Massively Parallel Bit-Precise Verification with Bitwuzla and Mallob

Dominik Schreiber[*], Aina Niemetz[+], **Mathias Preiner**[+]

TACAS, April 16, 2026

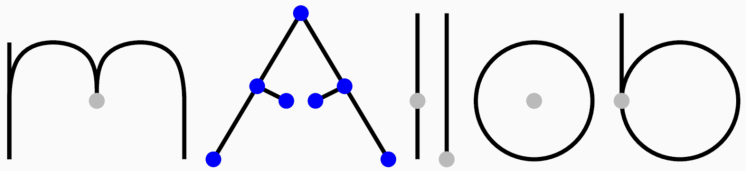
[*]Karlsruhe Institute of Technology, [+]Stanford University

Motivation

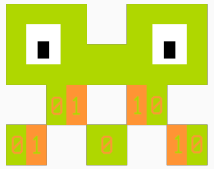
Combine the state of the art in
parallel and distributed SAT solving and bit-precise reasoning in SMT
for a parallel and distributed SMT solver for bit-precise reasoning tasks.

Motivation

Combine the state of the art in
parallel and distributed SAT solving and bit-precise reasoning in SMT
for a parallel and distributed SMT solver for bit-precise reasoning tasks.

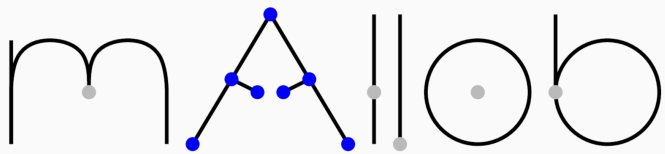


+



Mallob - Distributed Platform for Automated Reasoning

- Malleable job scheduling
  - Flexible balancing of available computational resources
  - Scheduling of individual SAT tasks in parallel and on demand

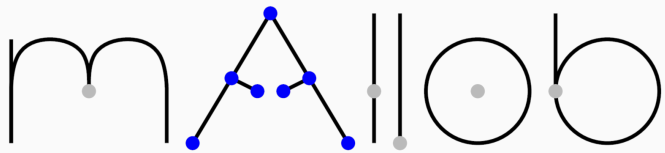


schedule · balance · solve · prove

https://github.com/domschrei/mallob

Mallob - Distributed Platform for Automated Reasoning

- Malleable job scheduling
  - Flexible balancing of available computational resources
  - Scheduling of individual SAT tasks in parallel and on demand
- Integrates parallel and distributed SAT engine MallobSat
  - Full support for incremental SAT solving
  - Clause-sharing

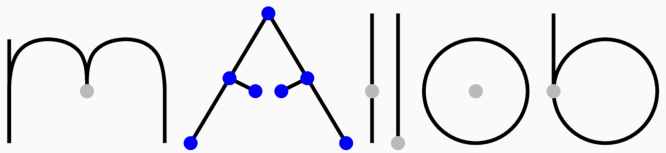


schedule · balance · solve · prove

https://github.com/domschrei/mallob

Mallob - Distributed Platform for Automated Reasoning

- Malleable job scheduling
  - Flexible balancing of available computational resources
  - Scheduling of individual SAT tasks in parallel and on demand
- Integrates parallel and distributed SAT engine MallobSat
  - Full support for incremental SAT solving
  - Clause-sharing
- Efficiently handles incremental SAT solving queries at a distributed scale
- Recently added support for distributed MaxSAT



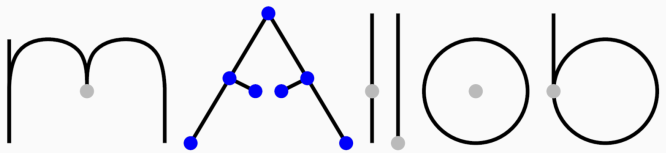
schedule · balance · solve · prove

https://github.com/domschrei/mallob

Mallob - Distributed Platform for Automated Reasoning

- Malleable job scheduling
  - Flexible balancing of available computational resources
  - Scheduling of individual SAT tasks in parallel and on demand
- Integrates parallel and distributed SAT engine MallobSat
  - Full support for incremental SAT solving
  - Clause-sharing
- Efficiently handles incremental SAT solving queries at a distributed scale
- Recently added support for distributed MaxSAT

➤ Now: Support for distributed SMT with Bitwuzla



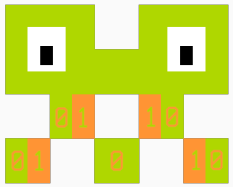
schedule · balance · solve · prove

https://github.com/domschrei/mallob

- Supported Theories: fixed-size bit-vectors, floating-point arithmetic, arrays, uninterpreted functions, quantifiers

Architecture

- Lemmas on demand over bit-vector abstraction
  - Abstraction-refinement approach
  - Bit-vector solver handles Boolean and bit-vector parts
  - Theory solvers handle corresponding theory atoms



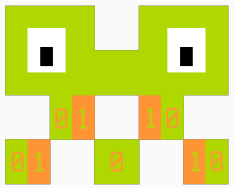
https://bitwuzla.github.io

Bitwuzla - An SMT Solver for Bit-Precise Reasoning

- Supported Theories: fixed-size bit-vectors, floating-point arithmetic, arrays, uninterpreted functions, quantifiers

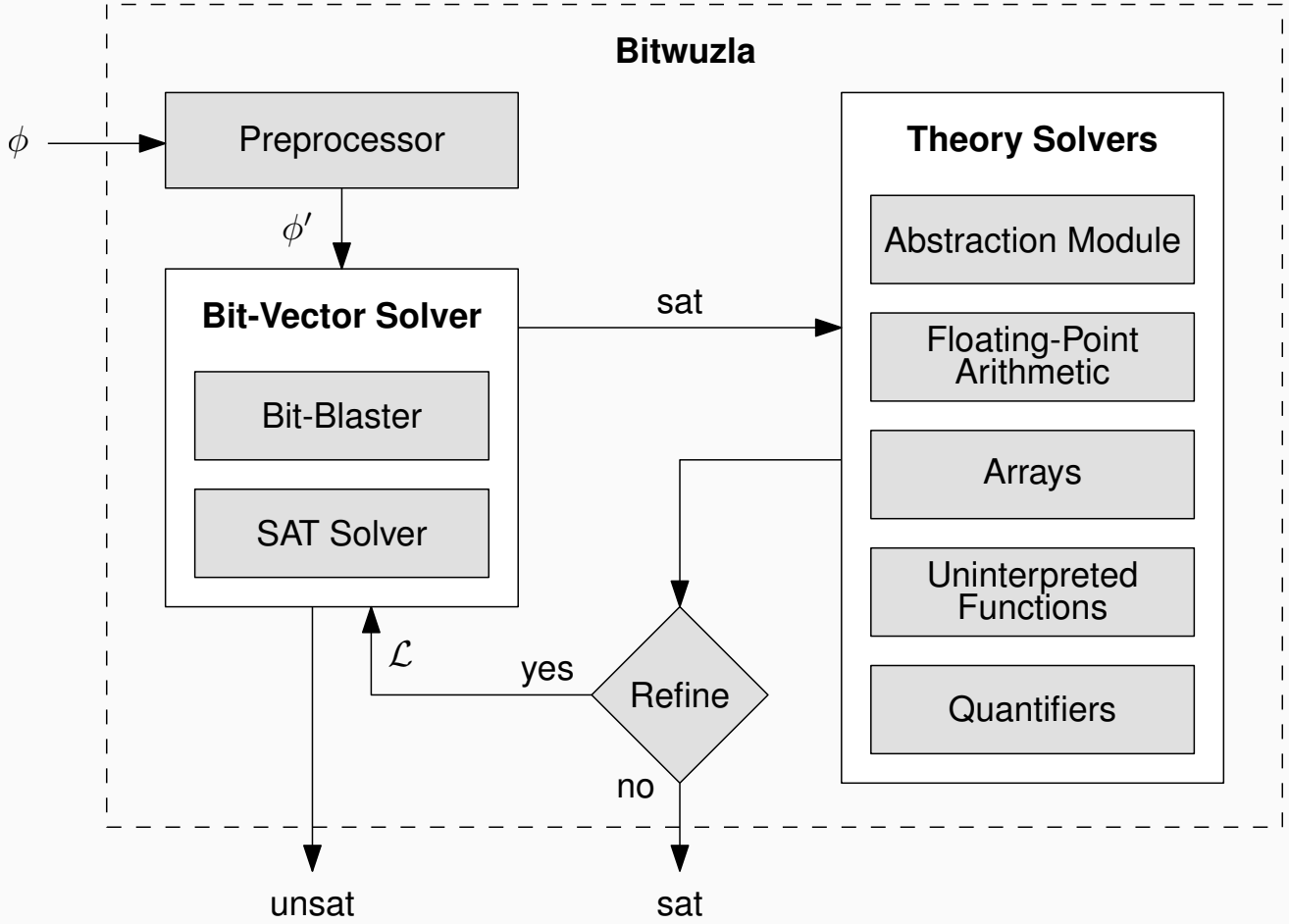
Architecture

- Lemmas on demand over bit-vector abstraction
  - Abstraction-refinement approach
  - Bit-vector solver handles Boolean and bit-vector parts
  - Theory solvers handle corresponding theory atoms
- Bit-vector solver core engine
  - Main technique: bit-blasting
  - Majority of solving time spent in SAT solver (offline)
  - Heavily relies on incremental SAT solving

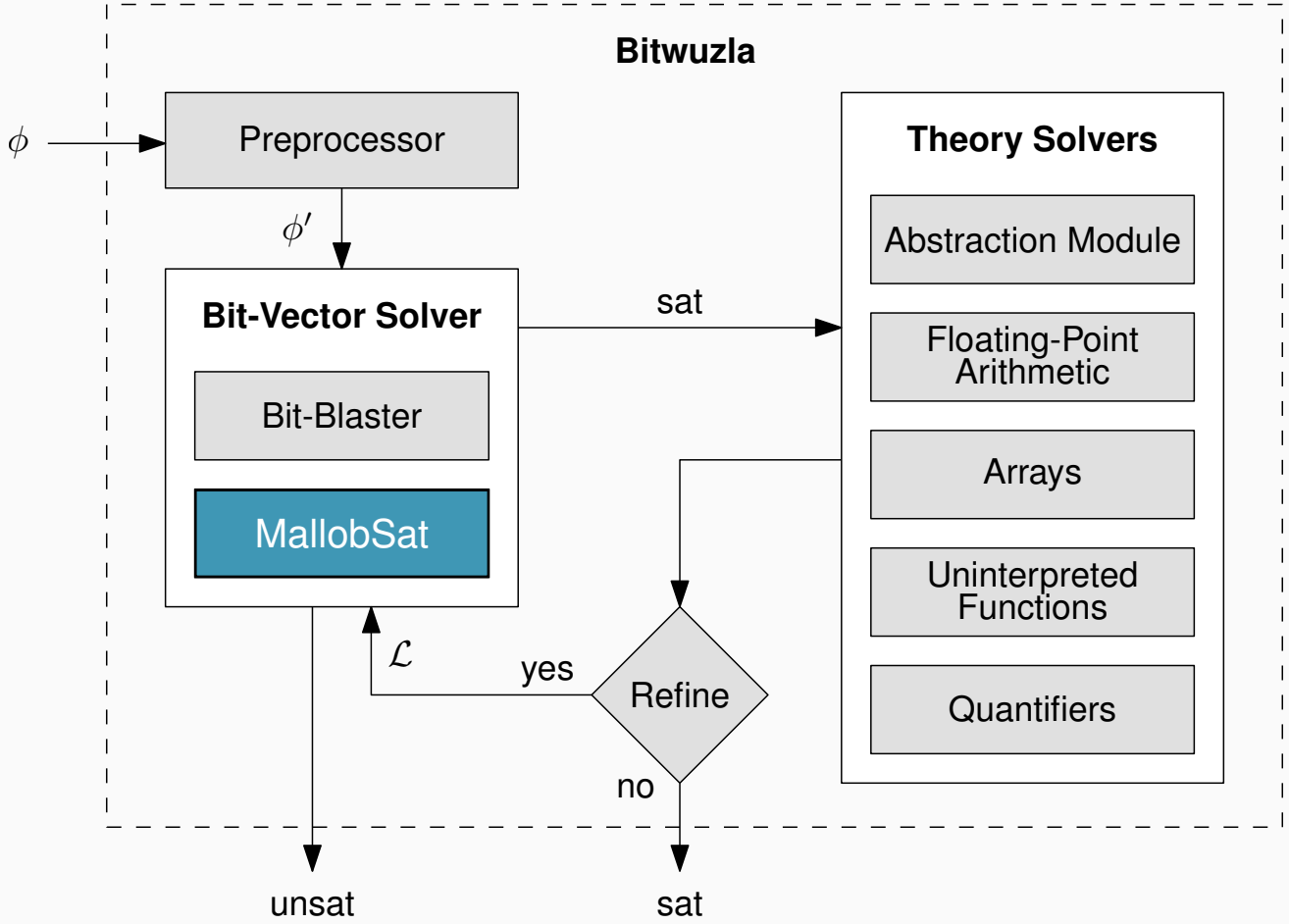


https://bitwuzla.github.io

Bitwuzla Architecture

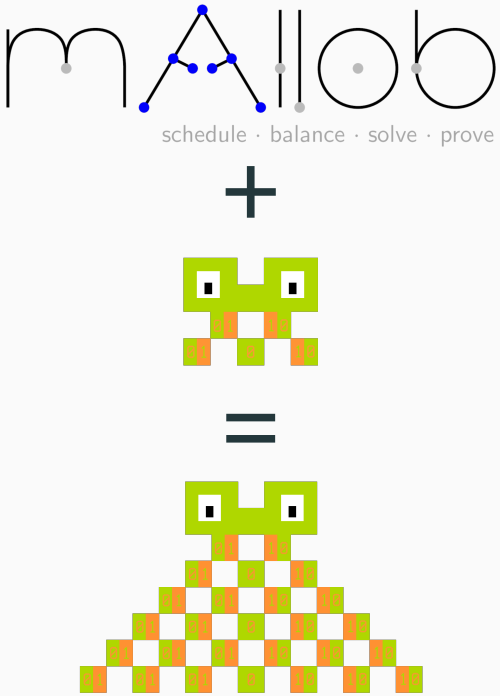


Bitwuzla Architecture



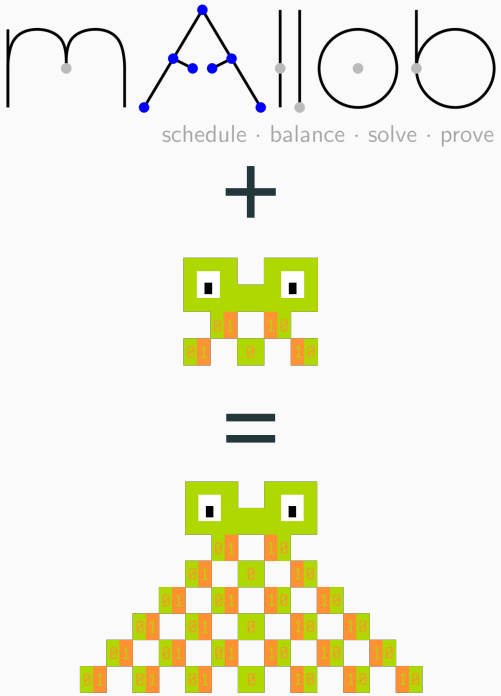
Bitwuzla + Mallob = Bitwuzllob

- Integration of Bitwuzla as Mallob application
  - Mallob app as a wrapper for Bitwuzla
  - Mallob connects MallobSat to Bitwuzla
    - Requires user-provided SAT solver factory in Bitwuzla
    - Direct communication between Bitwuzla and Mallob
  - Mallob schedules MallobSat instances created by Bitwuzla



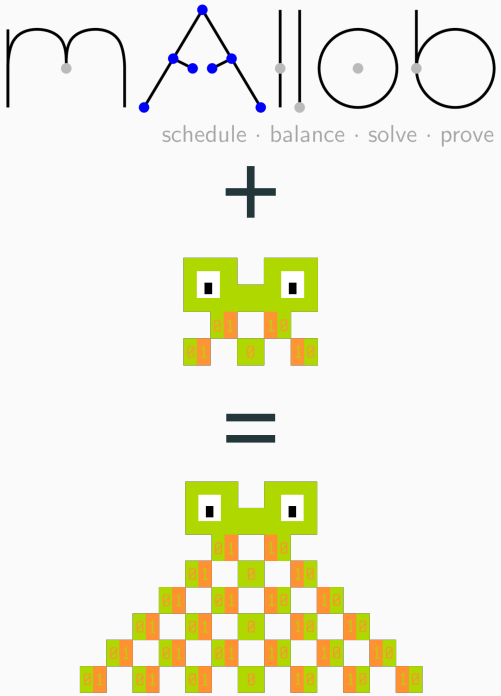
Bitwuzla + Mallob = Bitwuzllob

- Integration of Bitwuzla as Mallob application
  - Mallob app as a wrapper for Bitwuzla
  - Mallob connects MallobSat to Bitwuzla
    - Requires user-provided SAT solver factory in Bitwuzla
    - Direct communication between Bitwuzla and Mallob
  - Mallob schedules MallobSat instances created by Bitwuzla
- Modes
  - One-shot: solve single SMT2 problem
  - On demand: schedule and solve multiple SMT2 problems

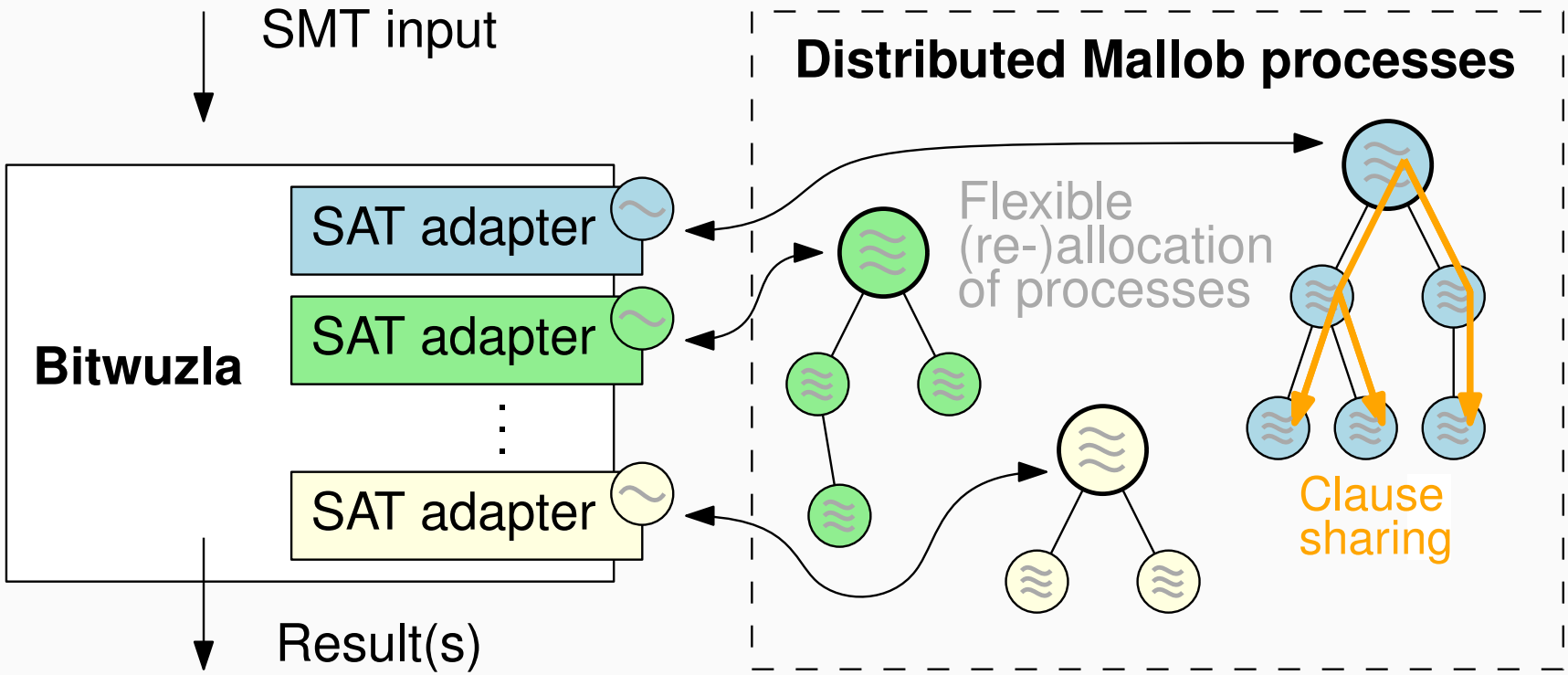


Bitwuzla + Mallob = Bitwuzllob

- Integration of Bitwuzla as Mallob application
  - Mallob app as a wrapper for Bitwuzla
  - Mallob connects MallobSat to Bitwuzla
    - Requires user-provided SAT solver factory in Bitwuzla
    - Direct communication between Bitwuzla and Mallob
  - Mallob schedules MallobSat instances created by Bitwuzla
- Modes
  - One-shot: solve single SMT2 problem
  - On demand: schedule and solve multiple SMT2 problems
- Supports majority of Bitwuzla features
  - All logics, incremental solving, models, unsat cores
  - No direct API access, but communication via input/output pipes supported



Bitwuzla + Mallob = Bitwuzllob



~ ... SAT Solver Thread

Challenge: Latency Reduction

Not every SAT query is hard

- Trivial SAT queries should not incur overhead
- Mallob optimized for low latency
  - But: latencies of 5-10 ms per call common

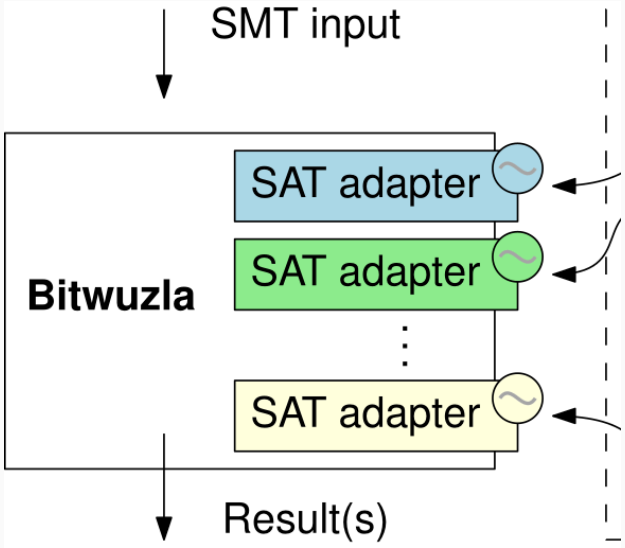
Challenge: Latency Reduction

Not every SAT query is hard

- Trivial SAT queries should not incur overhead
- Mallob optimized for low latency
  - But: latencies of 5-10 ms per call common

Latency-Hiding Head Thread (LHT)

- Runs on host process
- Single sequential SAT solver thread (local)
- Tries to solve same problem as distributed solver
- Delayed distributed task scheduling
  - Task scheduled if LHT does not solve within ~10ms



~ ... LHT Solver Thread

Challenge: Managing Concurrent Distributed Tasks

- More than one SAT solver instance created in Mallob
  - Bitwuzla creates separate SAT instances for subproblems (quantifiers)
    - Only one SAT instance active at any time

Challenge: Managing Concurrent Distributed Tasks

- More than one SAT solver instance created in Mallob
  - Bitwuzla creates separate SAT instances for subproblems (quantifiers)
    - Only one SAT instance active at any time
- Mallob distributed task exclusively blocks one MPI process until termination
  - With p available MPI processes and $T_1, ..., T_p$ deployed tasks
    - T_{p+1} would not be distributed, runs local SAT instance only

Challenge: Managing Concurrent Distributed Tasks

- More than one SAT solver instance created in Mallob
  - Bitwuzla creates separate SAT instances for subproblems (quantifiers)
    - Only one SAT instance active at any time
- Mallob distributed task exclusively blocks one MPI process until termination
  - With p available MPI processes and $T_1, ..., T_p$ deployed tasks
    - T_{p+1} would not be distributed, runs local SAT instance only
- Task eviction policy to avoid stalled solving tasks
  - Limit number of deployed tasks by parameter $s \leq p$
  - If new task to be deployed exceeds s, an inactive distributed task is evicted
    - Picks task with lowest total runtime so far
  - For our use case: $s = \frac{p}{2}$, active SAT task can use at least half of all processes

Evaluation Setup

Responsible Use of Computation Resources

- Distributed experiments are costly and resource intensive
  - Time limit: 300 seconds wallclock
  - Benchmarks: small, but diverse benchmark set on all supported logics
    - incremental and non-incremental SMT-LIB benchmarks

Solver Comparison

- *Bitwuzla* (sequential)
- *Bitwuzla+Gimsatul* (parallel)
- *Bitwuzllob* (parallel, distributed)
- *STP-Parti-Bitwuzla* (parallel) [Zhao et. al.]
- *Yices2* (parallel) [Dutertre et. al.]

Cluster Info

- SuperMUC-NG
- 2x24c Intel Xeon Platinum 8174 @ 2.7GHz
- 96GB memory

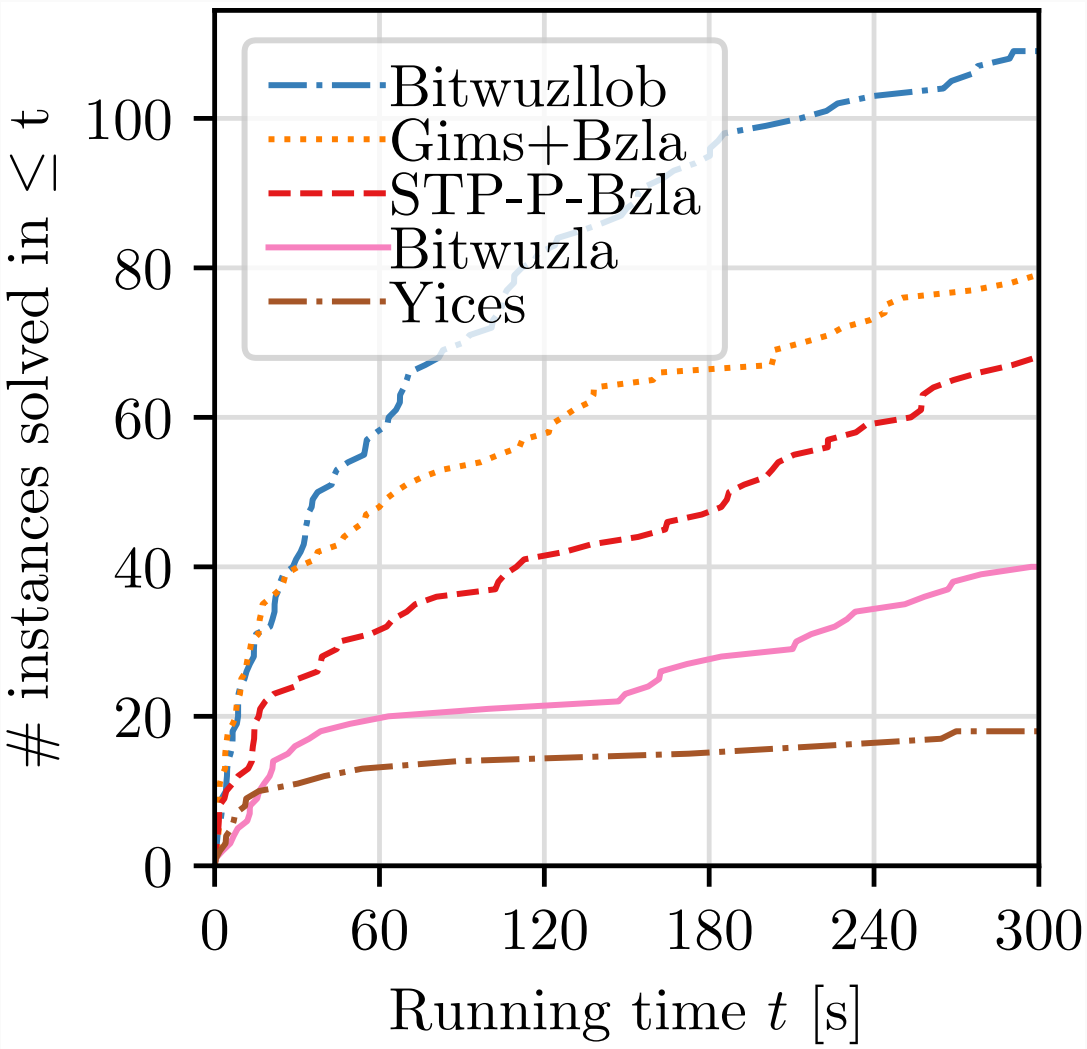
Benchmark Selection

- Goal: Small, but diverse set of easy to very hard benchmarks
  - Easy: Measure overhead of distributed approach
  - Hard: Measure scaling behavior

Benchmark Set

- SMT-LIB consists of 156k+ non-incremental and 25k+ incremental benchmarks
- Sample of 1098 non-incremental, 288 incremental (~93k queries) benchmarks
  - Based on runtime of sequential Bitwuzla with 1200 seconds/8GB limit
  - Each logic grouped into six sets $S_1 - S_5, S_u$
    - Runtime distr.: (5, 10), (10, 100), (100, 300), (300, 600), (600, 1200), (“unsolved”)
    - Benchmark distr.: 10, 10, 25, 25, 35, 50

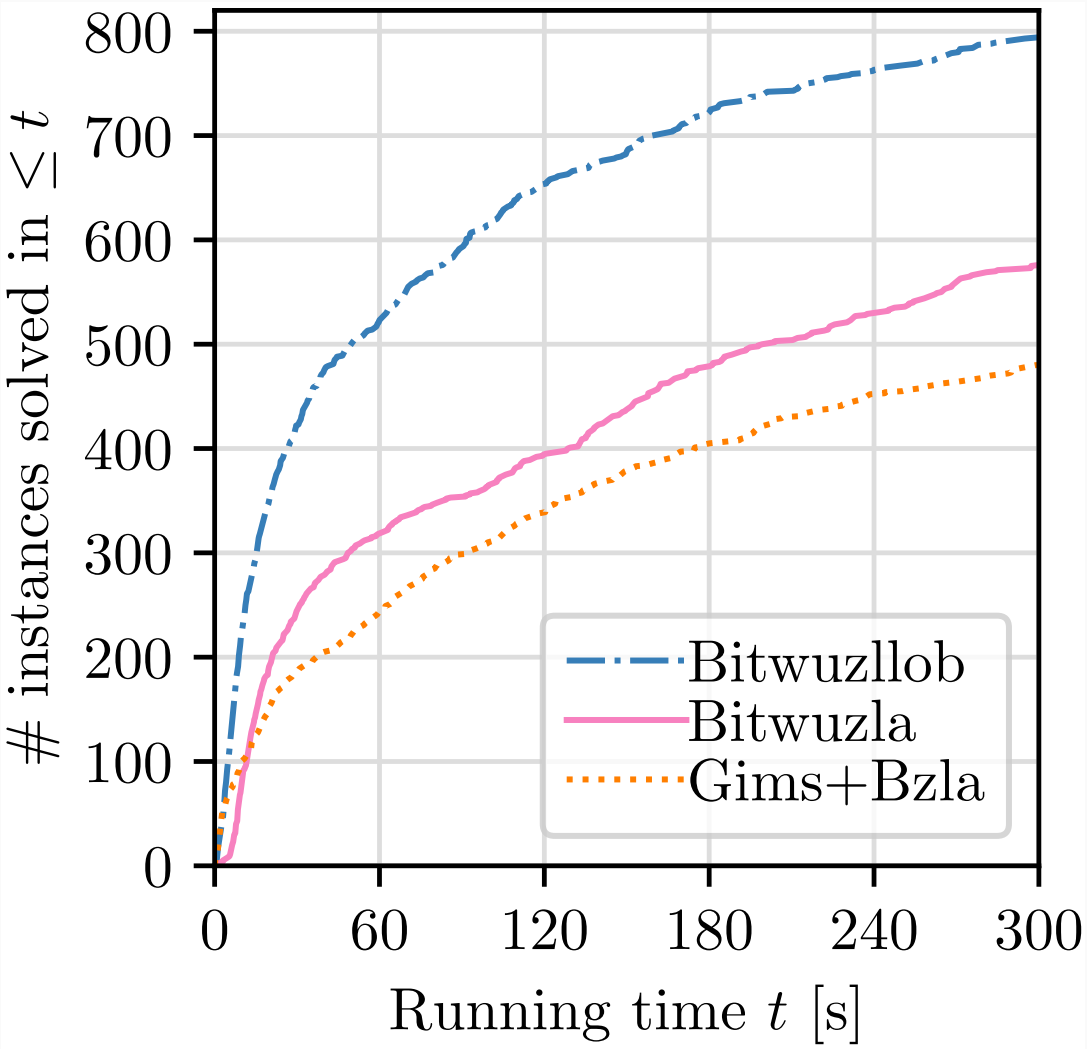
Comparison on non-incremental QF_BV



Setup

- 155 QF_BV benchmarks
- Single node
- 48 cores
- 300 seconds wallclock
- 96GB memory

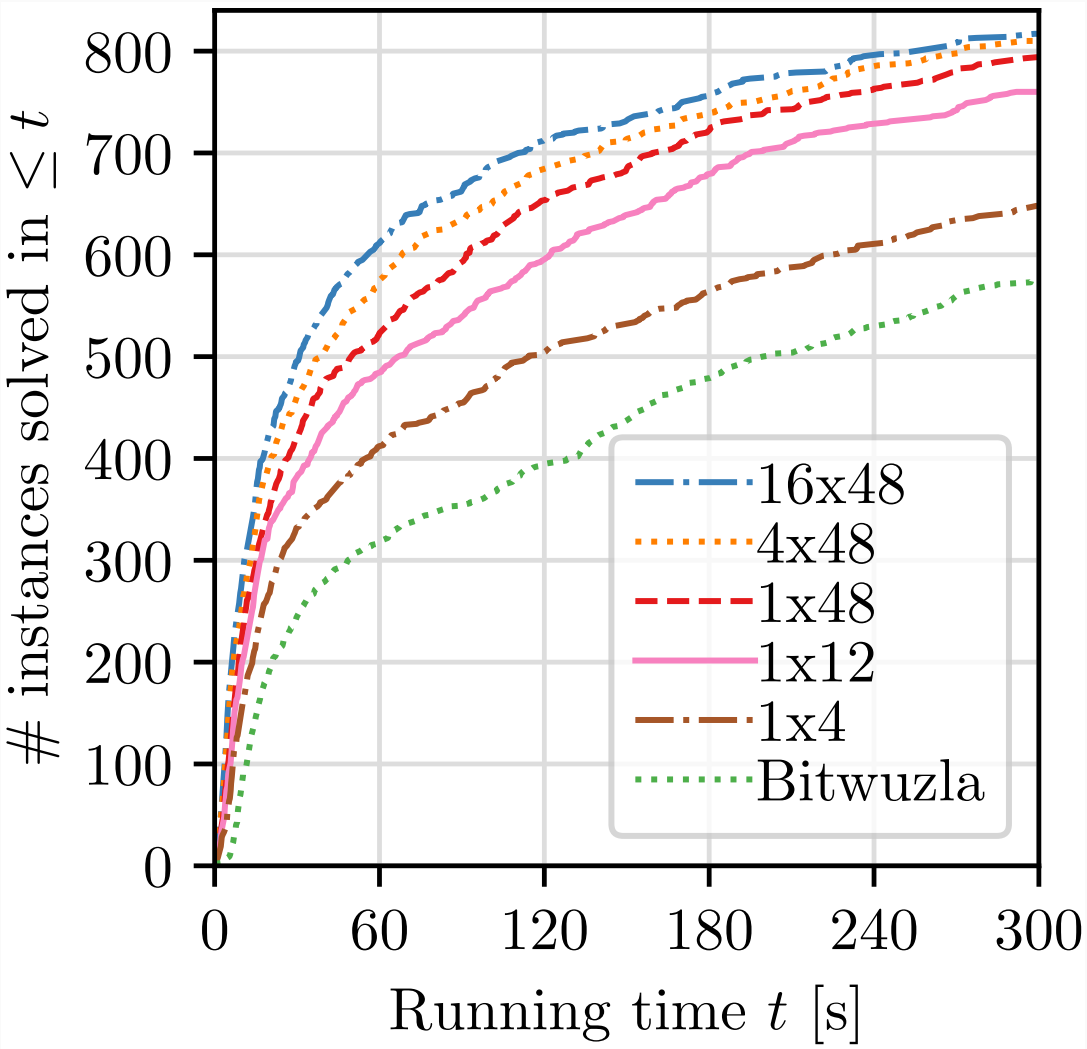
Comparison of Bitwuzla Configurations on All Benchmarks



Setup

- All benchmarks
- Single node
- 48 cores
- 300 seconds wallclock
- 96GB memory

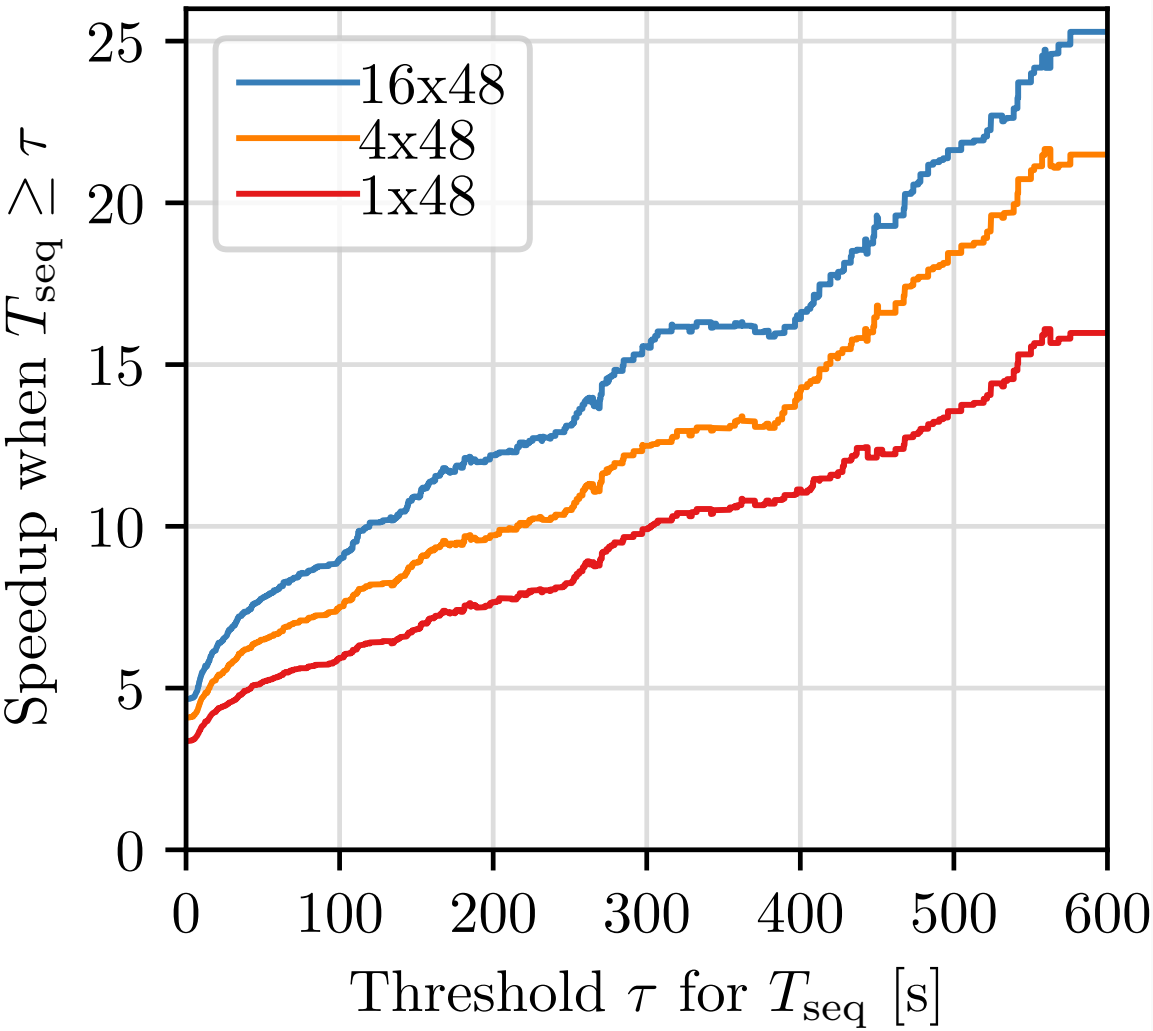
Scaling Performance of Bitwuzllob



Setup

- All benchmarks
- Bitwuzla (seq.) as baseline
- NxP: N nodes, P Threads
- 300 seconds wallclock
- 96GB memory

Weak Scaling Performance of Bitwuzllob vs. Bitwuzla (sequential)



Plot (x, y)

- Commonly solved
- Speedup w.r.t. increasing difficulty
- Bitwuzllob achieves mean speedup of y, where $T_{\text{seq}} \geq x$

Setup

- All benchmarks
- NxP: N nodes, P Threads
- 96GB memory

Conclusion

Bitwuzllob

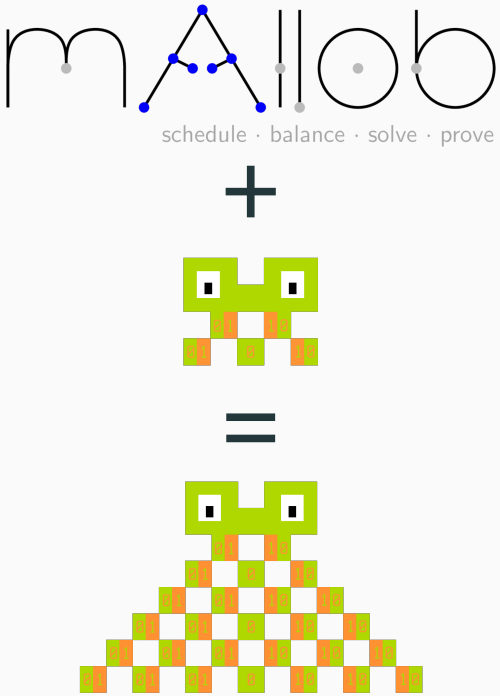
- Bitwuzla as integrated module in Mallob
  - Majority of solving time spent in SAT queries
    - Ideal for SAT-level parallelization
- Experiments up to 768 cores in distributed environment
- Significantly outperforms existing parallel approaches

Future Work

- Bitwuzllob solid foundation for SMT-level parallelization

Resources

- Mallob: https://github.com/domschrei/mallob
- Bitwuzla: https://bitwuzla.github.io
- Artifact: https://zenodo.org/records/17478481



Appendix

Interface: One-Shot

```
# Run Mallob+Bitwuzla with 32 threads on single SMT problem:
$ ./mallob -t=32 -mono-app=SMT -mono=smt-input.smt2 -q
c
c Mallob -- a parallel and distributed platform for job scheduling, load balancing, and
SAT solving
c Copyright (C) 2019-2025 Dominik Schreiber, Karlsruhe Institute of Technology, Germany
c
sat
(
  (define-fun s () (_ BitVec 512) #b00000000 ... 00000110)
  (define-fun t () (_ BitVec 512) #b00000000 ... 00000000)
)
```

Interface: On Demand

```
# Launch Mallob+Bitwuzla in 4 MPI processes with 48 threads each
$ mpirun -np 4 ./mallob -q -t=48 &

[1] 129857

# Create named pipe for output
$ mkfifo smt-output.pipe

# Show JSON submission file
$ cat job.json

{
  "user": "myname",
  "name": "job1",
  "application": "SMT",
  "files": ["smt-input.smt2"],
  "priority": 1.0,
  "configuration": {"smt-out-file": "smt-output.pipe"}
}

# Submit SMT job to Mallob
$ cp job.json .api/jobs.0/in/
```

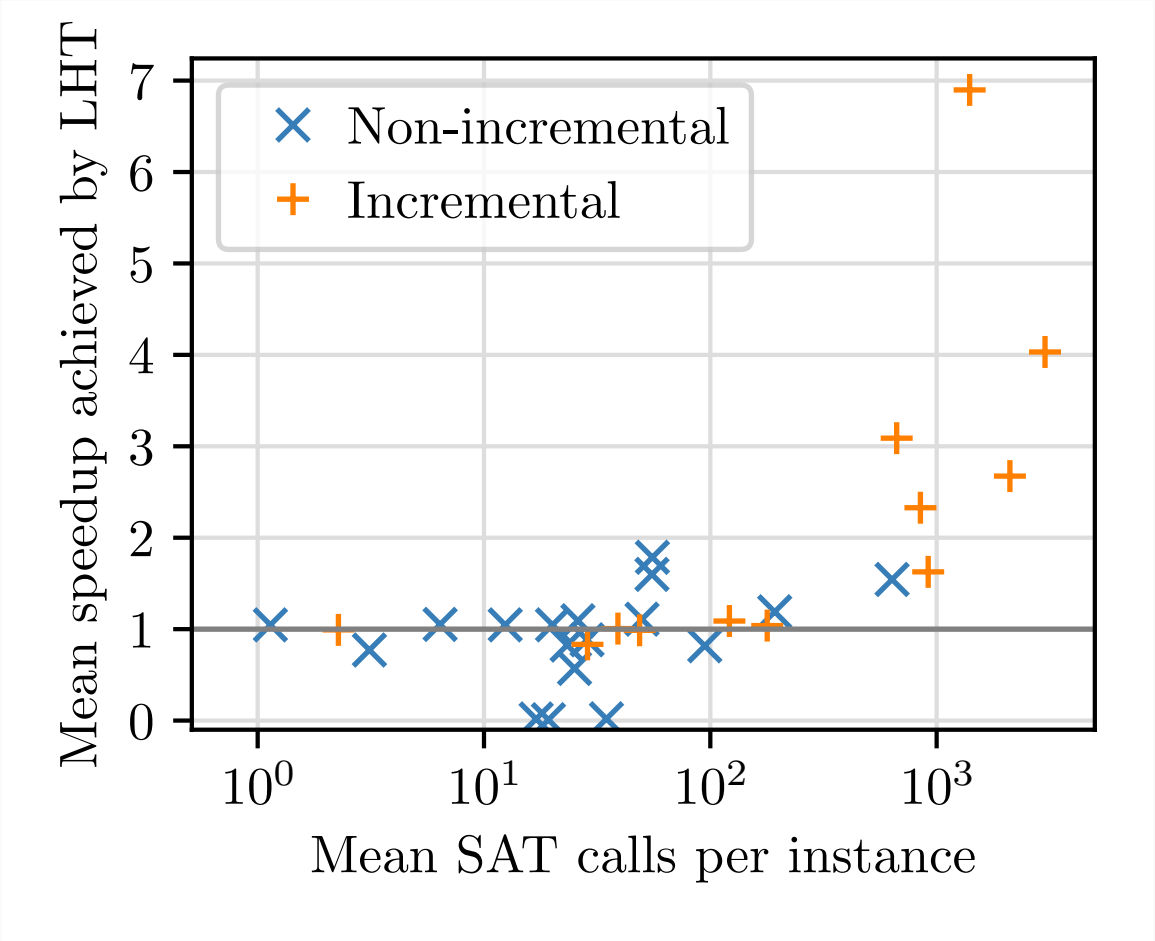
Interface: On Demand

```
# Fetch SMT solver output
$ cat smt-output.pipe

sat
(
  (define-fun s () (_ BitVec 512) #b00000000 ... 00000110)
  (define-fun t () (_ BitVec 512) #b00000000 ... 00000000)
)

# Clean up
$ kill -2 1079026 ; rm smt-output.pipe
```

Latency-Hiding Thread Impact



(x, y) **per Logic**

- x: Geom. mean #SAT calls
- y: Geom. mean speedup