Parallel MUS Extraction

Anton Belov[1], Norbert Manthey[2], and Joao Marques-Silva[1,3]

[1]Complex and Adaptive Systems Laboratory, University College Dublin, Ireland
[2]Institute of Artificial Intelligence, Technische Universität Dresden, Germany
[3]IST/INESC-ID, Lisbon, Portugal

SAT 2013
July 10, 2013
Helsinki, Finland

Minimal Unsatisfiable Subformulas (MUSes)

$$\begin{array}{ccc} C_1 & C_2 & C_3 \\ \boxed{(p)} & \boxed{(q)} & \boxed{(\neg p \vee \neg q)} \end{array}$$

$M = \{C_1, C_2, C_3\}$ is UNSAT

Minimal Unsatisfiable Subformulas (MUSes)

$$\underset{\boxed{(p)}}{C_1} \quad \underset{\boxed{(q)}}{C_2} \quad \underset{\boxed{(\neg p \vee \neg q)}}{C_3}$$

$M = \{C_1, C_2, C_3\}$ is UNSAT, and $\forall C \in M$, $M \setminus \{C\}$ is SAT.

Minimal Unsatisfiable Subformulas (MUSes)

$$\begin{array}{ccc} C_1 & C_2 & C_3 \\ \boxed{(p)} & \boxed{(q)} & \boxed{(\neg p \vee \neg q)} \end{array}$$

$M = \{C_1, C_2, C_3\}$ is *minimal unsatisfiable (MU)*.

Minimal Unsatisfiable Subformulas (MUSes)

C_1	C_2	C_3	C_4	C_5	C_6
(p)	(q)	$(\neg p \vee \neg q)$	$(\neg p \vee r)$	$(p \vee q)$	$(\neg q \vee \neg r)$

$M = \{C_1, C_2, C_3\}$ is *minimal unsatisfiable (MU)*.
$F = \{C_1, \ldots, C_6\}$ is UNSAT, but not MU.

Minimal Unsatisfiable Subformulas (MUSes)

C_1	C_2	C_3	C_4	C_5	C_6
(p)	(q)	$(\neg p \vee \neg q)$	$(\neg p \vee r)$	$(p \vee q)$	$(\neg q \vee \neg r)$

$M = \{C_1, C_2, C_3\}$ is *minimal unsatisfiable (MU)*.
$F = \{C_1, \ldots, C_6\}$ is UNSAT, but not MU.
M is a *minimal unsatisfiable subformula (MUS)* of F.

Minimal Unsatisfiable Subformulas (MUSes)

C_1	C_2	C_3	C_4	C_5	C_6
(p)	(q)	$(\neg p \vee \neg q)$	$(\neg p \vee r)$	$(p \vee q)$	$(\neg q \vee \neg r)$

$M = \{C_1, C_2, C_3\}$ is *minimal unsatisfiable (MU)*.
$F = \{C_1, \ldots, C_6\}$ is UNSAT, but not MU.
M is a *minimal unsatisfiable subformula (MUS)* of F.

Applications

Identification and repair of sources of inconsistency:

- circuit error diagnosis; error localization in product configuration

Identification of important/relevant features of systems:

- automatic abstraction in model checking
- environmental assumptions in formal equivalence checking

Complexity Decision: D^P-complete. Function: $\in FP^{NP}$

MUS Extraction

Based on detection of *necessary* (or, *transition*) clauses:

- $C \in F$ is *necessary* for F if $F \in$ UNSAT and $F \setminus \{C\} \in$ SAT.
- If C is necessary for F, then C is in every MUS of F.

MUS Extraction

Based on detection of *necessary* (or, *transition*) clauses:

- $C \in F$ is *necessary* for F if $F \in \text{UNSAT}$ and $F \setminus \{C\} \in \text{SAT}$.
- If C is necessary for F, then C is in every MUS of F.

```
Input ↦ Output: F ∈ UNSAT ↦ M ∈ MUS(F)
⟨F_w, M⟩ ← ⟨F, ∅⟩                    // Working formula, MUS under-approx.
while M ≠ F_w do                      // Inv:  M ⊆ F, and ∀C ∈ M is nec. for F_w
    C ← PickClause(F_w)
    st = SAT(F_w \ {C})                          // Test if C is nec. for F_w
    if st = true then                            // If SAT, C is nec. for F_w
        M ← M ∪ {C}
        RMR(F_w, M, τ)          // Model rotation: find more nec. clauses
    else
        F_w ← F_w \ {C}
return M                                                        // M ∈ MUS(F)
```

- Hybrid MUS extraction algorithm [Marques-Silva&Lynce'11]

MUS Extraction

Based on detection of *necessary* (or, *transition*) clauses:

- $C \in F$ is *necessary* for F if $F \in \text{UNSAT}$ and $F \setminus \{C\} \in \text{SAT}$.
- If C is necessary for F, then C is in every MUS of F.

Input $\mapsto$ **Output**: $F \in \text{UNSAT} \mapsto M \in \text{MUS}(F)$

```
⟨F_w, M⟩ ← ⟨F, ∅⟩                        // Working formula, MUS under-approx.
while M ≠ F_w do                         // Inv:  M ⊆ F, and ∀C ∈ M is nec. for F_w
    C ← PickClause(F_w)
    (st, U, τ) = SAT(F_w \ {C})          // U - unsat. core, τ - model
    if st = true then                    // If SAT, C is nec. for F_w
        M ← M ∪ {C}
        RMR(F_w, M, τ)                   // Model rotation: find more nec. clauses
    else
        F_w ← U    // Clause-set refinement: discard non-core clauses
return M                                 // M ∈ MUS(F)
```

- Hybrid MUS extraction algorithm [Marques-Silva&Lynce'11]

MUS Extraction: opportunities for parallelization

Input $\mapsto$ **Output**: $F \in \mathsf{UNSAT} \mapsto M \in \mathsf{MUS}(F)$
$\langle F_w, M\rangle \leftarrow \langle F, \emptyset\rangle$ `// Working formula, MUS under-approx.`
while $M \neq F_w$ **do** `// Inv:` $M \subseteq F$, `and` $\forall C \in M$ `is nec. for` F_w
    $C \leftarrow \texttt{PickClause}(F_w)$
    $(\mathsf{st}, U, \tau) = \texttt{SAT}(F_w \setminus \{C\})$ `//` U `- unsat. core,` τ `- model`
    if st = true **then** `// If SAT,` C `is nec. for` F_w
        $M \leftarrow M \cup \{C\}$
        $\texttt{RMR}(F_w, M, \tau)$ `// Model rotation: find more nec. clauses`
    else
        $F_w \leftarrow U$ `// Clause-set refinement: discard non-core clauses`
return M `//` $M \in \mathsf{MUS}(F)$

- Hybrid MUS extraction algorithm [Marques-Silva&Lynce'11]

MUS Extraction: opportunities for parallelization

1. Parallelize each SAT call

Input $\mapsto$ **Output**: $F \in \text{UNSAT} \mapsto M \in \text{MUS}(F)$
$\langle F_w, M \rangle \leftarrow \langle F, \emptyset \rangle$ `// Working formula, MUS under-approx.`
while $M \neq F_w$ **do** `// Inv:` $M \subseteq F$`, and` $\forall C \in M$ `is nec. for` F_w
    $C \leftarrow \texttt{PickClause}(F_w)$
    $(\text{st}, U, \tau) = \texttt{SAT}(F_w \setminus \{C\})$ `//` U `- unsat. core,` τ `- model`
    if st = true **then** `// If SAT,` C `is nec. for` F_w
        $M \leftarrow M \cup \{C\}$
        $\texttt{RMR}(F_w, M, \tau)$ `// Model rotation: find more nec. clauses`
    else
        $F_w \leftarrow U$ `// Clause-set refinement: discard non-core clauses`
return M `//` $M \in \text{MUS}(F)$

- Hybrid MUS extraction algorithm [Marques-Silva&Lynce'11]

MUS Extraction: opportunities for parallelization

1. Parallelize each SAT call
2. Parallelize the main loop, i.e. test multiple clauses

```
Input ↦ Output: F ∈ UNSAT ↦ M ∈ MUS(F)
⟨F_w, M⟩ ← ⟨F, ∅⟩                        // Working formula, MUS under-approx.
while M ≠ F_w do               // Inv:  M ⊆ F, and ∀C ∈ M is nec. for F_w
    C ← PickClause(F_w)
    (st, U, τ) = SAT(F_w \ {C})                // U - unsat. core, τ - model
    if st = true then                          // If SAT, C is nec. for F_w
        M ← M ∪ {C}
        RMR(F_w, M, τ)           // Model rotation: find more nec. clauses
    else
        F_w ← U   // Clause-set refinement: discard non-core clauses
return M                                                   // M ∈ MUS(F)
```

- Hybrid MUS extraction algorithm [Marques-Silva&Lynce'11]

MUS Extraction: opportunities for parallelization

1. Parallelize each SAT call
2. Parallelize the main loop, i.e. test multiple clauses
3. Parallel portfolio of MUS extractors

```
Input ↦ Output: F ∈ UNSAT ↦ M ∈ MUS(F)
⟨F_w, M⟩ ← ⟨F, ∅⟩                        // Working formula, MUS under-approx.
while M ≠ F_w do               // Inv:  M ⊆ F, and ∀C ∈ M is nec. for F_w
    C ← PickClause(F_w)
    (st, U, τ) = SAT(F_w \ {C})                // U - unsat. core, τ - model
    if st = true then                          // If SAT, C is nec. for F_w
        M ← M ∪ {C}
        RMR(F_w, M, τ)          // Model rotation: find more nec. clauses
    else
        F_w ← U  // Clause-set refinement: discard non-core clauses
return M                                                    // M ∈ MUS(F)
```

- Hybrid MUS extraction algorithm [Marques-Silva&Lynce'11]

MUS Extraction: opportunities for parallelization

1. Parallelize each SAT call
2. Parallelize the main loop, i.e. test multiple clauses $\leftarrow$ this talk/paper
3. Parallel portfolio of MUS extractors

Input $\mapsto$ **Output**: $F \in \text{UNSAT} \mapsto M \in \text{MUS}(F)$
$\langle F_w, M \rangle \leftarrow \langle F, \emptyset \rangle$ // Working formula, MUS under-approx.
while $M \neq F_w$ **do** // Inv: $M \subseteq F$, and $\forall C \in M$ is nec. for F_w
  $C \leftarrow \texttt{PickClause}(F_w)$
  $(\text{st}, U, \tau) = \texttt{SAT}(F_w \setminus \{C\})$ // U - unsat. core, τ - model
  if st = true **then** // If SAT, C is nec. for F_w
    $M \leftarrow M \cup \{C\}$
    $\texttt{RMR}(F_w, M, \tau)$ // Model rotation: find more nec. clauses
  else
    $F_w \leftarrow U$ // Clause-set refinement: discard non-core clauses
return M // $M \in \text{MUS}(F)$

- Hybrid MUS extraction algorithm [Marques-Silva&Lynce'11]

Parallelizing the main loop

$\langle F_w, M \rangle \leftarrow \langle F, \emptyset \rangle$ // Working formula, MUS under-approx.
while $M \neq F_w$ **do**
  $\{C_1, C_2\} \leftarrow \texttt{PickClauses}(F_w)$
  $(\mathrm{st}_1, U_1, \tau_1) = \texttt{SAT}(F_w \setminus \{C_1\}) \quad || \quad (\mathrm{st}_2, U_2, \tau_2) = \texttt{SAT}(F_w \setminus \{C_2\})$
  sleepUntilFinished() // Wait for both threads to finish
  if $\mathrm{st}_1 = \mathrm{true}$ **and** $\mathrm{st}_2 = \mathrm{true}$ **then**
    $M \leftarrow M \cup \{C_1, C_2\}$
    $\texttt{RMR}(F_w, M, \tau_1)$; $\texttt{RMR}(F_w, M, \tau_2)$
  else if $\mathrm{st}_1 = \mathrm{true}$ **and** $\mathrm{st}_2 = \mathrm{false}$ **then**
    $M \leftarrow M \cup \{C_1\}$
    $\texttt{RMR}(F_w, M, \tau_1)$; $F_w \leftarrow U_2$
  else if $\mathrm{st}_1 = \mathrm{false}$ **and** $\mathrm{st}_2 = \mathrm{true}$ **then**
    $M \leftarrow M \cup \{C_2\}$
    $\texttt{RMR}(F_w, M, \tau_2)$; $F_w \leftarrow U_1$
  else
    $F_w \leftarrow \texttt{PickCore}(U_1, U_2)$ // Pick one of the cores
return M // $M \in \mathrm{MUS}(F)$

Parallelizing the main loop

```
⟨F_w, M⟩ ← ⟨F, ∅⟩                                   // Working formula, MUS under-approx.
while M ≠ F_w do
    {C_1, C_2} ← PickClauses(F_w)
    (st_1, U_1, τ_1) = SAT(F_w \ {C_1})  ||  (st_2, U_2, τ_2) = SAT(F_w \ {C_2})
    sleepUntilFinished()                            // Wait for both threads to finish
    if st_1 = true and st_2 = true then
        M ← M ∪ {C_1, C_2}
        RMR(F_w, M, τ_1) ; RMR(F_w, M, τ_2)
    else if st_1 = true and st_2 = false then
        M ← M ∪ {C_1}
        RMR(F_w, M, τ_1) ; F_w ← U_2
    else if st_1 = false and st_2 = true then
        M ← M ∪ {C_2}
        RMR(F_w, M, τ_2) ; F_w ← U_1
    else
        F_w ← PickCore(U_1, U_2)                    // Pick one of the cores
return M                                            // M ∈ MUS(F)
```

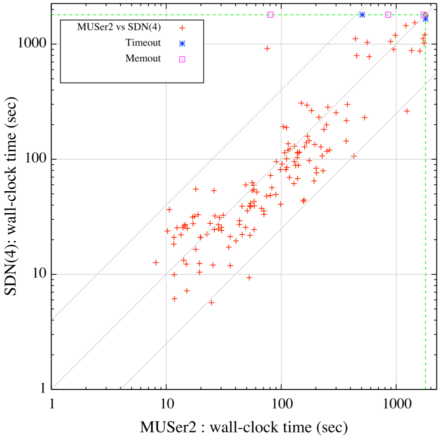
Parallelizing the main loop

$\langle F_w, M\rangle \leftarrow \langle F, \emptyset\rangle$ // Working formula, MUS under-approx.
while $M \neq F_w$ **do**
  $\{C_1, C_2\} \leftarrow \texttt{PickClauses}(F_w)$
  $(\text{st}_1, U_1, \tau_1) = \texttt{SAT}(F_w \setminus \{C_1\}) \quad || \quad (\text{st}_2, U_2, \tau_2) = \texttt{SAT}(F_w \setminus \{C_2\})$
  sleepUntilFinished() // Wait for both threads to finish
  if $\text{st}_1 = \text{true}$ **and** $\text{st}_2 = \text{true}$ **then**
    $M \leftarrow M \cup \{C_1, C_2\}$
    $\texttt{RMR}(F_w, M, \tau_1)$; $\texttt{RMR}(F_w, M, \tau_2)$
  else if $\text{st}_1 = \text{true}$ **and** $\text{st}_2 = \text{false}$ **then**
    $M \leftarrow M \cup \{C_1\}$
    $\texttt{RMR}(F_w, M, \tau_1)$; $F_w \leftarrow U_2$
  else if $\text{st}_1 = \text{false}$ **and** $\text{st}_2 = \text{true}$ **then**
    $M \leftarrow M \cup \{C_2\}$
    $\texttt{RMR}(F_w, M, \tau_2)$; $F_w \leftarrow U_1$
  else
    $F_w \leftarrow \texttt{PickCore}(U_1, U_2)$ // Pick one of the cores
return M // $M \in \text{MUS}(F)$

Parallelizing the main loop

$\langle F_w, M\rangle \leftarrow \langle F, \emptyset\rangle$ // Working formula, MUS under-approx.
while $M \neq F_w$ **do**
  $\{C_1, C_2\} \leftarrow \texttt{PickClauses}(F_w)$
  $(\mathrm{st}_1, U_1, \tau_1) = \texttt{SAT}(F_w \setminus \{C_1\}) \quad || \quad (\mathrm{st}_2, U_2, \tau_2) = \texttt{SAT}(F_w \setminus \{C_2\})$
  `sleepUntilFinished()` // Wait for both threads to finish
  if $\mathrm{st}_1 = \mathrm{true}$ **and** $\mathrm{st}_2 = \mathrm{true}$ **then**
    $M \leftarrow M \cup \{C_1, C_2\}$
    $\texttt{RMR}(F_w, M, \tau_1)$; $\texttt{RMR}(F_w, M, \tau_2)$
  else if $\mathrm{st}_1 = \mathrm{true}$ **and** $\mathrm{st}_2 = \mathrm{false}$ **then**
    $M \leftarrow M \cup \{C_1\}$
    $\texttt{RMR}(F_w, M, \tau_1)$; $F_w \leftarrow U_2$
  else if $\mathrm{st}_1 = \mathrm{false}$ **and** $\mathrm{st}_2 = \mathrm{true}$ **then**
    $M \leftarrow M \cup \{C_2\}$
    $\texttt{RMR}(F_w, M, \tau_2)$; $F_w \leftarrow U_1$
  else
    $F_w \leftarrow \texttt{PickCore}(U_1, U_2)$ // Pick one of the cores
return M // $M \in \mathrm{MUS}(F)$

Parallelizing the main loop



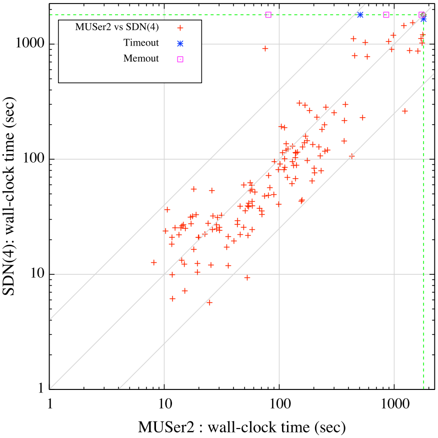
175 benchs, MUS track, SC'11.

wall-clock limit 1800 sec

memory limit 16 GB.

	#sol.	avg.time
(x) sequential	144	186.46
(y) parallel, 4 thr.	143	154.93

Parallelizing the main loop



175 benchs, MUS track, SC'11.

wall-clock limit 1800 sec

memory limit 16 GB.

	#sol.	avg.time
(x) sequential	144	186.46
(y) parallel, 4 thr.	143	154.93

Shortcomings

(i) Threads are under-utilized because of synchronization.

(ii) No *communication*, i.e. exchange of learned clauses between threads.

Parallelizing the main loop: de-synchronizing

Technicalities

"Outdated" SAT outcomes are OK — if C is necessary for F_w, it is also necessary for $F'_w \subset F_w$.

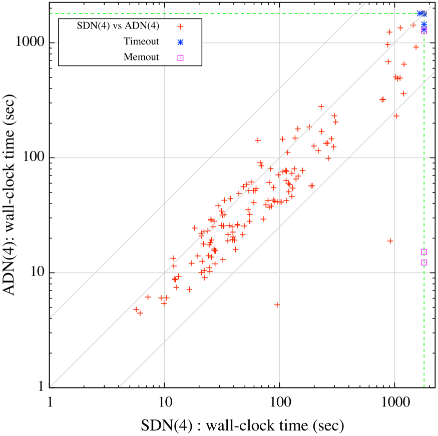
"Outdated" UNSAT cores might be not — test if $U \subseteq F_w$, if not drop it.

Parallelizing the main loop: de-synchronizing

Technicalities

"Outdated" SAT outcomes are OK — if C is necessary for F_w, it is also necessary for $F'_w \subset F_w$.

"Outdated" UNSAT cores might be not — test if $U \subseteq F_w$, if not drop it.



175 benchs, MUS track, SC'11.

wall-clock limit 1800 sec

memory limit 16 GB.

	#sol.	avg.time
(x) parallel, 4 thr. synchronous	143	154.93
(y) parallel, 4 thr. asynchronous	146	126.45

Parallelizing the main loop: communication

Would like to exchange clauses between threads

Problem: threads work on *different* formulas $\rightarrow$ clauses learned by one might be not valid for another.

Parallelizing the main loop: communication

Would like to exchange clauses between threads

Problem: threads work on *different* formulas $\rightarrow$ clauses learned by one might be not valid for another.

$$F \quad \overset{C_1}{\boxed{(p)}} \; \overset{C_2}{\boxed{(q)}} \; \overset{C_3}{\boxed{(\neg p \vee \neg q)}}$$

Thread 1: solves $SAT(F \setminus \{C_1\})$, derives $(\neg p)$.
Thread 2: works on $SAT(F \setminus \{C_2\})$, receives $(\neg p)$, returns UNSAT.

Parallelizing the main loop: communication

Would like to exchange clauses between threads

Problem: threads work on *different* formulas $\rightarrow$ clauses learned by one might be not valid for another.

$$F \quad \overset{C_1}{\boxed{(p)}} \; \overset{C_2}{\boxed{(q)}} \; \overset{C_3}{\boxed{(\neg p \vee \neg q)}}$$

Thread 1: solves $SAT(F \setminus \{C_1\})$, derives $(\neg p)$.
Thread 2: works on $SAT(F \setminus \{C_2\})$, receives $(\neg p)$, returns UNSAT.

Solution: assumption-based, incremental SAT [Eén, Sörensson, ENTCS 2003]

Note: most modern MUS extractors use assumption-based incremental SAT anyway.

Assumption-based incremental SAT [Eén, Sörensson, ENTCS 2003]

SAT solver interface

add($\{C_1, \ldots, C_n\}$) — add clauses $C_1, \ldots, C_n$ to the SAT solver.

solve($\{l_1, \ldots, l_k\}$) — determine the satisfiability of the current set of clauses under a partial assignment defined by literals $\{l_1, \ldots, l_k\}$.

Assumption-based incremental SAT [Eén, Sörensson, ENTCS 2003]

SAT solver interface

add($\{C_1, \ldots, C_n\}$) — add clauses $C_1, \ldots, C_n$ to the SAT solver.

solve($\{l_1, \ldots, l_k\}$) — determine the satisfiability of the current set of clauses under a partial assignment defined by literals $\{l_1, \ldots, l_k\}$.

$$F \qquad \overset{C_1}{\boxed{(p)}}\ \overset{C_2}{\boxed{(q)}}\ \overset{C_3}{\boxed{(\neg p \vee \neg q)}}\ \overset{C_4}{\boxed{(p \vee q)}}$$

$$F_A \qquad \overset{C_1}{\boxed{(a_1 \vee p)}}\ \overset{C_2}{\boxed{(a_2 \vee q)}}\ \overset{C_3}{\boxed{(a_3 \vee \neg p \vee \neg q)}}\ \overset{C_4}{\boxed{(a_4 \vee p \vee q)}}$$

Assumption-based incremental SAT [Eén, Sörensson, ENTCS 2003]

SAT solver interface

add($\{C_1, \ldots, C_n\}$) — add clauses $C_1, \ldots, C_n$ to the SAT solver.

solve($\{l_1, \ldots, l_k\}$) — determine the satisfiability of the current set of clauses under a partial assignment defined by literals $\{l_1, \ldots, l_k\}$.

	C_1	C_2	C_3	C_4
F	(p)	(q)	$(\neg p \vee \neg q)$	$(p \vee q)$
F_A	$(a_1 \vee p)$	$(a_2 \vee q)$	$(a_3 \vee \neg p \vee \neg q)$	$(a_4 \vee p \vee q)$

To test $F \setminus \{C_1\}$: add(F_A); solve($\{a_1, \neg a_2, \neg a_3, \neg a_4\}$) $\rightarrow$ SAT, model

To test $F \setminus \{C_4\}$: add(F_A); solve($\{\neg a_1, \neg a_2, \neg a_3, a_4\}$) $\rightarrow$ UNSAT, core

Assumption-based incremental SAT [Eén, Sörensson, ENTCS 2003]

SAT solver interface

add($\{C_1, \ldots, C_n\}$) — add clauses $C_1, \ldots, C_n$ to the SAT solver.

solve($\{l_1, \ldots, l_k\}$) — determine the satisfiability of the current set of clauses under a partial assignment defined by literals $\{l_1, \ldots, l_k\}$.

	C_1	C_2	C_3	C_4
F	$\boxed{(p)}$	$\boxed{(q)}$	$\boxed{(\neg p \vee \neg q)}$	$\boxed{(p \vee q)}$

	C_1	C_2	C_3	C_4
F_A	$\boxed{(a_1 \vee p)}$	$\boxed{(a_2 \vee q)}$	$\boxed{(a_3 \vee \neg p \vee \neg q)}$	$\boxed{(a_4 \vee p \vee q)}$

To test $F \setminus \{C_1\}$: add(F_A); solve($\{a_1, \neg a_2, \neg a_3, \neg a_4\}$) $\rightarrow$ SAT, model

To test $F \setminus \{C_4\}$: add(F_A); solve($\{\neg a_1, \neg a_2, \neg a_3, a_4\}$) $\rightarrow$ UNSAT, core

Note: learned clauses are entailed by *input* clauses — can be exchanged.

Assumption-based incremental SAT [Eén, Sörensson, ENTCS 2003]

SAT solver interface

add($\{C_1, \ldots, C_n\}$) — add clauses $C_1, \ldots, C_n$ to the SAT solver.

solve($\{l_1, \ldots, l_k\}$) — determine the satisfiability of the current set of clauses under a partial assignment defined by literals $\{l_1, \ldots, l_k\}$.

	C_1	C_2	C_3	C_4
F	(p)	(q)	$(\neg p \vee \neg q)$	$(p \vee q)$
F_A	$(a_1 \vee p)$	$(a_2 \vee q)$	$(a_3 \vee \neg p \vee \neg q)$	$(a_4 \vee p \vee q)$

To test $F \setminus \{C_1\}$: add(F_A); solve($\{a_1, \neg a_2, \neg a_3, \neg a_4\}$) $\rightarrow$ SAT, model

To test $F \setminus \{C_4\}$: add(F_A); solve($\{\neg a_1, \neg a_2, \neg a_3, a_4\}$) $\rightarrow$ UNSAT, core

Note: learned clauses are entailed by *input* clauses — can be exchanged.

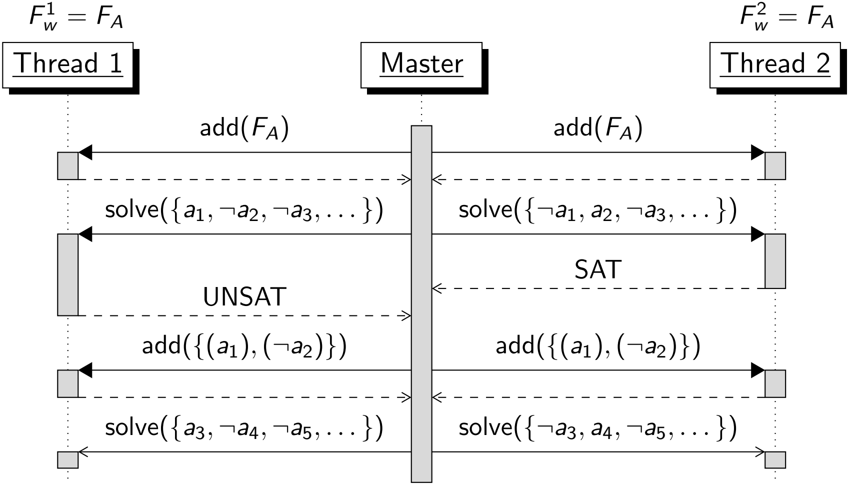
To "remove" C_4 from F_A : add({ (a_4) }).

To *finalize* C_1 in F_A : add({ $(\neg a_1)$ }).

Note: there is another approach [Marques-Silva, Sakallah, FTCS 1997; Nadel, Ryvchin, SAT 2012]

Incremental SAT and Parallel MUS Extraction (sync)

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

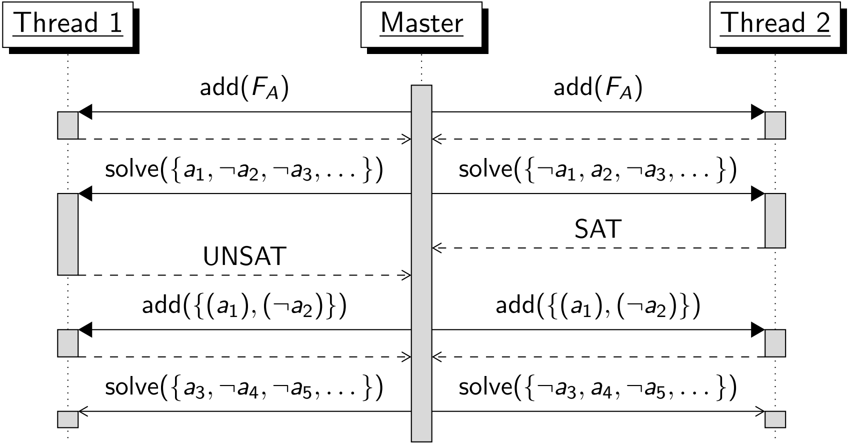


Incremental SAT and Parallel MUS Extraction (sync)

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

$F_w^1 = F_A \cup \{(a_1), (\neg a_2)\}$

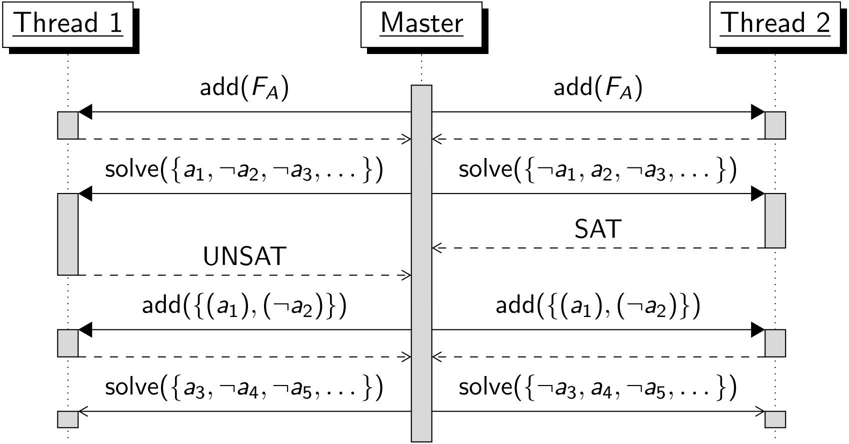
$F_w^2 = F_A \cup \{(a_1), (\neg a_2)\}$



Incremental SAT and Parallel MUS Extraction (sync)

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

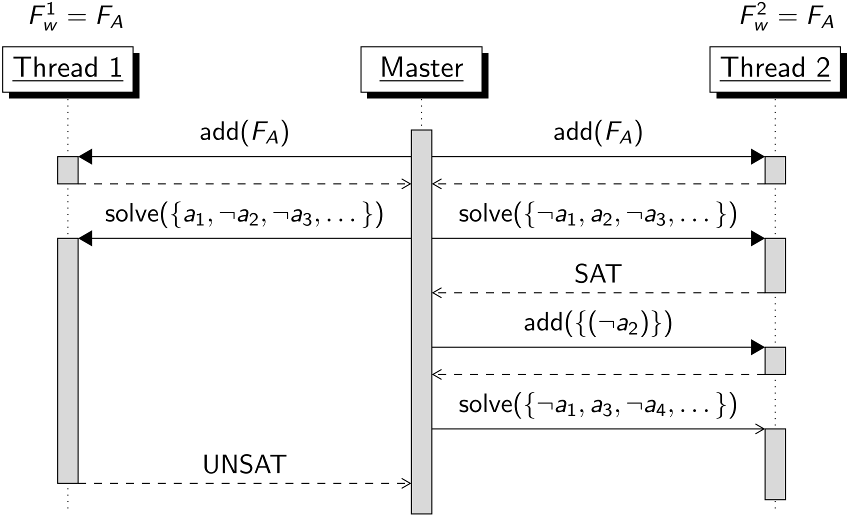
$F_w^1 = F_A \cup \{(a_1), (\neg a_2)\}$ $F_w^2 = F_A \cup \{(a_1), (\neg a_2)\}$



Threads always work on the *same* formula $\rightarrow$ unrestricted clause exchange.

Incremental SAT and Parallel MUS Extraction (async)

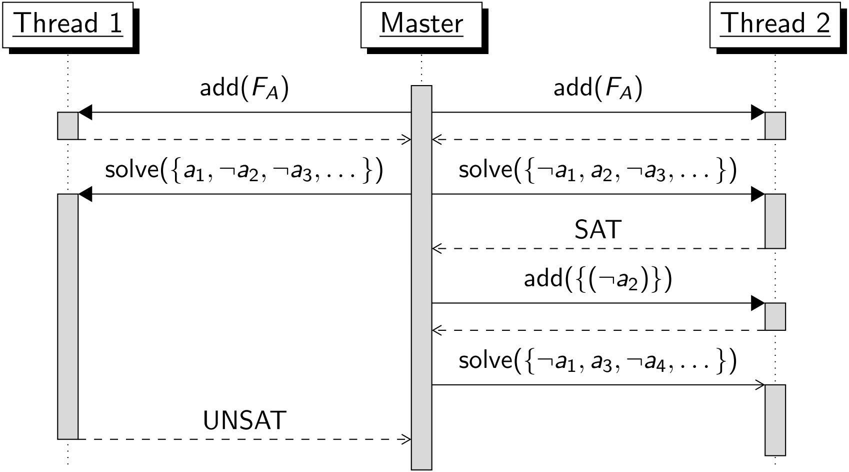
$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$



Incremental SAT and Parallel MUS Extraction (async)

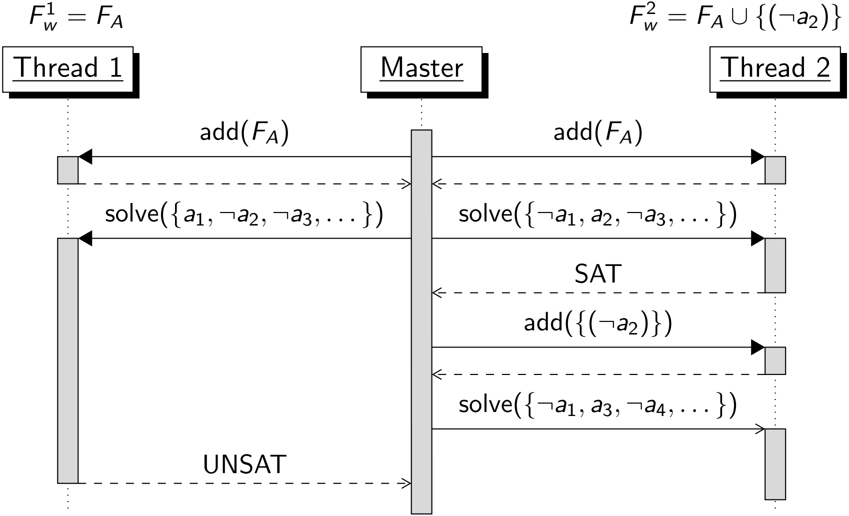
$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

$F_w^1 = F_A$ $F_w^2 = F_A \cup \{(\neg a_2)\}$



Incremental SAT and Parallel MUS Extraction (async)

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$



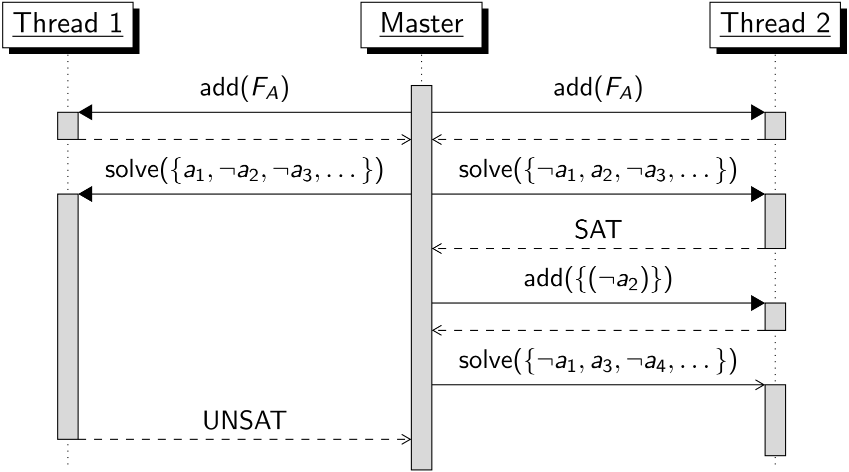
Threads work on different formulas: Thread 1 $\rightarrow$ Thread 2 is ok.

Incremental SAT and Parallel MUS Extraction (async)

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

$F_w^1 = F_A$

$F_w^2 = F_A \cup \{(\neg a_2)\}$



Threads work on different formulas: Thread 2 $\rightarrow$ Thread 1 ?

Soundness of “back” communication

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

Thread 1 (“behind”): $F_w^1 = F_A$, solve$(\{a_1, \neg a_2, \neg a_3, \dots\})$

Thread 2 (“ahead”): $F_w^2 = F_A \cup \{(\neg a_2)\}$, solve$(\{\neg a_1, a_3, \neg a_4, \dots\})$

Soundness of "back" communication

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

Thread 1 ("behind"): $F_w^1 = F_A$, solve$(\{a_1, \neg a_2, \neg a_3, \dots\})$

Thread 2 ("ahead"): $F_w^2 = F_A \cup \{(\neg a_2)\}$, solve$(\{\neg a_1, a_3, \neg a_4, \dots\})$

C – a clause learned by Thread 2. We have $F_A \cup \{(\neg a_2)\} \vDash C$.

C is not entailed by F_A, but since Thread 1 is solving under assumption $\neg a_2$, it is valid for the duration of the call.

Before the next call $(\neg a_2)$ will be added to Thread 1 by the Master, and C will be again entailed by the input clauses.

Soundness of "back" communication

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \ldots\}$$

Thread 1 ("behind"): $F_w^1 = F_A$, solve$(\{a_1, \neg a_2, \neg a_3, \ldots\})$

Thread 2 ("ahead"): $F_w^2 = F_A \cup \{(a_2)\}$, solve$(\{\neg a_1, a_3, \neg a_4, \ldots\})$

Soundness of "back" communication

$$F_A = \{(a_1 \vee C_1), (a_2 \vee C_2), (a_3 \vee C_3), (a_4 \vee C_4), \dots\}$$

Thread 1 ("behind"): $F_w^1 = F_A$, solve$(\{a_1, \neg a_2, \neg a_3, \dots\})$

Thread 2 ("ahead"): $F_w^2 = F_A \cup \{(a_2)\}$, solve$(\{\neg a_1, a_3, \neg a_4, \dots\})$

C – a clause learned by Thread 2.

Since a_2 appears only positively in F_A, no clause with a_2 will participate in the conflict. So, $F_A \vDash C$, and C can be used by Thread 1.

Soundness of "back" communication

The devil is in the details (and the details are in the paper)

Soundness of "back" communication

The devil is in the details (and the details are in the paper)

In the presence of model rotation and clause set refinement a worker may become "redundant".

Redundant workers *must* be aborted to ensure soundness.

Soundness of “back” communication

The devil is in the details (and the details are in the paper)

In the presence of model rotation and clause set refinement a worker may become “redundant”.

Redundant workers *must* be aborted to ensure soundness.

Workers that are “behind” may return a subset of UNSAT core — the Master can handle this with no overhead.

Soundness of “back” communication

The devil is in the details (and the details are in the paper)

In the presence of model rotation and clause set refinement a worker may become “redundant”.

Redundant workers *must* be aborted to ensure soundness.

Workers that are “behind” may return a subset of UNSAT core — the Master can handle this with no overhead.

Formal description of the algorithm and the correctness proof are in the paper.

Soundness of "back" communication

The devil is in the details (and the details are in the paper)

In the presence of model rotation and clause set refinement a worker may become "redundant".

Redundant workers *must* be aborted to ensure soundness.

Workers that are "behind" may return a subset of UNSAT core — the Master can handle this with no overhead.

Formal description of the algorithm and the correctness proof are in the paper.

Bottom line: *unrestricted* communication is possible — due to the assumption-based incremental SAT.

Improving communication

Would like to exchange *promising* clauses only.

- Restrict clause size (def: ≤ 10)
- Restrict clause LBD (def: ≤ 5)
- Optionally: change the limits dynamically
- Initialize ("bump") activity of received clauses.

Improving communication

Would like to exchange *promising* clauses only.

- Restrict clause size (def: ≤ 10)
- Restrict clause LBD (def: ≤ 5)
- Optionally: change the limits dynamically
- Initialize ("bump") activity of received clauses.

Important observation: assumptions are "second-class" citizens

A clause $(a_1 \vee \cdots \vee a_k \vee x)$ is *essentially* a unit clause. But might be either too long, or have a high LBD (each assumption has its own level).

Improving communication

Would like to exchange *promising* clauses only.

- Restrict clause size (def: ≤ 10)
- Restrict clause LBD (def: ≤ 5)
- Optionally: change the limits dynamically
- Initialize ("bump") activity of received clauses.

Important observation: assumptions are "second-class" citizens

A clause $(a_1 \vee \cdots \vee a_k \vee x)$ is *essentially* a unit clause. But might be either too long, or have a high LBD (each assumption has its own level).

Assumption "protection": ignore assumptions when computing the values for filters.

Improving communication

Would like to exchange *promising* clauses only.

- Restrict clause size (def: ≤ 10)
- Restrict clause LBD (def: ≤ 5)
- Optionally: change the limits dynamically
- Initialize ("bump") activity of received clauses.

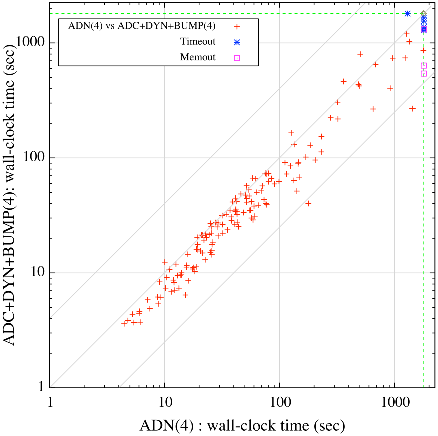
Important observation: assumptions are "second-class" citizens

A clause $(a_1 \vee \cdots \vee a_k \vee x)$ is *essentially* a unit clause. But might be either too long, or have a high LBD (each assumption has its own level).

Assumption "protection": ignore assumptions when computing the values for filters.

Note: a good idea for non-parallel MUS extraction as well [Audemard, Lagniez, Simon, SAT 2013] (tomorrow morning).

Parallelizing the main loop: communication



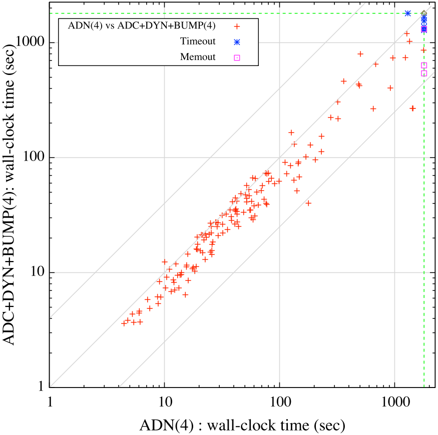
175 benchs, MUS track, SC'11.

wall-clock limit 1800 sec

memory limit 16 GB.

	#sol.	avg.time
(x) parallel, 4 thr. no communication	146	126.45
(y) parallel, 4 thr. communication	153	133.98

Parallelizing the main loop: communication



175 benchs, MUS track, SC'11.

wall-clock limit 1800 sec

memory limit 16 GB.

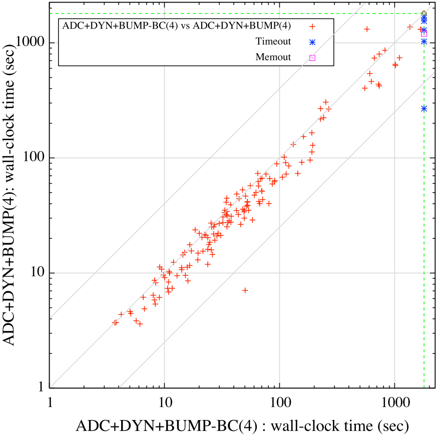
	#sol.	avg.time
(x) parallel, 4 thr. no communication	146	126.45
(y) parallel, 4 thr. communication	153	133.98

Communication is essential for performance.

Sound communication is enabled by *incremental SAT*.

Note: interestingly, sound resolution-based preprocessing for MUS extraction is also enabled by incremental SAT [Belov, Järvisalo, Marques-Silva, TACAS 2013]

Impact of “back” communication



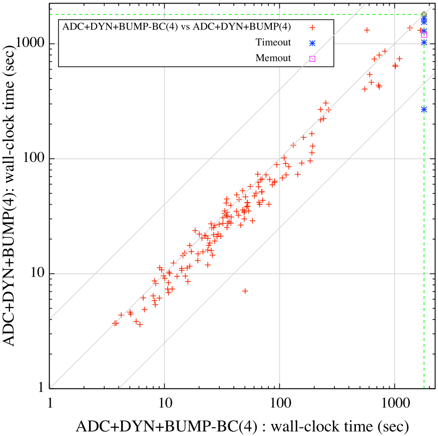
175 benchs, MUS track, SC’11.

wall-clock limit 1800 sec

memory limit 16 GB.

	#sol.	avg.time
(x) parallel, 4 thr. no back comm.	147	130.63
(y) parallel, 4 thr. full comm	153	133.98

Impact of “back” communication



175 benchs, MUS track, SC’11.

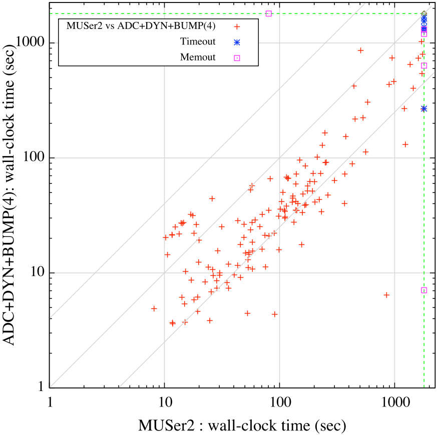
wall-clock limit 1800 sec

memory limit 16 GB.

	#sol.	avg.time
(x) parallel, 4 thr. no back comm.	147	130.63
(y) parallel, 4 thr. full comm	153	133.98

“Back” communication is actually quite crucial.

Parallelizing the main loop: communication



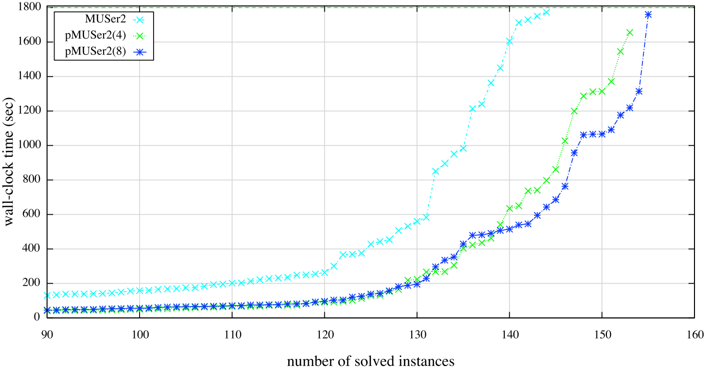
175 benchs, MUS track, SC'11.

wall-clock limit 1800 sec

memory limit 16 GB.

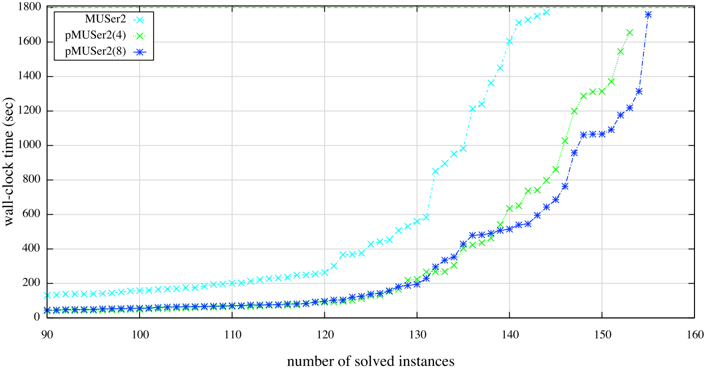
	#sol.	avg.time
(x) sequential	144	186.46
(y) parallel, 4 thr. async. + comm.	153	133.98

Performance and scalability from 4 to 8 cores



	Min. speedup	Avg. sp.	Max. sp.	Med. sp.
Seq. vs 4 cores	0.49x	4.09x	132.59x	2.94x
Seq. vs 8 cores	0.28x	4.01x	97.66x	3.38x

Performance and scalability from 4 to 8 cores



	Min. speedup	Avg. sp.	Max. sp.	Med. sp.
Seq. vs 4 cores	0.49x	4.09x	132.59x	2.94x
Seq. vs 8 cores	0.28x	4.01x	97.66x	3.38x

Possible reasons: (*i*) duplication of work; (*ii*) parallelization overhead on easy SAT calls.

Final Remarks

Also in the paper ...

- "Core-based" scheduling — a slight improvement on 8 cores.
- Results for group-MUS — less exciting than for plain-MUS.
- Comparison with TarmoMUS [Wieringa, CP 2012 and Wieringa, Heljanko, TACAS 2013] ... see the paper ☺

Main points

- Incremental SAT is a key technology for for enabling efficient parallel MUS extraction.
- Assumptions should be treated as superfluous during clause exchange.
- Good scalability to 4 cores; but not 8. Possible approaches:
  - A good partitioning/job distribution heuristic.
  - Parallel portfolio of MUS extractors ?

Final Remarks

Also in the paper ...

- "Core-based" scheduling — a slight improvement on 8 cores.
- Results for group-MUS — less exciting than for plain-MUS.
- Comparison with TarmoMUS [Wieringa, CP 2012 and Wieringa, Heljanko, TACAS 2013] … see the paper ☺

Main points

- Incremental SAT is a key technology for for enabling efficient parallel MUS extraction.
- Assumptions should be treated as superfluous during clause exchange.
- Good scalability to 4 cores; but not 8. Possible approaches:
  - A good partitioning/job distribution heuristic.
  - Parallel portfolio of MUS extractors ?

Thank you for your attention !