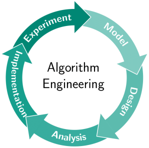
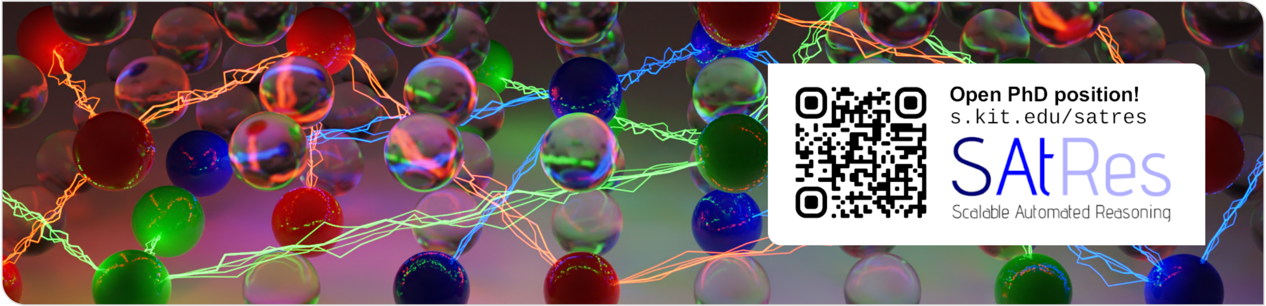
Karlsruher Institut für Technologie



MallobSat: Scalable SAT Solving by Clause Sharing

Journal of Artificial Intelligence Research (JAIR) article · Pragmatics of SAT 2024, Pune, India

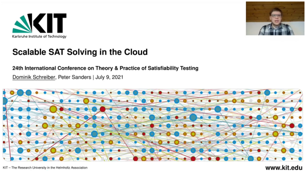
Dominik Schreiber, Peter Sanders | August 20, 2024



Open PhD position!
s.kit.edu/satres

SAtRes
Scalable Automated Reasoning

Context



Scalable SAT Solving in the Cloud

24th International Conference on Theory & Practice of Satisfiability Testing

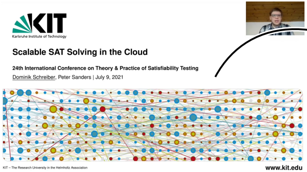
Dominik Schreiber, Peter Sanders | July 9, 2021

KIT – The Research University in the Helmholtz Association

www.kit.edu

SAT'21 paper "Scalable SAT Solving in the Cloud",
SAT Competition '20–23

Context



Karlsruhe Institute of Technology

Scalable SAT Solving in the Cloud

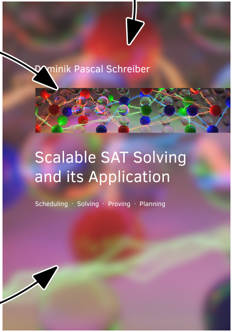
24th International Conference on Theory & Practice of Satisfiability Testing

Dominik Schreiber, Peter Sanders | July 9, 2021

KIT – The Research University in the Helmholtz Association

www.kit.edu

SAT'21 paper "Scalable SAT Solving in the Cloud",
SAT Competition '20–23



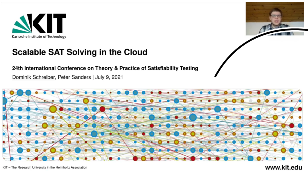
Dominik Pascal Schreiber

Scalable SAT Solving and its Application

Scheduling · Solving · Proving · Planning

2023: Dissertation

Context



Karlsruhe Institute of Technology

Scalable SAT Solving in the Cloud

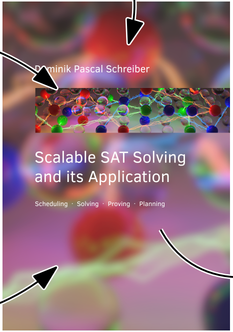
24th International Conference on Theory & Practice of Satisfiability Testing

Dominik Schreiber, Peter Sanders | July 9, 2021

KIT – The Research University in the Helmholtz Association

www.kit.edu

SAT’21 paper “Scalable SAT Solving in the Cloud”,
SAT Competition ’20–23



Dominik Pascal Schreiber

Scalable SAT Solving and its Application

Scheduling · Solving · Proving · Planning

2023: Dissertation

MallobSat: Scalable SAT Solving by Clause Sharing

Dominik Schreiber DOMINIK.SCHREIBER@KIT.EDU
Peter Sanders SANDERS@KIT.EDU
Karlsruhe Institute of Technology,
Kaiserstr. 12, 76131 Karlsruhe, Germany

Abstract

SAT solving in large distributed environments has previously led to some famous results and to impressive speedups for selected inputs. However, in terms of general-purpose SAT solving, prior approaches still cannot make efficient use of a large number of processors. We aim to address this issue with a complete and systematic overhaul of the distributed solver HordeSat with a focus on its algorithmic building blocks. In particular, we present a communication-efficient approach to clause sharing, careful buffering and filtering of produced clauses, and effective orchestration of state-of-the-art solver backends. In extensive evaluations, our approach named MallobSat significantly outperforms an updated HordeSat, doubling its mean speedup. Our clause sharing results in effective parallelization even if all threads execute *identical* solver programs that only differ based on which clauses they import at which times. We thus argue that MallobSat is not a portfolio solver with the added bonus of clause sharing but rather a *clause-sharing solver* where adding some explicit diversification is useful but not essential. We also discuss the last four iterations of the International SAT Competition (2020–2023), where our system ranked very favorably, and identify several previously unsolved competition problems that MallobSat solved successfully. Last but not least, our approach is *malleable*, i.e., supports running on a fluctuating set of resources, which allows us to combine parallel job processing and parallel SAT solving in a flexible manner for best resource efficiency.

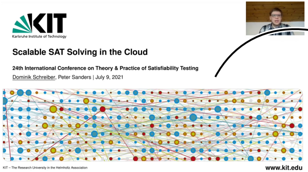
1. Introduction

The *Propositional Satisfiability* (SAT) problem, i.e., to satisfy a given Boolean expression or to report its unsatisfiability, is an essential building block at the core of automated reasoning, symbolic AI, and formal verification (Fichte et al., 2023). In today’s applied SAT solving, researchers and industrial users increasingly strive to exploit modern *distributed* environments with hundreds to thousands of cores (Heisinger et al., 2020; Froleyks et al., 2021; Co[illegible]; Burgess et al., 2022) in order to reduce processing times and to tackle difficult problems that are infeasible to solve with sequential algorithms. Prior massively parallel approaches that successfully solved open mathematical problems (Heule et al., 2016; Subercaseaux & Heule, 2023) are usually fine-tuned to the particular problem at hand. In terms of SAT solving that is *general purpose*, i.e., that works efficiently on application problems never seen before, existing distributed solvers still leave much to be desired. Linear or even “*superlinear speedups*” (Balyo et al., 2015) achieved for few individual instances must be set in relation with the total work invested in every single formula to achieve such peak speedups. Specifically, the prior state of the art in distributed SAT solving, HordeSat, achieved a median speedup of 13 on 2048 cores on industrial benchmarks (Balyo et al., 2015), implying a median efficiency of only 0.6%.

©2024 The Authors. Published by AI Access Foundation under Creative Commons Attribution License CC BY 4.0.

August 2024: JAIR article “MallobSat: Scalable SAT Solving by Clause Sharing”

Context



Scalable SAT Solving in the Cloud

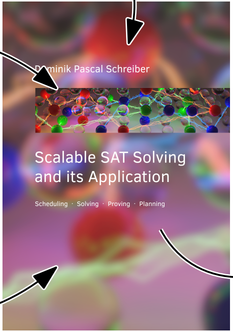
24th International Conference on Theory & Practice of Satisfiability Testing

Dominik Schreiber, Peter Sanders | July 9, 2021

KIT – The Research University in the Helmholtz Association

www.kit.edu

SAT'21 paper "Scalable SAT Solving in the Cloud",
SAT Competition '20–23

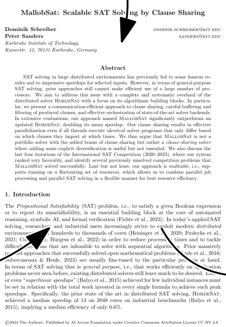


Dominik Pascal Schreiber

Scalable SAT Solving and its Application

Scheduling · Solving · Proving · Planning

2023: Dissertation



MallobSat: Scalable SAT Solving by Clause Sharing

Dominik Schreiber DOMINIK.SCHREIBER@KIT.EDU
Peter Sanders SANDERS@KIT.EDU
Karlsruhe Institute of Technology,
Kaiserstr. 12, 76131 Karlsruhe, Germany

Abstract

SAT solving in large distributed environments has previously led to some famous results and to impressive speedups for selected inputs. However, in terms of general-purpose SAT solving, prior approaches still cannot make efficient use of a large number of processors. We aim to address this issue with a complete and systematic overhaul of the distributed solver HordeSat with a focus on its algorithmic building blocks. In particular, we present a communication-efficient approach to clause sharing, careful buffering and filtering of produced clauses, and effective orchestration of state-of-the-art solver backends. In extensive evaluations, our approach named MallobSat significantly outperforms an updated HordeSat, doubling its mean speedup. Our clause sharing results in effective parallelization even if all threads execute *identical* solver programs that only differ based on which clauses they import at which times. We thus argue that MallobSat is not a portfolio solver with the added bonus of clause sharing but rather a *clause-sharing solver* where adding some explicit diversification is useful but not essential. We also discuss the last four iterations of the International SAT Competition (2020–2023), where our system ranked very favorably, and identify several previously unsolved competition problems that MallobSat solved successfully. Last but not least, our approach is *malleable*, i.e., supports running on a fluctuating set of resources, which allows us to combine parallel job processing and parallel SAT solving in a flexible manner for best resource efficiency.

1. Introduction

The *Propositional Satisfiability* (SAT) problem, i.e., to satisfy a given Boolean expression or to report its unsatisfiability, is an essential building block at the core of automated reasoning, symbolic AI, and formal verification (Fichte et al., 2023). In today's applied SAT solving, researchers and industrial users increasingly strive to exploit modern *distributed* environments with hundreds to thousands of cores (Heisinger et al., 2020; Froleyks et al., 2021; Co[illegible]; Burgess et al., 2022) in order to reduce processing times and to tackle difficult problems that are infeasible to solve with sequential algorithms. Prior massively parallel approaches that successfully solved open mathematical problems (Heule et al., 2016; Subercaseaux & Heule, 2023) are usually fine-tuned to the particular problem at hand. In terms of SAT solving that is *general purpose*, i.e., that works efficiently on application problems never seen before, existing distributed solvers still leave much to be desired. Lin[illegible] or even "*superlinear speedups*" (Balyo et al., 2015) achieved for few individual instances must be set in relation with the total work invested in every single formula to achieve such peak speedups. Specifically, the prior state of the art in distributed SAT solving, HordeSat, achieved a median speedup of 13 on 2048 cores on industrial benchmarks (Balyo et al., 2015), implying a median efficiency of only 0.6%.

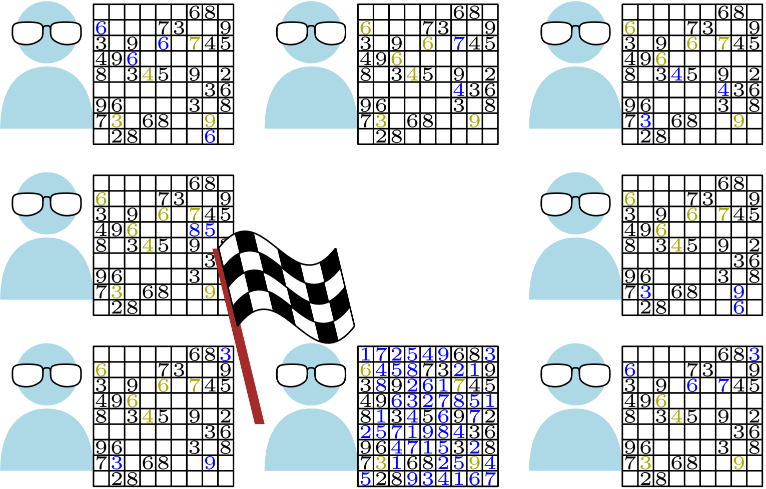
©2024 The Authors. Published by AI Access Foundation under Creative Commons Attribution License CC BY 4.0.

August 2024: JAIR article "MallobSat: Scalable SAT Solving by Clause Sharing"

● LIVE

PoS'24 presentation

Reminder from Two Talks Earlier



Cooperative portfolio

- All experts work on original problem independently
- Brief meetings to exchange crucial insights
- Insights accelerate solving
- Only one expert needs to find a solution!

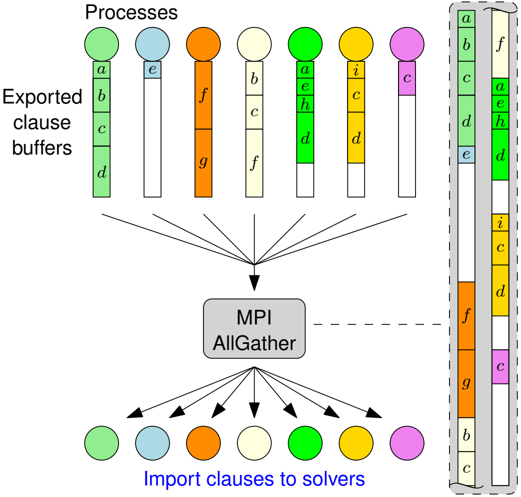
Parallel SAT

- Experts ≡ diversified sequential SAT solver threads
- Shared information ≡ conflict clauses

Distributed SAT: Prior State of the Art

Massively parallel solver **HordeSat** [Balyo et al. 2015]

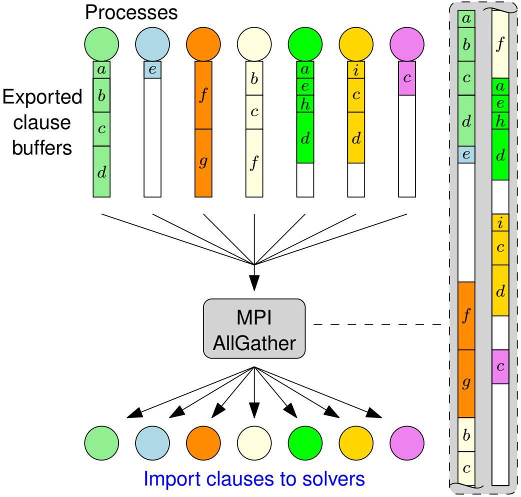
- Modular interface to plug in sequential solvers
- Periodic clause exchange
  - Concatenation of fixed-size clause buffers
  - Duplicates, unused space in buffers
    ⇒ often low number of distinct shared clauses



Distributed SAT: Prior State of the Art

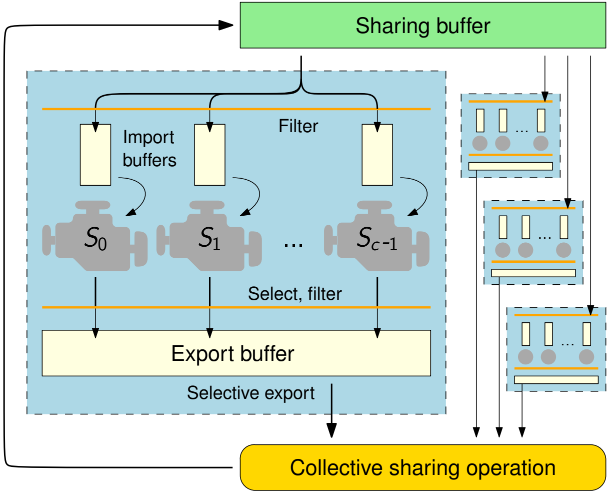
Massively parallel solver **HordeSat** [Balyo et al. 2015]

- Modular interface to plug in sequential solvers
- Periodic clause exchange
  - Concatenation of fixed-size clause buffers
  - Duplicates, unused space in buffers
    $\Rightarrow$ often low number of distinct shared clauses
- Experiments with $\leq$ 2048 cores
  - Surprisingly good scaling for difficult instances
  - Median speedup at 2048 cores: 13 (efficiency 0.6%)



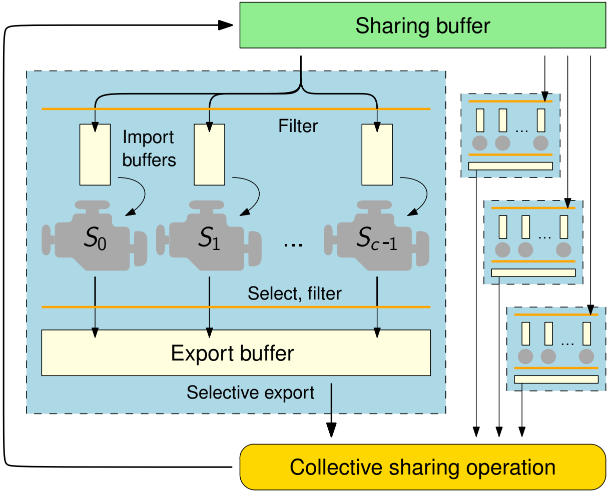
Design Decisions of MALLOBSAT

- **Fundament:** HORDESAT
  - Two-level hybrid parallelization
  - Periodic all-to-all clause sharing



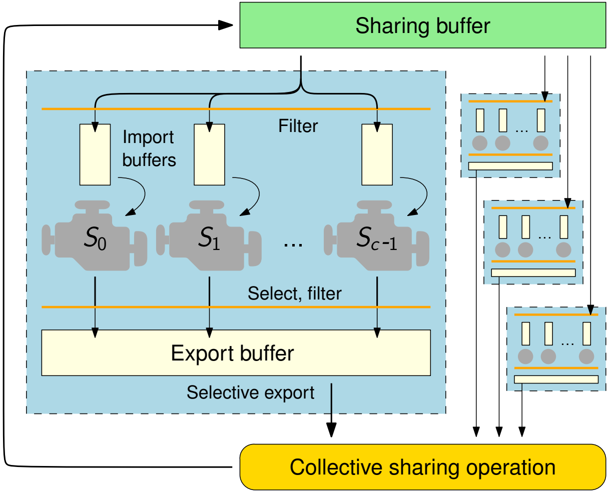
Design Decisions of MALLOBSAT

- **Fundament:** HORDESAT
  – Two-level hybrid parallelization
  – Periodic all-to-all clause sharing
- Fix a certain sharing volume, spend it on the globally most useful **distinct** clauses



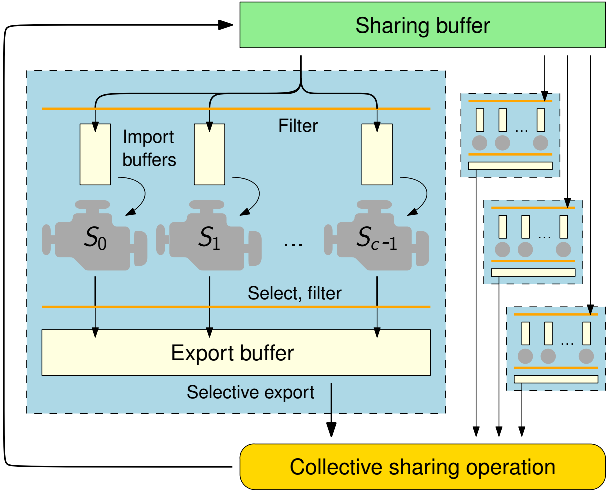
Design Decisions of MallobSat

- **Fundament:** HordeSat
  - – Two-level hybrid parallelization
  - – Periodic all-to-all clause sharing
- Fix a certain sharing volume, spend it on the globally most useful **distinct** clauses
- Prioritize clauses by clause length



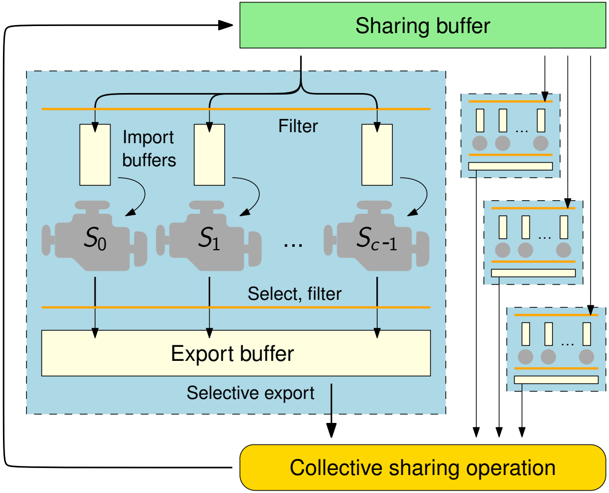
Design Decisions of MALLOBSAT

- **Fundament:** HORDESAT
  – Two-level hybrid parallelization
  – Periodic all-to-all clause sharing
- Fix a certain sharing volume, spend it on the globally most useful **distinct** clauses
- Prioritize clauses by clause length
- Minimize clause turnaround times



Design Decisions of MALLOBSAT

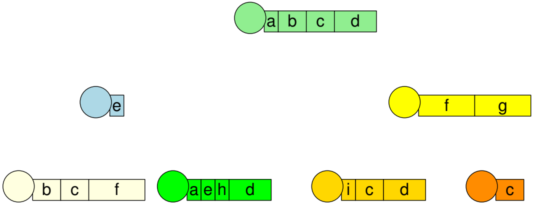
- **Fundament:** HORDESAT
  - – Two-level hybrid parallelization
  - – Periodic all-to-all clause sharing
- Fix a certain sharing volume, spend it on the globally most useful **distinct** clauses
- Prioritize clauses by clause length
- Minimize clause turnaround times
- Support fluctuating workers (*malleability*)



Clause sharing: Our approach

Exchange of useful clauses

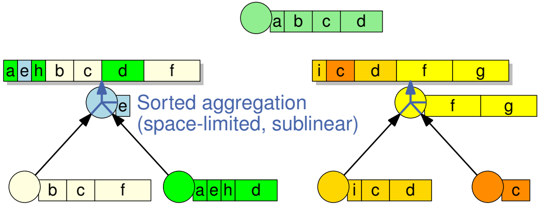
1.



Clause sharing: Our approach

Exchange of useful clauses

1.

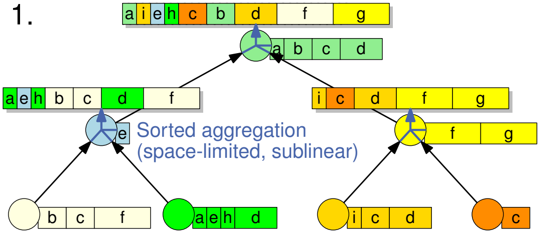


Sorted aggregation (space-limited, sublinear)

Clause sharing: Our approach

Exchange of useful clauses

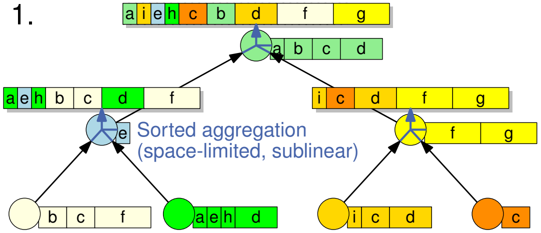
1.



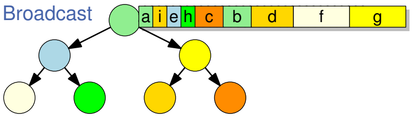
Clause sharing: Our approach

Exchange of useful clauses

1.



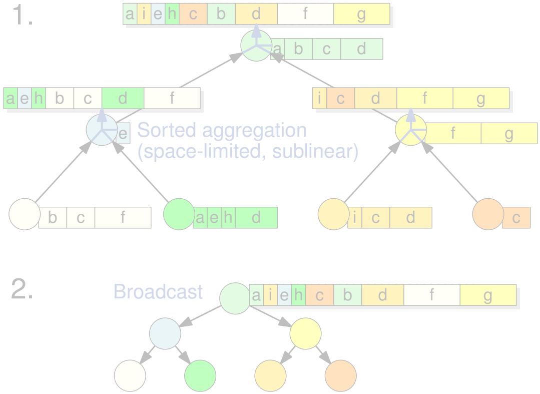
2.



Clause sharing: Our approach

Exchange of useful clauses

1.

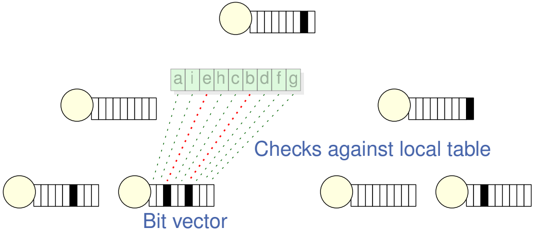


2.

Broadcast

Filtering of recently shared clauses

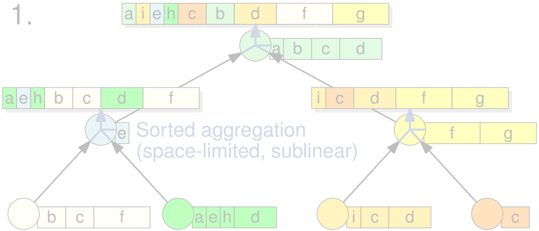
3.



Clause sharing: Our approach

Exchange of useful clauses

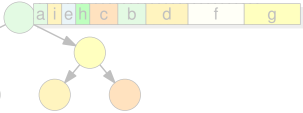
1.



Sorted aggregation
(space-limited, sublinear)

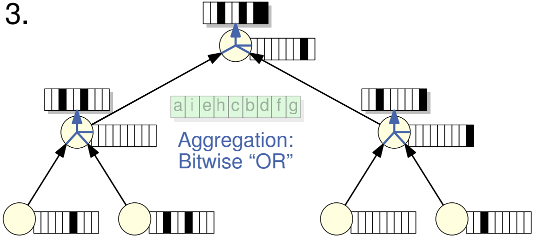
2.

Broadcast



Filtering of recently shared clauses

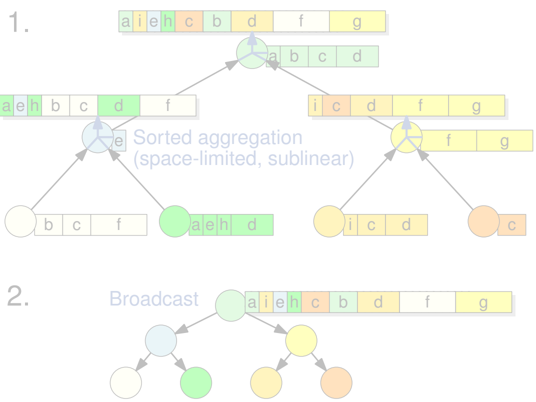
3.



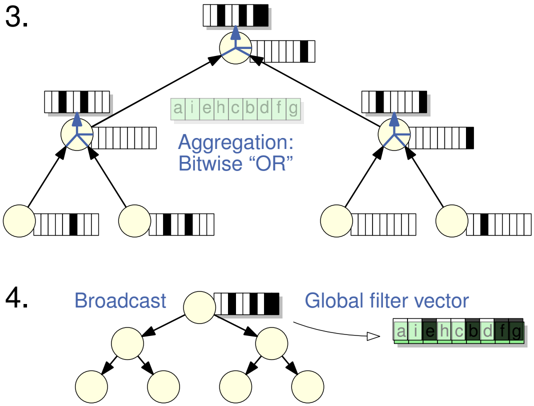
Aggregation:
Bitwise "OR"

Clause sharing: Our approach

Exchange of useful clauses



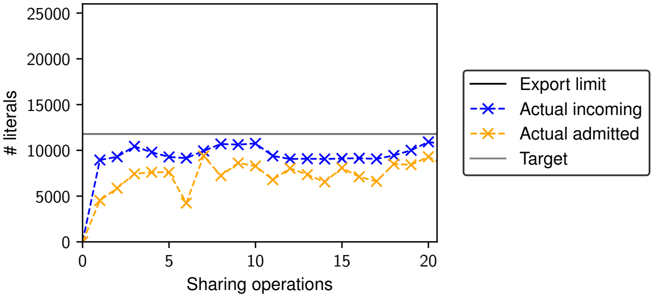
Filtering of recently shared clauses



Enforcing a Sharing Volume

We want to share L literals per sharing but may only get $L' < L$ successfully shared literals. Why?

1. Processes didn't produce, export enough clauses
2. Duplicate clauses were detected and eliminated during aggregation
3. Distributed filter blocked some of the transmitted clauses

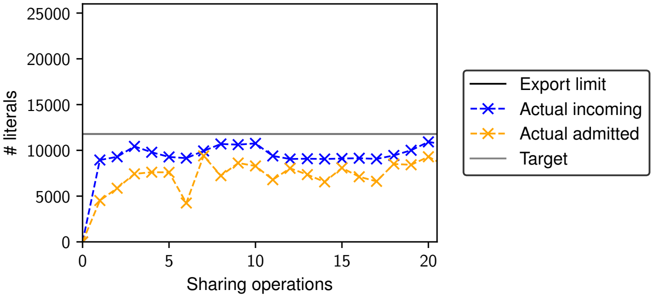


Enforcing a Sharing Volume

We want to share L literals per sharing but may only get $L' < L$ successfully shared literals. Why?

1. Processes didn't produce, export enough clauses
2. Duplicate clauses were detected and eliminated during aggregation
3. Distributed filter blocked some of the transmitted clauses

Fix: Elastic compensation for sharing volume unused for algorithmic reasons (2 , 3)

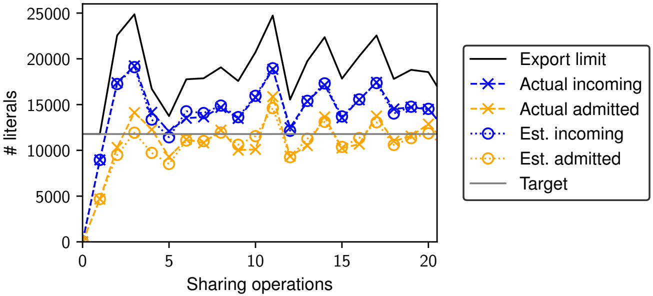


Enforcing a Sharing Volume

We want to share L literals per sharing but may only get $L' < L$ successfully shared literals. Why?

1. Processes didn't produce, export enough clauses
2. Duplicate clauses were detected and eliminated during aggregation
3. Distributed filter blocked some of the transmitted clauses

Fix: Elastic compensation for sharing volume unused for algorithmic reasons (2 , 3)



Handling LBD Values

- Seq. solving: central metric for whether to keep a clause
- **But:** LBD found by solver A not necessarily meaningful for solver B! → not as "global" as clause length

Handling LBD Values

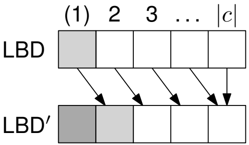
- Seq. solving: central metric for whether to keep a clause
- **But:** LBD found by solver A not necessarily meaningful for solver B! → not as “global” as clause length
- Some solvers keep clauses with LBD 2 indefinitely
  — but expect a single solver’s clause volume!
  ⇒ Growing overhead (time, space) from low-LBD clauses

Handling LBD Values

- Seq. solving: central metric for whether to keep a clause
- **But:** LBD found by solver A not necessarily meaningful for solver B! → not as “global” as clause length
- Some solvers keep clauses with LBD 2 indefinitely
  — but expect a single solver’s clause volume!
  ⇒ Growing overhead (time, space) from low-LBD clauses

Our current approach: Increment each LBD before import

- Maintains LBD-based prioritization of clauses
- Solver keeps more control over its LBD-2-clauses



	Median RAM	PAR-2
Orig. LBD	108.8 GiB	75.7
Reset LBD	95.6 GiB	74.3
LBD++	97.3 GiB	72.9

768 cores × 349 instances × 300 s

Portfolio & Diversification

Solver backends:

- LINGELING + YALSAT local search solver
- GLUCOSE + SYRUP clause sharing code
- CADICAL
- KISSAT

Diversification:

- Cycle through solver configuration options: restart intervals, pre–/inprocessing techniques, . . .
- Sparse random variable phases
- Seeds, input shuffling, Gaussian noise for numeric parameters

Pitfalls in Parallel Solver Evaluation

Balyo et al. (2016):

"*Experiments showed that* HordeSAT *can achieve superlinear average speedup on hard benchmarks.*"

Pitfalls in Parallel Solver Evaluation

Balyo et al. (2016):

"*Experiments showed that* HORDESAT *can achieve superlinear average speedup on hard benchmarks.*"

Superlinear speedups: erratic, often due to running time variance or some crucial solver configuration the sequential solver is missing

Pitfalls in Parallel Solver Evaluation

Arithmetic average of speedups:
no statistical meaning –
use geometric mean or median

Balyo et al. (2016):

“*Experiments showed that* HordeSAT *can achieve superlinear average speedup on hard benchmarks.*”

Superlinear speedups: erratic,
often due to running time variance
or some crucial solver configuration
the sequential solver is missing

Pitfalls in Parallel Solver Evaluation

Arithmetic average of speedups:
no statistical meaning –
use geometric mean or median

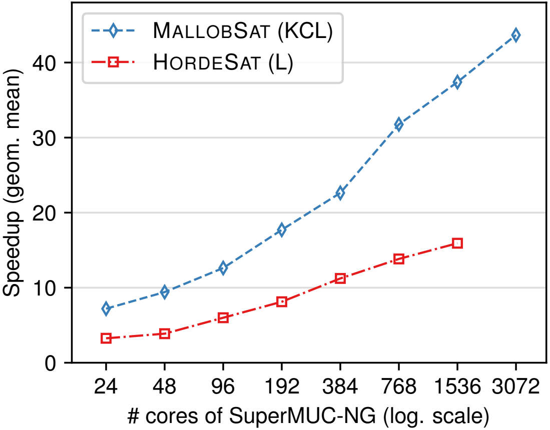
Balyo et al. (2016):

"*Experiments showed that* HordeSAT *can achieve superlinear average speedup on hard benchmarks.*"

Superlinear speedups: erratic,
often due to running time variance
or some crucial solver configuration
the sequential solver is missing

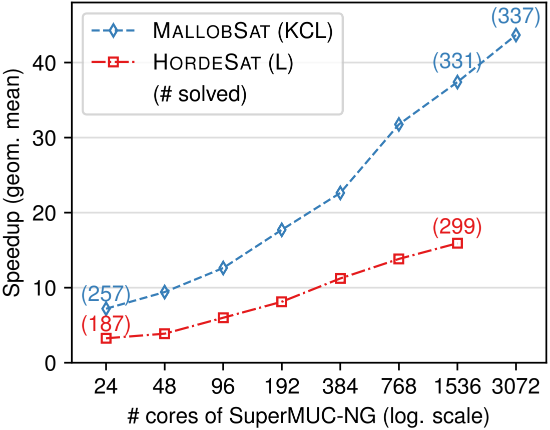
Counting sequential timeouts as
solved at the timeout:
makes speedups difficult to interpret,
especially for huge timeouts

Scaling



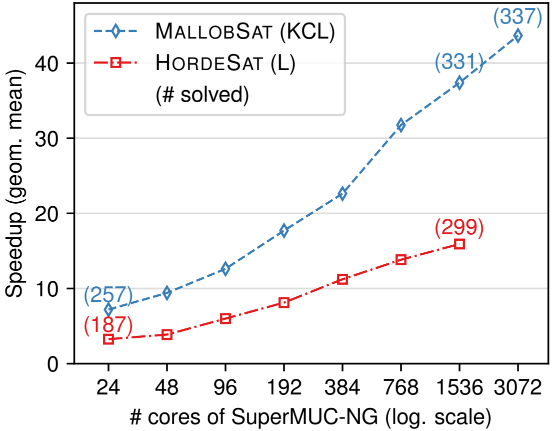
400 problems from SAT Comp. 2021 · Seq. baseline KISSAT_MAB-HYWALK · Seq. limit 32 h (331 solved) · Par. limit 300 s

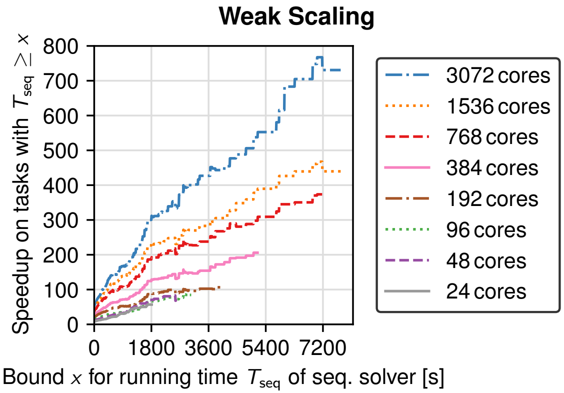
Scaling



400 problems from SAT Comp. 2021 · Seq. baseline KISSAT_MAB-HYWALK · Seq. limit 32 h (331 solved) · Par. limit 300 s

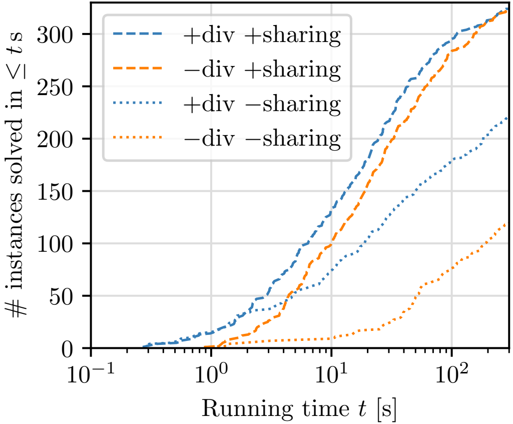
Scaling





400 problems from SAT Comp. 2021 · Seq. baseline KISSAT_MAB-HYWALK · Seq. limit 32 h (331 solved) · Par. limit 300 s

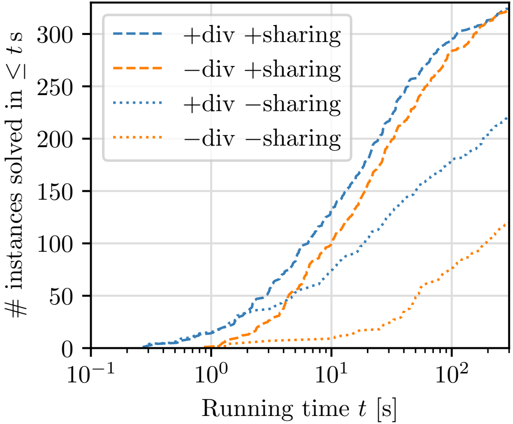
Impact of Diversification, Sharing @ 768 Cores



- Without sharing, diversification is highly effective

349 problems from SAT Comp. 2022 · KCL portfolio

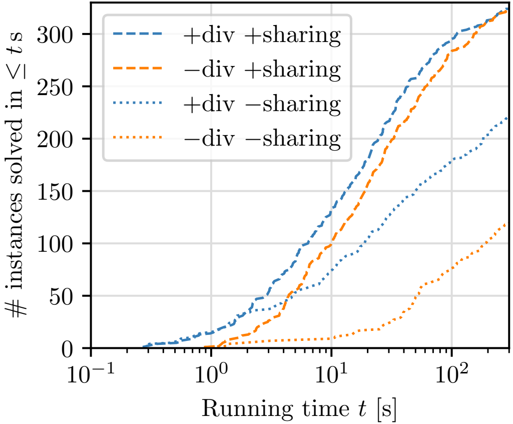
Impact of Diversification, Sharing @ 768 Cores



- Without sharing, diversification is highly effective
- With sharing: Only ≈ 40 distinct solver programs across 768 cores still perform competitively?!

349 problems from SAT Comp. 2022 · KCL portfolio

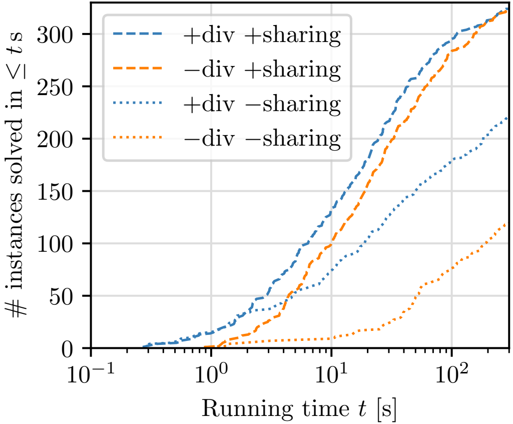
Impact of Diversification, Sharing @ 768 Cores



- Without sharing, diversification is highly effective
- With sharing: Only ≈ 40 distinct solver programs across 768 cores still perform competitively?!
  - Threads receive shared clauses at differing points in time
  - “Butterfly effect” ⇒ deviating exploration
  - Clause sharing as distributed search space pruning

349 problems from SAT Comp. 2022 · KCL portfolio

Impact of Diversification, Sharing @ 768 Cores



- Without sharing, diversification is highly effective
- With sharing: Only ≈ 40 distinct solver programs across 768 cores still perform competitively?!
  - Threads receive shared clauses at differing points in time
  - “Butterfly effect” ⇒ deviating exploration
  - Clause sharing as distributed search space pruning
- Similar findings @ 3072 cores
  - Default CaDiCaL with primitive diversification (seeds, phases) performs competitively
  - Fully diversified portfolio without clause sharing does not

349 problems from SAT Comp. 2022 · KCL portfolio

MallobSat: A Portfolio Solver?

Prevalent concept in literature: **Portfolio solver** *with clause sharing* / Clause-sharing portfolio

- "*each thread runs a different SAT solver on the same instance[, which] in combination with clause-sharing leads to surprisingly good performance for small portfolio sizes*" – Ozdemir et al., 2021

MallobSat: A Portfolio Solver?

Prevalent concept in literature: **Portfolio solver** *with clause sharing* / Clause-sharing portfolio

- "*each thread runs a different SAT solver on the same instance[, which] in combination with clause-sharing leads to surprisingly good performance for small portfolio sizes*" – Ozdemir et al., 2021

Our view, based on empirical observations:

MallobSat is a Clause-sharing solver *with diversification*

- Clause sharing = main driver of scalability
- Adding explicit diversification is beneficial but not essential
- Applicability to other solvers?

MallobSat: Further Notes

MallobSat ...

- ... was the best cloud solver 2020–23 and among the top parallel solvers 2021–23
- ... was able to solve 22/100 previously unsolved instances within 20 min @ 3072 cores
- ... performs well in on-demand settings coupled with malleable job scheduling
    - Solve hundreds of instances at once, redistributing resources based on perceived difficulty

MallobSat: Further Notes

MallobSat . . .

- . . . was the best cloud solver 2020–23 and among the top parallel solvers 2021–23
- . . . was able to solve 22/100 previously unsolved instances within 20 min @ 3072 cores
- . . . performs well in on-demand settings coupled with malleable job scheduling
  - Solve hundreds of instances at once, redistributing resources based on perceived difficulty
- . . . supports incremental SAT solving (CaDiCaL, Lingeling only; no proofs)
- . . . supports proof checking (CaDiCaL only; non-incremental only)
  – more on Thursday, 10:30 AM!

Wrap-Up

Testimonials

“*Mallob-mono is now, by a* ***wide*** *margin, the most powerful SAT solver on the planet.*” —Byron Cook, Amazon Science, 2021

https://www.amazon.science/blog/automated-reasonings-scientific-frontiers

Best cloud solver @ International SAT Competition 2020–2023

Mallob(Sat) @ GitHub

github.com/domschrei/mallob

JAIR article

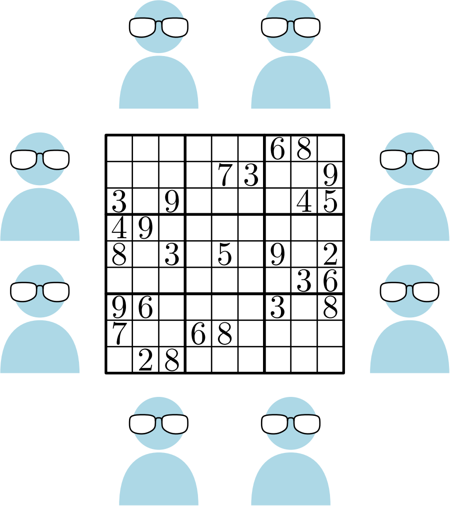
jair.org/index.php/jair/article/view/15827

Parallel Logical Reasoning

The assembly of logicians

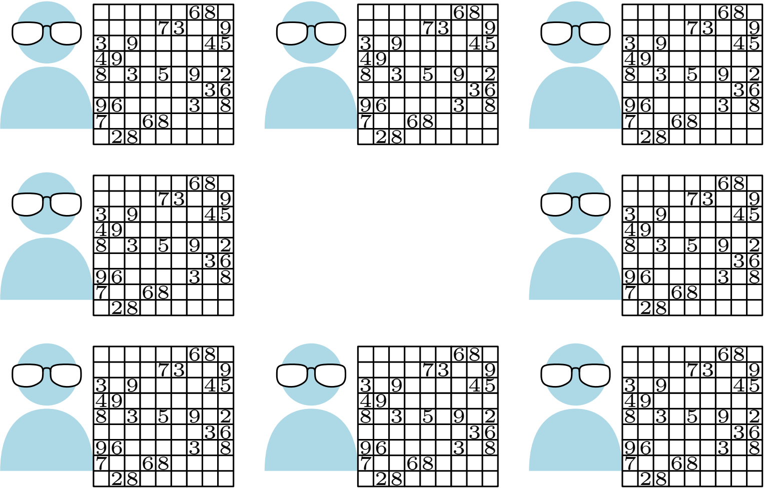
- Complex logic puzzle
- *n* logic experts want to solve the puzzle
- Experts tend to work the best undisturbed

How to coordinate our experts?



						6	8	
				7	3			9
3		9					4	5
4	9							
8		3		5		9		2
							3	6
9	6					3		8
7			6	8				
	2	8						

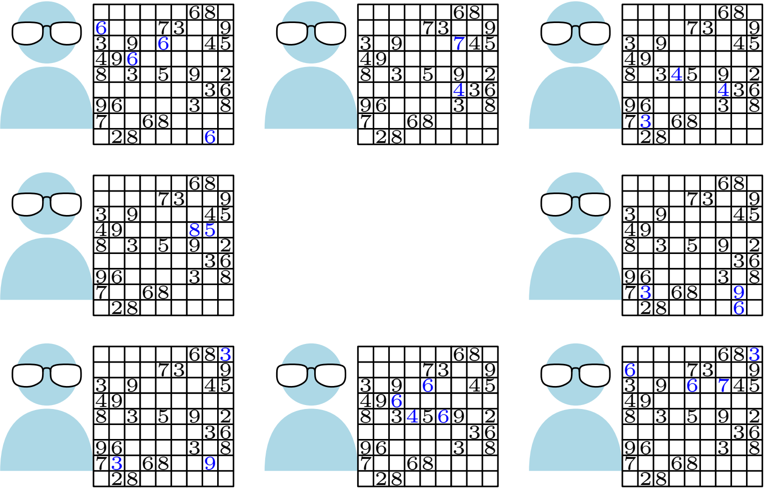
Pure Portfolio



Parallel portfolio

- All experts work on original problem independently

Pure Portfolio



Parallel portfolio

- All experts work on original problem independently
- Different approaches lead to different insights
- Only one expert needs to find a solution!

Pure Portfolio: Oracle view vs. Speedup view

Virtual Best Solver (VBS) / Oracle

Consider n algorithms $A_1, \dots, A_n$ where for each input x, algorithm A_i has run time $T_{A_i}(x)$.
The Virtual Best Solver (VBS) for $A_1, \dots, A_n$ has run time $T^*(x) = \min\{T_{A_1}(x), \dots, T_{A_n}(x)\}$.

Pure Portfolio: Oracle view vs. Speedup view

Virtual Best Solver (VBS) / Oracle

Consider n algorithms $A_1, \ldots, A_n$ where for each input x, algorithm A_i has run time $T_{A_i}(x)$.
The Virtual Best Solver (VBS) for $A_1, \ldots, A_n$ has run time $T^*(x) = \min\{T_{A_1}(x), \ldots, T_{A_n}(x)\}$.

Optimist: A pure portfolio simulates the VBS using parallel processing!

- On idealized hardware, we “select” best sequential solver for each instance

Pure Portfolio: Oracle view vs. Speedup view

Virtual Best Solver (VBS) / Oracle

Consider n algorithms $A_1, \ldots, A_n$ where for each input x, algorithm A_i has run time $T_{A_i}(x)$.
The Virtual Best Solver (VBS) for $A_1, \ldots, A_n$ has run time $T^*(x) = \min\{T_{A_1}(x), \ldots, T_{A_n}(x)\}$.

Optimist: A pure portfolio simulates the VBS using parallel processing!

- On idealized hardware, we “select” best sequential solver for each instance

Parallel speedup

Given parallel algorithm P and input x, the speedup of P is defined as $s_P(x) = T_Q(x)/T_P(x)$
where Q is the best available sequential algorithm.

Pure Portfolio: Oracle view vs. Speedup view

Virtual Best Solver (VBS) / Oracle

Consider n algorithms $A_1, \dots, A_n$ where for each input x, algorithm A_i has run time $T_{A_i}(x)$.
The Virtual Best Solver (VBS) for $A_1, \dots, A_n$ has run time $T^*(x) = \min\{T_{A_1}(x), \dots, T_{A_n}(x)\}$.

Optimist: A pure portfolio simulates the VBS using parallel processing!

- On idealized hardware, we “select” best sequential solver for each instance

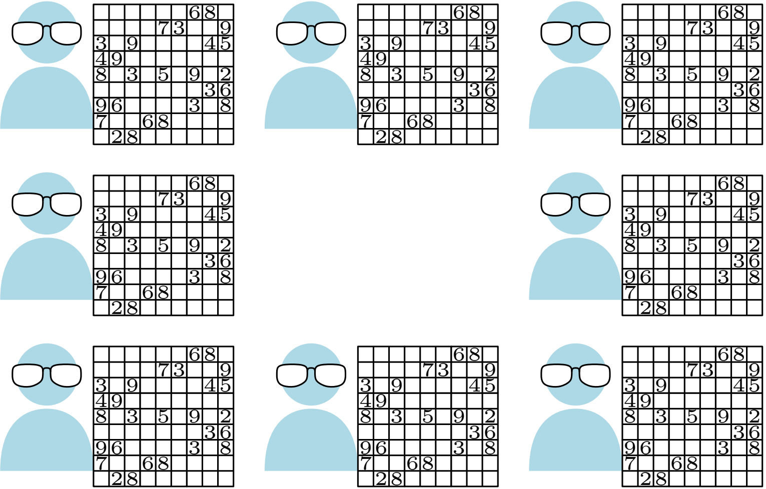
Parallel speedup

Given parallel algorithm P and input x, the speedup of P is defined as $s_P(x) = T_Q(x)/T_P(x)$
where Q is the best available sequential algorithm.

Pessimist: A pure portfolio never achieves actual speedups!

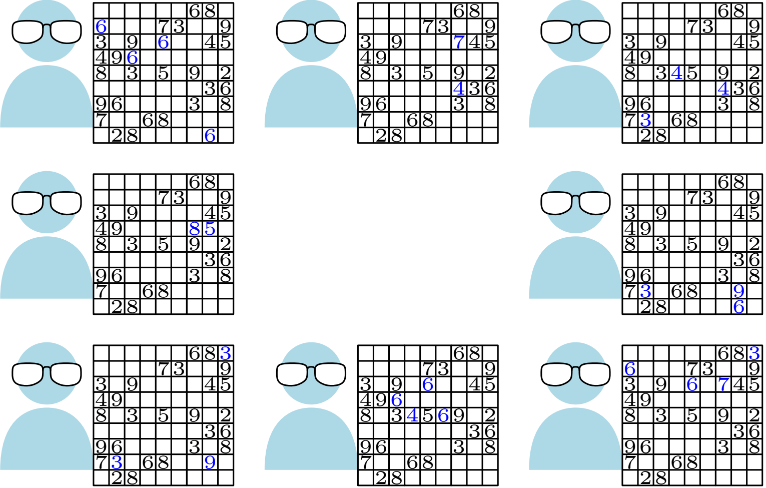
- There is always a sequential algorithm performing at least as well
- Consequence: Not resource efficient, not scalable

Cooperative Portfolio



- All experts work on original problem independently

Cooperative Portfolio



						6	8	
6				7	3			9
3		9		6			4	5
4	9	6						
8		3		5		9		2
							3	6
9	6					3		8
7			6	8				
	2	8					6	

						6	8	
				7	3			9
3		9				7	4	5
4	9							
8		3		5		9		2
						4	3	6
9	6					3		8
7			6	8				
	2	8						

						6	8	
				7	3			9
3		9					4	5
4	9							
8		3	4	5		9		2
						4	3	6
9	6					3		8
7	3		6	8				
	2	8						

						6	8	
				7	3			9
3		9					4	5
4	9					8	5	
8		3		5		9		2
							3	6
9	6					3		8
7			6	8				
	2	8						

						6	8	
				7	3			9
3		9					4	5
4	9							
8		3		5		9		2
							3	6
9	6					3		8
7	3		6	8			9	
	2	8					6	

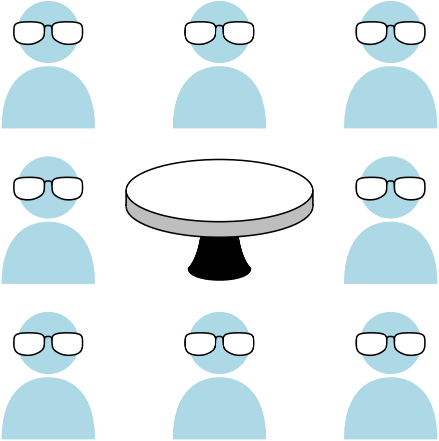
						6	8	3
				7	3			9
3		9					4	5
4	9							
8		3		5		9		2
							3	6
9	6					3		8
7	3		6	8			9	
	2	8						

						6	8	
				7	3			9
3		9		6			4	5
4	9	6						
8		3	4	5	6	9		2
							3	6
9	6					3		8
7			6	8				
	2	8						

						6	8	3
6				7	3			9
3		9		6		7	4	5
4	9							
8		3		5		9		2
							3	6
9	6					3		8
7			6	8				
	2	8						

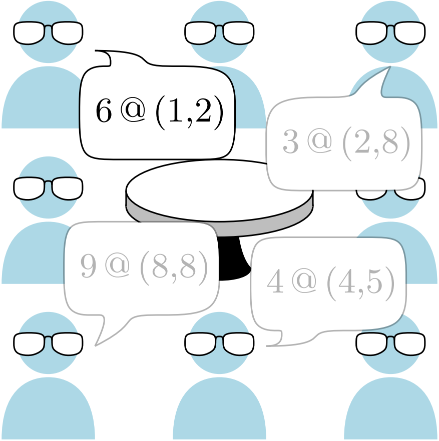
- All experts work on original problem independently

Cooperative Portfolio



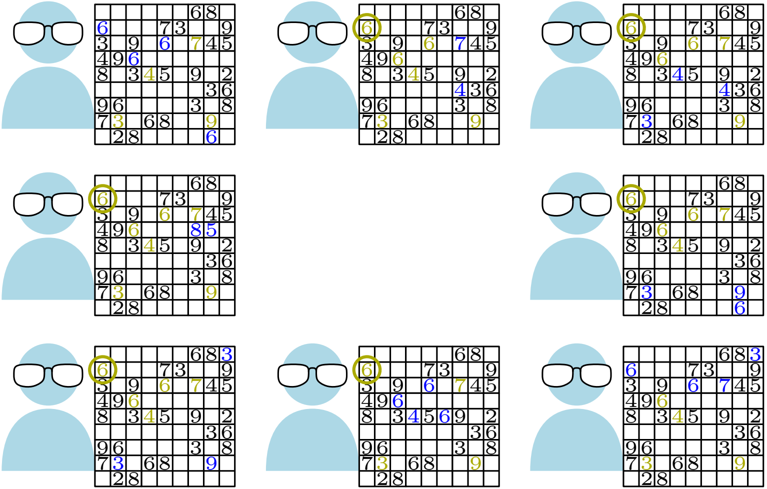
- All experts work on original problem independently
- Brief meetings to exchange crucial insights

Cooperative Portfolio



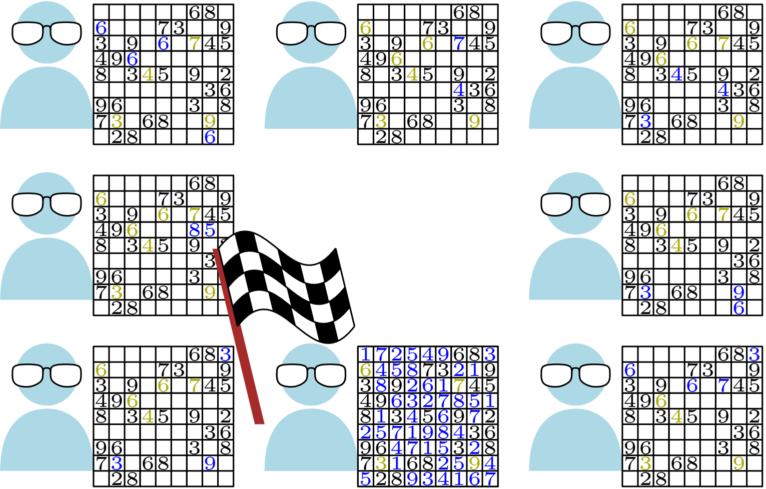
- All experts work on original problem independently
- Brief meetings to exchange crucial insights

Cooperative Portfolio



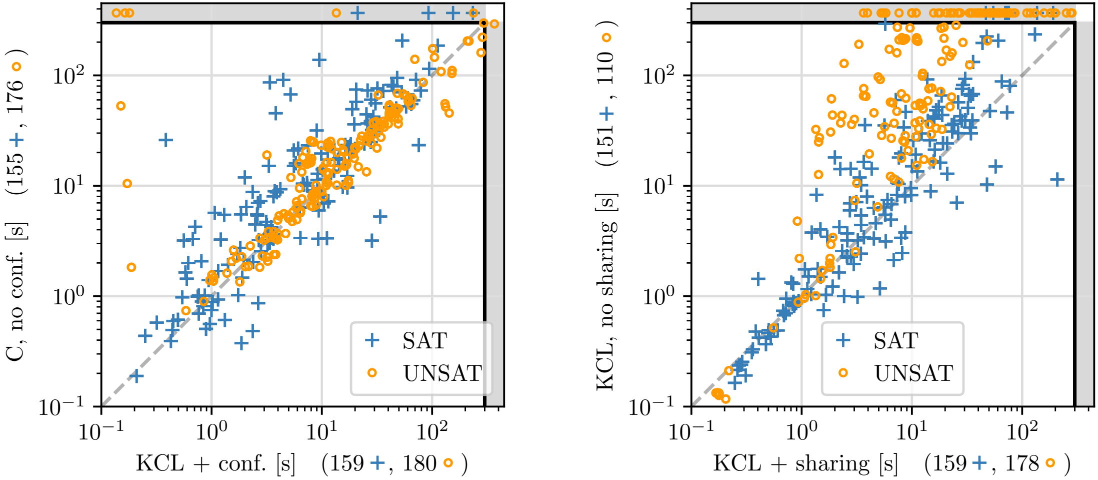
- All experts work on original problem independently
- Brief meetings to exchange crucial insights
- Insights accelerate solving

Cooperative Portfolio

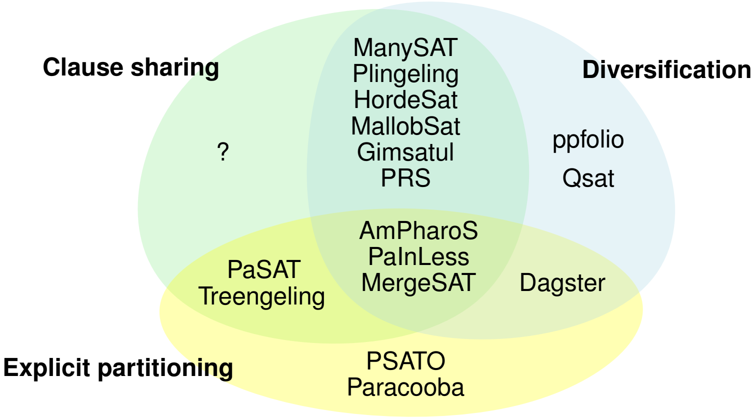


- All experts work on original problem independently
- Brief meetings to exchange crucial insights
- Insights accelerate solving
- Only one expert needs to find a solution!

Impact of Diversification, Sharing @ 3072 Cores



Parallel SAT Landscape



Clause sharing

Diversification

Explicit partitioning

?

ManySAT
Plingeling
HordeSat
MallobSat
Gimsatul
PRS

ppfolio

Qsat

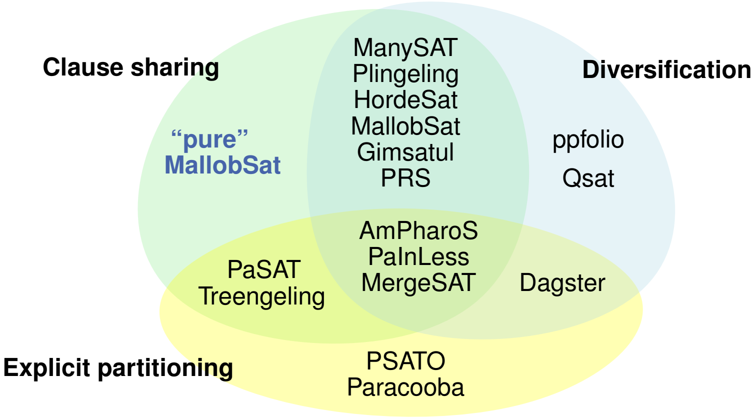
PaSAT
Treengeling

AmPharoS
PaInLess
MergeSAT

Dagster

PSATO
Paracooba

Parallel SAT Landscape



Clause sharing

Diversification

Explicit partitioning

“pure” MallobSat

ManySAT
Plingeling
HordeSat
MallobSat
Gimsatul
PRS

ppfolio
Qsat

PaSAT
Treengeling

AmPharoS
PaInLess
MergeSAT

Dagster

PSATO
Paracooba