DPS: A Framework for Deterministic Parallel SAT Solvers

Hidetomo Nabeshima[1], Tsubasa Fukiage[1], Yuto Obitsu[1], Xaio-Nan Lu[1], Katsumi Inoue[2]

[1] University of Yamanashi, Japan

{nabesima, xnlu}@yamanashi.ac.jp

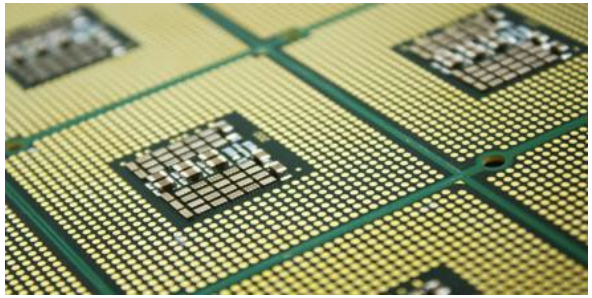
[2] National Institute of Informatics, Japan

inoue@nii.ac.jp

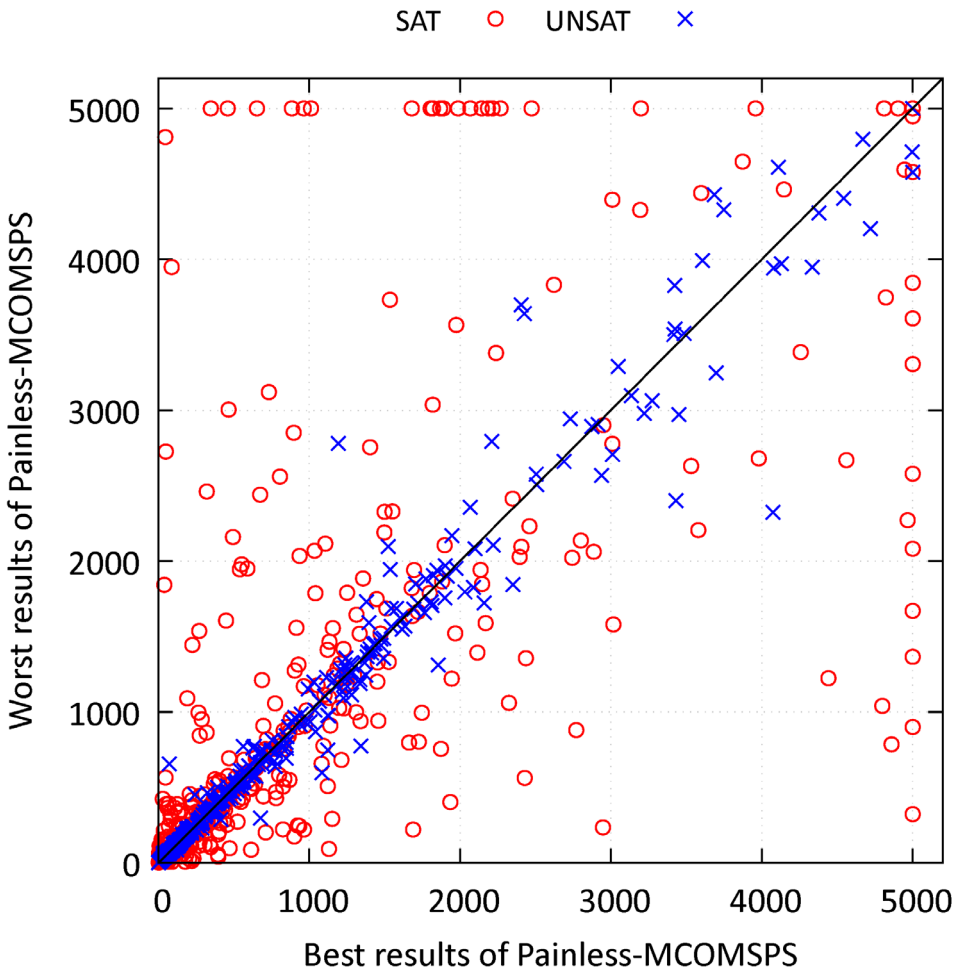
Introduction

- **SAT solvers are powerful tools for problem solving**
  - Hardware and software verification, Planning, Scheduling, etc.
- **Performance improvement is important for applications**

- **Sequential SAT solving**
  - Studying high-performance sequential SAT solvers is essential
  - Basis of the SAT-based problem solving
- **Parallel SAT solving**
  - With the spread of multi-core environments, important to utilize their computing resources



Non-deterministic Behavior in Parallel SAT Solvers



Results of PaInleSS-MapleCOMSPS, winner of the parallel track of the 2021 SAT competition, for 1200 instances from SAT Race 2019, SAT Competition 2020, and 2021

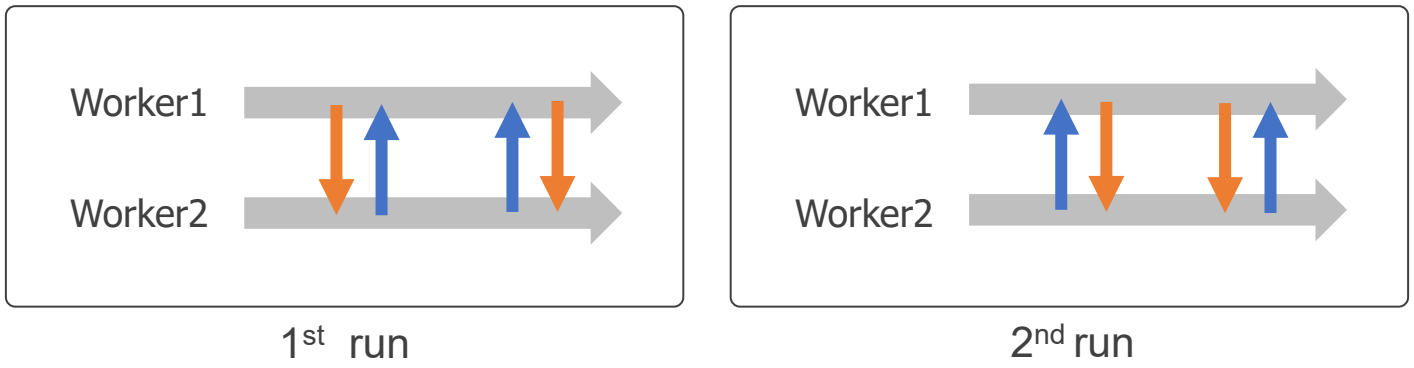
Issues of Non-deterministic Behavior

- When model checking, *different bugs may be found in different runs*
- In scheduling problem, even if a good solution is found, *it may not be reproduced next time*
- If a bug occurs in software with an embedded non-deterministic SAT solver, *the bug may not be reproduced*
- In the development of parallel SAT solvers, *instability of execution results leads to difficulty in tuning performance*

Reproducibility **is an important property that directly affects the *usability* of SAT solvers**

Cause of Non-deterministic Behavior

Asynchronous clause exchange between workers



- Timing of sending clauses is determined by the sender
  - *Affected by system workload, cache misses, and/or communication delays*
- Since each worker's search process is affected by the imported clauses, *the behavior may change from run to run*

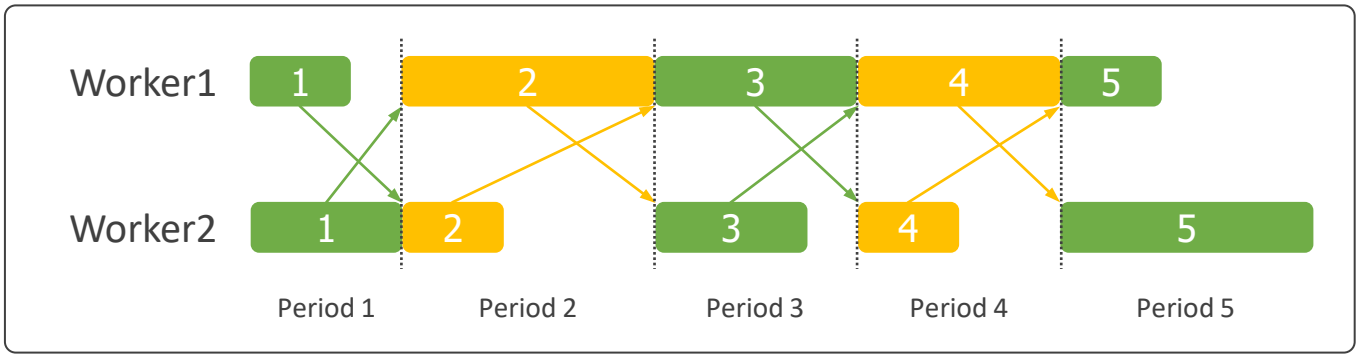
Deterministic Parallel SAT Solvers

- **ManySAT 2.0** [Hamadi+, 2011]
  - First deterministic parallel SAT solver
  - All workers synchronize periodically, then exchange clauses in a fixed order
- **MergeSat** [Manthey, 2021]
  - Recently supports deterministic parallel solving
  - Similar mechanism as ManySAT
- **ManyGlucose** [Nabeshima+, 2020]
  - Deterministic parallel SAT solver with *delayed clause exchange* to suppress synchronous waiting
  - 3rd place in the parallel track of the SAT Competition 2020

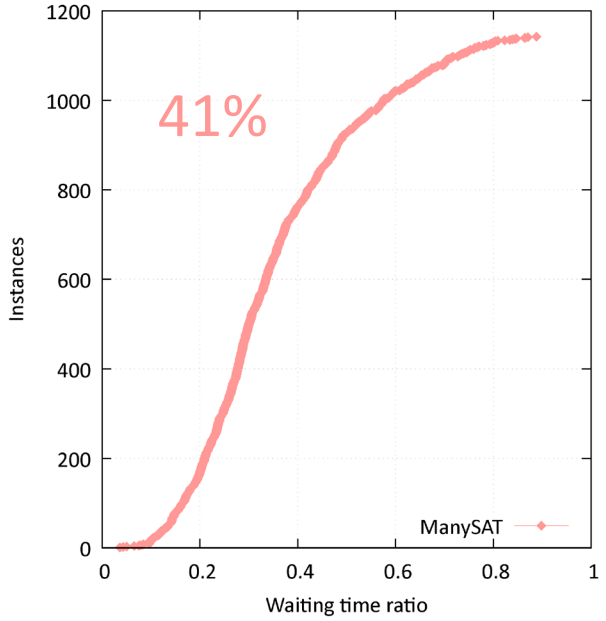
- Y. Hamadi, S. Jabbour, C. Piette, L. Sais: Deterministic Parallel DPLL, JSAT 7(4): 127-132 (2011)
- N. Manthey: The MergeSat Solver, SAT-2021:387-398 (2021)
- H. Nabeshima, K. Inoue: Reproducible Efficient Parallel SAT Solving, SAT-2020:123-138 (2020)

ManySAT 2.0

Y. Hamadi, S. Jabbour, C. Piette, L. Sais: Deterministic Parallel DPLL, JSAT 7(4): 127-132 (2011)



- **Pros**
  - Easily implemented with OpenMP
- **Cons**
  - Waiting time increases as workers increases



Results of 64 threads

ManyGlucose

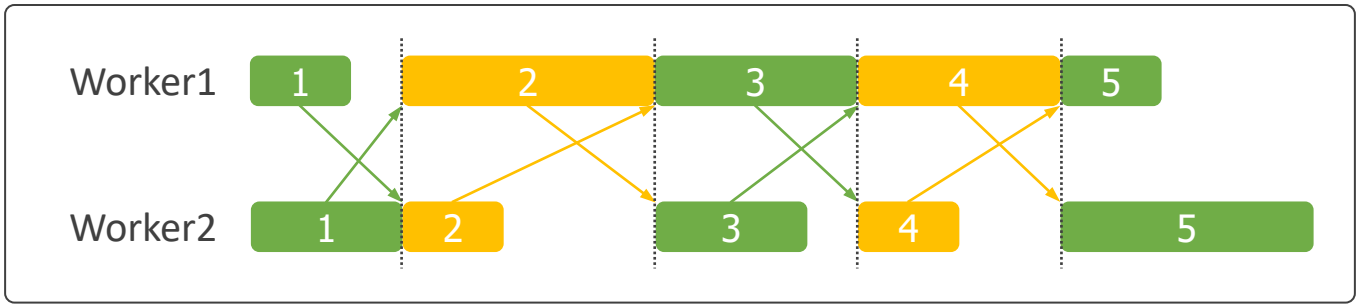
H. Nabeshima, K. Inoue: Reproducible Efficient Parallel SAT Solving, SAT-2020, pp.123-138

Delayed Clause Exchange

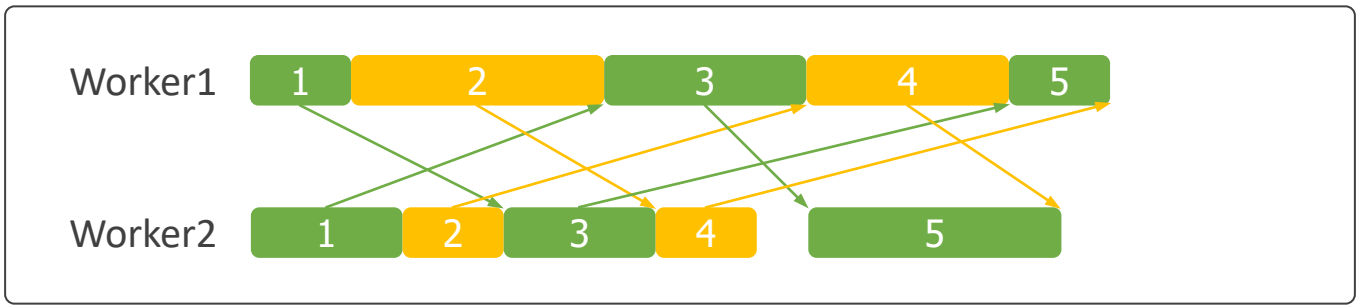
- Exchanges clauses acquired in period x at the end of period $x + m$ (*margin*)

$m = 0$

same as ManySAT



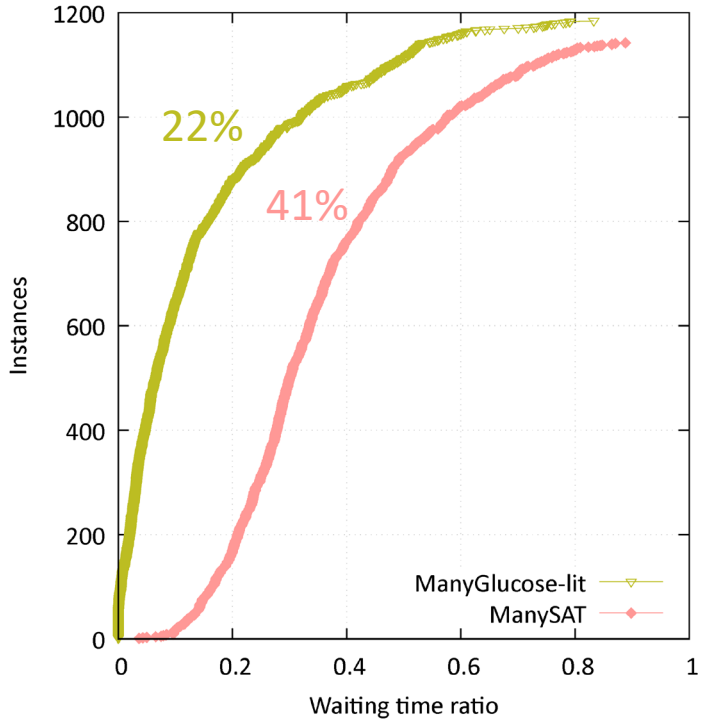
$m = 1$



No idle time is required If period difference between workers $\leq m$

However, the received clauses are always m *periods old*

Waiting Time Reduction by DCE



- **Pros**
  - DCE can reduce waiting time
- **Cons**
  - *Requires expertise in concurrent programming*
  - *More effort than building non-deterministic parallel SAT solvers*

Purpose

- A framework for easily constructing *efficient deterministic parallel SAT solvers*

Parallel Solver	Framework
Non-deterministic	PaInleSS [Frioux+, 2017]
Deterministic	This work

- PaInleSS is a framework to parallelize existing sequential SAT solvers with little effort, but does not support reproducible behavior

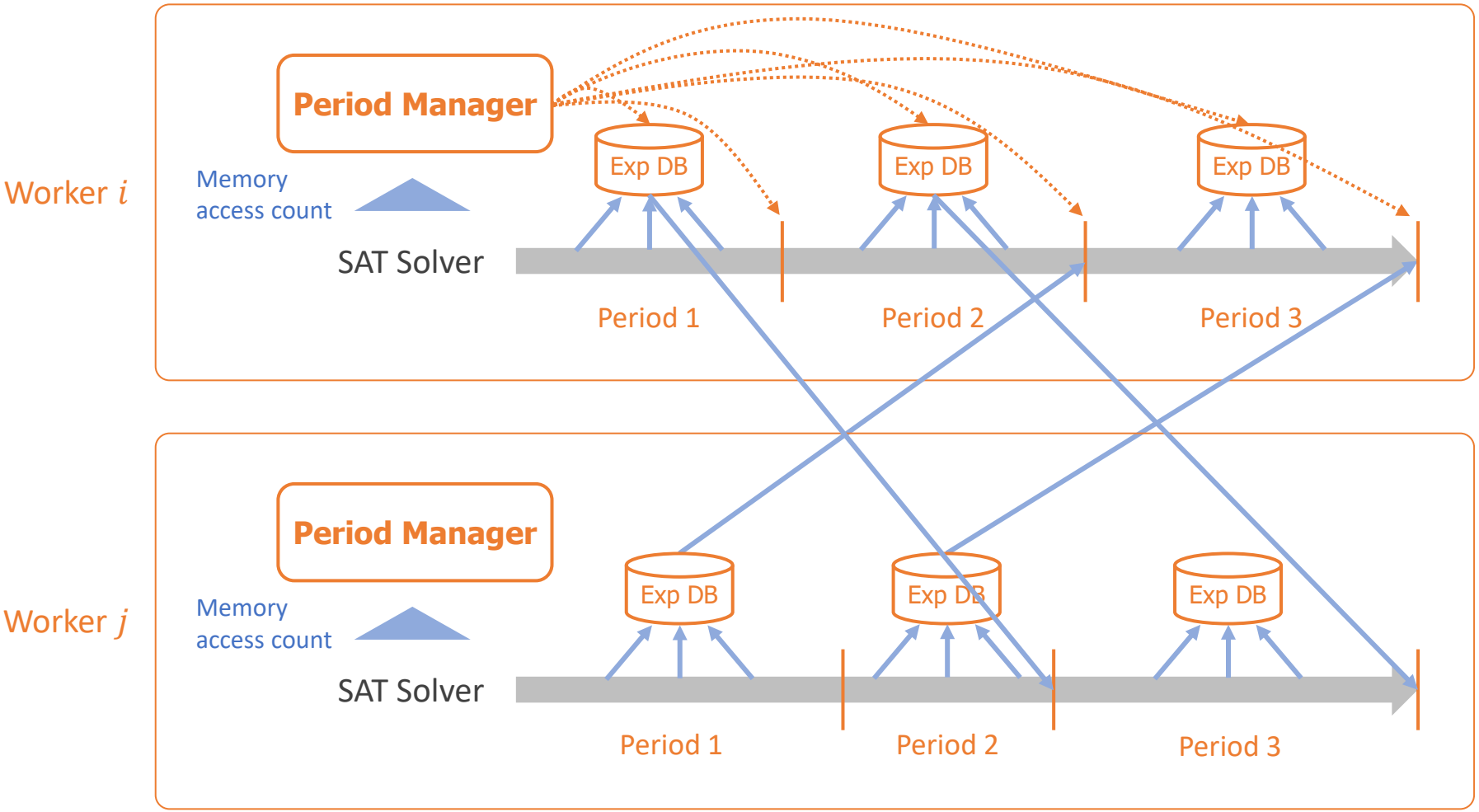
PaInleSS = PArallel INstantiabLE SAT Solver

L. Le Frioux, S. Baarir, J. Sopena, and F. Kordon: Painless: a Framework for Parallel SAT Solving, SAT-2017, pp.233-250

- **Parallelize existing sequential SAT solvers with little effort**
  - Provides adapter classes to incorporate popular SAT solvers
- **Provides representative strategies**
  - Parallelization
    - Portfolio, divide and conquer, and hybrid strategies of them
  - Clause exchange
    - Length, LBD and HordeSAT [Balyo+, 2015] based strategies
      - HordeSAT strategy shares 1500 literals every second
- **PaInleSS with MapleCOMSPS**
  - 1st in SAT Competition 2021, 2020 and 2018
  - Portfolio parallel SAT solver with HordeSAT strategy

PaInleSS supports building fast parallel SAT solvers without expertise in concurrent programming

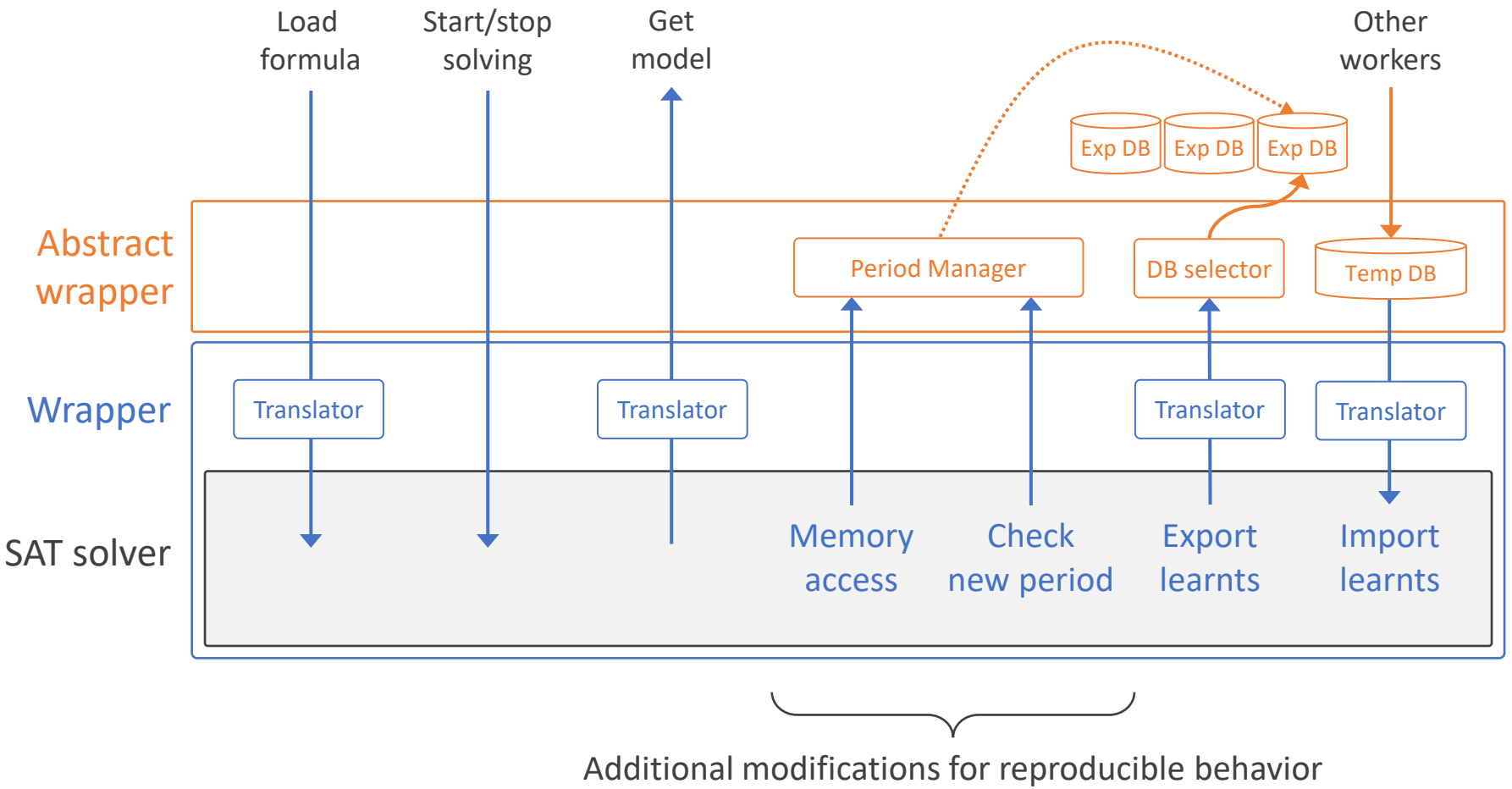
DPS Overview



Orange modules are provided by our framework DPS

Blue functions denote required modifications of solver

Modifications to embed SAT solver



Orange modules are provided by our framework DPS

Blue functions denote required modifications of solver

Counting Memory Accesses

- Keeping the run time of each worker's period as close as possible
  - *Define the length of period as elapsed time?*
    - Measurement error
    - Amount of processing within a given time may change
- Period length is defined by ***memory access count***
  - Corresponding to what Knuth calls “mems”
  - Reproducible measure

Counting in MiniSAT family

- MiniSAT family of solvers has Clause class
- Sufficient to count literal accesses within the class

```
namespace DPS {
  extern thread_local uint64_t num_mem_accesses;
}

class Clause {
                ⋮
  Lit&     operator [](int i)       { DPS::num_mem_accesses++; return data[i].lit; }
  Lit      operator [](int i) const { DPS::num_mem_accesses++; return data[i].lit; }
  operator const Lit* (void)  const { DPS::num_mem_accesses++; return (Lit*)data; }
  float&   activity   ()            { DPS::num_mem_accesses++; return data[header.size].act; }
  uint32_t abstraction() const      { DPS::num_mem_accesses++; return data[header.size].abs; }
                ⋮
};
```

- Thread-specific global variable since C++11
- Each worker (thread) has this counter accessible only by itself

Counting in Kissat

A. Biere, K. Fazekas, M. Fleury, and M. Heisinger. CaDiCaL, Kissat, Paracooba, Plingeling and Treengeling entering the SAT competition 2020, SAT Competition 2020 Solver Description

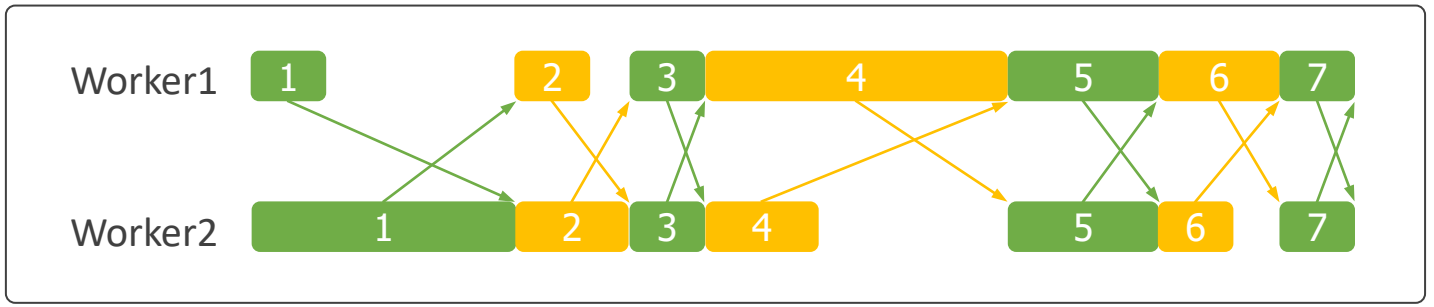
- A clause is defined as a C struct
- Not easy to count literal accesses since *members of struct are public*

- Kissat already has a mechanism to measure memory access count called "*ticks*"
  - Refinement of Knuth's "mems" and counts cache line accesses
  - Used to switch between various strategies and was also introduced to ensure reproducible behavior

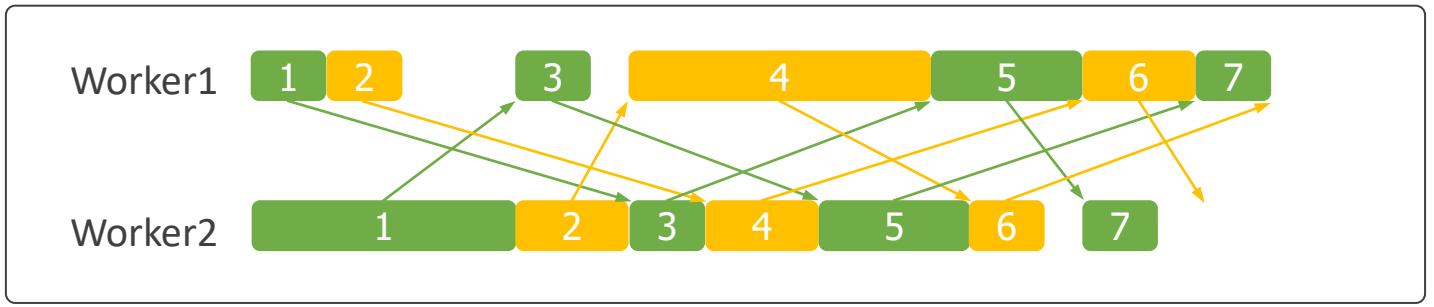
- *Relatively easy to incorporate Kissat into DPS by using ticks*

Support of Non-deterministic Run

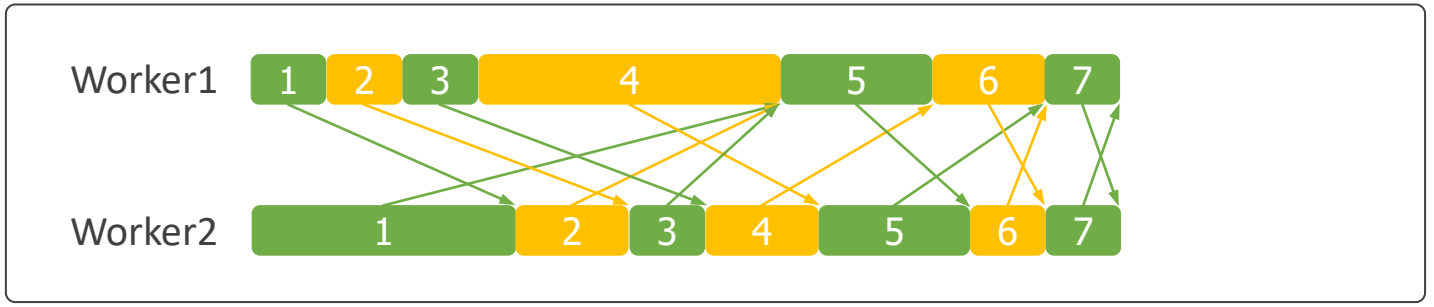
DPS
$(m = 0)$



DPS
$(m = 1)$



NPS
$(m = 0)$



No modifications are required to base solver for NPS

Implementation

- DPS is written in C++ and consists of about 3000 lines
  - https://github.com/nabesima/DPS-pos2022/
- MiniSAT, Glucose, MapleCOMSPS
  - Wrapper class and modification to base solver took about 300 lines
- Kissat
  - About 800 lines including interface between C++ and C

Solver	Diversity strategy	Sharing strategy
DPS-MiniSAT	Random decision until 1st conflict	Length $\leq$ 10
DPS-Glucose		Glue, or LBD $\leq$ 7 and length $\leq$ 24
DPS-MCOMSPS	Random decision until 1st conflict Four decision heuristics used in P-MCOMSPS	150 literals per period
DPS-Kissat	Random decision until 1st conflict Half of workers disable elimination technique	

Diversity is more important for the deterministic parallel SAT solvers

Experimental Evaluation

Base SAT solvers	Existing parallel solvers		Proposed parallel solvers	
	Det	Non-Det	Det	Non-Det
MiniSAT	ManySAT	-	DPS-MiniSAT	NPS-MiniSAT
Glucose	ManyGlucose-lit ManyGlucose-blk	Glucose-syrup	DPS-Glucose	NPS-Glucose
MapleCOMSPS	-	Painless-MCOMSPS 1st parallel track 2021	DPS-MCOMSPS	NPS-MCOMSPS
Kissat 1st main track 2020	-	-	DPS-Kissat DPS-Kissat-no-exchange	NPS-Kissat NPS-Kissat-no-exchange

- **Setting :** Margin of DCE is 20, 64 threads, 5000 sec / instance
- **Instances :** 1200 instances from SAT Race 2019, SAT Competition 2020 and 2021
- **Environment :** Cray XC40 (supercomputer system A in Kyoto University)
  Intel Xeon Phi KNL (1.4GHz, 68 cores), 96GB memory
- All experimental results (including additional results) are available at
  https://nabesima.github.io/DPS-pos2022/

Comparison of PaInleSS and NPS/DPS

Solver	# of solved instances				PAR-2
	2019	2020	2021	Total	
PaInleSS-MCOMSPS	156 + **107**	118 + **124**	134 + 166	805 (408 + 397)	4604576
	152 + 105	115 + 123	131 + **168**	794 (398 + 396)	4697998
	155 + **107**	107 + 124	134 + 167	794 (396 + 398)	4707163
NPS-MCOMSPS	160 + 105	135 + 123	**139** + **168**	**830** (434 + 396)	4386010
	159 + 104	**140** + 121	137 + **168**	829 (**436** + 393)	**4352294**
	163 + 105	126 + 122	138 + 167	821 (427 + 394)	4451212
DPS-MCOMSPS	156 + 101	129 + 119	137 + 166	808 (422 + 386)	4636427

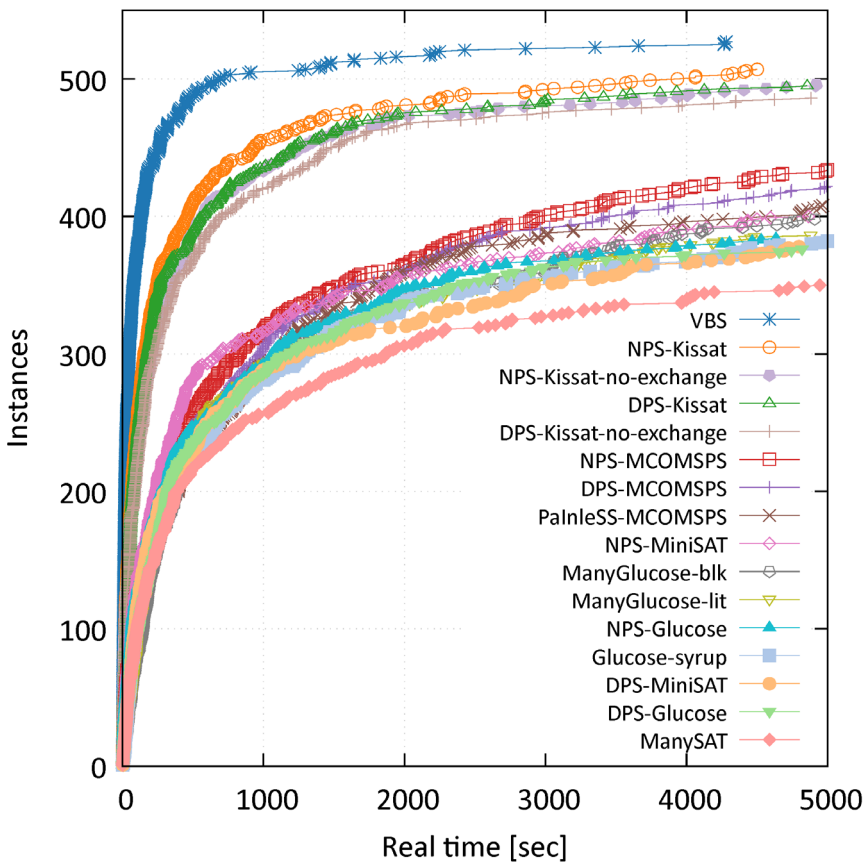
- NPS shows comparable on UNSAT but superior on SAT compared to PaInleSS
  - Random decisions and clause exchange strategies may have influenced
  - Capable of building efficient non-deterministic parallel solvers
- DPS also shows comparable performance to PaInleSS
  - Difference between NPS and DPS represents the cost of ensuring reproducible behavior

NPS-Kissat vs DPS-Kissat

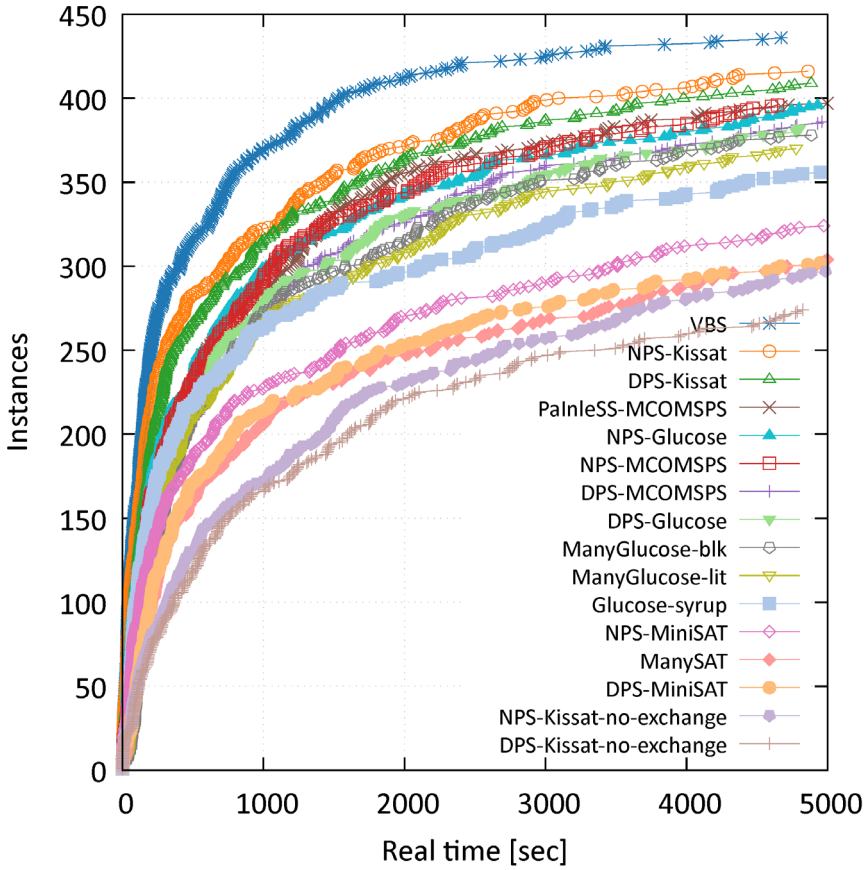
Solver	# of solved instances				PAR-2
	2019	2020	2021	Total	
NPS-Kissat	**175** + **114**	**178** + **134**	154 + **168**	**923** (**507** + 416)	**3245708**
	173 + 114	175 + 134	155 + **168**	919 (503 + 416)	3266045
	170 + 115	175 + 134	155 + **168**	917 (500 + **417**)	3296766
NPS-Kissat no exchange	171 + 69	173 + 100	152 + 128	793 (496 + 297)	4646767
DPS-Kissat	168 + 112	170 + 130	**157** + 167	904 (495 + 409)	3451073
	168 + 112	170 + 130	**157** + 167	904 (495 + 409)	3451074
	168 + 112	170 + 130	**157** + 167	904 (495 + 409)	3451445
DPS-Kissat no-exchange	168 + 64	168 + 95	150 + 115	760 (486 + 274)	4924475

- NPS-Kissat can solve 100 more instances than PaInleSS-MCOMSPS
- Clause exchange is essential for solving UNSAT efficiently, and a bit effective for SAT

CDF Plots



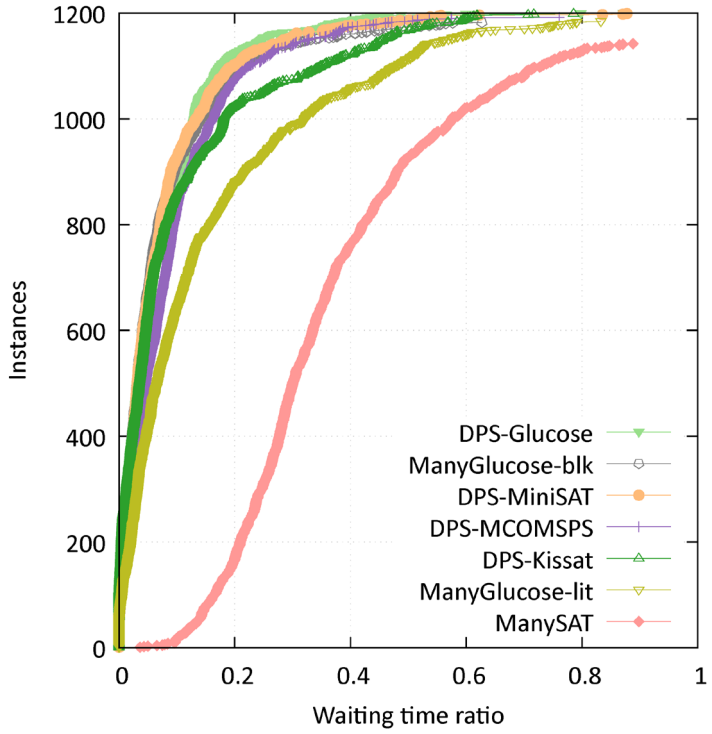
SAT instances



UNSAT instances

- Kissat is a SAT solver that has shown significant performance gains in SAT instances, which is also evident in parallelization

Waiting Time Ratio



Solver	Waiting Time
ManySAT	41.1%
ManyGlucose-lit	21.7%
DPS-MiniSAT	10.4%
DPS-Glucose	9.9%
DPS-MCOMSPS	12.4%
DPS-Kissat	15.0%

- DCE can reduce waiting time
- Complex diversity strategies produce variations in period execution times, making it difficult to reduce latency

Conclusion

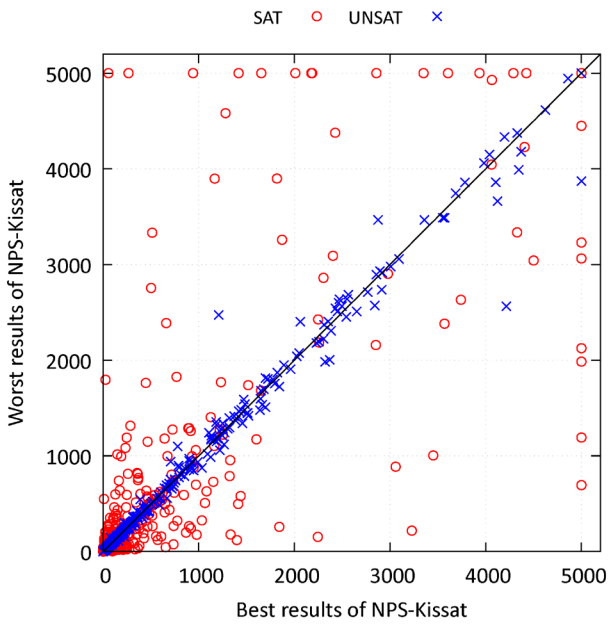
- Reproducible behavior will facilitate the application of parallel solvers in practical fields and promote research in parallel SAT solving
- DPS makes it easy to build efficient deterministic parallel SAT solvers
- NPS can achieve higher performance if performance is important

Future Work

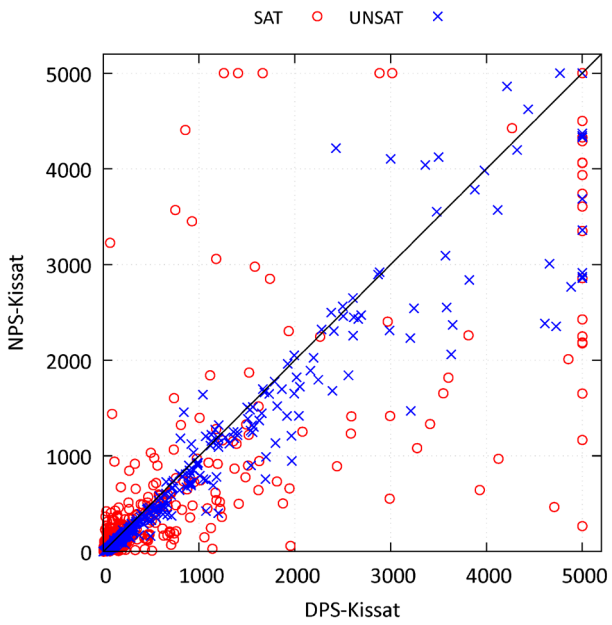
- Extending DPS to non-shared-memory environment
- Efficient clause exchange between heterogeneous solvers with various strategies

Thank you for your attention

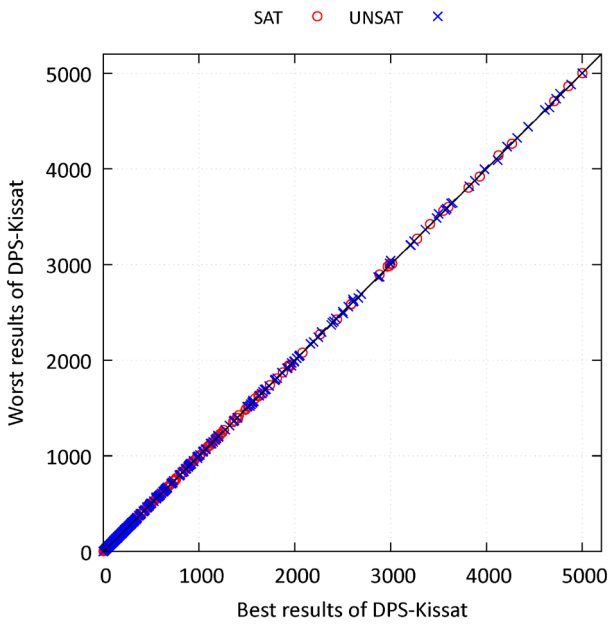
Comparison of DPS and NPS



NPS



NPS vs DPS



DPS

- NPS execution time (especially for SAT instances) varies widely
- DPS has *reproducible behavior* (all points on the diagonal)
- NPS can solve more instances than DPS