Parallel Automated Reasoning

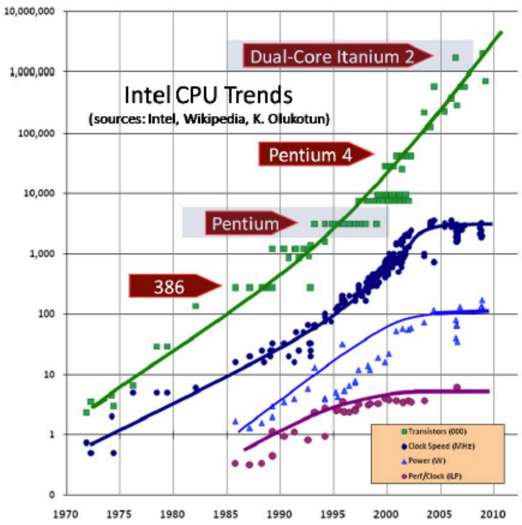
Ruben Martins

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f19/

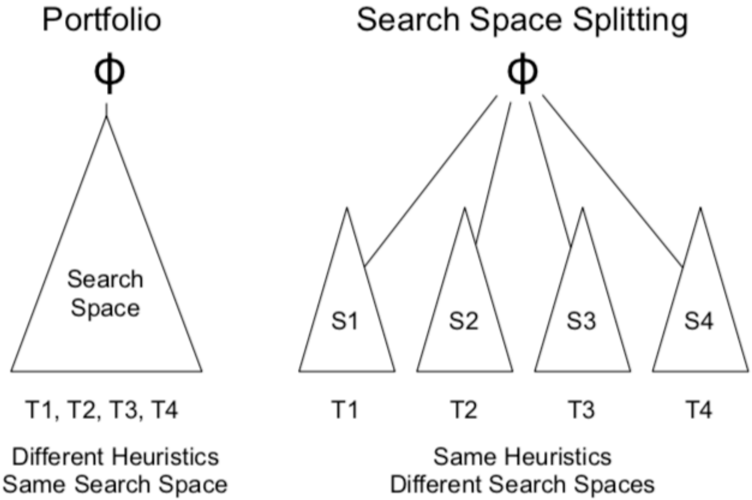
Automated Reasoning and Satisfiability, October 3, 2019

Why Parallelization? Power Wall



How to parallelize SAT?

How to parallelize SAT?



Portfolio Approach

Basic Idea:

- Run several solvers in parallel
- Stop when the first solver finds a solution or proves unsatisfiability

SAT Competition 2011:

- `ppfolio//` wins 11 medals, best solver in competition
- This solver is equivalent to type:
  $\texttt{solver}_1$ `&` $\texttt{solver}_2$ `& ... &` $\texttt{solver}_n$

Portfolio Approach

Basic Idea:

- Run several solvers in parallel
- Stop when the first solver finds a solution or proves unsatisfiability

SAT Competition 2011:

- `ppfolio//` wins 11 medals, best solver in competition
- This solver is equivalent to type:
  $\texttt{solver}_1$ `&` $\texttt{solver}_2$ `& ... &` $\texttt{solver}_n$

Can we do better?

Portfolio Approach

Basic Idea:

- Run several solvers in parallel
- Stop when the first solver finds a solution or proves unsatisfiability

SAT Competition 2011:

- `ppfolio//` wins 11 medals, best solver in competition
- This solver is equivalent to type:
  $\texttt{solver}_1$ `&` $\texttt{solver}_2$ `& ... &` $\texttt{solver}_n$

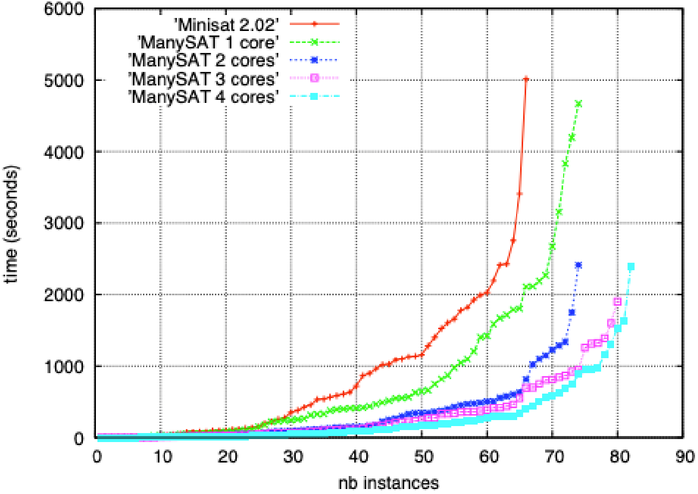
Can we do better?

- Exchange learned clauses
- Increase diversity between solvers

Portfolio Solvers: ManySAT

	Restart	Heuristic	Polarity	Learning
Core 0	Geometric $x_1 = 100$ $x_i = 1.5 \times x_{i-1}$	VSIDS (3% rand.)	if $\#occ(l) > \#occ(\neg l)$ $l = true$ else $l = false$	CDCL (extended)
Core 1	Dynamic (fast) $\alpha = 1200$ $x_1 = 100, x_2 = 100$ $x_i = f(y_{i-1}, y_i), i > 2$ if $y_{i-1} < y_i$ $f(y_{i-1,y_i}) =$ $\frac{\alpha}{y_i} \times \left\| cos(1 - \frac{y_{i-1}}{y_i}) \right\|$ else $f(y_{i-1,y_i}) =$ $\frac{\alpha}{y_i} \times \left\| cos(1 - \frac{y_i}{y_{i-1}}) \right\|$	VSIDS (2% rand.)	Progress saving	CDCL
Core 2	Arithmetic $x_1 = 16000$ $x_i = x_{i-1} + 16000$	VSIDS (2% rand.)	false	CDCL
Core 3	Luby 512	VSIDS (2% rand.)	Progress saving	CDCL (extended)

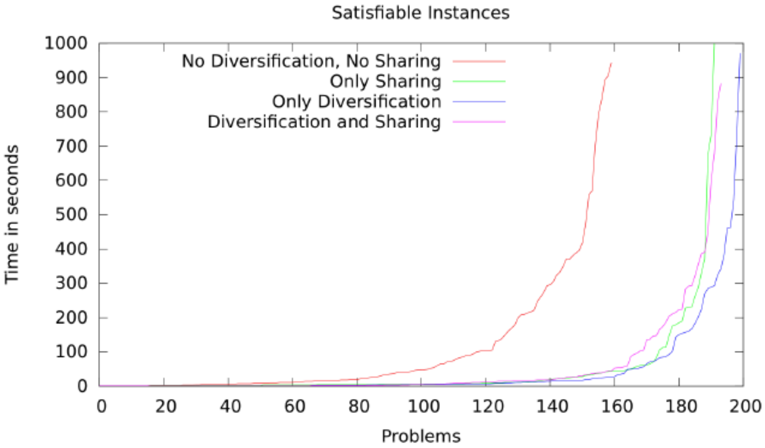
Portfolio Solvers: ManySAT



Portfolio Solvers: HordeSAT

What is the impact of diversification and clause exchange?

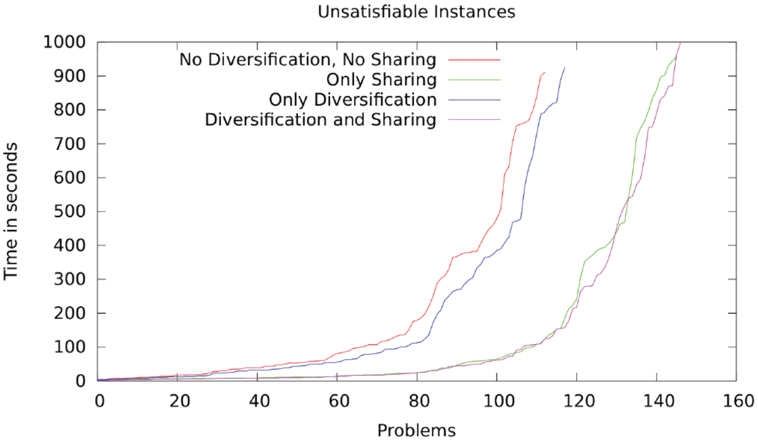
- 16 processes with 1 thread each
- Random 3-SAT, only satisfiable instances



Portfolio Solvers: HordeSAT

What is the impact of diversification and clause exchange?

- 16 processes with 1 thread each
- Random 3-SAT, only unsatisfiable instances



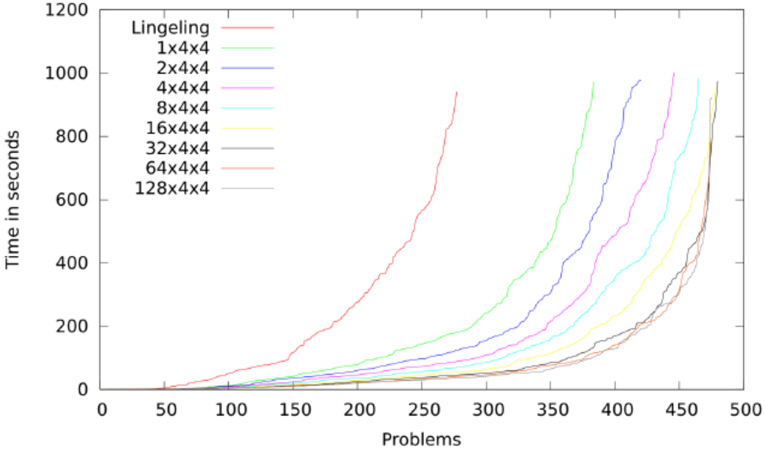
Portfolio Solvers: HordeSAT

How scalable are portfolio approaches?

Portfolio Solvers: HordeSAT

How scalable are portfolio approaches?

- (#nodes)x(#processes/node)x(#threads/process)



Search Space Splitting Approach

Basic idea:

- Split the search space into disjoint subspaces
- Each process searches in a disjoint subspace
- Load balancing mechanism to maintain all processes busy

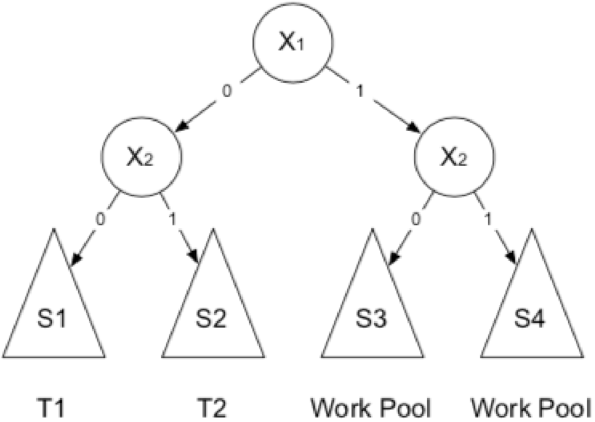
Search Space Splitting Approach

Basic idea:

- Split the search space into disjoint subspaces
- Each process searches in a disjoint subspace
- Load balancing mechanism to maintain all processes busy

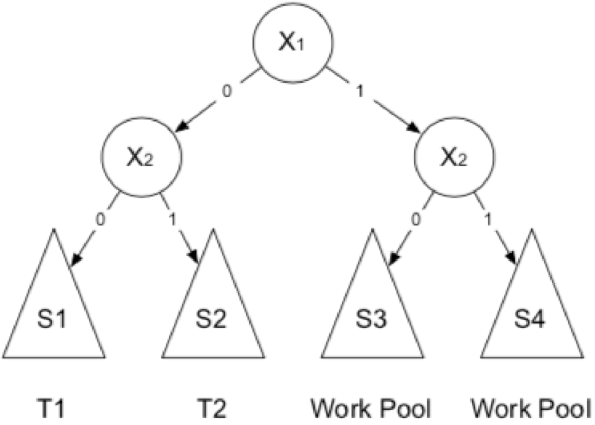
How to split the search space?

Search Space Splitting



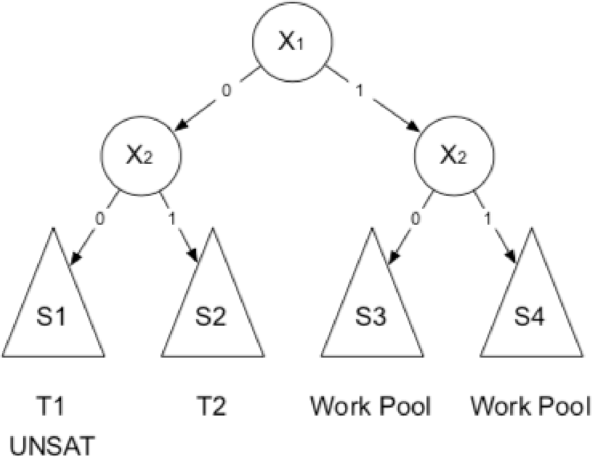
- Guiding path S1: $x_1 = 0, x_2 = 0$
- Restricts the search space of a given process

Search Space Splitting



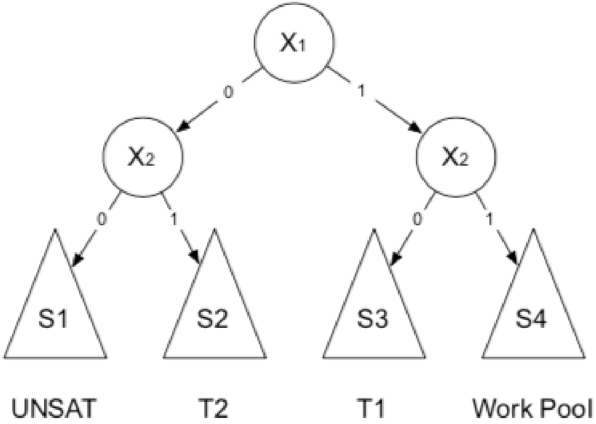
- Unused guiding paths are stored in the work queue

Search Space Splitting



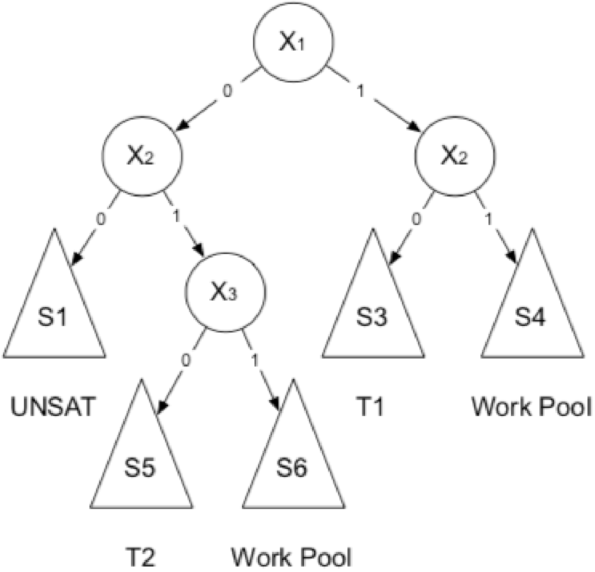
- If a subspace is unsatisfiable, then the process gets a new subspace

Search Space Splitting



- Dynamic work stealing procedure guarantees that all processes are always working

Search Space Splitting



Can we do a better split of the search space?

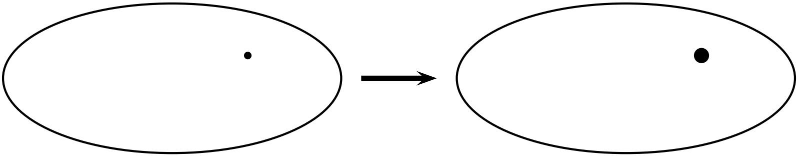
Conflict-Driven Clause Learning solvers

Highlights:

- goal: find small effective conflict clauses
- decisions: assign variables that occur in recent conflicts
- strength: powerful on "easy" problems

General CDCL situation:

- hit a conflict that can be generalized / analyzed to a large clause



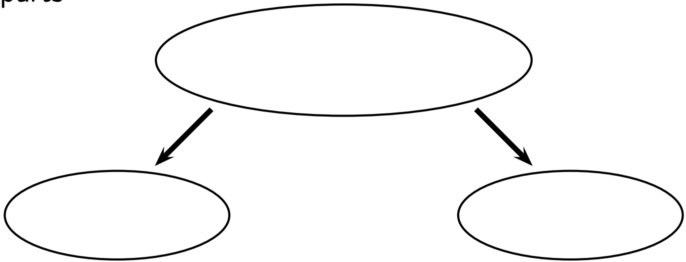
Lookahead solvers

Highlights:

- goal: construct a small binary search tree
- decisions: assign variables that cause a large reduction
- strength: powerful on small hard problems

Ideal lookahead situation:

- split the search space into two equally large but smaller parts



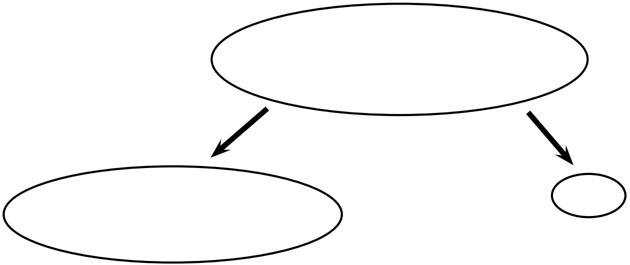
Lookahead solvers

Highlights:

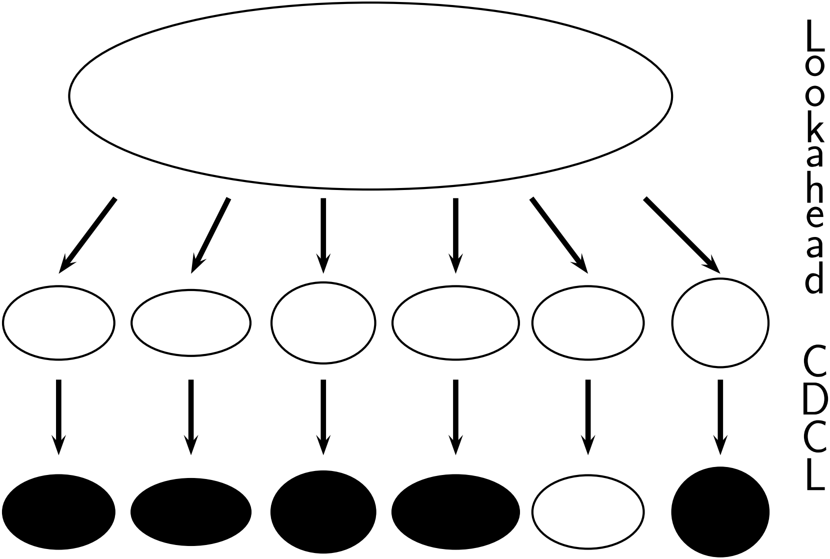
- goal: construct a small binary search tree
- decisions: assign variables that cause a large reduction
- strength: powerful on small hard problems

General lookahead situation:

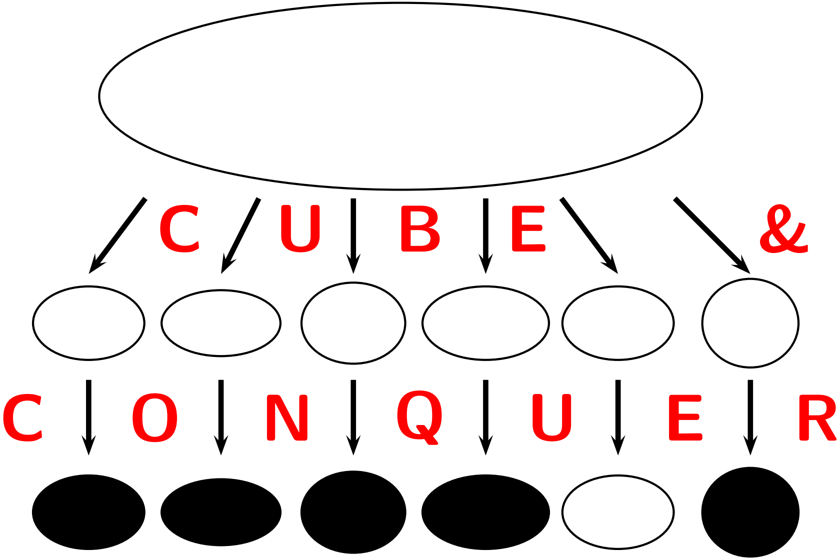
- the search space is split into a large and a small part



Best of both worlds: Combining Lookahead and CDCL



Best of both worlds: Cube and Conquer



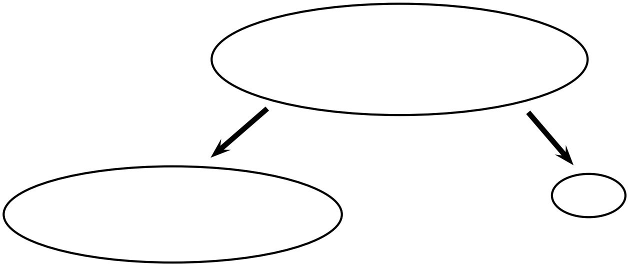
Cube: key observation

Split until the problem becomes easy

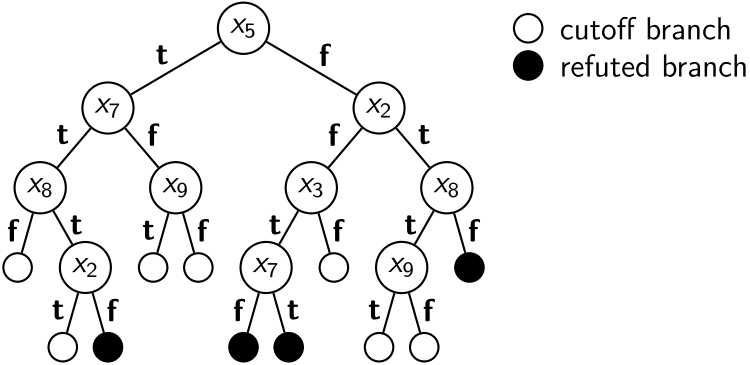
- do not have a fixed cut off depth
- determine hardness by number of assigned variables
- create many thousands or even millions of cubes

General lookahead situation:

- the search space is split into a large and a small part



Cube: example

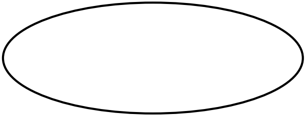


$$F_1 := F \wedge (x_5 \wedge x_7 \wedge \neg x_8)$$
$$F_2 := F \wedge (x_5 \wedge x_7 \wedge x_8 \wedge x_2)$$
$$F_3 := F \wedge (x_5 \wedge \neg x_7 \wedge x_9)$$
$$F_4 := F \wedge (x_5 \wedge \neg x_7 \wedge \neg x_9)$$
$$F_5 := F \wedge (\neg x_5 \wedge \neg x_2 \wedge \neg x_3)$$
$$F_6 := F \wedge (\neg x_5 \wedge x_2 \wedge x_8 \wedge x_9)$$
$$F_7 := F \wedge (\neg x_5 \wedge x_2 \wedge x_8 \wedge \neg x_9)$$

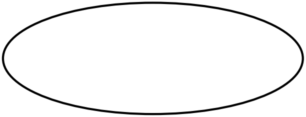
Conquer: describing cubes

How much information to send to the CDCL solver?

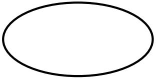
- Only the decisions

 $\wedge\ \varphi_{\mathrm{dec}}$

- The full assignment (including failed literals)

 $\wedge\ \varphi_{\mathrm{dec}}\ \wedge\ \varphi_{\mathrm{imp}}$

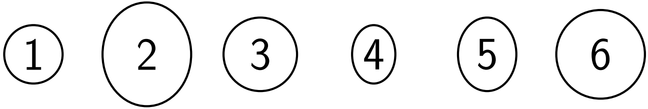
- The simplified formula (including local learnt clauses)



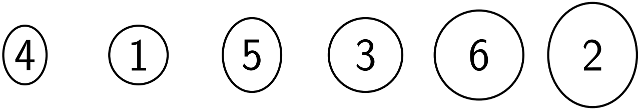
Conquer: ordering cubes

What is the optimal order to solve the cubes?

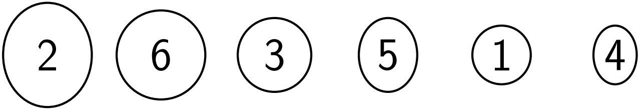
- Depth-first search (in lookahead order)



- Solves cubes with increasing (approximated) search space



- Solves cubes with decreasing (approximated) search space



Conquer: parallel solving

Strategies to solve cubes in parallel:

1. cores solve different cubes in parallel
2. cores solve the same cube in parallel
3. start with (1) till no new cubes are available, continue with (2)

Conquer: parallel solving

Strategies to solve cubes in parallel:

1. cores solve different cubes in parallel
2. cores solve the same cube in parallel
3. start with (1) till no new cubes are available, continue with (2)

What to share between cores?

- nothing, so hardly communication required (only ask / receive cubes)
- sharing the learned clauses (maybe only to master)
- sharing the short conflict clauses, units (maybe also binaries)

Results: two experiments

1^{st} experiment: single core on Van der Waerden numbers

- hard combinatorial problem in Ramsey Theory
- comparison with the best solver for each instance
- cube solver: `OKsolver`
- conquer solver: `minisat`

2^{nd} experiment: multi core on challenging applications

- unsolved application instances from the SAT09 benchmarks
- comparison with the best parallel solvers
- cube solver: `march`
- conquer solver: `lingeling`

Results: palindromic Van der Waerden numbers

- k_1 : arithmetic progression of first set;
- k_2 : arithmetic progression of second set;
- n : number of elements to partition;
- best solver : time of fastest sequential solver;
- D : cut off depth.

k_1	k_2	n	#cls	?	best solver	D	#cubes	C&C
3	25	586	45779	S	$\sim$ 13 days	45	9120	6.5 hours
3	25	607	49427	U	$\sim$ 13 days	45	13462	2 days
4	12	387	15544	S	$>$ 14 days	30	132131	2 days
4	12	394	15889	U	$>$ 14 days	34	147237	8 hours
5	8	312	9121	S	3.5 days	20	2248	5 hours
5	8	313	9973	U	53 days	20	87667	40 hours

Results: parallel SAT solving

Portfolio solvers:

- run multiple versions of the same solver (different seeds)
- share short conflict clauses such as units
- solver `pLingeling` (`pLing`), on a 12-core machine

Grid based SAT solving approach:

- run solvers with different cubes on a grid
- grid constraints: limited communication, possible delay and timeout
- solver `PartitionTree` (`PTree`) on a grid, up to 60 jobs in parallel

Results: hard application benchmarks

Benchmark	?	#cubes	I total	II total	II 12-core	pLing 12-core
9dlx_vliw_at_b_iq8	U	121	150	—	—	**3256**
9dlx_vliw_at_b_iq9	U	100	179	—	—	**5164**
AProVE07-25	U	84247	89	100340	**8690**	—
dated-5-19-u	U	57716	418	3214	**1451**	4465
eq.atree.braun.12	U	86541	85	3261	**273**	—
eq.atree.braun.13	U	81313	77	18165	**1517**	—
gss-24-s100	S	18237	48	4975	**415**	2930
gss-26-s100	S	19455	57	37259	**3108**	18173
gus-md5-14	U	60102	961	—	—	—
ndhf_xits_09_UNS	U	37358	82	71096	12041	—
rbcl_xits_09_UNK	U	54669	132	94911	11542	—
rpoc_xits_09_UNS	U	30681	114	48028	**8366**	—
sortnet-8-ipc5-h19	S	724	153	48668	4067	**2700**
total-10-17-u	U	9192	288	5638	4517	**3672**
total-5-15-u	U	14914	215	—	—	—

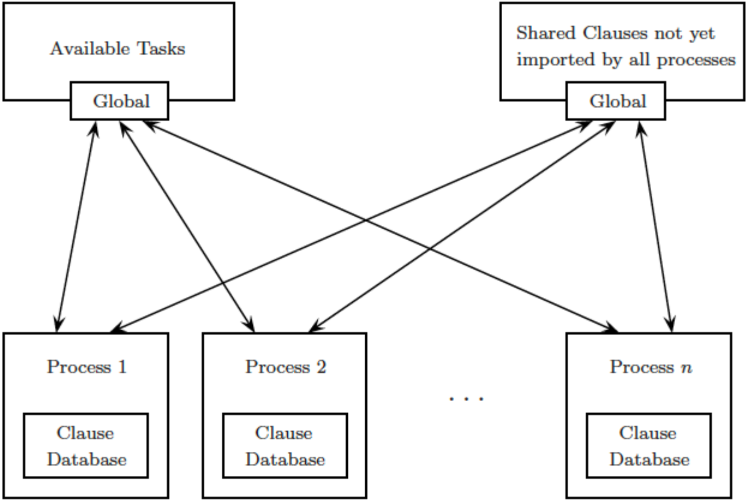
Results: hard application benchmarks

Benchmark	?	#cubes	I total	II total	II 12-core	pLing 12-core
9dlx_vliw_at_b_iq8	U	121	150	—	—	**3256**
9dlx_vliw_at_b_iq9	U	100	179	—	—	**5164**
AProVE07-25	U	84247	89	100340	**8690**	—
dated-5-19-u	U	57716	418	3214	**1451**	4465
eq.atree.braun.12	U	86541	85	3261	**273**	—
eq.atree.braun.13	U	81313	77	18165	**1517**	—
gss-24-s100	S	18237	48	4975	**415**	2930
gss-26-s100	S	19455	57	37259	**3108**	18173
gus-md5-14	U	60102	961	—	—	—
ndhf_xits_09_UNS	U	37358	82	71096	12041	—
rbcl_xits_09_UNK	U	54669	132	94911	11542	—
rpoc_xits_09_UNS	U	30681	114	48028	**8366**	—
sortnet-8-ipc5-h19	S	724	153	48668	4067	**2700**
total-10-17-u	U	9192	288	5638	4517	**3672**
total-5-15-u	U	14914	215	—	—	—

Results: hard application benchmarks

Benchmark	?	#cubes	I total	II total	II 12-core	pLing 12-core
9dlx_vliw_at_b_iq8	U	121	150	—	—	**3256**
9dlx_vliw_at_b_iq9	U	100	179	—	—	**5164**
AProVE07-25	U	84247	89	100340	**8690**	—
dated-5-19-u	U	57716	418	3214	**1451**	4465
eq.atree.braun.12	U	86541	85	3261	**273**	—
eq.atree.braun.13	U	81313	77	18165	**1517**	—
gss-24-s100	S	18237	48	4975	**415**	2930
gss-26-s100	S	19455	57	37259	**3108**	18173
gus-md5-14	U	60102	961	—	—	—
ndhf_xits_09_UNS	U	37358	82	71096	12041	—
rbcl_xits_09_UNK	U	54669	132	94911	11542	—
rpoc_xits_09_UNS	U	30681	114	48028	**8366**	—
sortnet-8-ipc5-h19	S	724	153	48668	4067	**2700**
total-10-17-u	U	9192	288	5638	4517	**3672**
total-5-15-u	U	14914	215	—	—	—

Architecture of parallel SAT solvers



- Most parallel SAT solvers has its own clause database!

Other forms of parallelization

Parallel unit propagation:

- More than 90% of the SAT solver is spent doing unit propagations

Number of new implied literals	Ratio
2	13%
4	4%

Can we find these implied literals in parallel?

Other forms of parallelization

Parallel unit propagation:

- More than 90% of the SAT solver is spent doing unit propagations

Number of new implied literals	Ratio
2	13%
4	4%

Can we find these implied literals in parallel?

Does not scale beyond 2 cores!

Other forms of parallelization

Parallel unit propagation:

- More than 90% of the SAT solver is spent doing unit propagations

In the worst case unit propagation is inherently sequential:

$$\varphi = (\neg x_1) \wedge (x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_2 \vee \neg x_3 \vee x_4) \dots$$

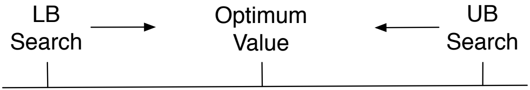
Applying unit propagation to φ results in the following chain of successive (sequential) and unique implications:

- $x_1 = 0 \rightarrow x_2 = 1 \rightarrow x_3 = 1 \rightarrow x_4 = 1 \dots$

What about parallel MaxSAT?

Parallel MaxSAT solver (2 threads)

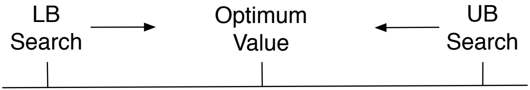
- Search in the lower and upper bound values of the optimal solution:



- The optimum value is found when:
  - LB or UB search terminates with a solution;
  - or when LB value = UB value.

Parallel MaxSAT solver (2 threads)

- Search in the lower and upper bound values of the optimal solution:

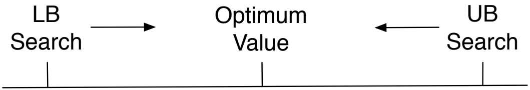


- The optimum value is found when:
  - LB or UB search terminates with a solution;
  - or when LB value = UB value.

T_0												T_1
LB												UB
0	1	2	3	4	5	6	7	8	9	10	11	12

Parallel MaxSAT solver (2 threads)

- Search in the lower and upper bound values of the optimal solution:



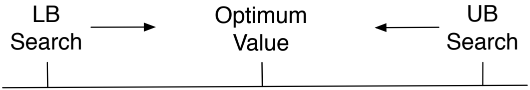
- The optimum value is found when:
  - LB or UB search terminates with a solution;
  - or when LB value = UB value.

	T_0											T_1
	LB											UB
0	1	2	3	4	5	6	7	8	9	10	11	12

T_0 returns UNSAT; update lower bound value

Parallel MaxSAT solver (2 threads)

- Search in the lower and upper bound values of the optimal solution:



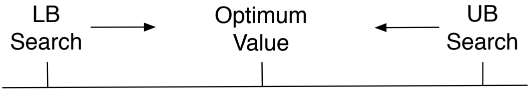
- The optimum value is found when:
  - LB or UB search terminates with a solution;
  - or when LB value = UB value.

	T_0								T_1			
	LB								UB			
0	1	2	3	4	5	6	7	8	9	10	11	12

T_1 returns SAT; update upper bound value

Parallel MaxSAT solver (2 threads)

- Search in the lower and upper bound values of the optimal solution:



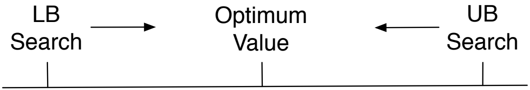
- The optimum value is found when:
  - LB or UB search terminates with a solution;
  - or when LB value = UB value.

		T_0							T_1			
		LB							UB			
0	1	2	3	4	5	6	7	8	9	10	11	12

T_0 returns UNSAT; update lower bound value

Parallel MaxSAT solver (2 threads)

- Search in the lower and upper bound values of the optimal solution:



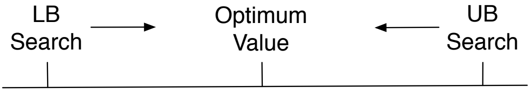
- The optimum value is found when:
  - LB or UB search terminates with a solution;
  - or when LB value = UB value.

		T_0; T_1										
		LB; UB										
0	1	2	3	4	5	6	7	8	9	10	11	12

T_1 returns SAT; update upper bound value

Parallel MaxSAT solver (2 threads)

- Search in the lower and upper bound values of the optimal solution:



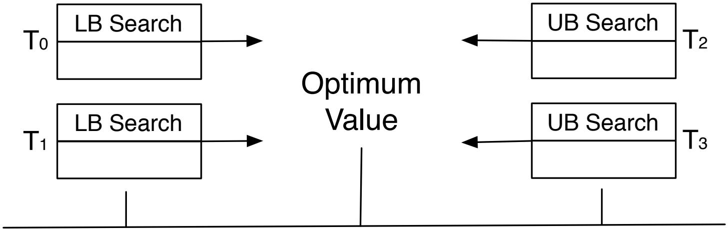
- The optimum value is found when:
  - LB or UB search terminates with a solution;
  - or when LB value = UB value.

		T_0; T_1										
		LB; UB										
0	1	2	3	4	5	6	7	8	9	10	11	12

LB value = UB value, optimal value has been found

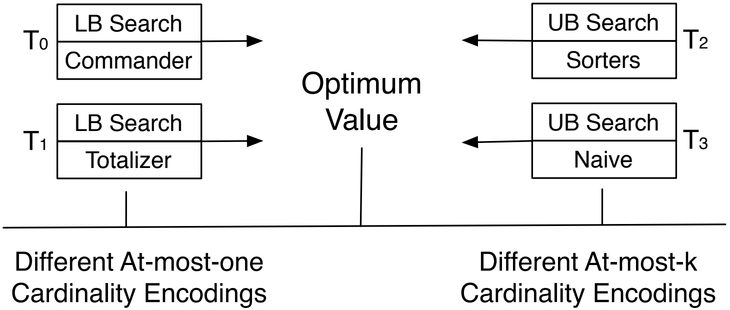
Parallel MaxSAT solver (n threads)

- Search in the lower and upper bound values of the optimal solution:



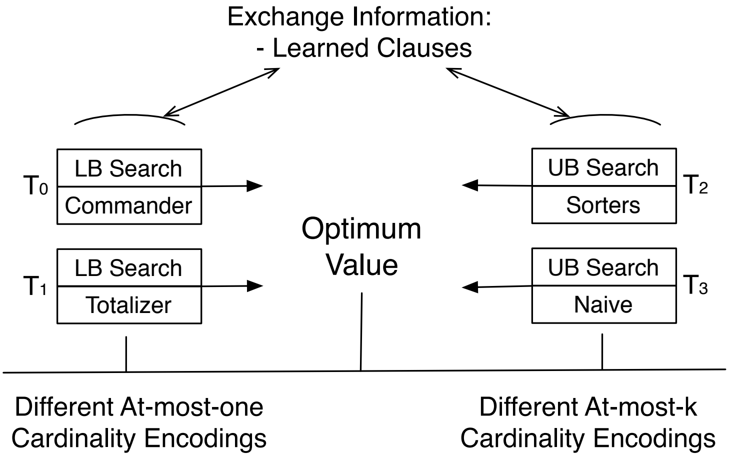
Parallel MaxSAT solver (n threads)

- Search in the lower and upper bound values of the optimal solution:

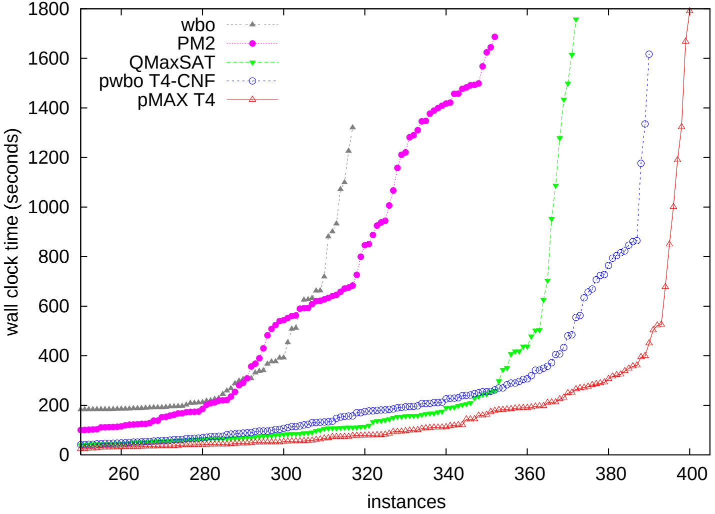


Parallel MaxSAT solver (n threads)

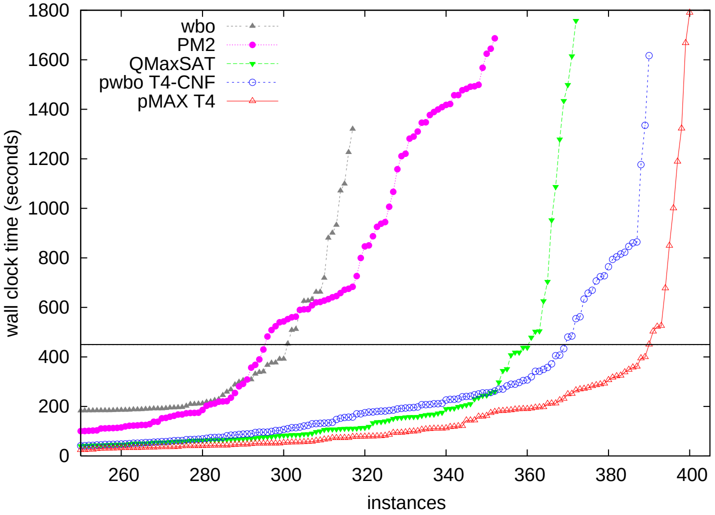
- Search in the lower and upper bound values of the optimal solution:



Experimental Results

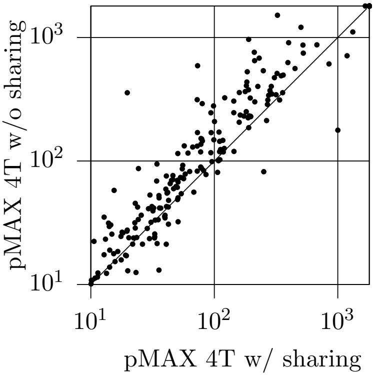


Experimental Results



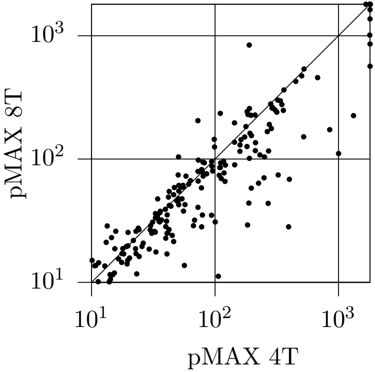
Experimental Results

- Impact of sharing learned clauses (seconds):



Experimental Results

- Scalability of pMAX (seconds):



Experimental Results

- Speedup on instances solved by all solvers:

Solver	Time (s)	Speedup
wbo	67,947.41	1.00
pwbo 4T-CNF	18,015.69	3.77
pMAX 4T	11,382.91	5.97
pMAX 8T	7,990.10	8.50

Experimental Results

- Speedup on instances solved by all solvers:

Solver	Time (s)	Speedup
wbo	67,947.41	1.00
pwbo 4T-CNF	18,015.69	3.77
pMAX 4T	11,382.91	5.97
pMAX 8T	7,990.10	8.50

- Does not scale beyond 8 cores!

Conclusions

Parallelization of SAT algorithms is hard!

Success stories:

- Cube and Conquer on thousands of nodes to solve hard combinatorial problems:
  - Pythagorean Triples
  - Schur Number Five
- Variable Elimination using GPUs
  - $66\times$ speedup
  - NVIDIA Titan Xp GPU (30 SMs with 128 cores each)

Still many open research directions for scalability in parallel SAT solving!

Parallel Automated Reasoning

Ruben Martins

Carnegie Mellon University

http://www.cs.cmu.edu/~mheule/15816-f19/

Automated Reasoning and Satisfiability, October 3, 2019