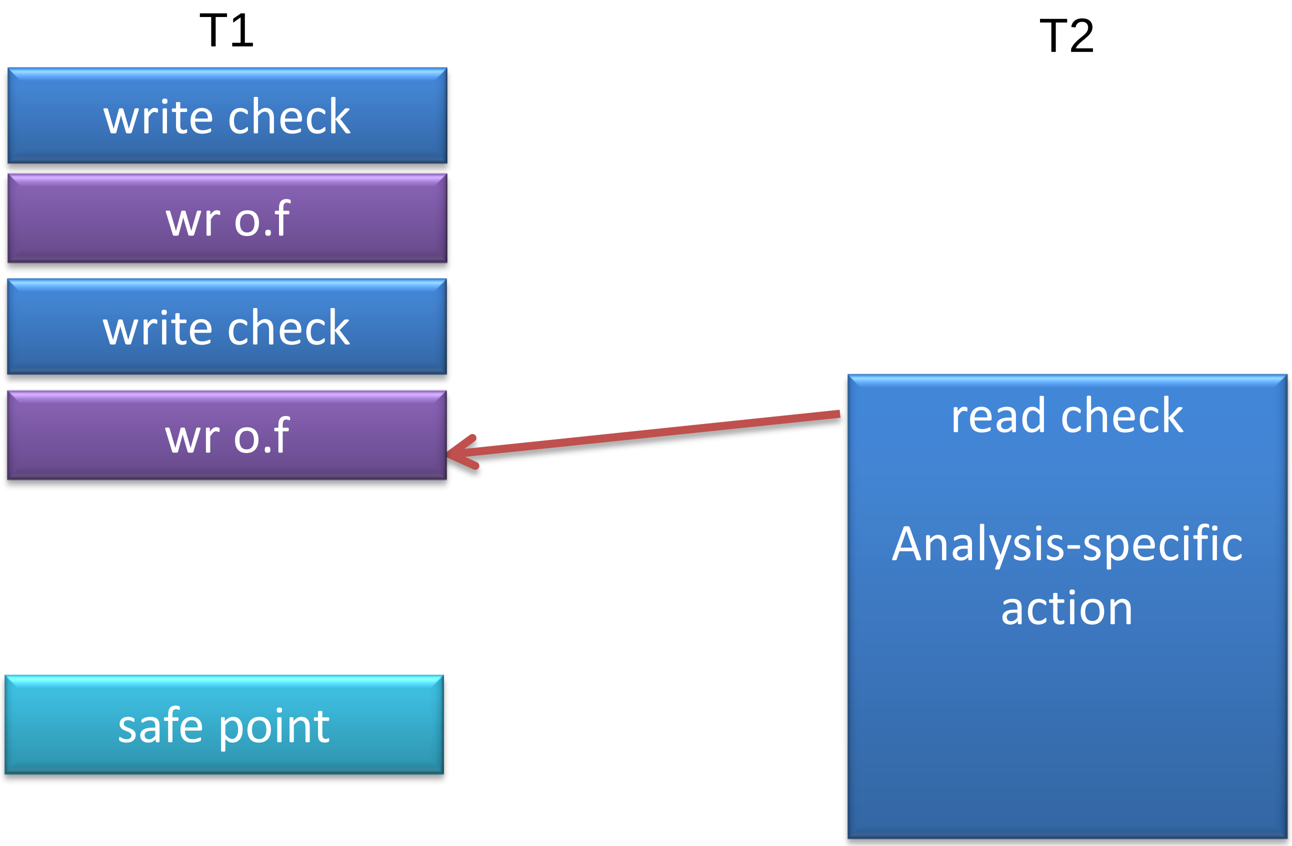
Optimistic Synchronization

- Avoid synchronization for non-conflicting accesses
- Heavyweight coordination for conflicting accesses

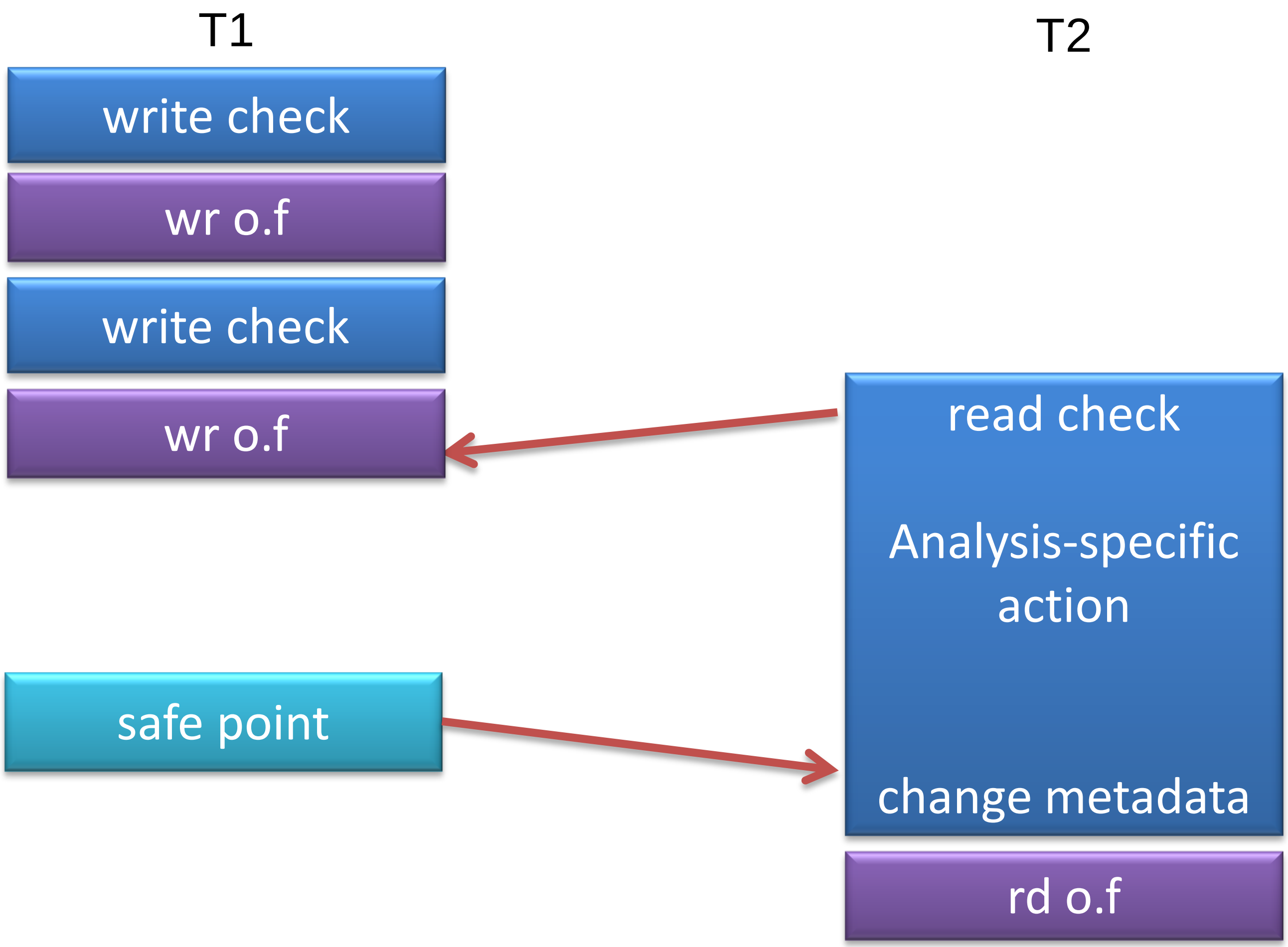
Optimistic Synchronization (Cont.)

Optimistic Synchronization (Cont.)

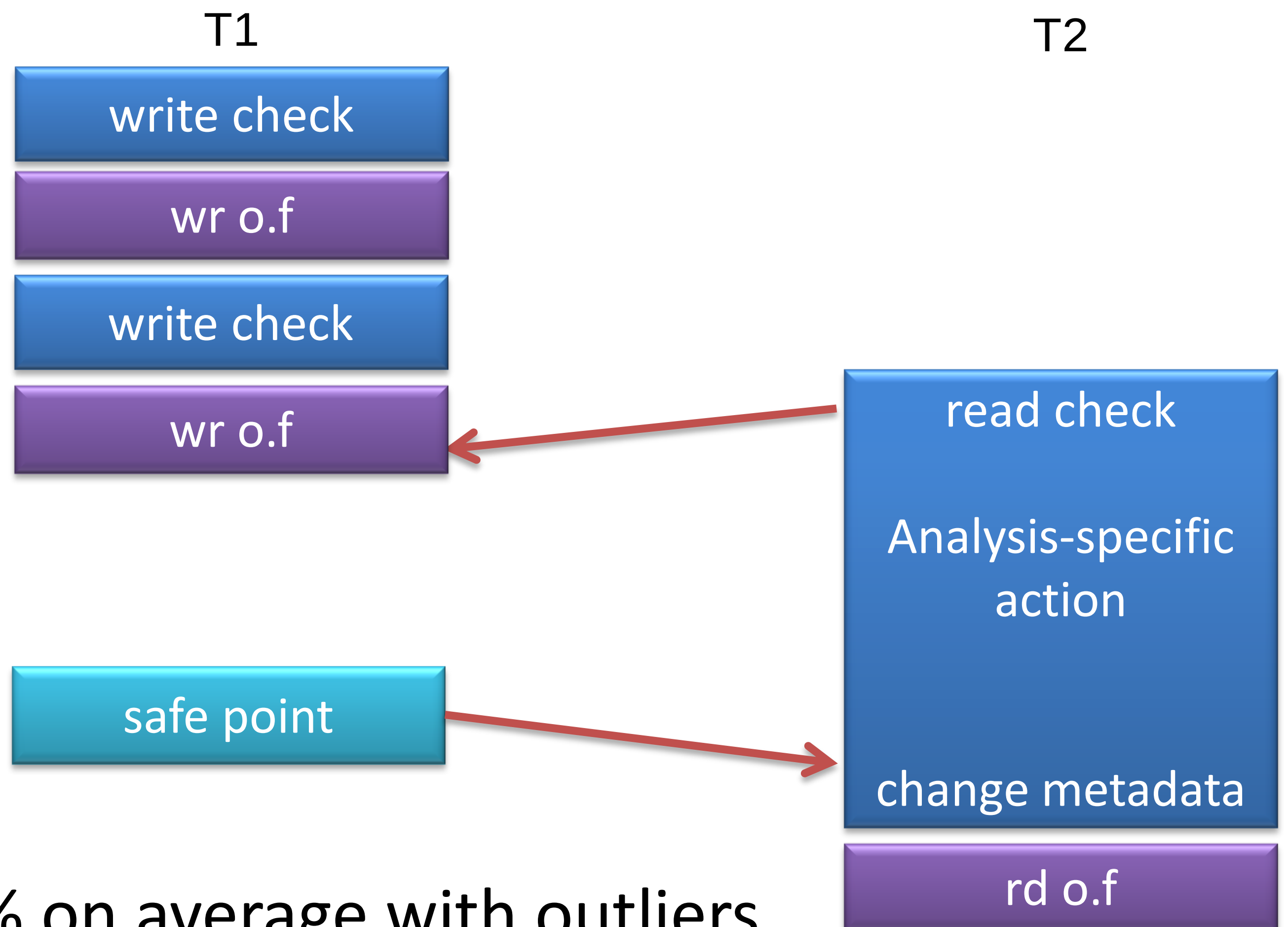
Optimistic Synchronization (Cont.)



Optimistic Synchronization (Cont.)



Optimistic Synchronization (Cont.)



- 26% on average with outliers
  - Expensive if there are many conflicting accesses

Optimistic synchronization performs best if there are few conflicting accesses.

Pessimistic synchronization is cheaper for conflicting accesses.

Drink from both glasses?

- Goal:
  - Optimistic sync. for most non-conflicting accesses
  - Pessimistic sync. for most conflicting accesses
- Our approach:
  - Hybrid state model
  - Adaptive policy
  - Support for detecting and controlling cross-thread dependences

Adaptive Policy

- Decides when to perform Pess → Opt and Opt → Pess transitions

- Cost—Benefit model
  – Formulates the problem
- Online profiling
  – Efficiently collects information and approximates the Cost-Benefit model

Cost—Benefit model

- Compares total time spent in transitions if an object were optimistic or pessimistic
  - Whichever takes less time is beneficial

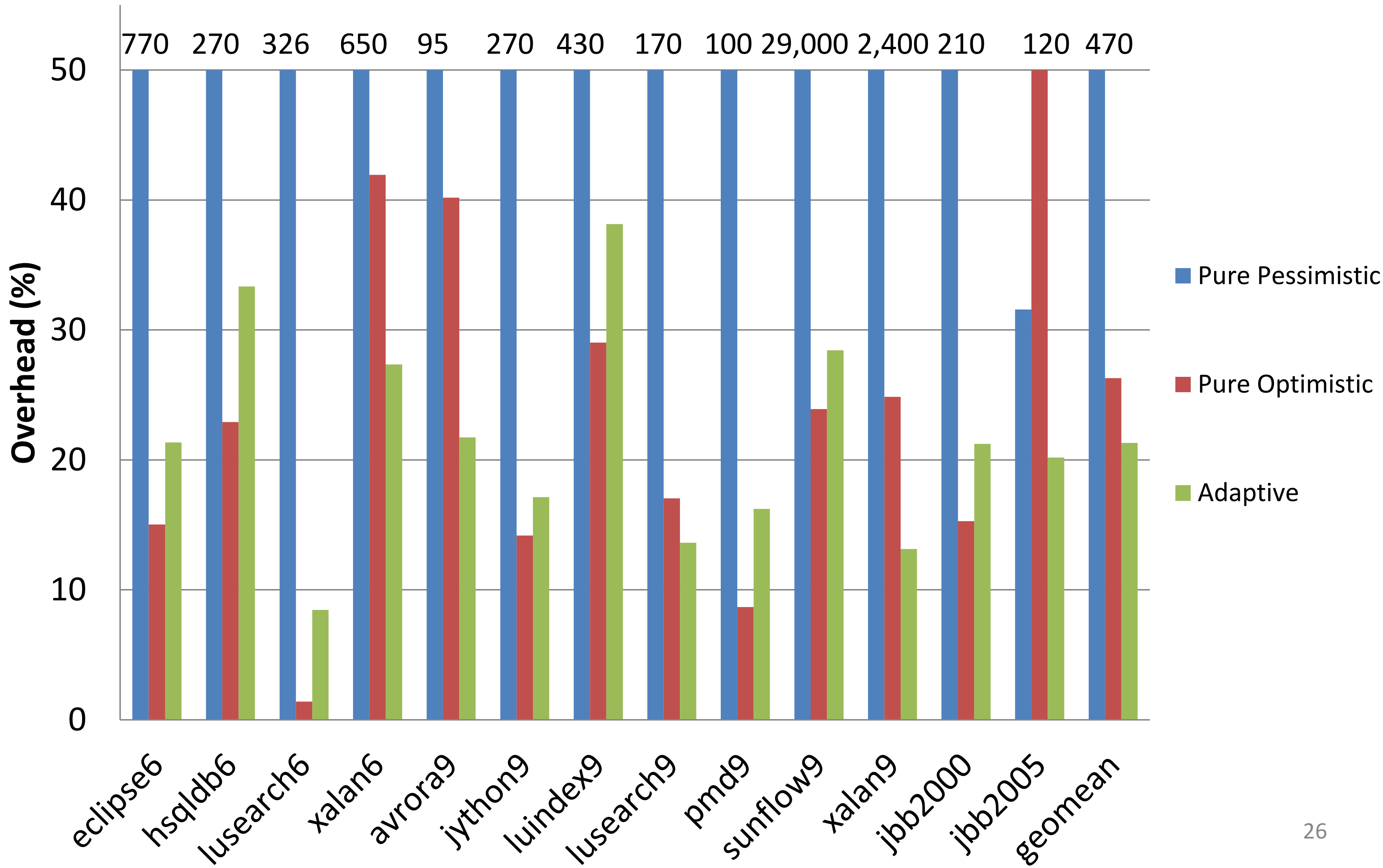
$$T_{total\_Opt} > T_{total\_Pess} \ ? \ Pess \ : \ Opt$$

- Only relies on numbers (or just the ratio) of non-conflicting and conflicting transitions

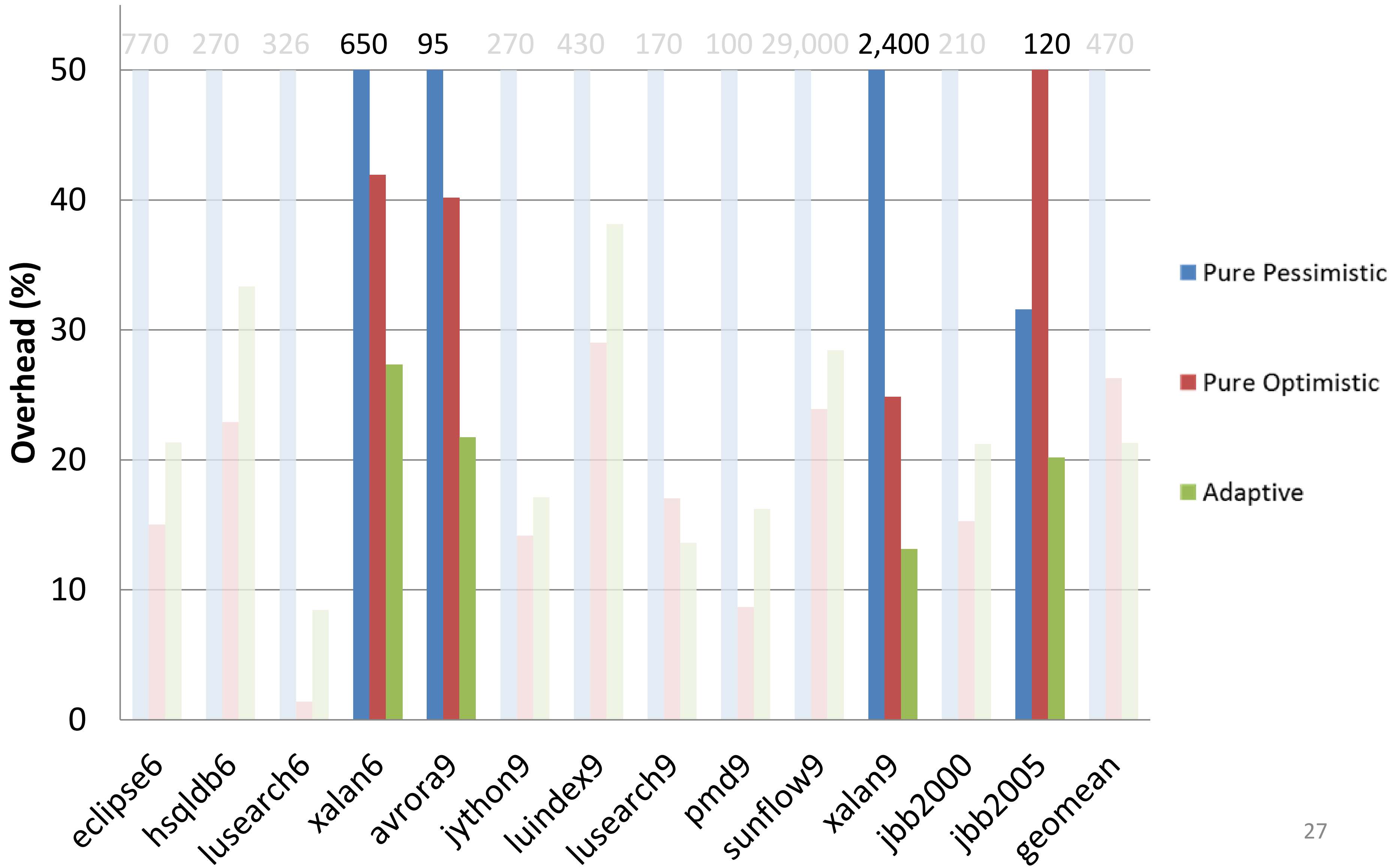
Evaluation

- Implementation
  - Jikes RVM 3.1.3
- Parallel programs
  - DaCapo Benchmarks 2006 & 2009
  - SPEC JBB 2000 & 2005
- Platform
  - 32 cores (AMD Opteron 6272)

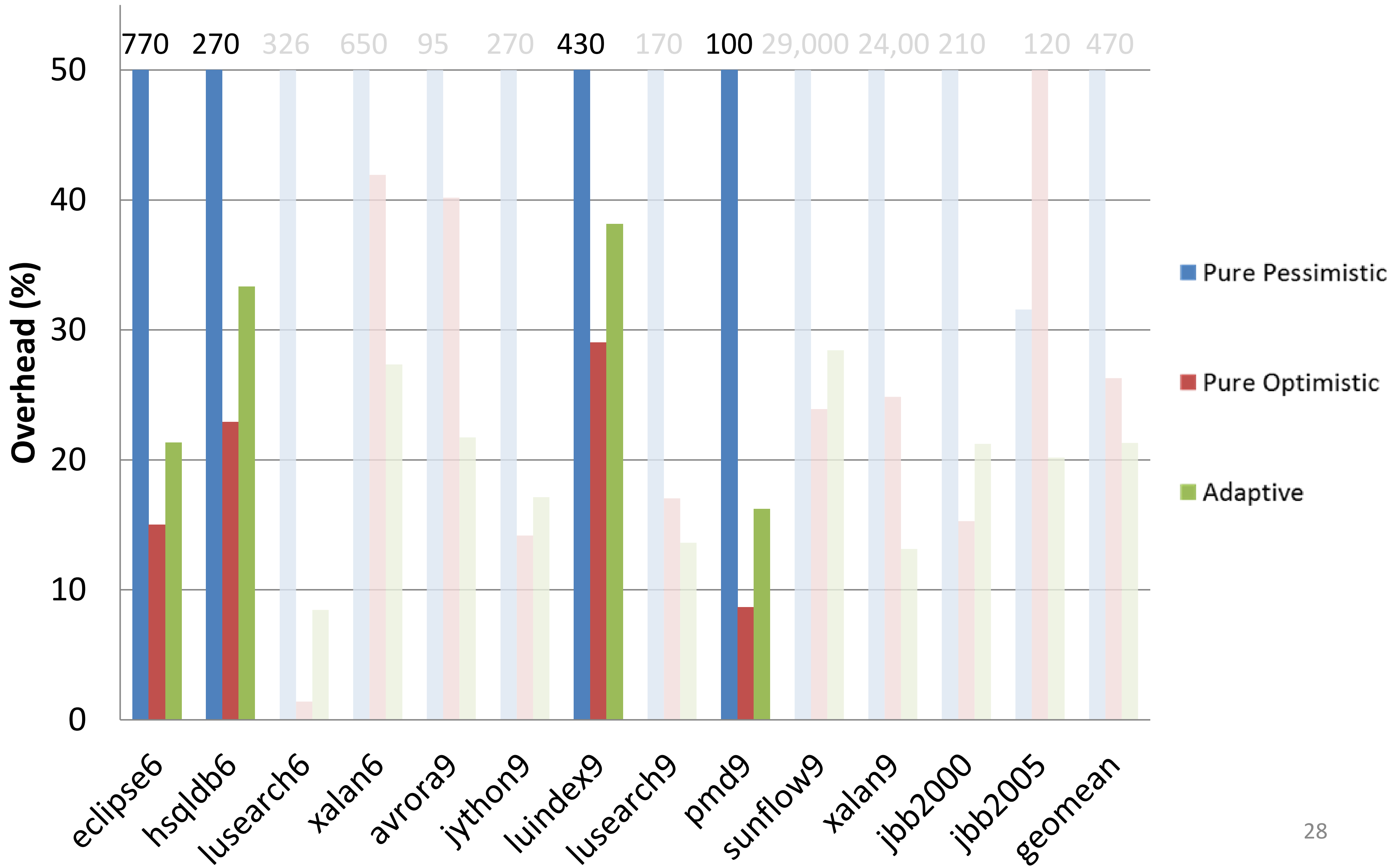
Performance



Performance



Performance



Framework support

- Detecting cross-thread dependences
  - dependence recorder
- Key challenge
  - Identify the source location of a happens-before edge for a pessimistic conflicting transition
  - Current solution requires acquiring a lock and writing to remote thread's log

Framework support (Cont.)

- Controlling cross-thread dependences
  - enforcing Region Serializability (in progress)
- Key challenge
  - Need to keep locking pessimistic objects until the end of a region

Framework support (Cont.)

- Controlling cross-thread dependences
  - enforcing Region Serializability (in progress)
- Key challenge
  - Need to keep locking pessimistic objects until the end of a region
- Possible solution
  - Defer unlocking of pessimistic objects until program lock releases
    - Helps dependence recorder
    - Simplifies instrumentation

Conclusion & Future work

- Hybrid, adaptive synchronization achieves better performance
  - never significantly degrades performance
  - sometimes improves performance substantially
- Future directions
  - Explore different adaptive policies (e.g. aggregate profiling)
  - Reduce instrumentation cost by deferring unlock operations of pessimistic synchronization
  - Apply to *control* cross-thread dependences