TECHNISCHE UNIVERSITÄT DRESDEN

DESIGNING SCALABLE PARALLEL SAT SOLVERS

Antti E. J. Hyvärinen and Norbert Manthey

Trento, 18.06.2012

Solving Scenarios

There are two ways of using (parallel) resources:

- There is a set of formulas that needs to be solved, e.g. scheduling trains for countries
- Solve a single instance fast, e.g. verify hardware design

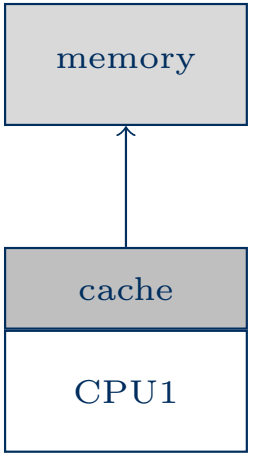
Solving Scenarios

There are two ways of using (parallel) resources:

- There is a set of formulas that needs to be solved, e.g. scheduling trains for countries
- Solve a single instance fast, e.g. verify hardware design

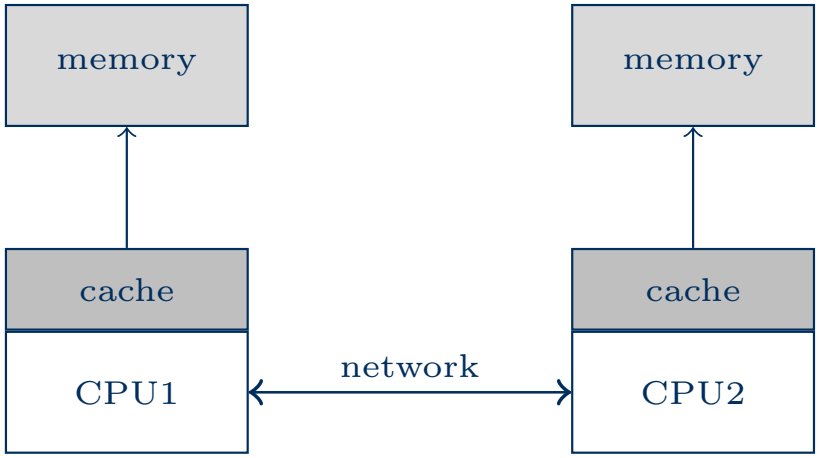
- This work focuses on solving a single formula
- The criterion is the wall clock time

Modern Architecture



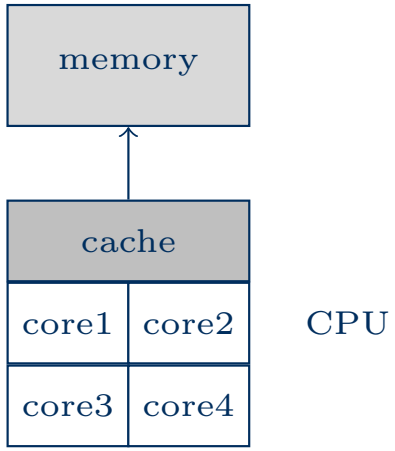
- Two decades ago, a computer had a single CPU

Modern Architecture



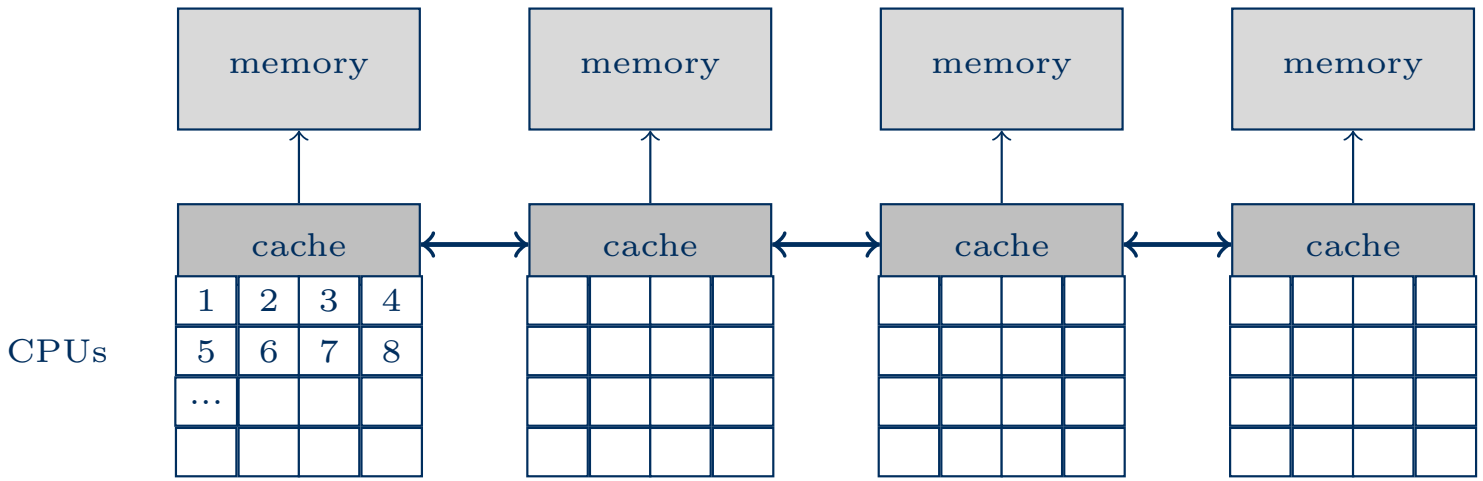
- One decade ago, there have been two CPUs in one computer

Modern Architecture



- Additionally, multi-core CPUs have been invented

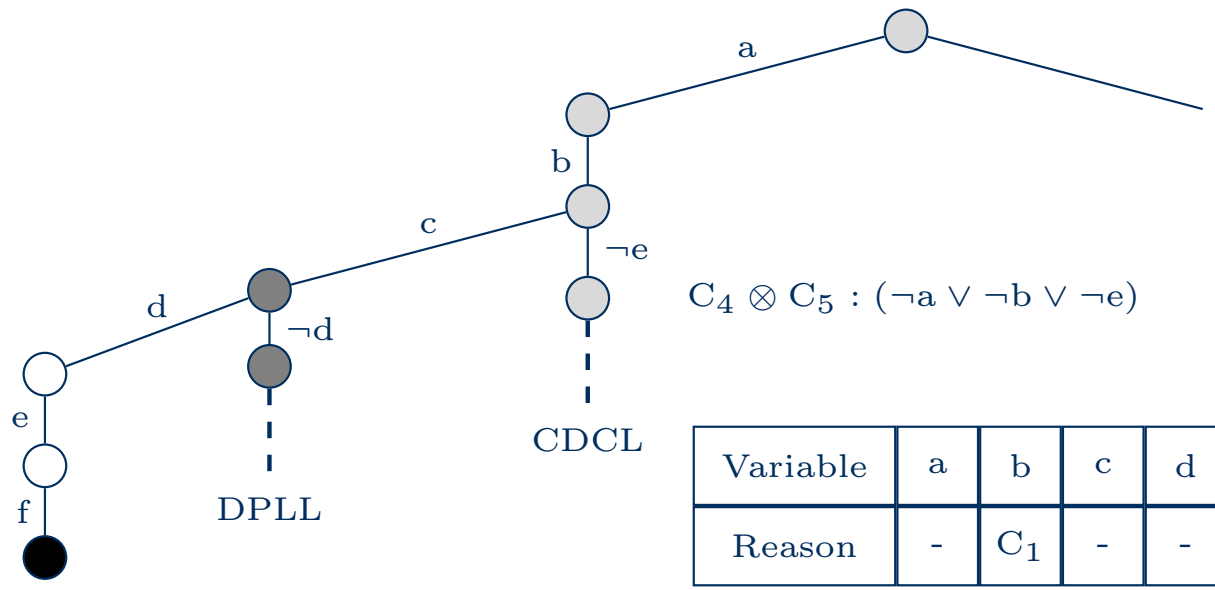
Modern Architecture



- Todays architecture offers 4 16-core CPUs on a single board
- The memory bandwidth is limited
- Single thread performance decreased

The CDCL Algorithm

$(\neg a \vee b) \wedge (\neg b \vee \neg d \vee e) \wedge (c \vee d) \wedge (\neg a \vee \neg b \vee \neg e \vee f) \wedge (\neg a \vee \neg e \vee \neg f) \wedge (d \vee \neg f) \wedge (\neg c \vee e \vee f)$



$C_4 \otimes C_5 : (\neg a \vee \neg b \vee \neg e)$

Variable	a	b	c	d	e	f
Reason	-	C_1	-	-	C_2	C_4

$C_5 = (\neg a \vee \neg e \vee \neg f)$

Properties of the CDCL Algorithm

- Clause learning disables structured backtracking (improvement)
- Learned clauses prune the search space
- Decisions are usually based on learned clauses

- Restarts have been introduced
- Based on randomization, there is a run time distribution

Properties of the CDCL Algorithm

- Clause learning disables structured backtracking (improvement)
- Learned clauses prune the search space
- Decisions are usually based on learned clauses

- Restarts have been introduced
- Based on randomization, there is a run time distribution

- It is a sequential algorithm, but architecture becomes parallel

Properties of the CDCL Algorithm

- Clause learning disables structured backtracking (improvement)
- Learned clauses prune the search space
- Decisions are usually based on learned clauses

- Restarts have been introduced
- Based on randomization, there is a run time distribution

- It is a sequential algorithm, but architecture becomes parallel

- A parallelization of the CDCL algorithm is not intuitive

Outline

This work focuses on

- exploit parallelism to solve hard formulas
- comparing
  - portfolio solvers and non-determinism
  - partitioning solvers
  - iterative partitioning (exist for grids already)
- trying to answer how to use multi-core CPUs

Contributions:

- provide iterative partitioning for multi-core
- plain partitioning can increase the expected run time

proof in the paper

Outline

Simple Parallel SAT Solving (SPSAT)

Task: Solve the formula F

- Solve F with n different solver configurations S_i
- Return the result from the first solver

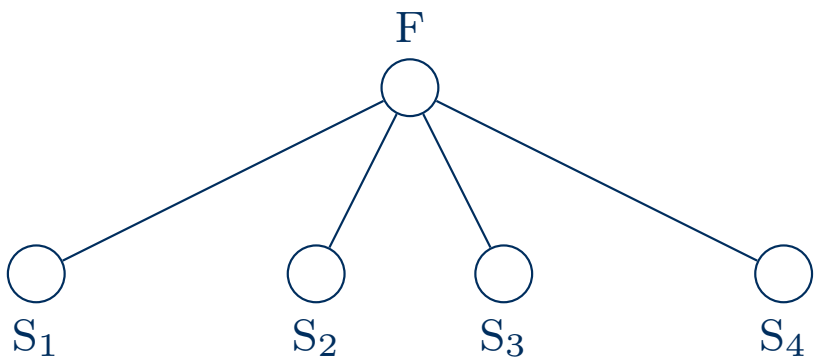
Simple Parallel SAT Solving (SPSAT)

Task: Solve the formula F

- Solve F with n different solver configurations S_i
- Return the result from the first solver
- Some details:
  - The minimum run time of the solvers is the overall run time
  - Modern systems share information (there has been much work)
  - We evaluated simplified version of parallel portfolio solving (some random decisions)

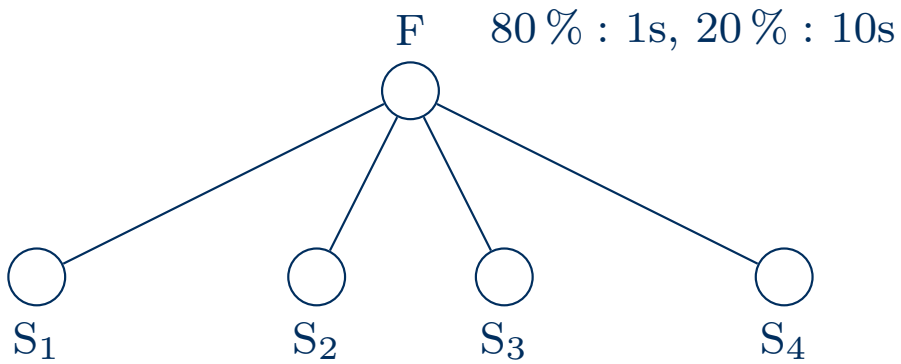
Simple Parallel SAT Solving

Task: Solve the formula F



Simple Parallel SAT Solving

Task: Solve the formula F



- Assume, solving F sequentially requires the expected run time (2.8 s)
- The expected run time for the parallel solver is (1.01 s)

Plain Partitioning

Task: Solve the formula F

- Solve F by partitioning into n partitions $F_1, \ldots, F_n$

  $$F \equiv \bigvee_i F_i \text{ and } F_i \wedge F_j \equiv \bot \text{ for all } 1 \leq i < j \leq n$$

- The sub formulas can be solved independently in parallel

Plain Partitioning

Task: Solve the formula F

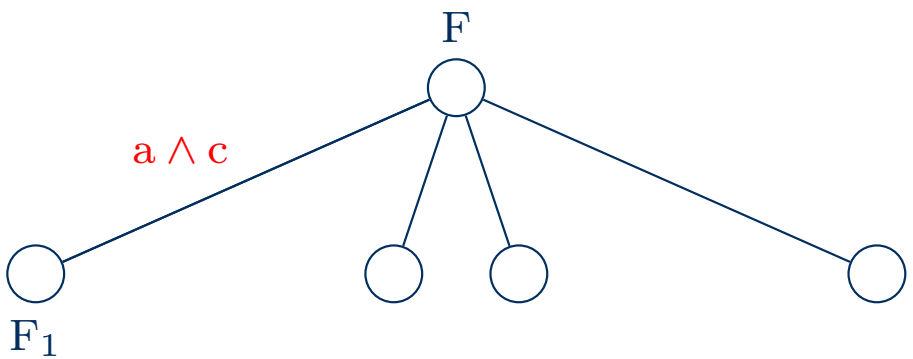
- Solve F by partitioning into n partitions $F_1, \ldots, F_n$

  $F \equiv \bigvee_i F_i$ and $F_i \wedge F_j \equiv \bot$ for all $1 \leq i < j \leq n$
- The sub formulas can be solved independently in parallel
- Some details:
  - Unsatisfiability is shown by proving each F_i unsatisfiable
  - The maximum run time of the solvers is the overall run time
  - Here: sharing information is to keep unary clauses at failures
  - We use VSIDS scores and scattering to create sub formulas

Partitioning technique

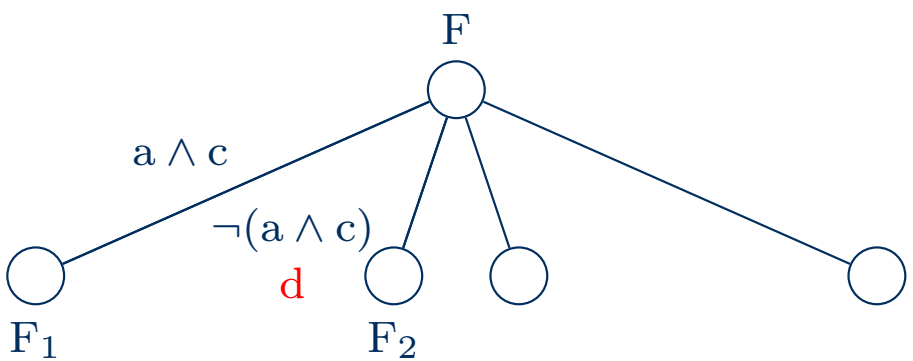
Plain Partitioning

Task: Solve the formula F



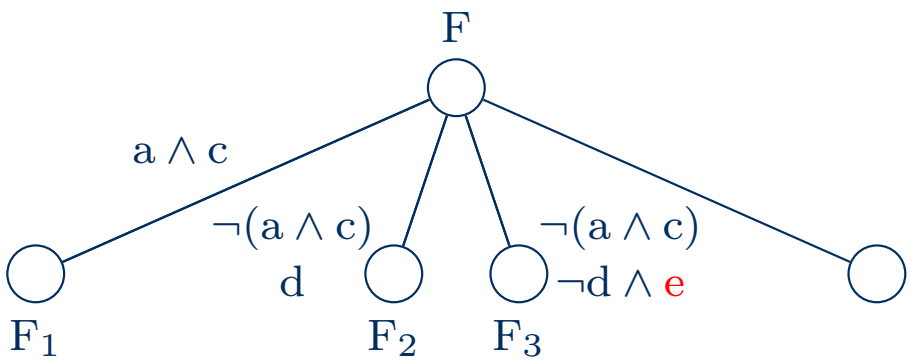
Plain Partitioning

Task: Solve the formula F



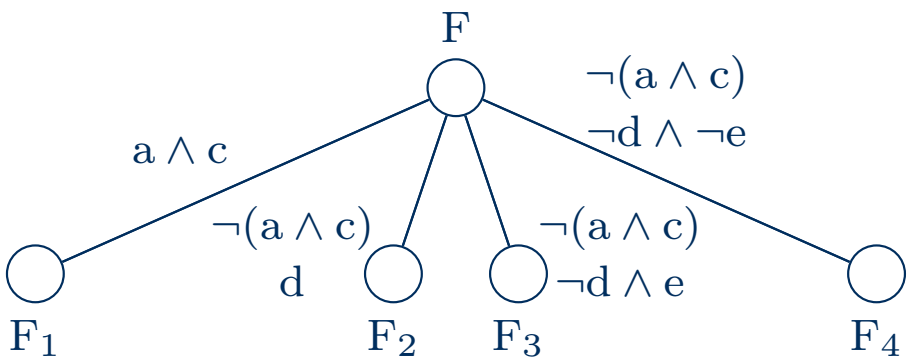
Plain Partitioning

Task: Solve the formula F



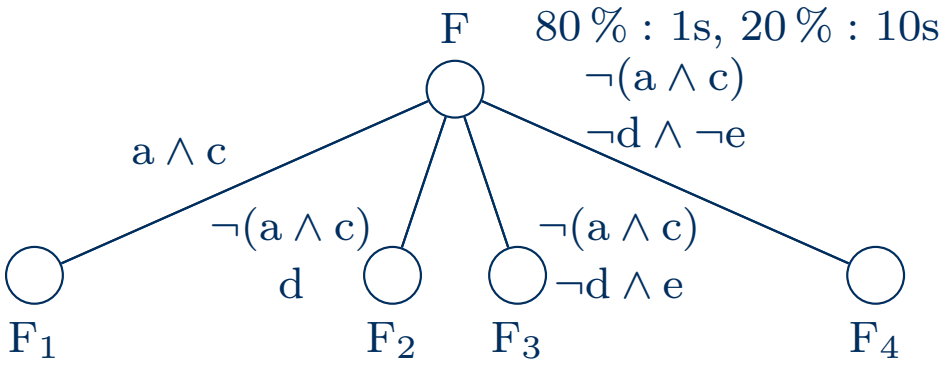
Plain Partitioning

Task: Solve the formula F



Plain Partitioning

Task: Solve the formula F



- Assume, F can be solved with the given expected run time (2.8 s)
- The expected run time for the parallel solver is (3.16 s)
- Under some assumptions, this effect can be proven

Iterative Partitioning

Task: Solve the formula F

- Solve F by partitioning F and the sub formulas iteratively
- Solve each sub formula, and F with limited resources (time, conflicts)
- Each created formula can be solved independently

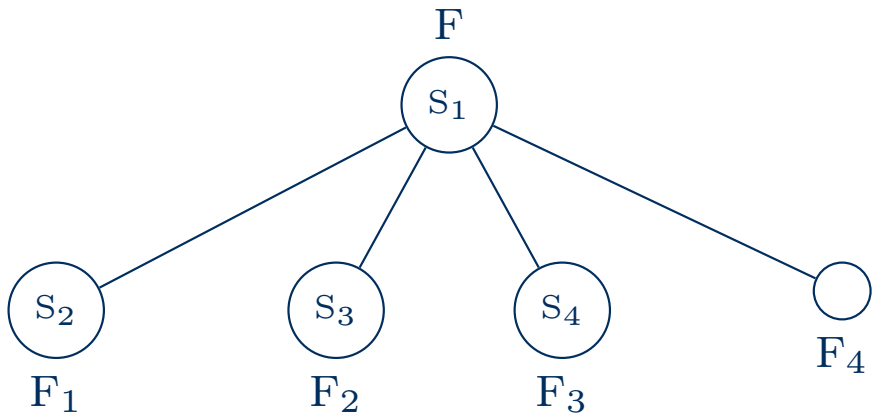
Iterative Partitioning

Task: Solve the formula F

- Solve F by partitioning F and the sub formulas iteratively
- Solve each sub formula, and F with limited resources (time, conflicts)
- Each created formula can be solved independently
- Some details:
  - The maximum run time does not exceed solving F
  - Unsatisfiability: having an unsatisfiable sub formula per path
  - We use scattering to create sub formulas
  - We share unary clauses downwards

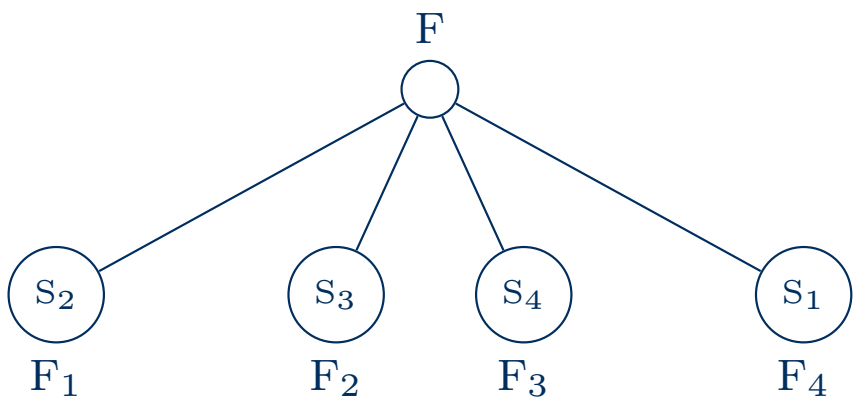
Iterative Partitioning

Task: Solve the formula F



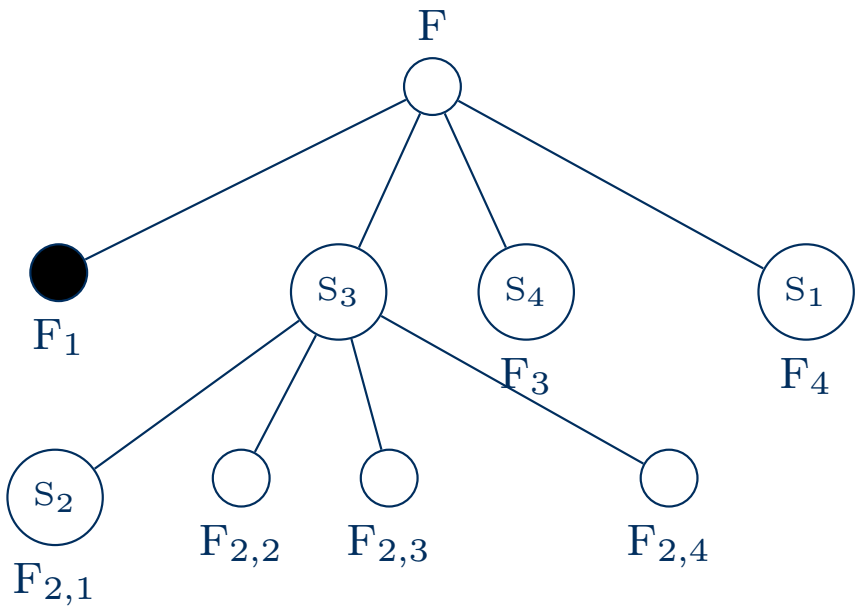
Iterative Partitioning

Task: Solve the formula F



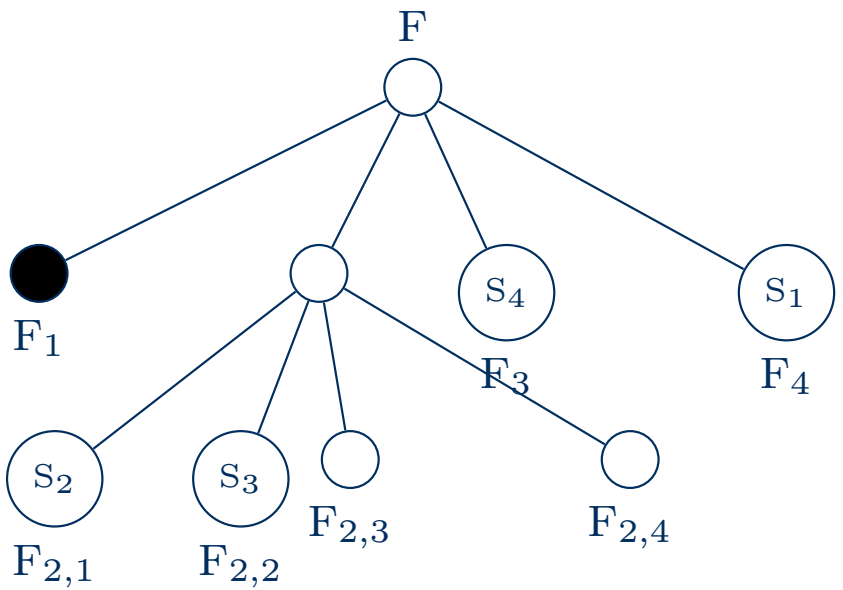
Iterative Partitioning

Task: Solve the formula F



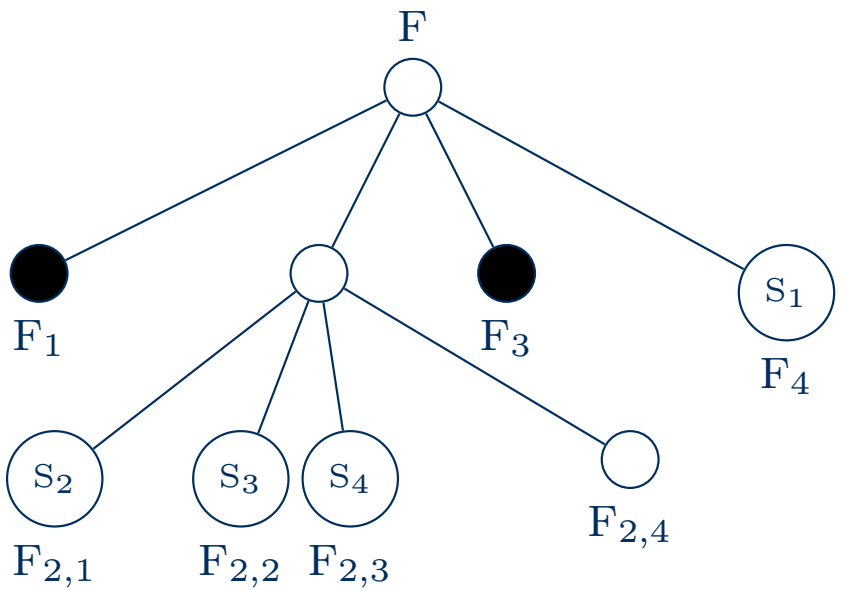
Iterative Partitioning

Task: Solve the formula F



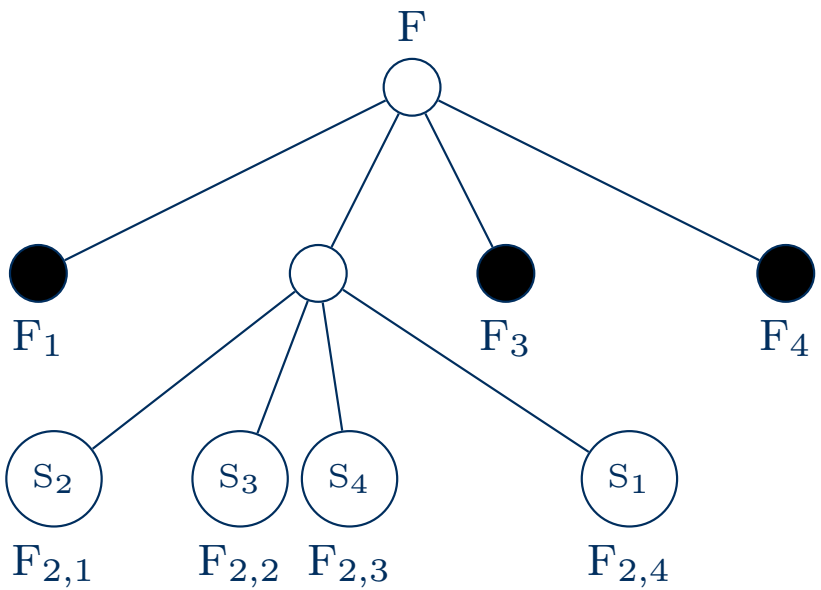
Iterative Partitioning

Task: Solve the formula F



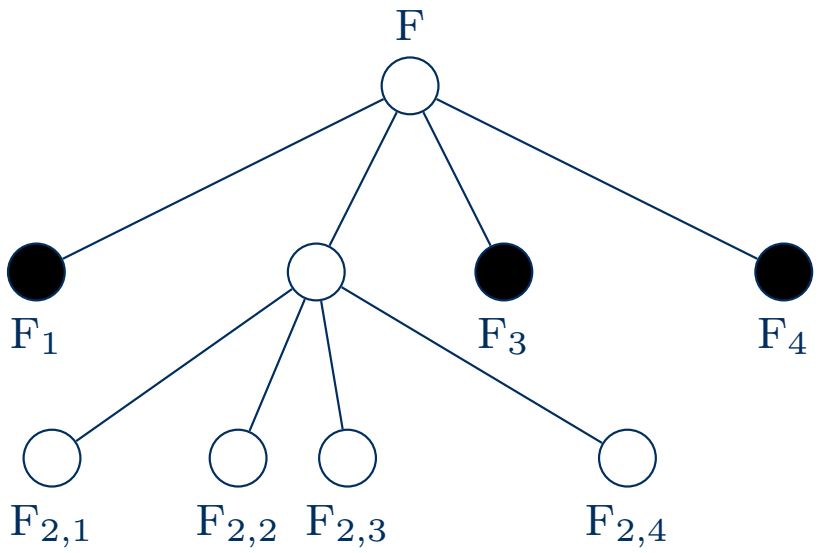
Iterative Partitioning

Task: Solve the formula F



Iterative Partitioning

Task: Solve the formula F



- Solvers perform a breadth first search
- Unsatisfiable sub spaces can be cut off early

Outline

Scalability Analysis

We want to reduce the wall clock time.

- Assumption: adding resources improves the performance

Scalability Analysis

We want to reduce the wall clock time.

- Assumption: adding resources improves the performance
- For multi-core CPUs, this does not hold in general
  - T_1 := run time for running a single solver
  - T_m := run time for running m times the same solver simultaneously
  - Usually $T_1 \cdot m < T_m$

Scalability Analysis

We want to reduce the wall clock time.

- Assumption: adding resources improves the performance
- For multi-core CPUs, this does not hold in general
  - T_1 := run time for running a single solver
  - T_m := run time for running m times the same solver simultaneously
  - Usually $T_1 \cdot m < T_m$
- Furthermore, plain partitioning suffers from the theoretical slow down

Scalability Analysis

We want to reduce the wall clock time.

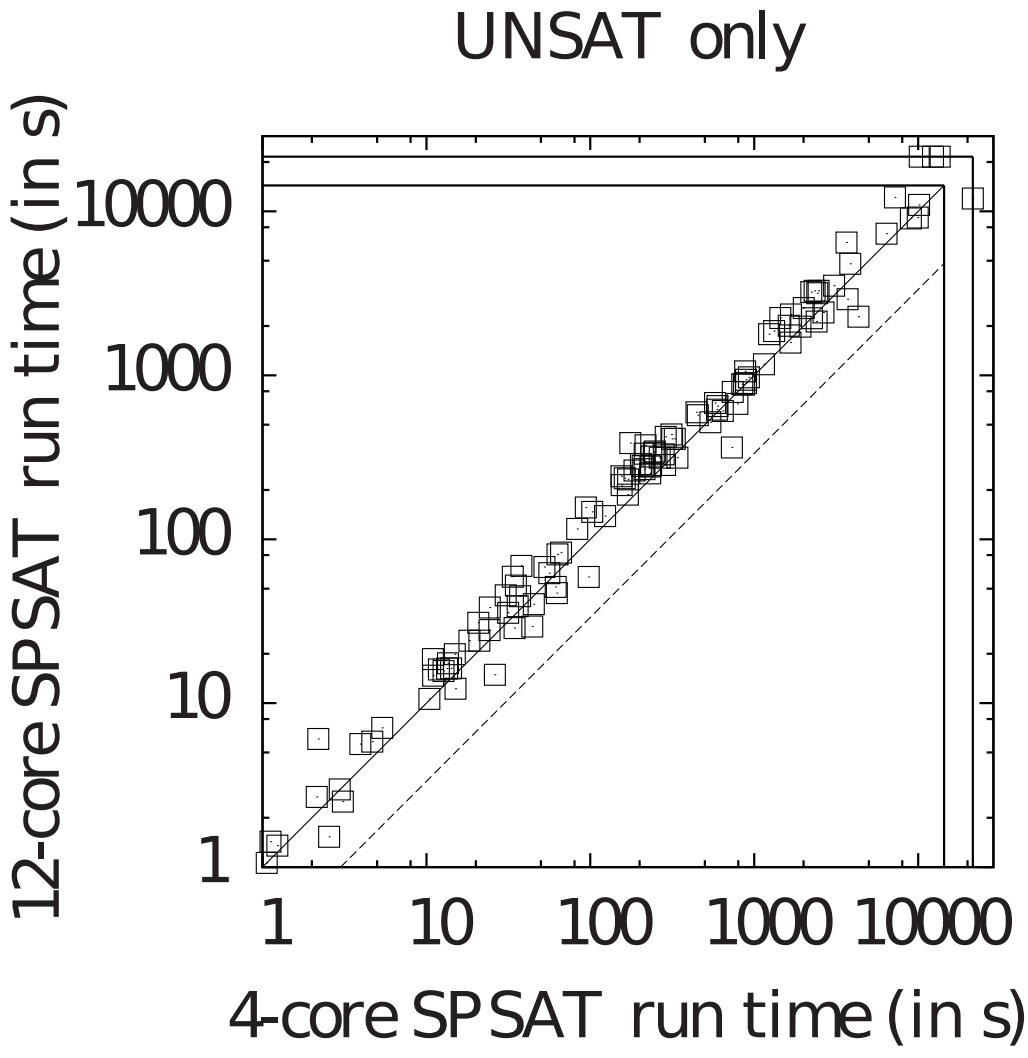
- Assumption: adding resources improves the performance
- For multi-core CPUs, this does not hold in general
  - T_1 := run time for running a single solver
  - T_m := run time for running m times the same solver simultaneously
  - Usually $T_1 \cdot m < T_m$
- Furthermore, plain partitioning suffers from the theoretical slow down
- For unsatisfiable instances, the run time is more stable
- For easy instances, parallelization will introduce overhead

Scalability Analysis

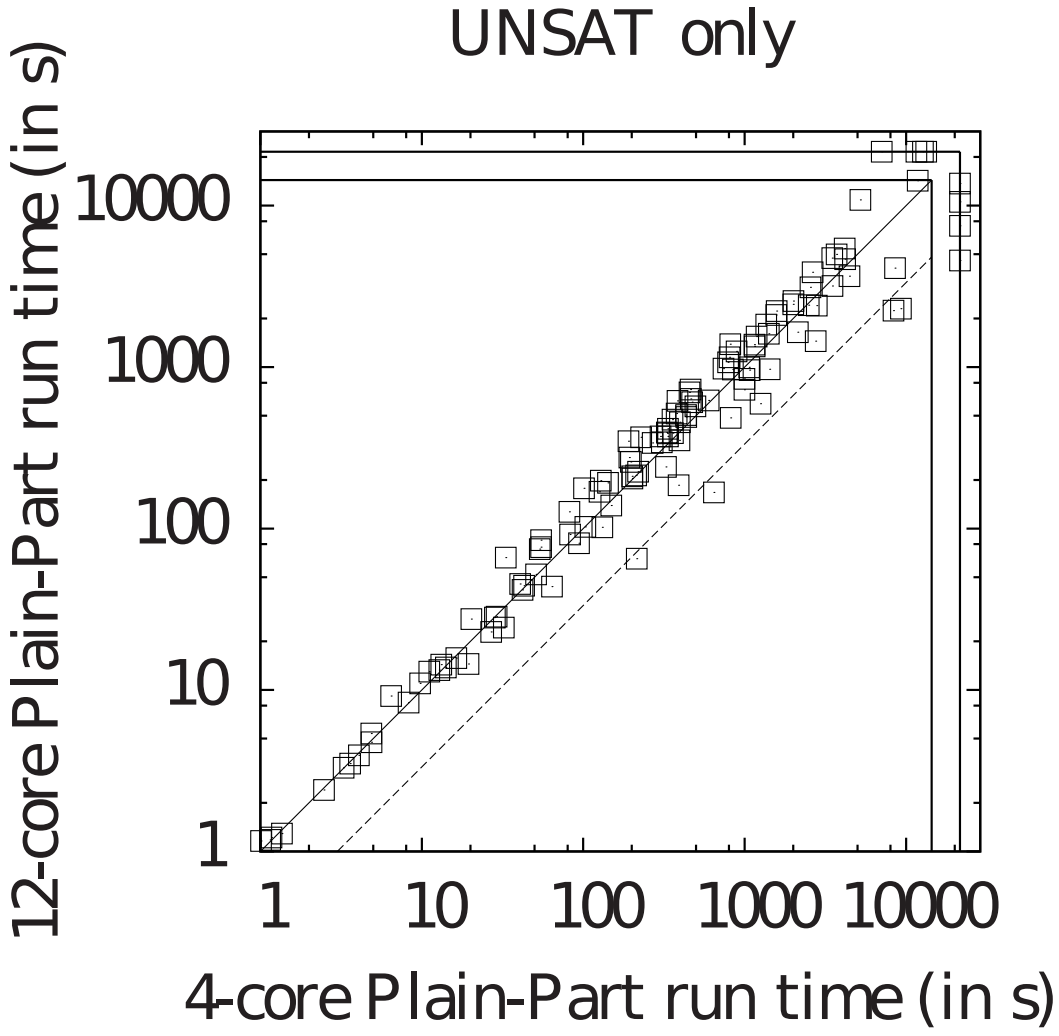
We want to reduce the wall clock time.

- Assumption: adding resources improves the performance
- For multi-core CPUs, this does not hold in general
  - T_1 := run time for running a single solver
  - T_m := run time for running m times the same solver simultaneously
  - Usually $T_1 \cdot m < T_m$
- Furthermore, plain partitioning suffers from the theoretical slow down
- For unsatisfiable instances, the run time is more stable
- For easy instances, parallelization will introduce overhead
- All solvers are based on MiniSAT 2.2 without any special optimizations

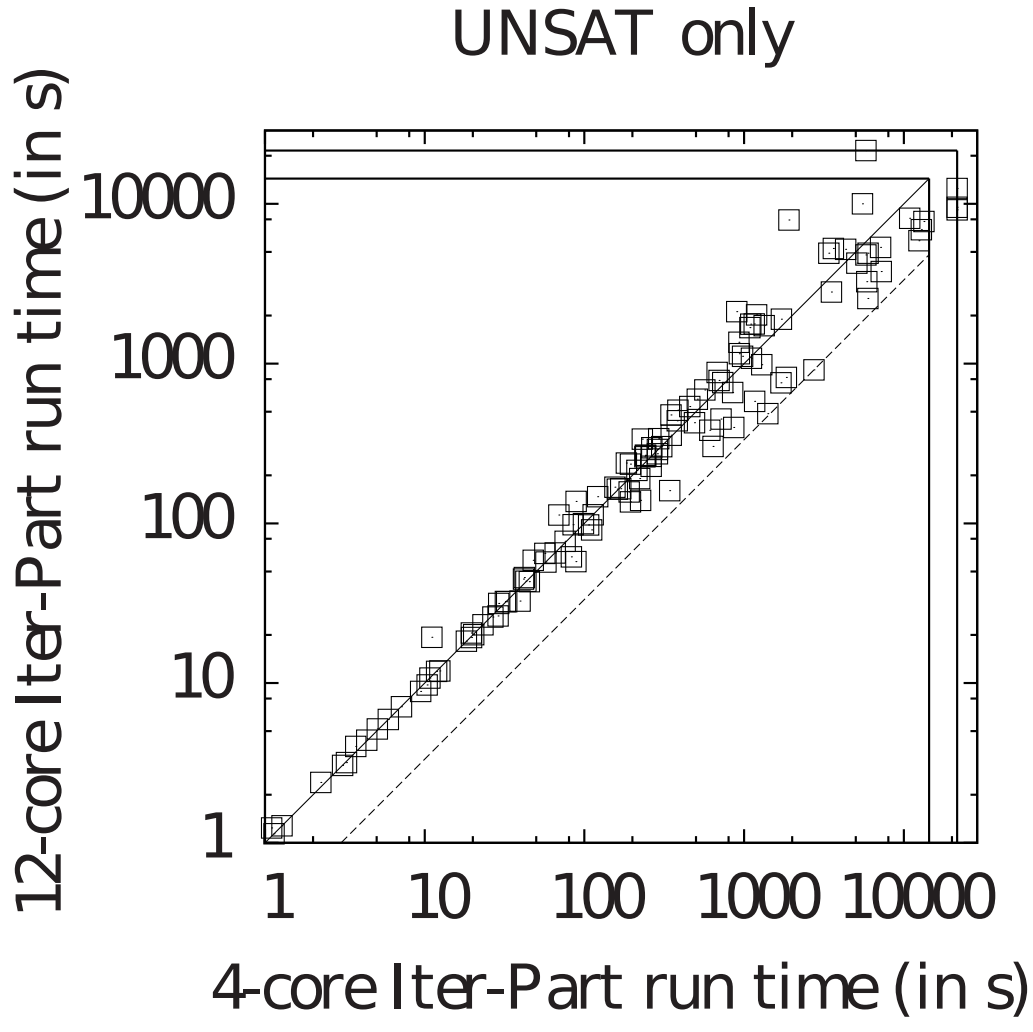
Scalability Analysis



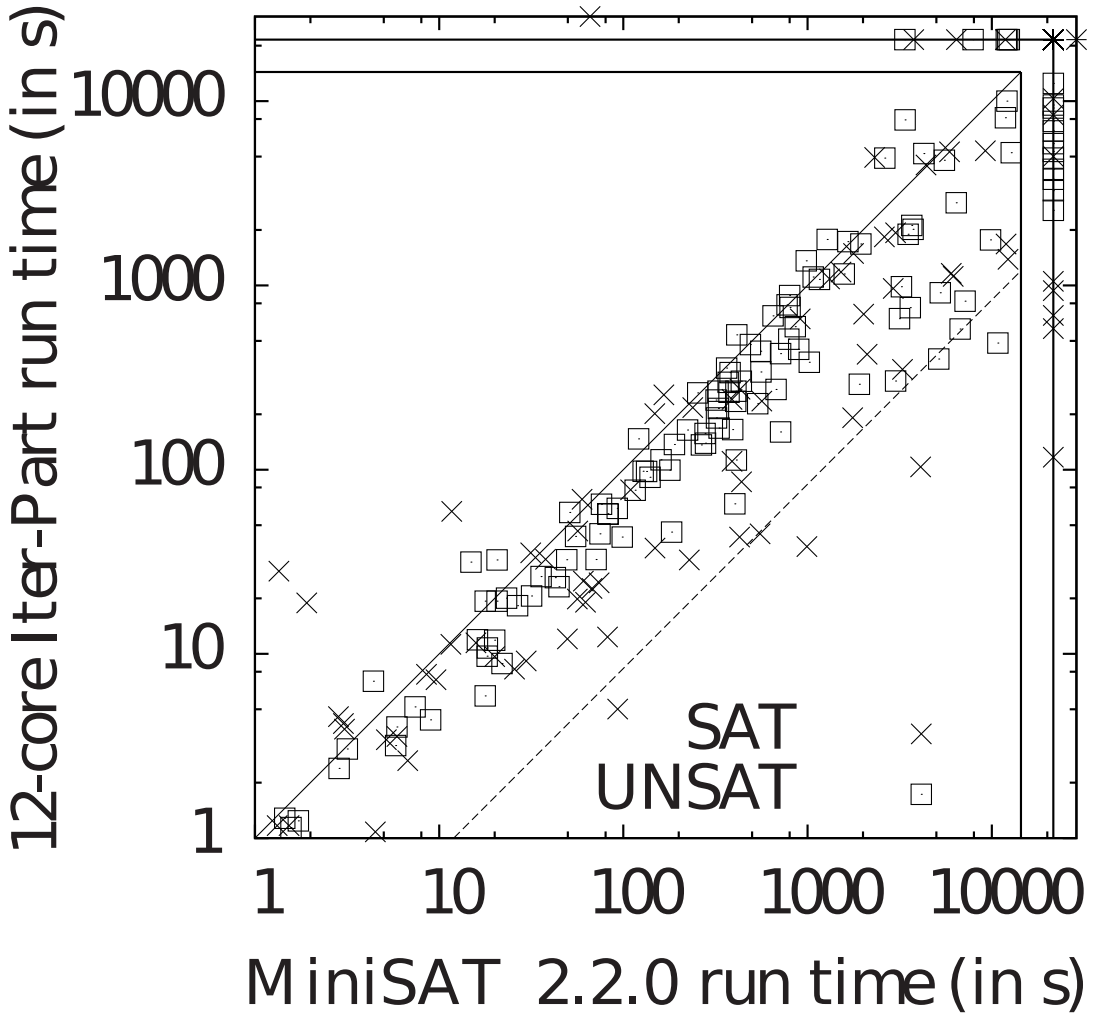
Scalability Analysis



Scalability Analysis



Scalability Analysis



Scalability Analysis

Summarizing the results:

- Plain Partitioning slows down solving when 1 core is compared to 12 cores
- Neither SPSAT nor Plain Partitioning scale well from 4 to 12 cores
- For hard instances, adding cores to Iterative Partitioning increases the performance

Scalability Analysis

Summarizing the results:

- Plain Partitioning slows down solving when 1 core is compared to 12 cores
- Neither SPSAT nor Plain Partitioning scale well from 4 to 12 cores
- For hard instances, adding cores to Iterative Partitioning increases the performance

	SAT		UNSAT	
Approach	slower	faster	slower	faster
SPSAT	47	36	93	23
Plain Partitioning	45	34	77	39
Iterative Partitioning	33	45	61	58

Number of faster solved instances when 12 cores are compared to 4 cores

Scalability Analysis

Summarizing the results:

- Plain Partitioning slows down solving when 1 core is compared to 12 cores
- Neither SPSAT nor Plain Partitioning scale well from 4 to 12 cores
- For hard instances, adding cores to Iterative Partitioning increases the performance

	SAT run time		UNSAT run time	
Approach	4-core	12-core	4-core	12-core
SPSAT	61784	57380	111152	127462
Plain Partitioning	61681	60934	121432	119925
Iterative Partitioning	53918	50642	153726	131521

Run time comparison

Outline

Outlook

With the algorithms available, further research can be done:

- Sequential solvers perform inprocessing, restarts and keep knowledge
  - Can we simulate this behavior?
  - Should sub formulas be preprocessed?
- Portfolio solvers share learned clauses
  - Which clauses can be shared in a partitioning solvers?
  - Are the heuristics for portfolio solvers adequate?
- With 64 core computing resources, does Iterative Partitioning scale further?

Conclusion

- We present three parallel solving techniques and evaluated their scalability.

- Important to remember
  - Plain Partitioning can slow down the search
  - Portfolio solving does not scale well with many cores
  - Iterative Partitioning provides interesting results

The solver is available at `http://tools.computational-logic.org`

Thanks for your attention

The solver is available at `http://tools.computational-logic.org`