Multithreaded SAT Solving

Matthew Lewis, Tobias Schubert, Bernd Becker

University of Freiburg, Germany
Overview

- Introduction to the Boolean Satisfiability Problem
- Sequential SAT Solving
- Distributed SAT Solving and Previous Work
- MiraXT
  - Motivation
  - Design & Implementation
- Results and Comparison to Other Solvers
- Summary & Future Work
SAT Introduction

- What are Boolean Satisfiability Problems?
  - Contain Boolean variables
  - Problems are defined in Conjunctive Normal Form (cnf)
    - Consists of a conjunction of Clauses
    - Each clause consist of the inclusive disjunction of literals
    - If problem is SAT every clause must be satisfied
  - Complete solver must find a solution or prove unsolvable
  - Hard problems (NP-Complete)

- Example: \( F(x_1,..x_n) = (\neg x_1 + x_2) \cdot (\neg x_1 + \neg x_2 + x_3) \cdot (\neg x_1 + \neg x_2 + \neg x_3) \cdot \ldots \)
SAT Introduction

What are SAT solvers used for?
- Verification (Equivalence Checking, Bounded Model Checking)
- Automatic Test Pattern Generation
- FPGA routing
- AI Planning

Existing complete powerful SAT solvers:
- Sequential: zChaff, MiniSat, Siege, Mira, ...
- Parallel: GridSAT, PaSAT, PSATA, ySAT, MiraXT, ...
- All these solvers are based on the classical Davis-Putnam method
- Current solvers can handle industrial problems with millions of clauses
Sequential SAT Solvers

\[ F(x_1,..x_n) = (\neg x_1 + x_2) \cdot (\neg x_1 \neg x_2 + x_3) \cdot (\neg x_1 \neg x_2 \neg x_3) \cdot \ldots \]

1. Load the problem and perform some preprocessing
   - Elimination of unused or one sided variables
2. Decision
   - Select a variable and assigns it a value
3. Boolean Constraint Propagation Procedure (BCP)
   - Find all implications and consequences of the decision
   - Signal Conflict
4. Conflict Analysis Procedure
   - Finds the reason for the conflict and backtracks if possible
   - Records a conflict clause to prevent future possible conflicts
Parallel SAT Solving

- A Parallel SAT Solver in theory is simple:
  - Divide the problem space and use multiple sequential SAT solvers

- However there are some important issues:
  - How do we divide the search space?
  - Communication
    - Control and Synchronization
    - Knowledge Sharing?
Parallel SAT Solving – Implementation

- Previous work based on loose integration (GridSAT, PSATO, ...)
  - Normally, an existing sequential SAT solver can be used
    - GridSAT uses zChaff, PSATO uses SATO
  - Communication is done with Message Passing
    - Relatively slow form of communication
    - Significant overhead for every message
  - Highly scalable:
    - 100’s of machines can be connected with Ethernet

Master

Clients
MiraXT – Motivation

- Focus on workstations not grids
  - Multi-core and multi-CPU workstations
- Tighter integration of SAT Solver threads
  - Shared memory used for communication and clause database
    - Provides high bandwidth, low latency, low overhead communication
- Improve Knowledge Sharing
  - In GridSAT clause with 3 literals or less were shared
    - In PaSATs “Clause Store” clauses with 5 literals or less were shared
  - Due to message passing limitations clauses are sent in bundles
    - Introduces significant latency in the knowledge sharing system
  - Problem - clients choose the clauses they want to share
    - Client should select the clauses they want to use based on their current state!
- Provide competitive single threaded performance
MiraXT – Design

- Tight integration of SAT Solver threads
- Use of shared memory clause database
- No Master Process
  - Shared object used for control signals and communication
MiraXT – Shared Clause Database

- What we need from the Clause Database:
  - Fast insertion of new clauses
    - Very little or No Lock Contention (only a pointer insertion)
  - Unimpeded use of clauses by BCP procedure
    - Clause are read only
  - Fast independent deletion of unused clauses
MiraXT – Shared Clause Database

- What we need from the Clause Database:
  - Fast insertion of new clauses
    - Very little or No Lock Contention (only a pointer insertion)
  - Unimpeded use of clauses by BCP procedure
    - Clause are read only
  - Fast independent deletion of unused clauses

![Diagram](image-url)
Problem:
- Clauses are read only (cannot easily mark WL)
- Where are the clauses physically located?
  - Remote memory is slower than local memory
Solution: Watched Literal Reference List (WLRL)
- Basic idea: each thread has a partial local copy of every clause
  - Contains and provides fast access to both WL (fast BCP)
  - Contains a cache variable
  - CV is easy to calculate, it is the old WL if the WL is replaced
- Unit, binary, and ternary clauses are fully contain within structure
MiraXT – Watched Literal Reference List

- WLRL – Contains a partial reference copy of every clause
MiraXT – Sub-problem Generation

- First come first serve (not served)
  - Optimal first DL splitting variable taken with 2 threads
  - Very low overhead technique
- Pre-processing important
  - Elimination of many possible splitting variables
  - Reduces redundancy in the problem
All communication is done in a passive way
- Threads poll a simple Boolean flag to check for new events
- If there are new events a more complicated procedure is run
  - This procedure requires locks

What object does it contain?
- Decision Stack Queue
  - First client thread to respond to message donates a decision stack
  - Once a sub-problem is added, a sleeping thread is signalled
- Queue of sleeping threads
  - Idle threads are put to sleep so they don’t waste CPU cycles
  - Sleeping threads are served randomly with new sub problems
- Statistics
MiraXT Solver Threads

- Sequential MiraXT solver Highlights:
  - Pre-processing
    - Variable and Clause Elimination (satELite)
    - Unit Propagation Look Ahead (Berre)
  - Advanced Decision Heuristic with Random Restarts
    - Modified Variable State Independent Decaying Sum (zChaff)
  - Boolean Constraint Propagation (BCP)
    - Use of watched literals (zChaff)
    - Early Conflict Detection Based BCP with IQS (Mira)
  - Conflict Analysis
    - First UIP based, with non-chronological backtracking and conflict clause recording (zChaff/Grasp)
    - Conflict Clause Deletion (Berkmin)
MiraXT Design Results

- Almost No Lock Contention
  - Only a few contention per second (2-3/s with 2 threads)
  - Problem with ySAT (up to 10% of time spent waiting)
- Cache Variable Performance
  - 84% of clauses are evaluated with only the WLRL
  - Can store entire unit, binary, and ternary clauses
- Excellent BCP/Sec Scaling (Industrial example)
MiraXT Design Results – BCP

Solver Efficiency - 2P AMD System

Solver

SatElite

Mira1T

Mira2T

zChaff

MCE/s
MiraXT – Comparison to Other Solvers

- Benchmarks
  - 2004 IBM Bounded Model Checking Benchmarks (over 1000)
  - Industrial 2005 Benchmarks from the SAT 2005 Competition
  - All benchmarks were pre-processed by SatELite

- Used an AMD Dual Processor Linux System
  - 2 Opteron processors running @ 2.6 GHz
  - 2 GB of memory per processor (4 GB total)
  - SMP Linux Kernel 2.6.*
## Results

### Comparison of Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>IBM BMC 2004</th>
<th>Industrial 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T^2$</td>
<td>#S</td>
</tr>
<tr>
<td>Mira2Ta</td>
<td>81.4</td>
<td>923</td>
</tr>
<tr>
<td>Mira2Tb</td>
<td>86.5</td>
<td>923</td>
</tr>
<tr>
<td>Mira1T</td>
<td>120.2</td>
<td>900</td>
</tr>
<tr>
<td>SatELite</td>
<td>116.9</td>
<td>901</td>
</tr>
<tr>
<td>zChaff</td>
<td>327.4</td>
<td>784</td>
</tr>
<tr>
<td>ySat2T</td>
<td>509.6</td>
<td>709</td>
</tr>
<tr>
<td>ySat1T</td>
<td>615.0</td>
<td>649</td>
</tr>
</tbody>
</table>
## Results

### Comparison of Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>IBM BMC 2004</th>
<th>Industrial 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T^2</td>
<td>#S</td>
</tr>
<tr>
<td>Mira2Ta</td>
<td>81.4</td>
<td>923</td>
</tr>
<tr>
<td>Mira2Tb</td>
<td>86.5</td>
<td>923</td>
</tr>
<tr>
<td>Mira1T</td>
<td>120.2</td>
<td>900</td>
</tr>
<tr>
<td>SatELite</td>
<td>116.9</td>
<td>901</td>
</tr>
<tr>
<td>zChaff</td>
<td>327.4</td>
<td>784</td>
</tr>
<tr>
<td>ySat2T</td>
<td>509.6</td>
<td>709</td>
</tr>
<tr>
<td>ySat1T</td>
<td>615.0</td>
<td>649</td>
</tr>
</tbody>
</table>
## Results

### Comparison of Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>IBM BMC 2004</th>
<th>Industrial 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$T^2$</td>
<td>#S</td>
</tr>
<tr>
<td>Mira2Ta</td>
<td>81.4</td>
<td>923</td>
</tr>
<tr>
<td>Mira2Tb</td>
<td>86.5</td>
<td>923</td>
</tr>
<tr>
<td>Mira1T</td>
<td>120.2</td>
<td>900</td>
</tr>
<tr>
<td>SatELite</td>
<td>116.9</td>
<td>901</td>
</tr>
<tr>
<td>zChaff</td>
<td>327.4</td>
<td>784</td>
</tr>
<tr>
<td>ySat2T</td>
<td>509.6</td>
<td>709</td>
</tr>
<tr>
<td>ySat1T</td>
<td>615.0</td>
<td>649</td>
</tr>
</tbody>
</table>
## Results

Comparison of Solvers

<table>
<thead>
<tr>
<th>Solver</th>
<th>IBM BMC 2004</th>
<th>Industrial 2005</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>T²</td>
<td>#S</td>
</tr>
<tr>
<td>Mira2Ta</td>
<td>81.4</td>
<td>923</td>
</tr>
<tr>
<td>Mira2Tb</td>
<td>86.5</td>
<td>923</td>
</tr>
<tr>
<td>Mira1T</td>
<td>120.2</td>
<td>900</td>
</tr>
<tr>
<td>SatELite</td>
<td>116.9</td>
<td>901</td>
</tr>
<tr>
<td>zChaff</td>
<td>327.4</td>
<td>784</td>
</tr>
<tr>
<td>ySat2T</td>
<td>509.6</td>
<td>709</td>
</tr>
<tr>
<td>ySat1T</td>
<td>615.0</td>
<td>649</td>
</tr>
</tbody>
</table>
Summary

- Single threaded performance is Competitive
  - Competitive with SatELite, 2005 & 2006 SAT competition winner
  - Introduced efficient data structures for multithreaded SAT solving
    - Excellent BCP scaling
  - Threaded speedup of $\approx 45\%$ on BMC and Industrial benchmarks
    - 51 % and 41% speedup for SAT and UNSAT benchmarks

- Future Work
  - Combining MiraXT with message passing to allow better scalability