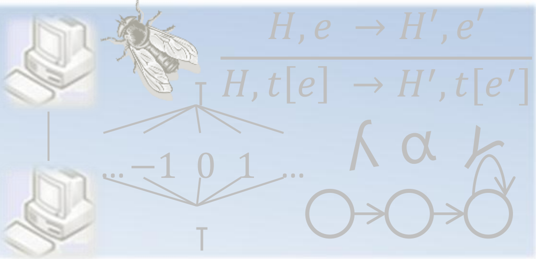


ALTER: Exploiting Breakable Dependences for Parallelization

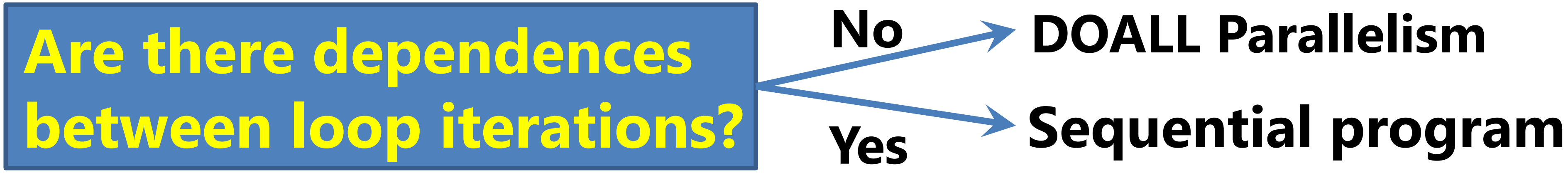
Kaushik Rajan

Abhishek Udupa

William Thies

Rigorous Software Engineering
Microsoft Research, India

Parallelization Reconsidered

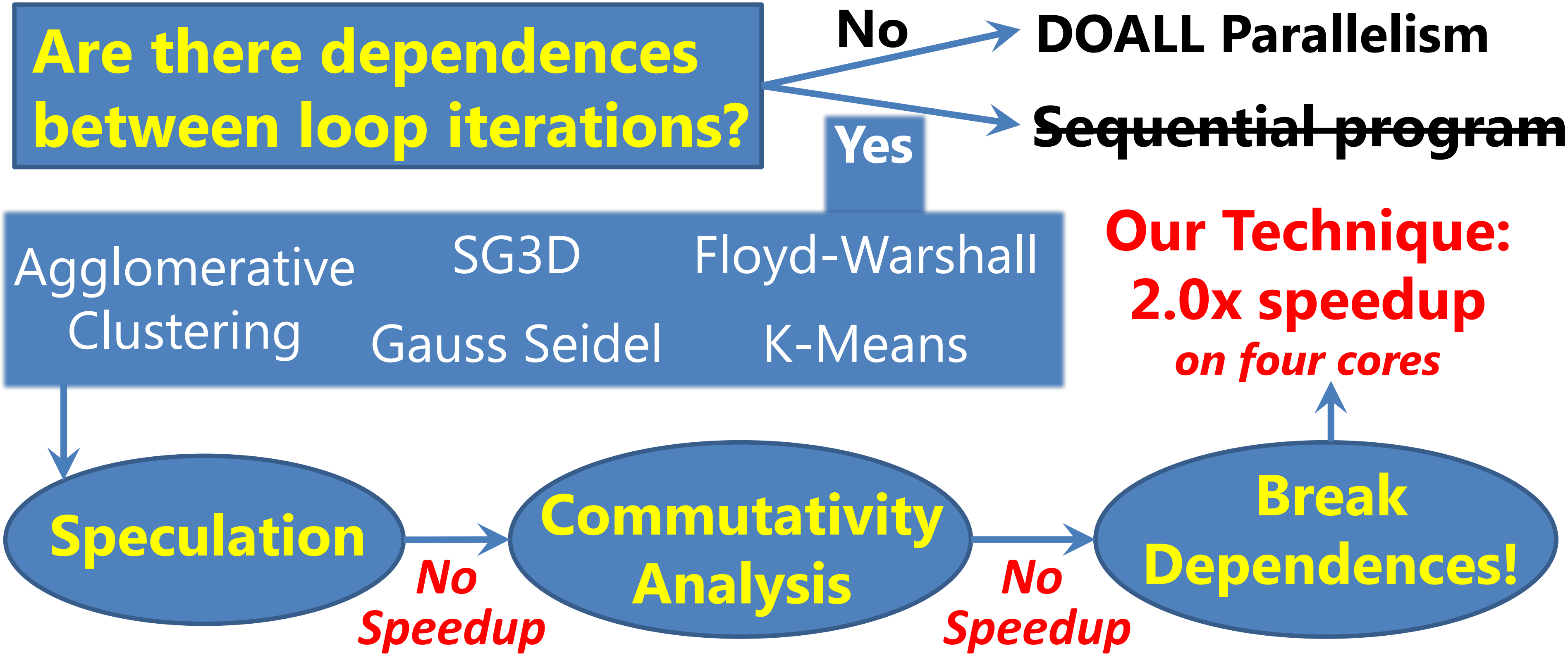


Are there dependences between loop iterations?

No → **DOALL Parallelism**

Yes → **Sequential program**

Parallelization Reconsidered



Are there dependences between loop iterations?

No → **DOALL Parallelism**

Yes → **~~Sequential program~~**

Agglomerative Clustering

SG3D

Floyd-Warshall

Gauss Seidel

K-Means

Speculation → ***No Speedup*** → **Commutativity Analysis** → ***No Speedup*** → **Break Dependences!**

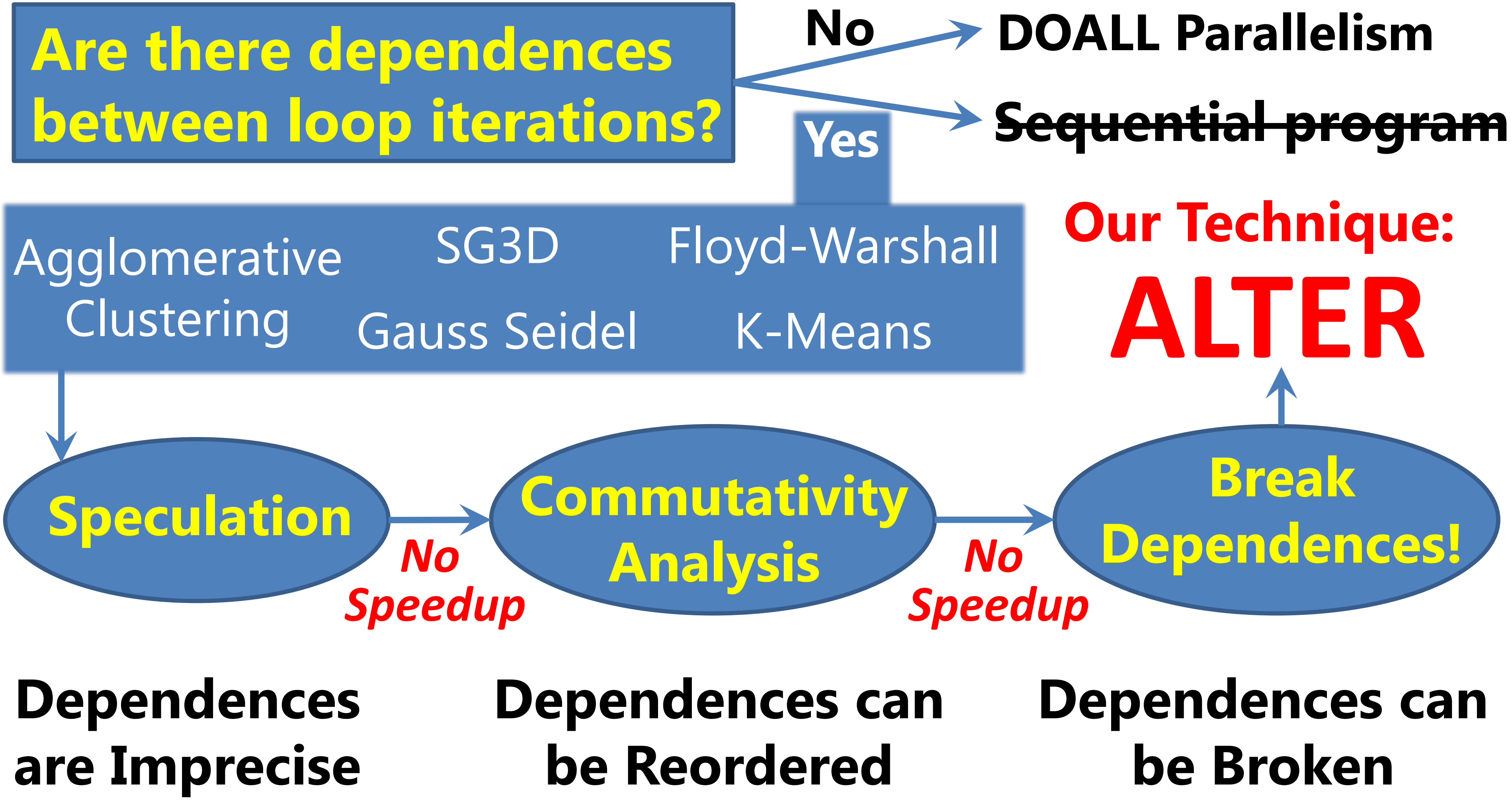
Our Technique: 2.0x speedup ***on four cores***

Dependences are Imprecise

Dependences can be Reordered

Dependences can be Broken

Parallelization Reconsidered



Are there dependences between loop iterations?

No → **DOALL Parallelism**

→ **~~Sequential program~~**

Yes

Agglomerative Clustering

SG3D

Floyd-Warshall

Gauss Seidel

K-Means

Our Technique:

ALTER

Speculation → ***No Speedup*** → **Commutativity Analysis** → ***No Speedup*** → **Break Dependences!**

Dependences are Imprecise

Dependences can be Reordered

Dependences can be Broken

Outline

- Breakable Dependences: Stale Reads
- Deterministic Runtime System
- Assisted Parallelization
- Results

other details in the paper

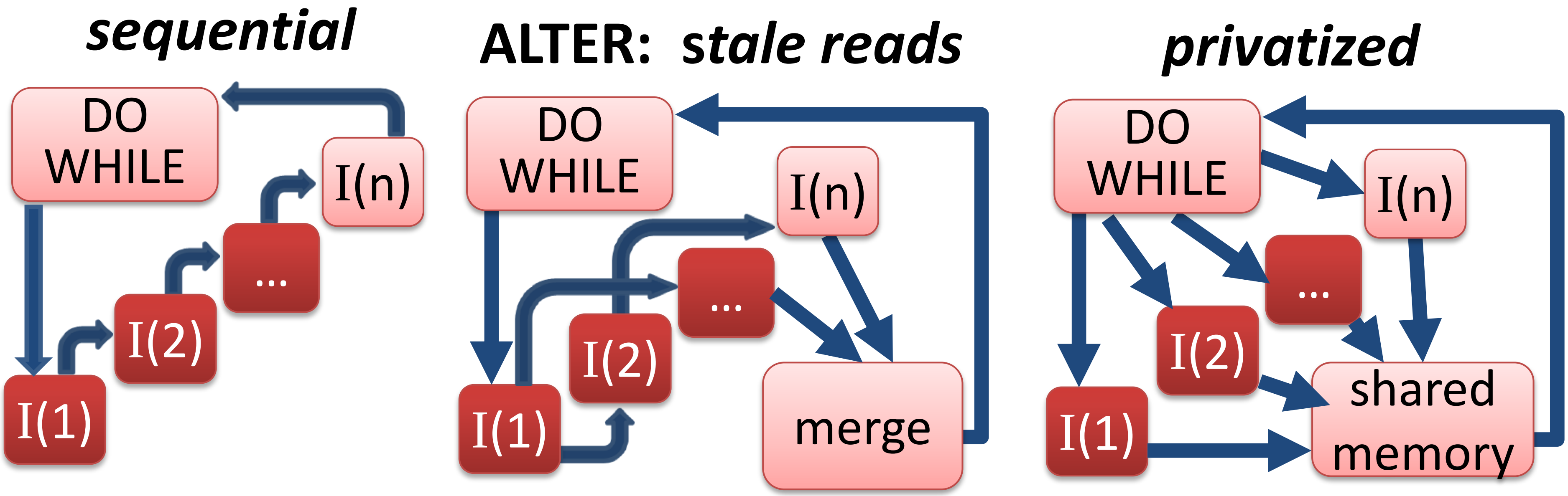
Breakable Dependences

in an Iterative Convergence Algorithm

```
while(!converged) {
   for i = 1 to n {
      refine(soln[i])
    }
}
```

Examples:

- Floyd Warshall algorithm
- Monotonic data-flow analyses
- Linear algebra solvers
- Stencil computations

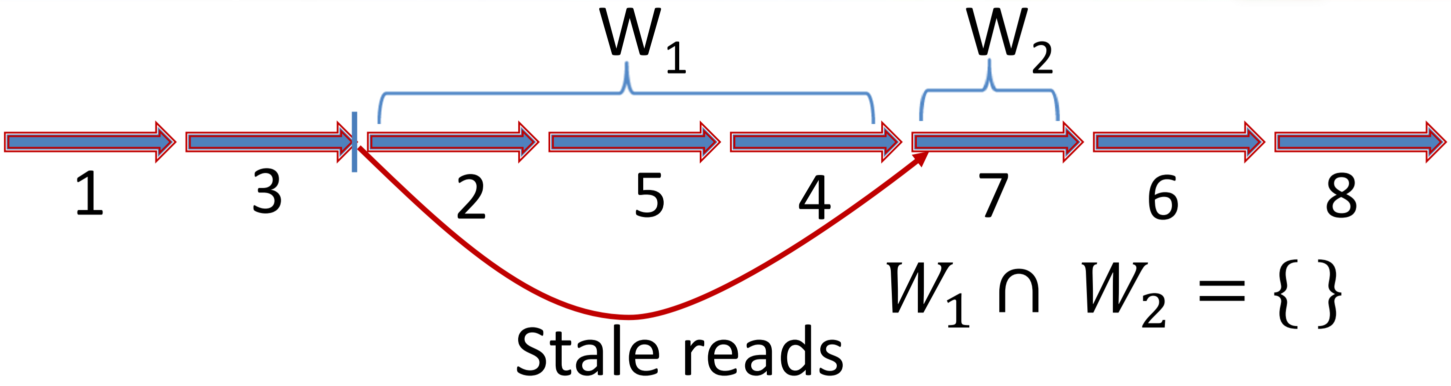


sequential

ALTER: *stale reads*

privatized

Stale Reads Execution Model

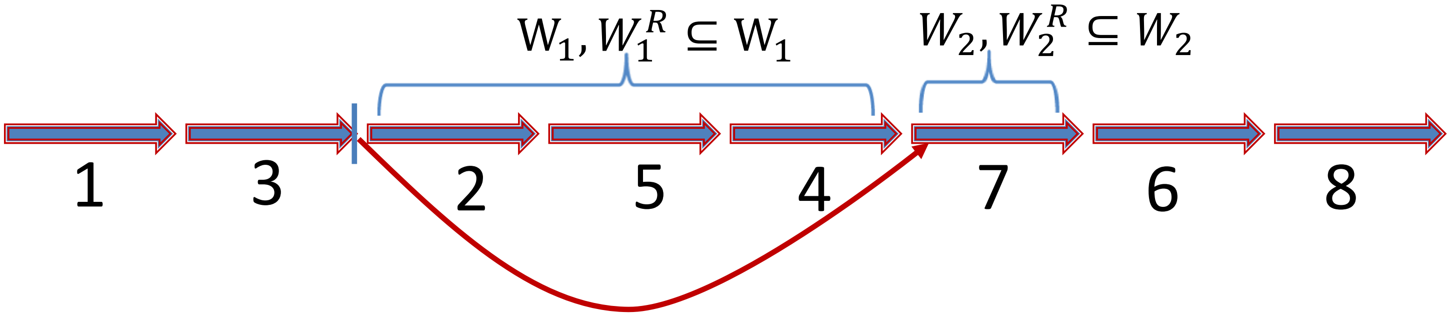


$W_1 \cap W_2 = \{\,\}$

- Execution valid under staleReads model iff
  - Commit order is some serial order of iterations (can be different from sequential order)
  - Each iteration reads a stale but consistent snapshot
  - Staleness is bounded: no intersecting writes by intervening iterations

Akin to Snapshot Isolation for databases

Stale Reads with Reduction

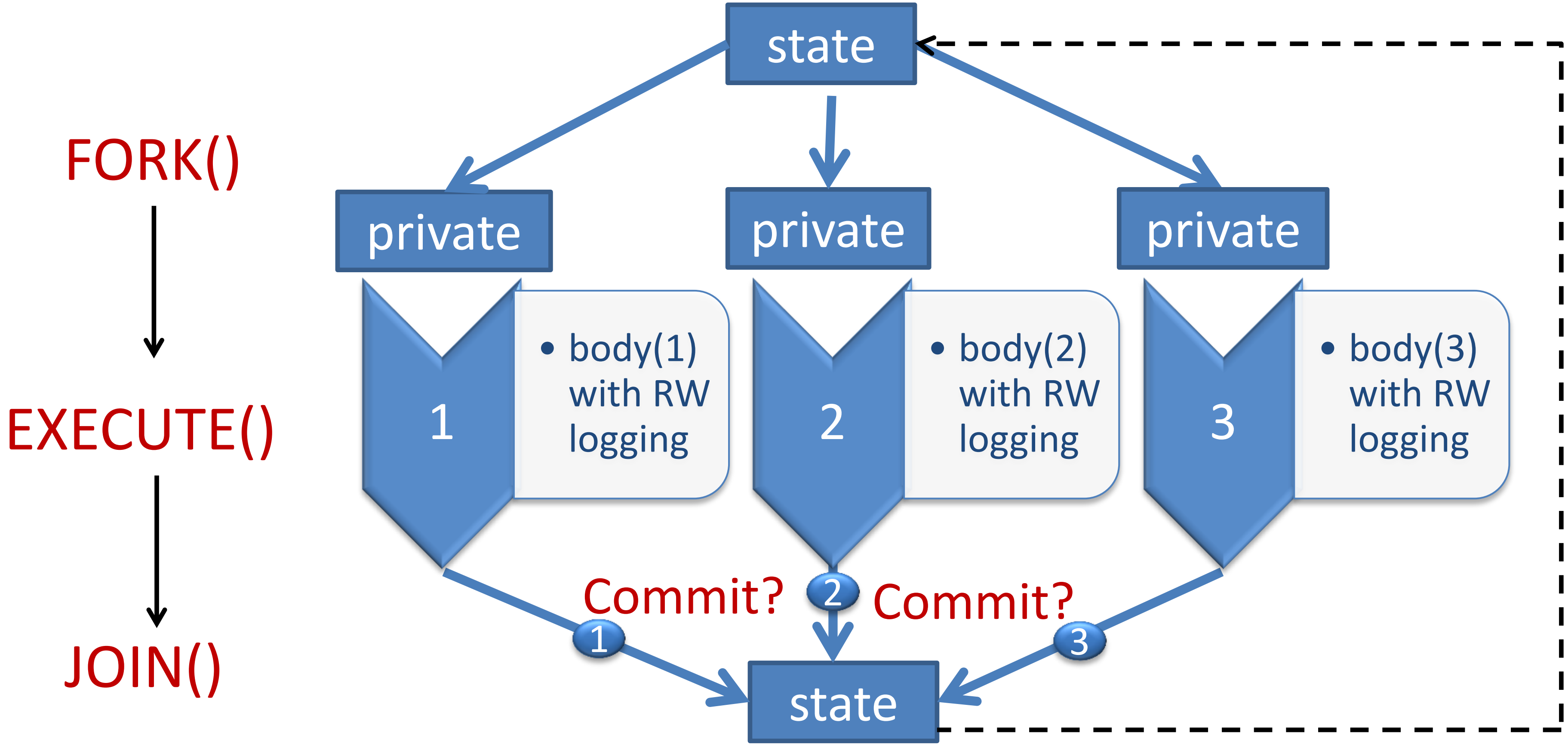


$$(W_1 \setminus W_1^R) \cap (W_2 \setminus W_2^R) = \{\,\}$$

$$reduction\ \ R \coloneqq (var, O)\ where$$

1. Every access to *var* is an update using operation O
2. Operator O is commutative and associative

Deterministic Runtime System



StaleReads Commit(i):

$$\forall_{j\ st.j<i}\ writes(i) \cap writes(j) = \{\}$$

Alter Annotations

```
while(error < EPSILON) { //convergence loop
   error = 0.0;
   for(uint32_t i = 1; i < grid->xmax - 1; ++i) {
     [StaleReads, (error, max)]
     for(uint32_t j = 1; j < grid->ymax - 1; ++j) {
         for(uin32_t k = 1; k < grid->zmax - 1; ++k) {
             oldValue = grid[i][j][k]
             grid[i][j][k] = a * grid[i][j][k] + b * AddDirectNbr(grid)
                   + c * AddSquareNbr(grid) + d * AddCubeNbr(grid);
             error = max(error, (OldValue,GridPtr[i][j][k])));
      }
}
```

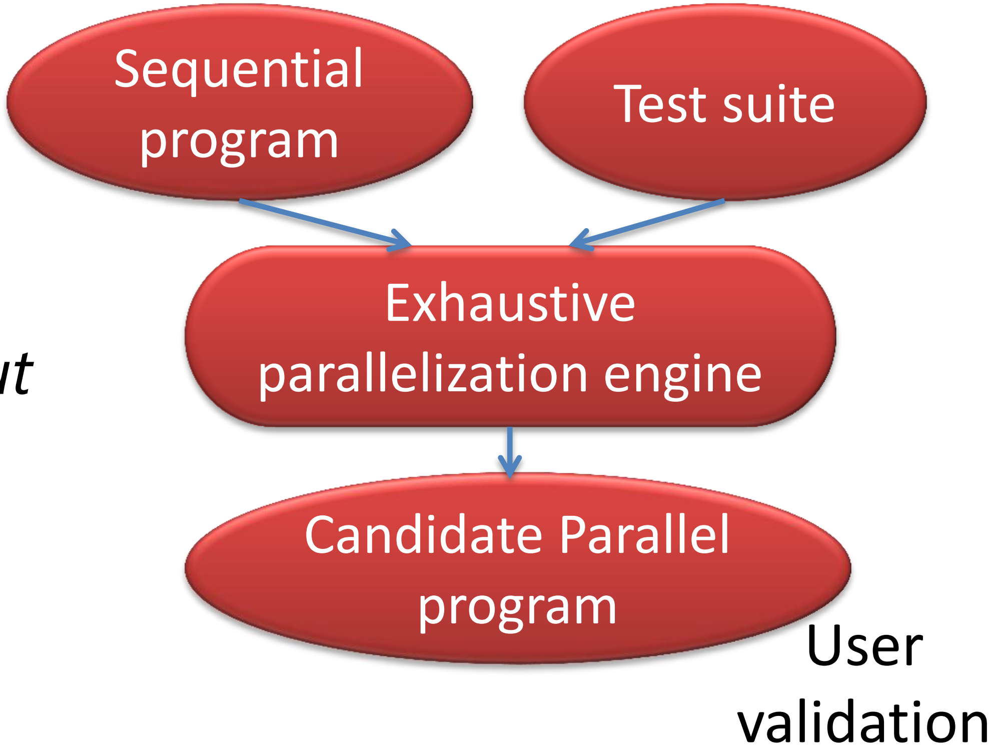
Test Driven Parallelism Inference

Exhaustive parallelization engine

- For each annotation run all test cases, record outcome
- outcome of a single run

$success, failure \in$ (*crash, timeout, high contention, output mismatch*)

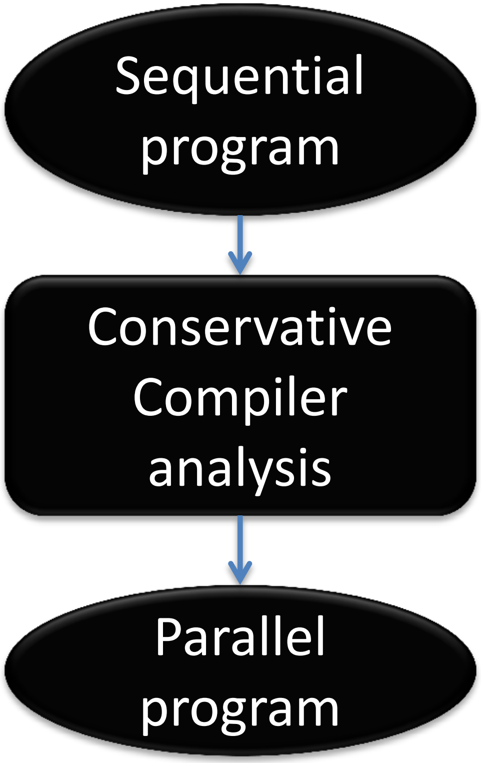
➢ Output mismatch: assertion failures or floating point difference < 0.01%



Assisted Parallelism

Prior art

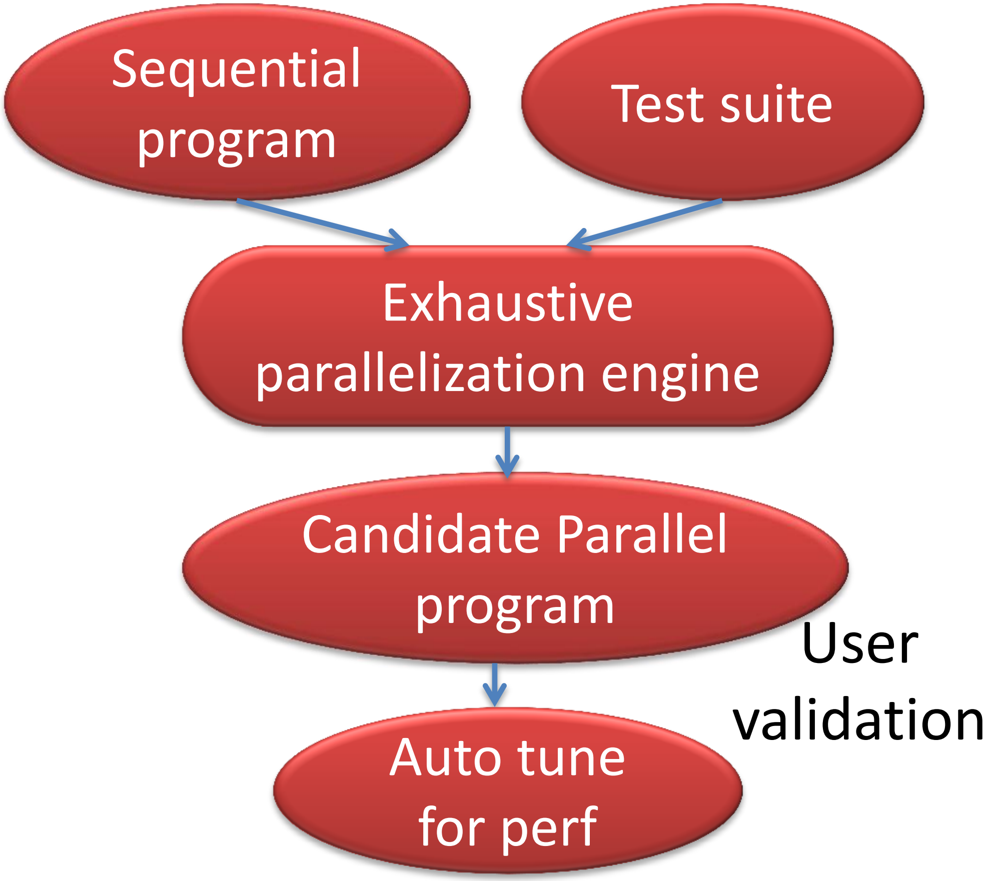
Automatic parallelism



Preserve program dependences

ALTER

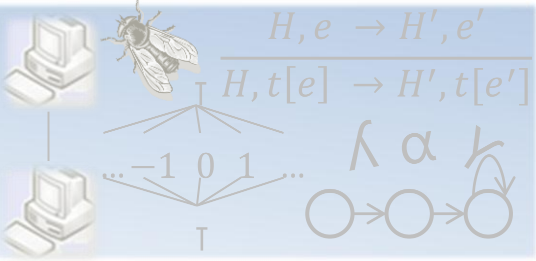
Assisted parallelism



Preserve functionality

Benchmarks

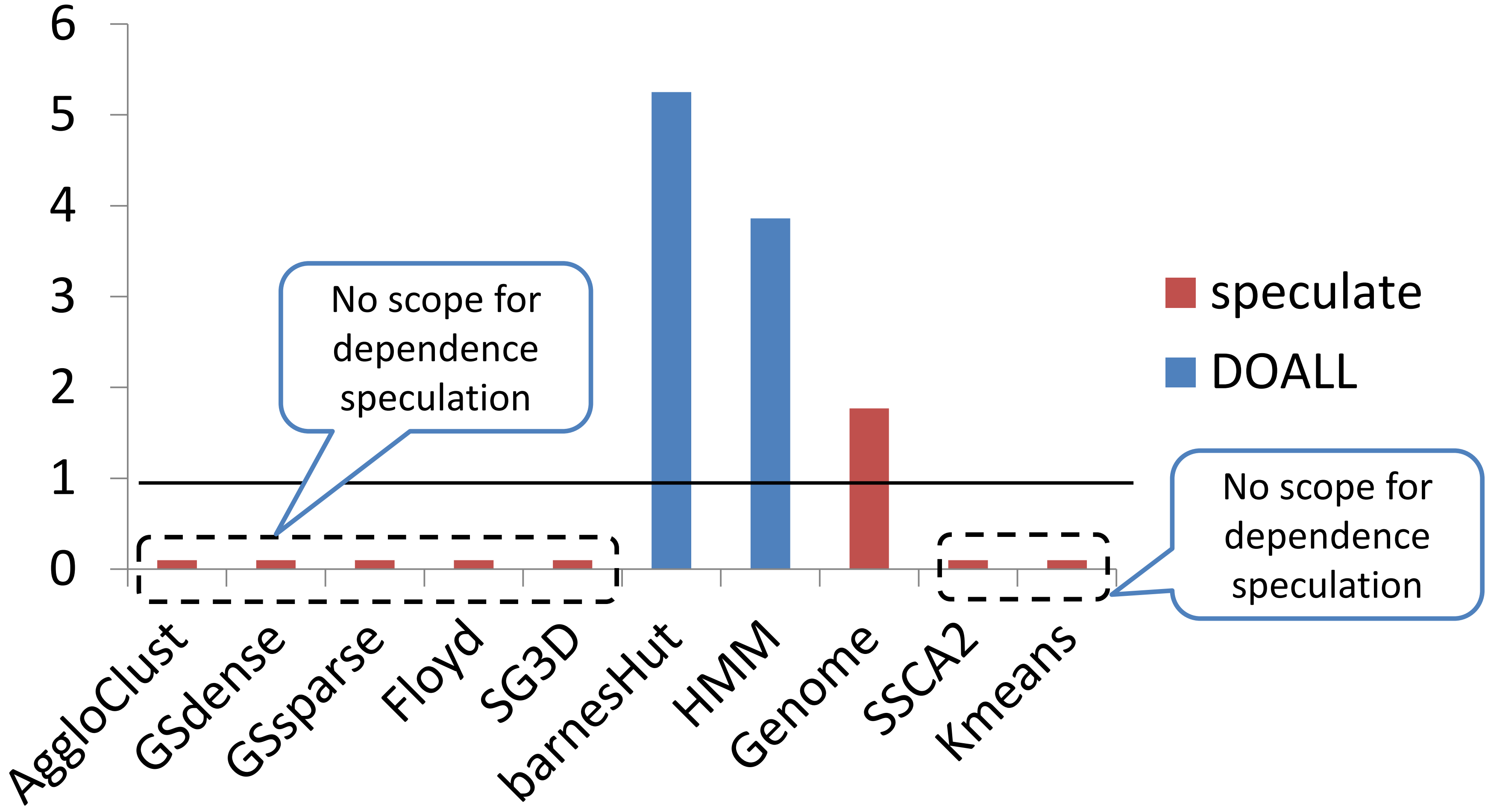
BENCHMARK	ALGORITHM TYPE	PARALLELISM	LOOP WGT
AggloClust	Branch & bound	STALE READS	89%
GSdense	Dense algebra	STALE READS	100%
GSsparse	Sparse algebra	STALE READS	100%
FloydWarshall	Dynamic programming	STALE READS	100%
SG3D	Structured grids	STALE READS, (error, max)	96%
BarnesHut	N-body methods	DOALL	99.6%
FFT	Spectral methods	DOALL	100%
HMM	Graphical models	DOALL	100%
Genome	*Bioinformatics*	STALE READS	89%
SSCA2	*Scientific*	STALE READS	76%
K-means	*Data mining*	STALE READS, (delta, +)	89%
Labyrinth	*Engineering*	_	99%



Experimental Setup

- Experiments on a 2 x quad core Xeon processor
- Alter transformations in Microsoft Phoenix compiler framework
- Comparison with dependence speculation and manual parallelization of 2 applications

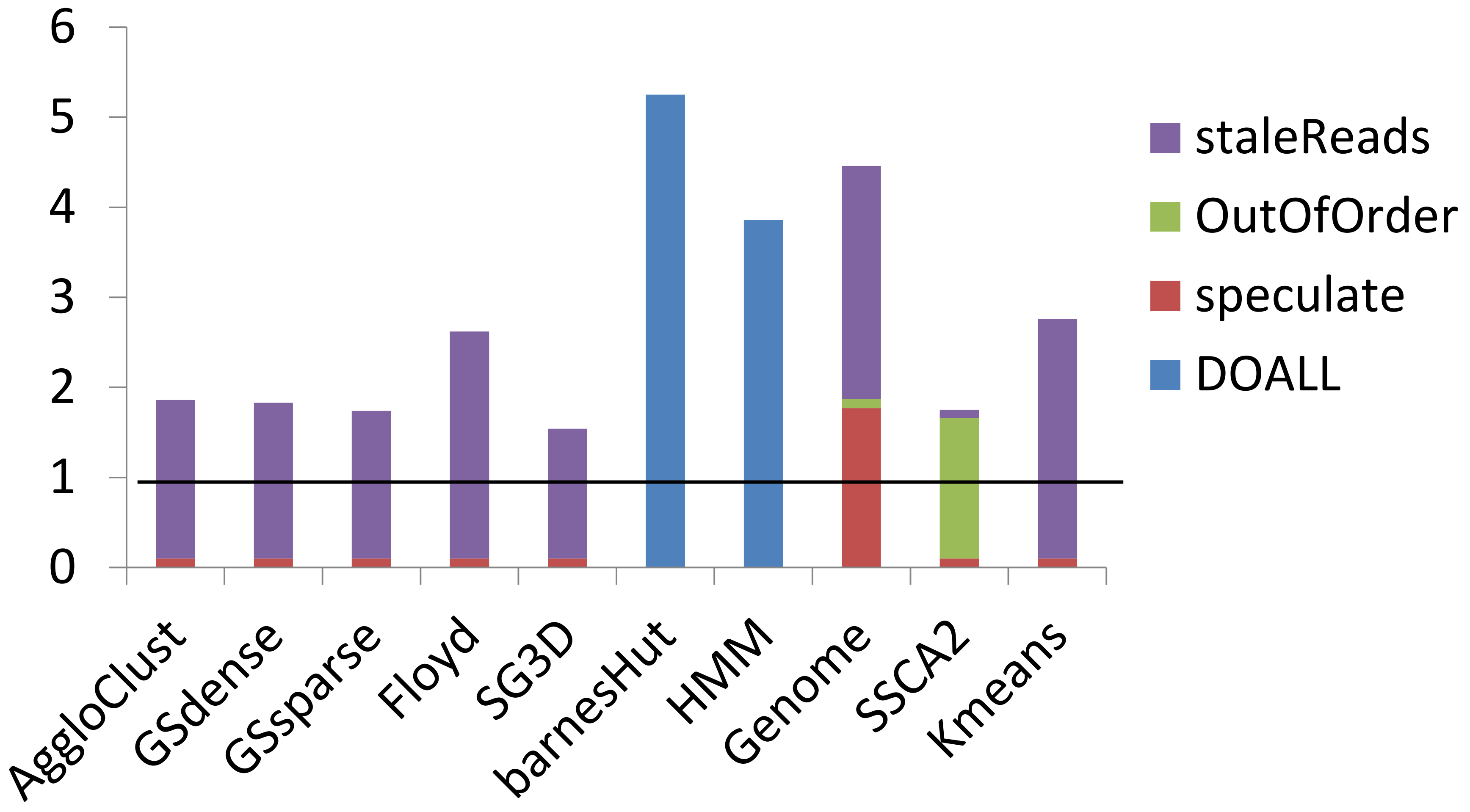
Results : Baseline



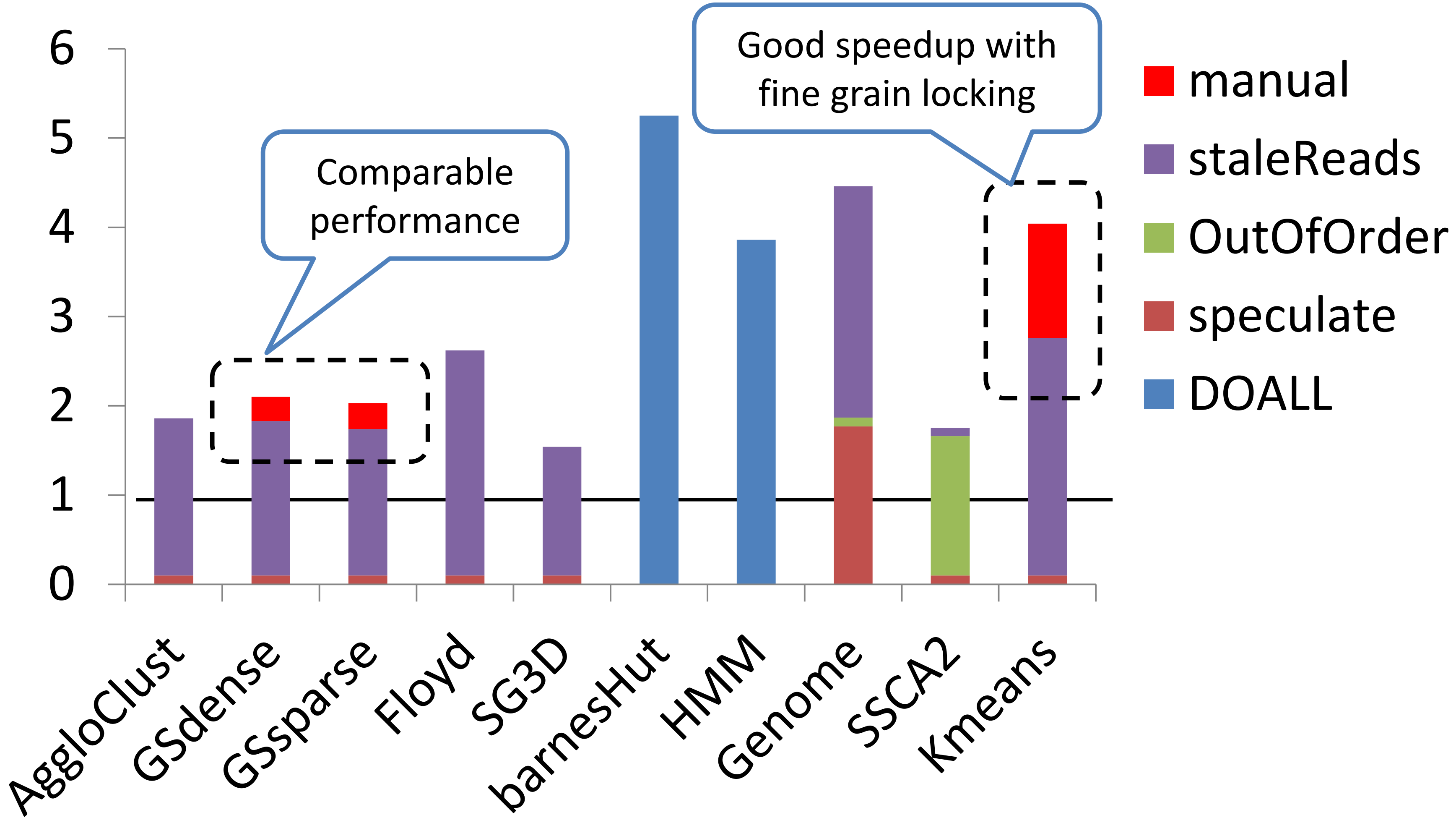
No scope for dependence speculation

No scope for dependence speculation

Results : Alter



Results: Manual Parallelization



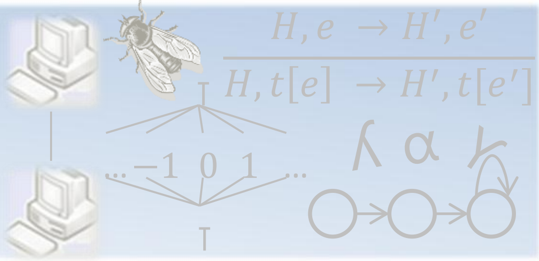
Comparable performance

Good speedup with fine grain locking

In the Paper…

- ALTER multi-process memory allocator
- ALTER collections
- Usage scenario’s for ALTER
- Profiling and instrumentation overhead
- DOALL parallelism and speculation within ALTER

Related Work



- Test-driven parallelization
  – QuickStep: similar testing methods for non-deterministic programs, offers accuracy bounds [Rinard 2010]
- Assisted parallelization [Taylor 2011] [Tournavitis 2009]
  – Paralax: annotations improve precision of analysis, but dependences respected [Vandierendonck 2010]
- Implicit parallelization [Burckhardt 2010]
  – Commutative annotation for reordering[August 2007, 11]
  – Optimistic execution of irregular programs [Pingali 2008]
  – As far as we know, stale reads execution model is new

Conclusions

- **Breakable dependences** must be exploited in order to parallelize certain classes of programs
- We propose a new execution model, **StaleReads**, that violates dependences in a principled way
- Adopt database notion of **Snapshot Isolation** for loop parallelization
- **ALTER** is a compiler and deterministic runtime system that discovers new parallelism in programs
- We believe tools for **assisted parallelism** can help to overcome the limits of automatic parallelization