MS Project Proposal:
Hybrid Parallel Complete SAT Solver
Using
Parallel Java

By Jiten Patel
Email: jrp5375@rit.edu

<table>
<thead>
<tr>
<th>Committee</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Committee Chairman</td>
<td>Prof. Alan Kaminsky</td>
</tr>
<tr>
<td>Reader</td>
<td>Prof. Edith Hemaspaandra</td>
</tr>
<tr>
<td>Observer</td>
<td>Prof. Zack Butler</td>
</tr>
</tbody>
</table>
Document History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>04/26/13</td>
<td>Initial Draft</td>
</tr>
<tr>
<td>1.1</td>
<td>05/06/13</td>
<td>Grammar corrections</td>
</tr>
<tr>
<td>1.2</td>
<td>05/10/13</td>
<td>Integrated changes suggested by Prof. Butler and Prof. Hemaspaandra</td>
</tr>
<tr>
<td>1.3</td>
<td>05/13/13</td>
<td>Accepted Proposal by entire project committee</td>
</tr>
</tbody>
</table>

Table of Contents

1. Summary
2. Background Introduction
3. Hypothesis
4. Evaluation
5. Design and Architecture Overview
   5.1. Guiding path Divide and Conquer Strategy
   5.2. Load Balancing
   5.3. Global Restart Policy
   5.4. Clause Learning
   5.5. Clause Sharing and Communication between Threads
6. Principal Deliverables
7. References
8. Tentative Schedule
1. Summary

In this project, a new hybrid approach of parallel complete SAT solver will be proposed which will use portfolio based approach combined with classical divide and conquer strategy called guiding path. This parallel SAT solver will be developed in 100% Java using Parallel Java (PJ) Library. Here, the portfolio would consist of multiple Conflict Driven Clause Learning (CDCL) sequential SAT solvers with different heuristics and random decisions for diversification. Each of these solvers would be running on a separate thread and behaves as a master which spawns multiple slaves. Each master would use guiding path strategy to split its entire search space into different areas and run them on different slaves. In guiding path strategy, a subset of variables can be pre-assigned different values so that different parallel slave processes search for a solution in different parts of the problem search space. The load balancing between parallel solving processes would be the challenging part of this guiding path strategy.

Performance of this hybrid parallel SAT solver will be evaluated on various structured industrial SAT instances from SAT Race 2010 [8] and its performance comparison will be done with sequential SAT solver MiniSAT [3] as well as parallel SAT solvers like ManySAT [4] and SArTagnan [6].

2. Background Introduction

The Boolean Satisfiability Problem (SAT) was the first known NP-complete problem and is one of the most-researched problems in Computer Science. A wide range of other decision and optimization problems can be transformed into instances of an SAT so that SAT solvers can efficiently solve many SAT instances which are useful in various areas such as circuit design and automatic theorem proving. SAT solvers have also been directly useful in various areas of computer science including theoretical computer science, artificial intelligence, hardware design, electronic design automation, and verification.

The Davis-Putnam-Logemann-Loveland (DPLL) algorithm is one of the oldest complete, backtracking-based search algorithms for solving SAT problems and it has been the basis for most of the modern more efficient sequential SAT solvers [1]. Despite the several advancements introduced in DPLL by recent algorithms, many important SAT problems are still beyond the reach of present sequential SAT solvers. The inherent complexity of the SAT limits the possibility of improvements in sequential algorithms. Parallelization of the SAT solvers seems to be the only potential approach to break the performance barrier. In last two decades, parallelization of SAT solvers has made it possible to solve SAT instances with 1,000,000+ variables and 5,000,000+ constraints in a few seconds to a few minutes [7].

Most of the existing parallel SAT solvers for multi-core computers use the DPLL algorithm as a starting point followed by partitioning the search space effectively using some divide-and-conquer techniques like classical heuristic based partitioning, guiding-paths, or dividing the Boolean formula itself. The parallel solvers in another category use a portfolio of sequential SAT solvers, each of which would run DPLL based independent sequential solvers on the same SAT instance. Each solver in portfolio would exploit a particular parameter set so their
combinations represent a set of complementary orthogonal strategies. A solver in an individual thread could perform knowledge exchange in order to improve the performance globally. The communication between individual solvers of a portfolio can be organized through lockless queues that contain the lemmas that a particular core wants to exchange [4].

ManySAT is a CDCL based, portfolio-based, multi-core (4 cores) parallel SAT solver engine which tries to address the weaknesses of other existing parallel solvers by including techniques like modern dynamic restart policy, heuristic based random noise enabled diversification, polarity strategy for literal assignment, learning based on implication graphs, and clause sharing with the addition of classical features like two-watched-literal, unit propagation, activity-based decision heuristics, and lemma deletion strategies. ManySAT runs different incarnations of sequential solvers on the same instance and performs knowledge sharing in order to improve overall system performance but it does not guide different solving processes in any way.

On the other hand, SArTagn is another portfolio-based, multi-core, parallel SAT solver which uses a guided search where a thread uses a partial variable assignment called an Autarky which may change the set of models for satisfying formulae. SArTagn solver also uses physical clause sharing and communication between threads to benefit them from simplification and clause minimization techniques of other threads.

After getting inspired by the above mentioned two parallel SAT solvers, in this project, a hybrid parallel SAT solver will be proposed which combines the portfolio approach with Guiding Path divide and conquer technique. I would like to develop a SMP parallel Java program using Parallel Java (PJ) [2] API for a portfolio-based complete SAT solver that can solve very large structured industrial SAT instance faster than most of the existing sequential complete SAT solvers. MiniSAT is a Conflict Driven Cause Learning (CDCL) based sequential SAT solver developed in C where as ManySAT is parallel version of it developed in C++ using OpenMP library. SAT4j [5] is a general purpose Java API for solving SAT and optimization problems. Since no well-known CDCL based SAT solver implemented in Java is available as freeware, I would like to implement my program from the scratch with the reference of the above mentioned three implementations.

3. **Hypothesis**

Portfolio and Guiding Path divide and conquer technique based hybrid parallel SAT solver should be able to solve selected structured industrial SAT instances from SAT Race 2010 much faster than other sequential and parallel SAT solvers.

4. **Evaluation**

I. Performance measurements of this hybrid parallel solver on a range of structured large industrial SAT instances from SAT Race 2010.

II. Performance comparison of this hybrid parallel solver with existing complete sequential solver MiniSAT as well as parallel solvers ManySAT and SArTagn.
5. Design and Architectural Overview

In this project, a hybrid parallel SAT solver will be implemented which will include a portfolio of two independent CDCL SAT solvers. These two solvers would work on the same SAT instance in parallel to decide it is satisfiable. Each solver would include the below mentioned features.

5.1 Guiding path Divide and Conquer Strategy

Each CDCL solver would behave as a master thread and divide the search space using guiding path divide and conquer strategy. This approach seems more obvious in SAT solving where a subset of variables can be pre-assigned to different values for different parallel solving processes. Pre-assigning variables forces all parallel processes to search for a solution in a different part of the search space.

5.2 Load Balancing

After creating parallel solving processes, one process may encounter conflict and finish execution while other processes are still working on their own search spaces. On SMP architectures the guiding path approach can be implemented by including dynamic work stealing [9] where an inactive thread can request work from any active thread where as the active thread can divide its own guiding path into two paths, one of which is given to the requesting thread. The problem with this approach is that the requesting thread has to interrupt the active thread which eventually leads to waste of CPU cycles. Moreover, if a requesting thread gets a fairly negligible chunk of work then work stealing becomes even more frequent. To solve this issue, a central work queue can be maintained which is topped by the longer running threads. The idle threads can then steal from this central queue without having to wait for other threads to respond. When the central queue runs out, the requesting thread continues stealing work from the active thread.

5.3 Global Restart Policy

SAT solvers use a restart policy to compact the assignment stack and improve the order of assumptions. The performance of various restart policies widely depends on the type of considered SAT instance. Based on the observation of early works [5] [6], usually frequent restart policy drastically improves the SAT solver performance on industrial instances. In this project, one CDCL solver would implement the well known Luby policy [10] where as the second CDCL solver would implement the adaptive restart policy [11]. The former policy dynamically measures the search process agility which in turn measures the average number of recently flipped assignments. Low agility means frequent restarts; where as high agility avoids restarts.

5.4 Clause Learning

Clause Learning is an essential feature of most of the modern SAT solvers. In this hybrid SAT solver, each individual CDCL solver would implement the implication graph based clause
learning scheme described in [4]. This technique uses inverse arcs which can be obtained by considering satisfied clauses of the SAT formula, which are typically ignored by classical conflict analysis. These inverse arcs detect that even some decision literals admit a reason which is ignored by classical implication graphs. It results in increased size of the back jumps, which eventually speeds up the search process.

5.5 Clause Sharing and Communication between Threads

From the observation of previous SAT solvers, the size of the learnt clause that should be exchanged is supposed to be less than or equal to 8. The communication between the solvers of the portfolio can be organized through lockless queues which contain the lemmas that a particular core wants to exchange. [4] [6]

6. Principal Deliverables

I. A final comprehensive project report which includes project design, features, challenges with their solution, discussion of related existing parallel solvers, and performance comparison of this SAT parallel solver with existing sequential solver MiniSAT as well as parallel solvers ManySAT and SArTagnan.

II. A source code of SMP parallel complete SAT solver in Java using PJ, executable binary and its documentation.

III. Demo of project functionalities and features.

IV. Project presentation slides in Microsoft power point and PDF formats.

V. All above mentioned design documents, source files, user manual and final report will be delivered in a single zipped file.

7. References


8. Tentative Schedule

<table>
<thead>
<tr>
<th>Target Date</th>
<th>Topic</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>05/13/13</td>
<td>Approval of project Design and Proposal</td>
<td>Accepted</td>
</tr>
<tr>
<td>05/13/13-05/17/13</td>
<td>Develop project website</td>
<td>In Progress</td>
</tr>
<tr>
<td>05/18/13-05/31/13</td>
<td>Study MiniSAT, ManySAT and SArTagnan solver</td>
<td></td>
</tr>
<tr>
<td>06/01/13-06/15/13</td>
<td>Design and develop two individual CDCL SAT solvers based on ManySAT and SArTagnan</td>
<td></td>
</tr>
<tr>
<td>06/16/13-06/30/13</td>
<td>Modify each CDCL solver to incorporate master-slave pattern using guiding path divide- &amp;-conquer technique.</td>
<td></td>
</tr>
<tr>
<td>07/01/13-07/15/13</td>
<td>Coordinate two solvers to parallelize them</td>
<td></td>
</tr>
<tr>
<td>07/16/13-07/31/13</td>
<td>Measure performance of the final solver and compare it against performance of MiniSAT</td>
<td></td>
</tr>
<tr>
<td>08/01/13-08/15/13</td>
<td>Write final report</td>
<td></td>
</tr>
<tr>
<td>09/01/13-09/20/13</td>
<td>MS project defense</td>
<td></td>
</tr>
</tbody>
</table>