SArtagnan
A parallel portfolio SAT solver
with lockless physical clause sharing

Stephan Kottler and Michael Kaufmann

University of Tuebingen
OUTLINE

1 Motivation

2 Parallel Solving
   - Physical clause sharing
   - Communication of threads

3 Portfolio Solving

4 Summary
STATE-OF-THE-ART SOLVING

CDCL

- partial assignment
- decisions based on variable activity
- conflict analysis
- restarts
STATE-OF-THE-ART SOLVING

CDCL
- partial assignment
- decisions based on variable activity
- conflict analysis
- restarts

DMRP
- complete assignment (ref. point)
- decisions based on unsat clauses
- slower than CDCL but less decisions
KINDS OF PARALLELISATION

- Division of search space (guiding path)
- Portfolio solving

CLAUSE SHARING

Most solvers: copy learnt clauses of other threads
KINDS OF PARALLELISATION

- Division of search space (guiding path)
- Portfolio solving

CLAUSE SHARING

Most solvers: copy learnt clauses of other threads

MAIN AIM

- Real / physical sharing of data
- Threads work together
  \[ \Rightarrow \] Any thread may benefit from strengthened clause
- No use of OS locks
Basic Concept to Share Data

- Shared data / objects contain *user-mask*
  - user-mask initialised by creating thread
  - Any thread can release object (clear bit)
  - Last thread destructs object
- Compare and Swap operation

```c
void release(SharedObj obj, tId)
{
    do{
        SharedObj
        old = obj.umask;
        umask;
        new = clear bit 'tId' in old;
        ...
    }while(!exchange(obj.umask,old,new));
}
if(new == 0) destruct(obj);
```
Basic Concept to Share Data

- Shared data / objects contain *user-mask*
  - user-mask initialised by creating thread
  - Any thread can release object (clear bit)
  - Last thread destructs object

- Compare and Swap operation

```c
void release(SharedObj obj, tId){
    do{
        old = obj.umask;
        new = clear bit 'tId' in old;
    }while(!exchange(obj.umask,old,new));
    if(new == 0) destruct(obj);
}
```
PHYSICAL SHARING OF CLAUSES

Have one instance of a clause
Indirection to access clause (thread private data)
II. A Physical clause sharing

II Parallel Solving

**SHARED CLAUSE’ ARCHITECTURE**
REFERENCING CLAUSES IN CDCL

- Unit propagation
- Conflict analysis
- Garbage collection
Referencing clauses in CDCL

- Unit propagation
- Conflict analysis
- Garbage collection

Observation
Whenever a clause is referenced at least one watched literal is known.
II.a Physical clause sharing

II Parallel Solving

Referencing clauses in CDCL

- Unit propagation
- Conflict analysis
- Garbage collection

Observation

Whenever a clause is referenced at least one watched literal is known

Lemma

Two watched literals $l_i, l_j$ can be stored by one value:

$$C_w = l_i \text{xor} l_j.$$

($l_i \text{xor} C_w \rightarrow l_j$)
SHARED CLAUSE’ ARCHITECTURE II

II.a Physical clause sharing
II Parallel Solving

XOR of 2 watched lits

Pointer to shared clause

Optional: Index into clause

Thread 1

priv 1

priv 2

priv 3

priv ...

priv m

Thread 2

priv 1

priv 2

priv 3

priv ...

priv m

Clause 1

Clause 2

Clause 3

Clause ...

Clause m

priv 1

priv 2

priv 3

priv ...

priv m
DIGRESS TO SEQUENTIAL SOLVERS

- Order of literals may be modified
- Store clause $C$ with $|C| - 1$ integers

![Diagram showing XOR of watched literals and clauses](image-url)
OVERHEAD OF CLAUSE ORGANISATION

Comparison of different implementations with single thread

![Graph showing effects of clause organisation]
COMMUNICATION OF THREADS

Message queues used to send . . .

- a new clause (may be new version)
- notification on variable elimination
- variable replacement
- heuristic information
COMMUNICATION OF THREADS

Message queues used to send . . .
  - a new clause (may be new version)
  - notification on variable elimination
  - variable replacement
  - heuristic information

! Messages not only for heuristics
! Keep order of messages
! No OS locks
LOCKLESS QUEUES

- one reading / one writing thread
- `writeHead` points to next write position
- `readHead` points to next read position
- queue empty if `writeHead = readHead`
**Lockless Queues**

- One reading / one writing thread
- `writeHead` points to next write position
- `readHead` points to next read position
- Queue empty if `writeHead = readHead`
**Lockless Queues**

- One reading / one writing thread
- `writeHead` points to next write position
- `readHead` points to next read position
- Queue empty if `writeHead = readHead`
Lockless queues

- one reading / one writing thread
- writeHead points to next write position
- readHead points to next read position
- queue empty if writeHead = readHead
**Lockless Queues**

- One reading / one writing thread
- `writeHead` points to next write position
- `readHead` points to next read position
- Queue empty if `writeHead = readHead`
DYNAMIC SIZE

- Write operation may fail
- Write operation may overwrite unseen data

IDEA

Queue links to available update
**Dynamic Size**

- Write operation may fail
- Write operation may overwrite unseen data

**Idea**

Queue links to available update

![Diagram showing queue links to available update](image-url)
**Dynamic size**

- Write operation may fail
- Write operation may overwrite unseen data

**Idea**

Queue links to available update
**Dynamic size**

- Write operation may fail
- Write operation may overwrite unseen data

**Idea**

Queue links to available update

Queue has several reading threads!
**Different Strategies**

- 6 of 8 threads apply CDCL (different settings)
  - Activity of Variables / Literals
  - Glucose / Static / Geometric / Luby restart schemes
- Dedicated simplification thread
  - satElite like simplification
  - Asymmetric branching / vivification
  - SCC computation and removal of redundant binaries
- Connect work - DMRP
  - At each restart: init reference point to set each variable to predominant value among all threads
  - Learn ’interesting’ clauses
TAKE ADVANTAGE OF SHARING

- Simplification of clause DB is shared immediately
- On-the-fly clause subsumption done by any thread
  ⇒ Any thread may benefit
- Lazy hyper binary resolution
DMRP & Literals Activity

Configuration of solving threads

Time [s] vs. Number of Instances

- CDCL with variable activity
- CDCL (var, lit activity) and DMRP
IV Summary

Summary

- Physical clause sharing
- XOR idea to store watched literals parallel and sequential solvers
- Communication without OS locks

Challenges

- Has to run in parallel
  - Difficult to measure speedup
  - Computation time
- Logging without influencing course of events
### Clause Copying Still Faster

<table>
<thead>
<tr>
<th></th>
<th>plingeling</th>
<th>ManySAT 1.5</th>
<th>ManySAT 1.1</th>
<th>SArTagnan</th>
<th>antom</th>
</tr>
</thead>
<tbody>
<tr>
<td>#solved (in 1st run):</td>
<td>78</td>
<td>75</td>
<td>72</td>
<td>70</td>
<td>67</td>
</tr>
<tr>
<td>#solved SAT/UNSAT (in</td>
<td>23/55</td>
<td>19/56</td>
<td>18/54</td>
<td>18/52</td>
<td>19/48</td>
</tr>
<tr>
<td>first run):</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>average time per</td>
<td>97.7</td>
<td>143.9</td>
<td>124.0</td>
<td>86.5</td>
<td>83.1</td>
</tr>
<tr>
<td>solved instance:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>#solved in 2nd run:</td>
<td>79</td>
<td>74</td>
<td>73</td>
<td>70</td>
<td>65</td>
</tr>
<tr>
<td>#solved in 3rd run:</td>
<td>79</td>
<td>74</td>
<td>71</td>
<td>72</td>
<td>68</td>
</tr>
<tr>
<td>#solved in at least one</td>
<td>80</td>
<td>78</td>
<td>76</td>
<td>76</td>
<td>69</td>
</tr>
<tr>
<td>out of three runs:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rank:</td>
<td><strong>1</strong></td>
<td><strong>2</strong></td>
<td><strong>3</strong></td>
<td><strong>4</strong></td>
<td><strong>5</strong></td>
</tr>
</tbody>
</table>

Thank you for your attention!