Parallel Variable Elimination on CNF Formulas

Kilian Gebhardt   Norbert Manthey

Knowledge Representation and Reasoning Group
Technische Universität Dresden

July 8, 2013

Motivation for Parallel Preprocessing

- Formula simplification / preprocessing important part of SAT-Solving tool chain

Motivation for Parallel Preprocessing

- Formula simplification / preprocessing important part of SAT-Solving tool chain
- Simplification techniques polynomial bounded, but application until fixpoint takes long

Motivation for Parallel Preprocessing

- Formula simplification / preprocessing important part of SAT-Solving tool chain
- Simplification techniques polynomial bounded, but application until fixpoint takes long
- Simplification is limited in current SAT-solvers

Motivation for Parallel Preprocessing

- Formula simplification / preprocessing important part of SAT-Solving tool chain
- Simplification techniques polynomial bounded, but application until fixpoint takes long
- Simplification is limited in current SAT-solvers
- Research on parallel SAT-solving, but no parallel preprocessing

Motivation for Parallel Preprocessing

- Formula simplification / preprocessing important part of SAT-Solving tool chain
- Simplification techniques polynomial bounded, but application until fixpoint takes long
- Simplification is limited in current SAT-solvers
- Research on parallel SAT-solving, but no parallel preprocessing
- Parallel preprocessing without/less strict limits beneficial for solving hard formulas

Notation and Basic Concepts

- The set of variables occurring in a formula F: atoms(F).
- We assume an arbitrary, but fixed linear order on atoms(F).
- $F_l := \{C \in F \mid l \in C\}$
- The *neighbor variables* (neighbors) of v: atoms$(F_v \cup F_{\overline{v}})$

- A clause C *subsumes* a clause D iff $C \subseteq D$.
- If $C = \{x, a_1, \ldots, a_n\}$ and $D = \{\overline{x}, b_1, \ldots, b_m\}$ then $C \otimes_x D := \{a_1, \ldots, a_n, b_1, \cdots, b_m\}$.
- Resolution on sets: $F \otimes_x G := \{C \otimes_x D \mid C \in F, D \in G\}$

- Worker: part of an algorithm executed in parallel.

Section 2

SAT Preprocessing

Subsumption

Subsumption (CNF formulas F, G)
removes all clauses D from a Formula F,
where $\exists C \in G$: $C \subseteq D \wedge C \neq D$.

Strengthening / Self-Subsuming Resolution

C can *strengthen* D
iff $C \otimes_l D \subseteq D$.
iff C subsumes D except for one literal $l \in C$,
which occurs in D with the opposite sign.

Strengthening(CNF formulas F, G)
successively removes literals from F
by removing $\bar{l}$ from D,
if $C \in G$ can strengthen $D \in F$.

SubSimp

SubSimp (CNF formulas F, G)

```
Subsumption(F,G)
Strengthening(F,G)
H := {C | C ∈ F, changed}
Subsumption(F,H)
```

If the algorithms are executed in this way, a fixed point is reached. Since strengthening is not confluent, there is in general **no unique** fixed point.

Variable Elimination

F is satisfiable iff $(F \setminus (F_x \cup F_{\overline{x}})) \cup (F_x \otimes_x F_{\overline{x}})$ is satisfiable.

Variable Elimination

F is satisfiable iff $(F \setminus (F_x \cup F_{\overline{x}})) \cup (F_x \otimes_x F_{\overline{x}})$ is satisfiable.

$$F = \{\underbrace{\underbrace{\{a, \overline{x}\}, \{b, \overline{x}\}, \{d, \overline{x}\}}_{= F_{\overline{x}}}, \quad \underbrace{\{\overline{a}, \overline{b}, x\}, \{e, x\}}_{= F_x}}_{F_x \otimes_x F_{\overline{x}} = \{\{\overline{a}, \overline{b}, d\}, \{a, e\}, \{b, e\}, \{d, e\}\}}, \quad \{a, e, d\}\}$$

$$\begin{aligned} & (F \setminus (F_x \cup F_{\overline{x}})) \cup (F_x \otimes_x F_{\overline{x}}) \\ = \ & \{\{\overline{a}, \overline{b}, d\}, \{a, e\}, \{b, e\}, \{d, e\}, \{a, e, d\}\} \end{aligned}$$

Variable Elimination

F is satisfiable iff $(F \setminus (F_x \cup F_{\overline{x}})) \cup (F_x \otimes_x F_{\overline{x}})$ is satisfiable.

VariableElimination (CNF formula F)

$Q = \text{atoms}(F)$
do
    SubSimp(F,F)
    for $v \in Q$ **do**
        $Q := Q \setminus \{v\}$
        $S := F_v \otimes_v F_{\overline{v}}$
        if $|S| \leq |F_v| + |F_{\overline{v}}|$ **then**
            $F := F \setminus (F_v \cup F_{\overline{v}})$
            $F := F \cup S$
            SubSimp(F,S)
    update(Q)
while changed

Section 3

Parallel SAT Preprocessing

How can we parallelize these techniques?

(a) Divide the formula in disjoint parts.
Run the workers without synchronization.
  - Optimal graph bisection is $\mathcal{NP}$-hard.
  - Limits simplifications, if the necessary clauses are assigned to different workers.

How can we parallelize these techniques?

(a) Divide the formula in disjoint parts.
Run the workers without synchronization.
- Optimal graph bisection is $\mathcal{NP}$-hard.
- Limits simplifications, if the necessary clauses are assigned to different workers.

(b) Synchronize the workers with the help of locks.
- Synchronization overhead.
- Steps to construct a parallel algorithm:
  (i) Identify critical sections.
  (ii) Introduce a locking scheme.
  (iii) Check if deadlocks can occur.

Analysis for Parallel Variable Elimination

VariableElimination (CNF F)

```
Q = atoms(F)
do
    SubSimp(F,F)
    for v ∈ Q do
        Q := Q \ {v}
        S := F_v ⊗ F_v̄
        if |S| ≤ |F_v| + |F_v̄| then
            F := F \ (F_v ∪ F_v̄)
            F := F ∪ S
            SubSimp (F,S)
    update(Q)
while changed
```

Analysis for Parallel Variable Elimination

```
    VariableElimination (CNF F)
1   Q = atoms(F)
2   do
3       SubSimp(F,F)
4       for v ∈ Q do
5           Q := Q \ {v}
6           S := F_v ⊗ F_v̄
7           if |S| ≤ |F_v| + |F_v̄| then
8               F := F \ (F_v ∪ F_v̄)
9               F := F ∪ S
10              SubSimp (F,S)
11      update(Q)
12  while changed
```

- Lines 6-9: exclusive access to $F_v \cup F_{\overline{v}}$.

Analysis for Parallel Variable Elimination

```
VariableElimination (CNF F)
1  Q = atoms(F)
2  do
3    SubSimp(F,F)
4    for v ∈ Q do
5      Q := Q \ {v}
6      S := F_v ⊗ F_v̄
7      if |S| ≤ |F_v| + |F_v̄| then
8        F := F \ (F_v ∪ F_v̄)
9        F := F ∪ S
10       SubSimp (F,S)
11   update(Q)
12 while changed
```

- Lines 6-9: exclusive access to $F_v \cup F_{\overline{v}}$.
- Line 9: new clauses are created
  $\Longrightarrow$ synchronization of Clause Database (CDB).

Analysis for Parallel Variable Elimination

VariableElimination (CNF F)

```
1   Q = atoms(F)
2   do
3     SubSimp(F,F)
4     for v ∈ Q do
5       Q := Q \ {v}
6       S := F_v ⊗ F_v̄
7       if |S| ≤ |F_v| + |F_v̄| then
8         F := F \ (F_v ∪ F_v̄)
9         F := F ∪ S
10        SubSimp (F,S)
11    update(Q)
12  while changed
```

- Lines 6-9: exclusive access to $F_v \cup F_{\overline{v}}$.
- Line 9: new clauses are created
  $\Longrightarrow$ synchronization of Clause Database (CDB).
- Line 10: SubSimp executed in parallel to Variable Elimination.

Parallization concept for Parallel Variable Elimination

- Locks (in this order $\implies$ no deadlocks):
  variable locks, RW-Lock for CDB, lock per clause.

Parallization concept for Parallel Variable Elimination

- Locks (in this order $\implies$ no deadlocks):
  variable locks, RW-Lock for CDB, lock per clause.
- Accessing clauses: read-lock to CDB

Parallization concept for Parallel Variable Elimination

- Locks (in this order $\implies$ no deadlocks):
  variable locks, RW-Lock for CDB, lock per clause.
- Accessing clauses: read-lock to CDB
- Creating space reservation for new clauses: write-lock to CDB

Parallization concept for Parallel Variable Elimination

- Locks (in this order $\Longrightarrow$ no deadlocks):
  variable locks, RW-Lock for CDB, lock per clause.
- Accessing clauses: read-lock to CDB
- Creating space reservation for new clauses: write-lock to CDB
- Conditions for exclusive access to a clause:
  - Locking all variables of the clause **or**
  - Locking a variable of the clause and the clause itself

Parallization concept for Parallel Variable Elimination

- Locks (in this order $\implies$ no deadlocks):
  variable locks, RW-Lock for CDB, lock per clause.
- Accessing clauses: read-lock to CDB
- Creating space reservation for new clauses: write-lock to CDB
- Conditions for exclusive access to a clause:
  - Locking all variables of the clause **or**
  - Locking a variable of the clause and the clause itself
- Eliminating v:
  exclusive access to all clauses consisting of neighbors of v

Parallization concept for Parallel Variable Elimination

- Locks (in this order $\implies$ no deadlocks):
  variable locks, RW-Lock for CDB, lock per clause.
- Accessing clauses: read-lock to CDB
- Creating space reservation for new clauses: write-lock to CDB
- Conditions for exclusive access to a clause:
  - Locking all variables of the clause **or**
  - Locking a variable of the clause and the clause itself
- Eliminating v:
  exclusive access to all clauses consisting of neighbors of v

$F = \{\{a, b, v\}, \quad \{\overline{v}, c\}, \quad \{a, c\}\}$ atoms$(F_v \cup F_{\overline{v}}) = \{a, b, c, v\}$

Algorithm Sketch for Parallel Variable Elimination

ParallelVariableElimination (CNF F, $Q \subseteq \text{atoms}(F)$)

```
while Q ≠ ∅ do
    take v from Q
    determine neighbors: atoms(F_v ∪ F_v̄)
    lock neighbors
    check if neighbors of v changed
    simulate elimination of v
    if elimination of v beneficial then
        reserve memory for F_v ⊗_v F_v̄
        F := (F \ (F_v ∪ F_v̄)) ∪ (F_v ⊗_v F_v̄)
        add F_v ⊗_v F_v̄ to SubSimp-queue
    unlock neighbors
    SubSimp
```

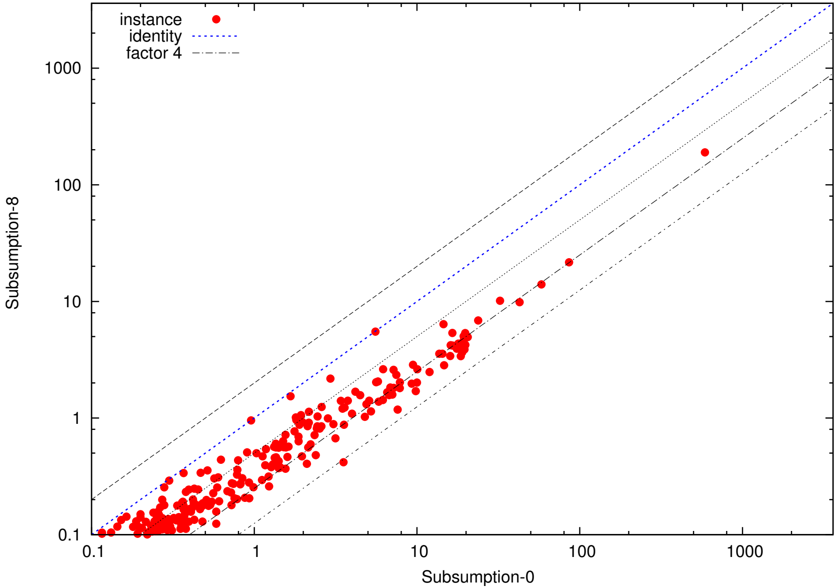
Section 4

Evaluation

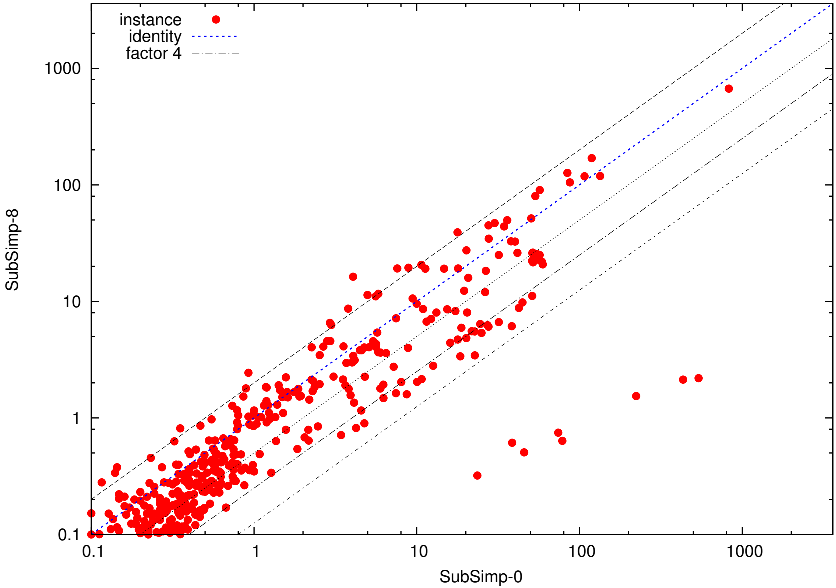
Benchmark Setup

- 880 Instances from SAT competition 2009, SAT Challange 2012 (+ "too hard"-submissions)
- No preprocessing limits
- Timeout of 3600 seconds
- AMD Opteron 6274
  (2.2 GHs, 16 Cores, 16MB L3-Cache per 8 Cores)
- Implementations are part of Coprocessor 3

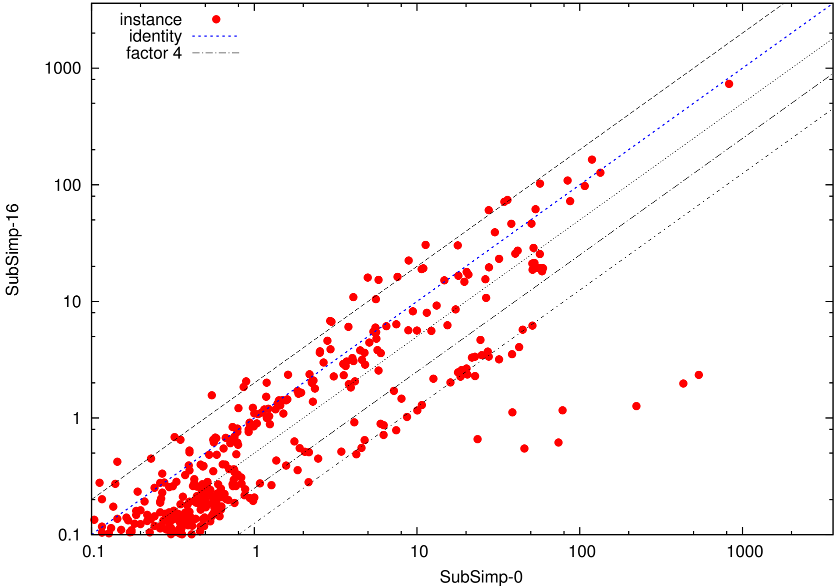
Subsumption - 8 Cores



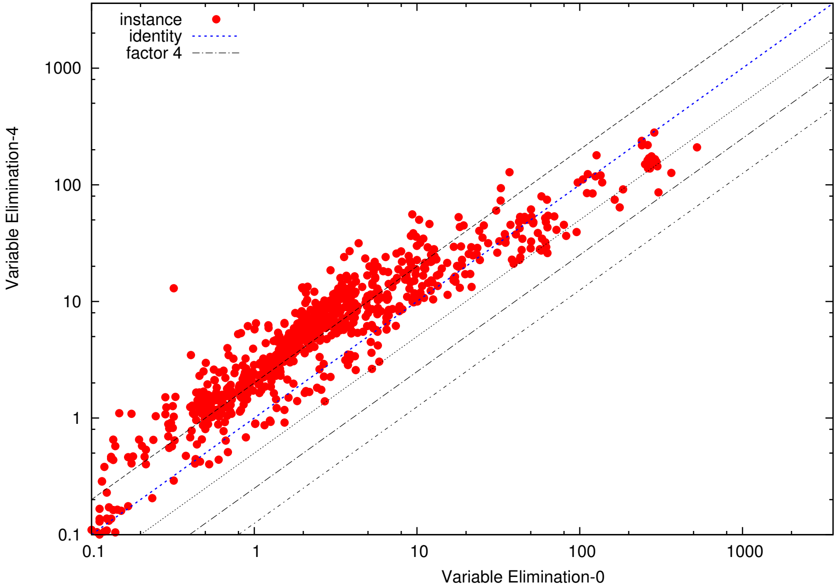
SubSimp - 8 Cores



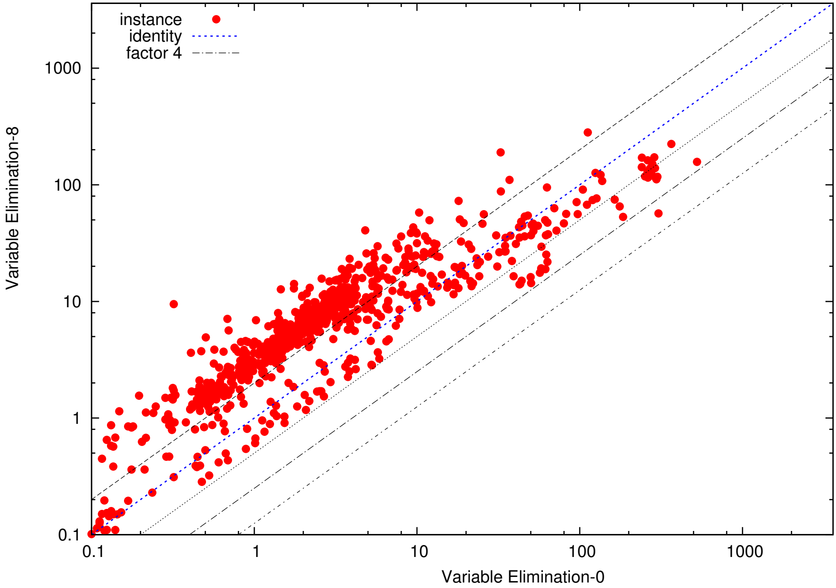
SubSimp - 16 Cores



Variable Elimination - 4 Cores



Variable Elimination - 8 Cores



Simplification Quality

	Subsumption				SubSimp				
Cores	0	1	8	16	0	1	4	8	16
T_{Simp}	1.8	2.5	**0.5**	1.8	7.7	8.7	5.6	4.0	**3.7**
SAT	240	240	**268**	263	246	254	262	**264**	263
UNSAT	229	227	259	**261**	222	245	257	**259**	256
total	469	467	**527**	524	468	499	519	**523**	519

	Variable Elimination						
Cores	0	R_0	1	4	8	R_8	16
T_{Simp}	19.8	18.2	27.6	20.4	19.6	**16.8**	28.8
SAT	255	**270**	249	**270**	264	269	262
UNSAT	252	283	257	**295**	287	284	286
total	507	553	506	**565**	551	553	548

Conclusion

- Variable elimination is one of the most important simplification techniques.
- We showed how variable elimination, subsumption and strengthening, can be parallelized with the help of a low-level locking approach.
- A speedup of almost 2 could be reached for variable elimination.
- For subsumption and strengthening
  an average speedup of 4 is obtained.
- Future work:
  - Influence of formula topology on the performance
    of parallel variable elimination.
  - Better understanding of strengthening.
  - Heuristics when to use parallel preprocessing
  - Parallelization of other techniques

Thank you for your attention!