Enhancing State-of-the-Art Parallel SAT Solvers Through Optimized Sharing Policies

Vincent Vallade\textsuperscript{1} Julien Sopena\textsuperscript{1} Souheib Baarir\textsuperscript{2}
Sorbonne Université, CNRS, LIP6, UMR 7606, Paris, France\textsuperscript{1}
EPITA, Toulouse, France\textsuperscript{2}

Pragmatics of SAT 2023
Sequential enumeration reaches its limits

CDCL solvers are efficient thanks to:

• Preprocessing \[\text{EB05, PHS08}\]
• Branching Heuristics \[\text{ZMMM01, LGPC16}\]
• Qualifying learned clauses for garbage collection
Sequential enumeration reaches its limits

CDCL solvers are efficient thanks to:

- Preprocessing [EB05, PHS08]
- Branching Heuristics [ZMMM01, LGPC16]
- Qualifying learned clauses for garbage collection

Sequential solving reaches limits:

- Instances became bigger and more complex over time
- Rarity of new heuristics
- Hardware limits (end of Moore’s law)
Sequential enumeration reaches its limits

CDCL solvers are efficient thanks to:

- Preprocessing [EB05, PHS08]
- Branching Heuristics [ZMMM01, LGPC16]
- Qualifying learned clauses for garbage collection

Sequential solving reaches limits:
- Instances became bigger and more complex over time
- Rarity of new heuristics
- Hardware limits (end of Moore’s law)

Developing parallel SAT solvers able to exploit new multicore machines become a necessity.
Interest in parallel SAT solving
Interest in parallel SAT solving

Divide and conquer [ZBH96]

Dynamically divide the search space between each worker
Interest in parallel SAT solving

Divide and conquer
[ZBH96]

Dynamically divide the search space between each worker

Portfolio
[HJS09]

Multiple workers on the whole search space
Interest in parallel SAT solving

Divide and conquer

Dynamically divide the search space between each worker

Portfolio

Multiple workers on the whole search space

Information can be shared between the solvers.
Clause Sharing

\[
\begin{align*}
& z \\
& \neg z \\
& x \\
& \neg x \\
& k \\
& \neg k
\end{align*}
\]
Clause Sharing

Enhancing State-of-the-Art Parallel SAT Solvers Through Optimized Sharing Policies
Clause Sharing

Enhancing State-of-the-Art Parallel SAT Solvers Through Optimized Sharing Policies

July 4, 2023
Clause Sharing

Enhancing State-of-the-Art Parallel SAT Solvers Through Optimized Sharing Policies

July 4, 2023
Clause Sharing

Enhancing State-of-the-Art Parallel SAT Solvers Through Optimized Sharing Policies

July 4, 2023
Clause Sharing

Enhancing State-of-the-Art Parallel SAT Solvers Through Optimized Sharing Policies

July 4, 2023
Clause Sharing
Challenges

In practice, sharing too much clauses can impair performances.

**Algorithmic reasons:**
- slow down unit propagation
- slow down garbage collection

**Concurrency/Hardware reasons:**
- memory contention (cacheline, alloc)
- memory footprint
- synchronization

**How to select clauses to find the right trade-offs between cost and gain?**
(One of the 7 challenges of parallel SAT solving [HW13])
Sharing in P-MCOMSPS

What is shared?
Clauses with a low LBD value [AS09].
Sharing in P-MCOMSPS

*What is shared?*
Clauses with a low LBD value [AS09].

*To whom?*
All solvers in the portfolio are producers and consumers.
Sharing in P-MCOMSPS

What is shared?
Clauses with a low LBD value [AS09].

To whom?
All solvers in the portfolio are producers and consumers

How?
A thread applies the hordesat sharing strategy [BSS15]:
• Try to share 1500 literals
• Prioritize small clauses
• Increase/decrease LBD threshold when producer under/overproduces
Sharing in P-MCOMSPS

*What is shared?*
Clauses with a low LBD value [AS09].

*To whom?*
All solvers in the portfolio are producers and consumers.

*How?*
A thread applies the hordesat sharing strategy [BSS15]:
• Try to share 1500 literals
• Prioritize small clauses
• Increase/decrease LBD threshold when producer under/overproduces

Added mechanisms:
• Bloom filter for exchanged learned clauses [SS21]
• Asynchronous clauses minimization [VFBSK20]
Progressive integration of sharing strategies

P-MCOMSPS sequential engine does not use recent discoveries

Kissat [BFFH20] dominates sequential and parallel solving

Contributions:

• **Implementation on another parallel SAT solver:**
  • We incrementally integrated each mechanism
  • We evaluated them on the SAT 2022 competition bench
  • We demonstrated combinations of mechanisms that improved performance

• **Scaling study performed on 48 and 64 core machines:**
  • Shows good scaling for SAT instances
  • Limited results for UNSAT instances
Incremental integration of sharing components

**Parkissat**: winner of the parallel track 2022 [ZCC22]

![Diagram]

- **Parallelisation**: Portfolio → SequentialWorker
- **Sequential**: 1 → Kissat
- **Sharing**: 1 → Static Sharer

Legend:
- Green: thread
- Orange: algorithm
Incremental integration of sharing components

Parkissat: winner of the parallel track 2022 [ZCC22]

- **STR**: Asynchronous minimisation of clauses
Enhancing State-of-the-Art Parallel SAT Solvers Through Optimized Sharing Policies

Incremental integration of sharing components

**Parkissat**: winner of the parallel track 2022 [ZCC22]

- **STR**: Asynchronous minimisation of clauses
- **Dynamic sharing strategy**: same than P-MCOMSPS
Incremental integration of sharing components

**Parkissat:** winner of the parallel track 2022 [ZCC22]

- **STR:** Asynchronous minimisation of clauses
- **Dynamic sharing strategy:** same than P-MCOMSPS
- **Dup:** Bloom filter for exchanged clauses
**Parkissat**: winner of the parallel track 2022 [ZCC22]

- **STR**: Asynchronous minimisation of clauses
- **Dynamic sharing strategy**: same than P-MCOMSPS
- **Dup**: Bloom filter for exchanged clauses
- **2G**: Add of a second sharing thread
Performance study

SAT instances: race to the solution
- Using more threads for sharing is useful
- Other heuristics are not fruitful
Performance study

SAT instances: race to the solution
- Using more threads for sharing is useful
- Other heuristics are not fruitful

UNSAT instances: race to the unsat core
- Strengthening and horde improve performance
- Duplicates management does not
SAT instances:
The probability of finding a solution increases with the number of threads.
SAT instances:
The probability of finding a solution increases with the number of threads.

UNSAT instances:
Each path must be explored, more computational resources do not remove algorithm limitations.
Conclusion

Contributions:

• Evaluation of multiple sharing strategies

• Boost the performance of the best parallel solver

• Detect a scaling problem in the UNSAT resolution for this solver


Bibliography


