A Framework for Parallel SAT Solving Using Batch Schedulers

David Bremner, Eric Aubanel, Sajjad Asghar
Faculty of Computer Science, University of New Brunswick Fredericton, NB, Canada

Introduction

Solvers for the Boolean satisfiability problem (SAT) are enabling technology for diverse set of applications.
Real Life Applications of SAT solvers are
- •Electronic Design Automation (EDA) and verification
- •Field Programmable Gate Array(FPGA) routing
- •Automatic Theorem proving
- •Bounded model checking
- •AI planning
- •Formal verification of hardware and software design
- •Cryptography and crypto Analysis

Generally a SAT problem is represented as conjunctive Normal Form(CNF) .Following is an example of CNF for a SAT instance that have five variables and five clauses.

$$(x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_3 \vee x_4) \wedge (\neg x_2 \vee \neg x_3 \neg x_5) \wedge (\neg x_4 \vee x_5)$$

Davis, Putnam, Logemann and Loveland(DPLL) Algorithm is most popular algorithm for SAT solving.
- DPLL is based on backtracking technique
- Most state of the art SAT solvers are based on DPLL algorithm
- In addition to DPLL modern SAT solvers have introduced the concepts of
  - Clause learning
  - Non-chronological backtracking(back jumping)
  - Two watched literals per clause
  - Variable State Independent, Decaying Sum (VSIDS) for variable selection heuristics
  - Restarts
- Some popular SAT Solvers are
  - miniSAT
  - Tsat
  - Crypto minSAT

Parallel SAT Solving

Parallel SAT solving is challenging task due to the irregular structure of Search space a of SAT problem.
- There are very few dedicated parallel SAT solvers.
- Most of the parallel SAT solvers are based on divide-and-conquer paradigm.
- In parallel SAT solving, it is difficult to handle workload management and work scheduling of Jobs.

Objectives

The main objectives of our research work are to
- Design a framework that utilizes parallel tree search techniques for the solving of SAT problems
- Design framework that is
  - Easy to use and easy to integrate with currently available parallel computing infrastructures and software.
  - Can run with available state of the art SAT solvers without making a lot of changes to the original code of these SAT solvers.
  - Should be compatible with different cluster resource managers and batch schedulers

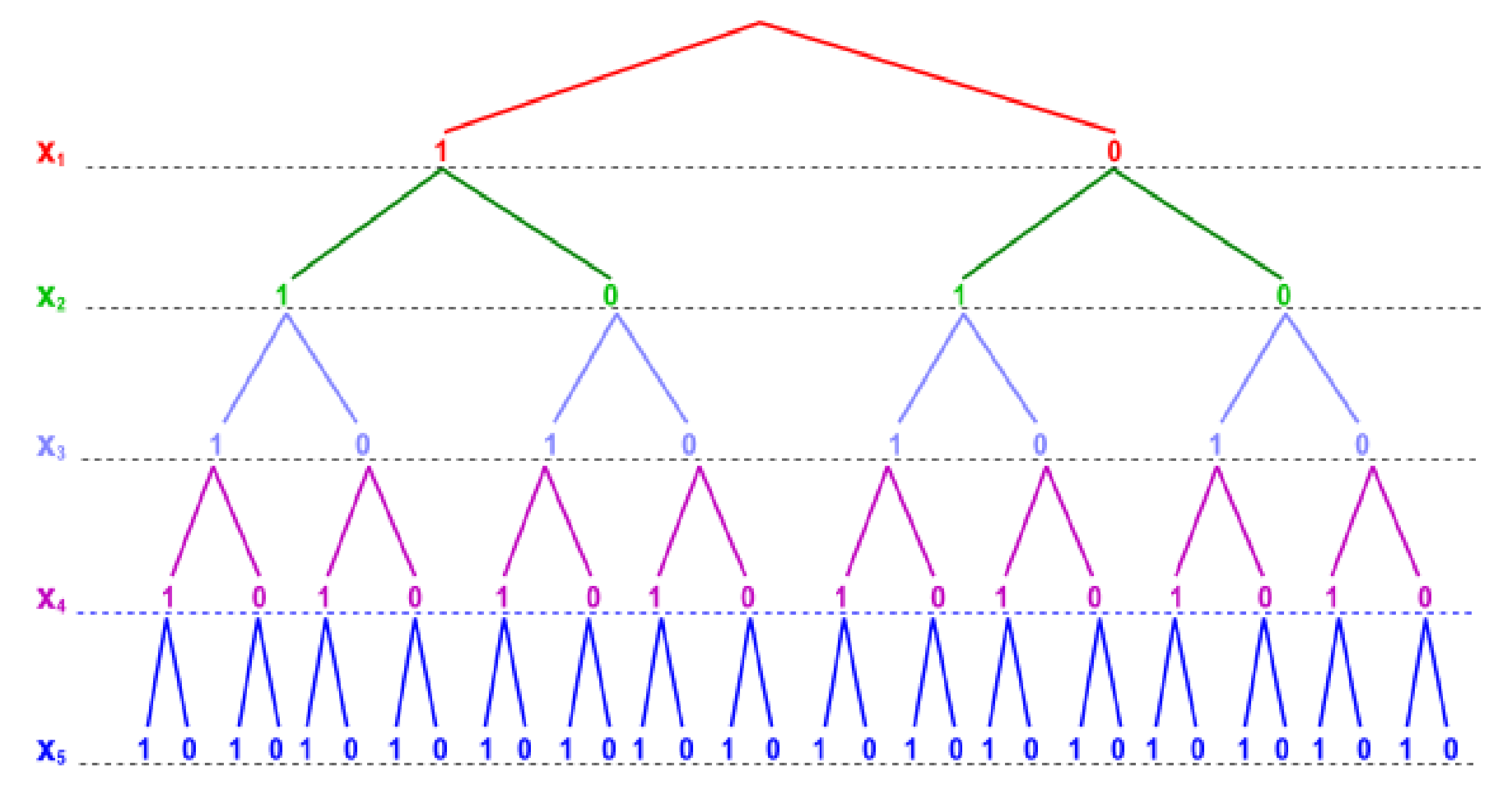
Methodology

Representing a SAT instance as Tree:

A SAT instance can be represented as a binary tree. For example given a SAT instance with five variables

$(\neg x_2 \vee x_5) \wedge (x_1 \vee \neg x_3 \vee x_4) \wedge (x_4 \vee \neg x_5) \wedge (x_2 \vee x_1)$

this SAT formula can be represented as tree with all the 32 assignments that are possible for this formula



Source http://www.mqasem.net/sat/sat/index.php

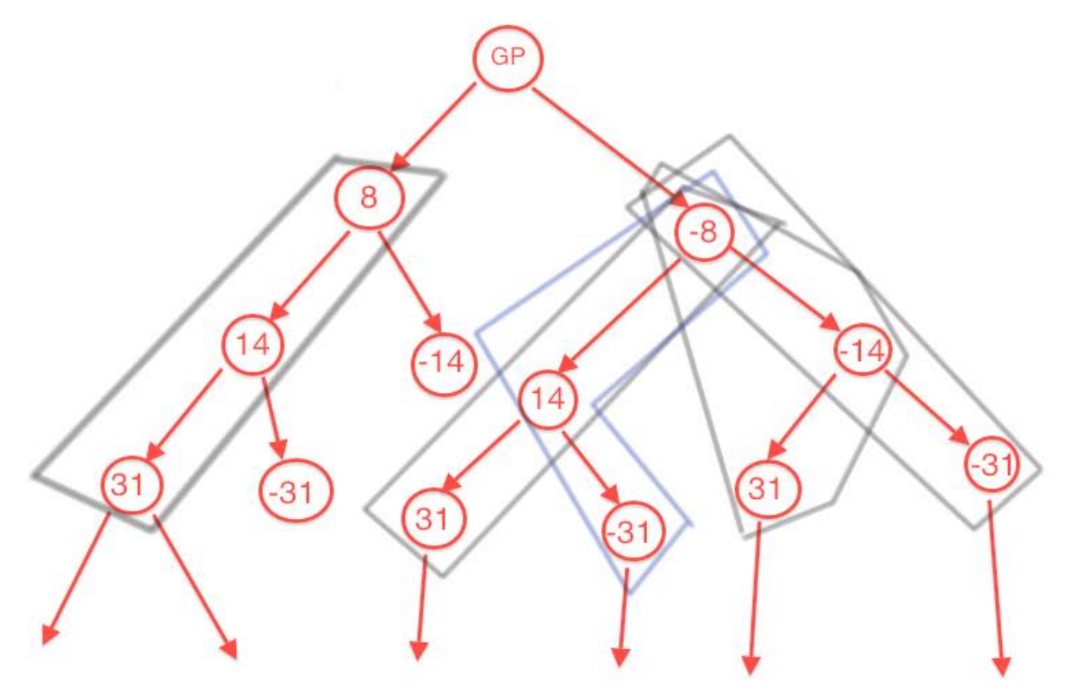
Splitting of Search tree:

To solve a SAT instance with the help of parallel computing infrastructure, the search tree of a SAT instance is split into different branches, and then each branch is submitted to a workstation to search for the solution in different parts of the tree. A tree for SAT solving is split by using the idea of guiding Path.

Guiding Paths:

To find the solution of SAT instance in parallel we have to split the search tree into different paths, each path represents a different part of the search tree. Guiding paths give us good estimation for search tree splitting. Guiding paths are calculated after running a SAT solver for a given duration serially, for example If 8, 14, 31 is a sequence of assignments in a given formula, then the corresponding guiding paths are {-8, 14, 31}, {-8, -14, 31}, {-8,14, -31},{-8, -14, -31}.

Each sequence of above mentioned guiding paths lead us to a different branch of a search tree this illustrated in the following diagram.



For our framework we have identified and experimented with two different types of guiding paths based tree splitting techniques and these techniques are

1. **Static Tree Splitting:**
   - At first a SAT solver is run on the master node to discover the guiding paths.
   - Initial formula of SAT instance is modified according to the newly discovered guiding paths.
   - Depending on the number of Paths the jobs are submitted to the workstations on the cluster.
   - Jobs are run independently of each other
   - Once the solution is found by on job, all the other running jobs are notified to stop searching, and results are returned to the master node.

Static splitting has been very effective for our experiments, but it suffers with following limitations
- Once jobs are submitted the number of processes, that we can use for solving of a SAT instance become fixed and even though if we have free resources available on the cluster, we can not use them.
- Static splitting also suffers from the work repetition and cycles over the search

Dynamic Tree Splitting

Dynamic splitting of the search tree is done on the bases of available CPU resources and before submitting the new jobs, the framework also collects the up-to-date information from already running jobs. Dynamic splitting helps to achieve efficient use of resources and it also reduces the work repetition. The work related to dynamic splitting is still under progress.

Architecture of Framework

The main components that are part of the architecture of our developed framework are

1. **DRMS**

Distributed Resource Management System(DRMS) are consist of batch schedulers that enables the users of a cluster to submit, control and monitor jobs on remote workstations. Few of widely used DRMS are
- Oracle Grid Engine(OGE)
- Condor
- Open PBS/PBS pro
- LSF
- SLURM
- Torque

We are using ACEnet infrastructure at UNB, ACEnet cluster is running with OGE as DRMS.

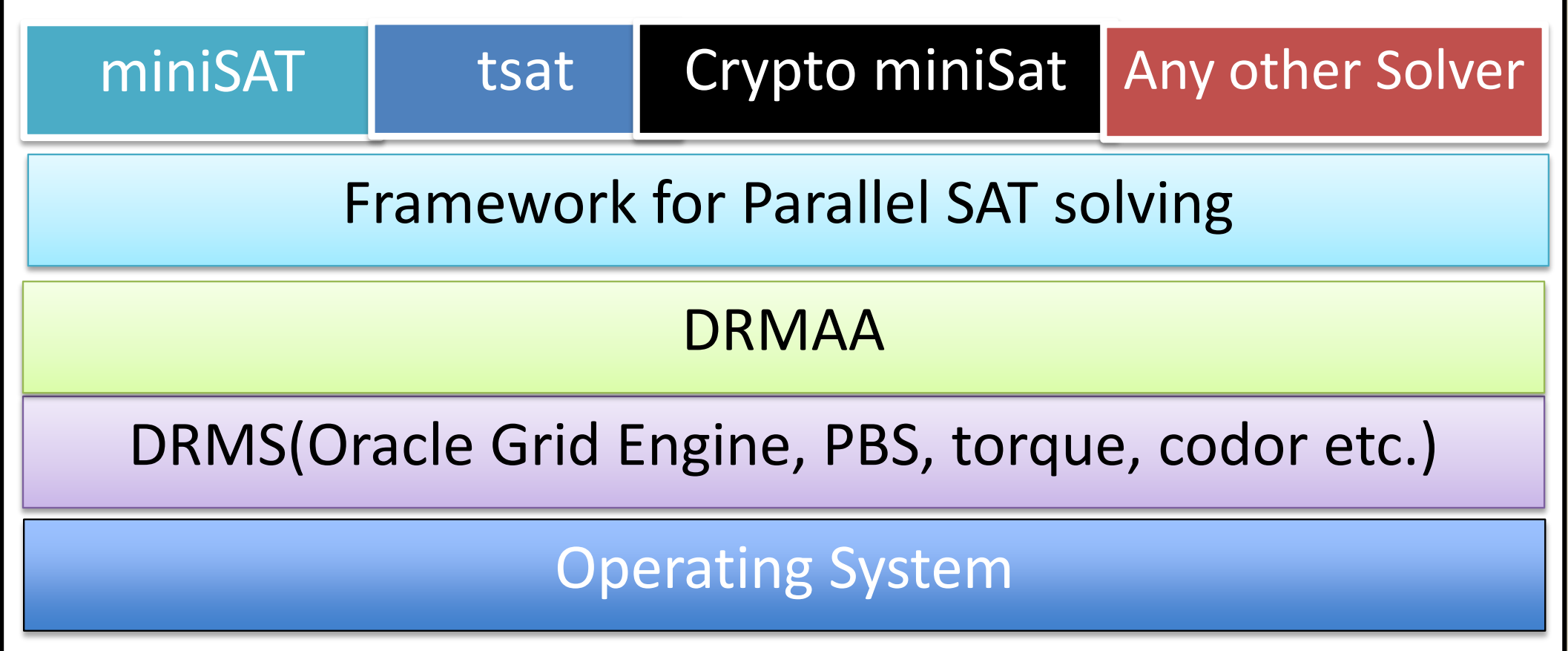
2. **DRMAA**

Batch schedulers based clusters provides commodity computing with low cost per CPU cycle. We are using ACEnet infrastructure and oracle Grid Engine(OGE) for running and testing of our SAT solver framework. Job control and management in our developed framework is done through Distributed Resource Management Application API (DRMAA).

DRMAA is an Open Grid Forum specification that standardizes job submission, job monitoring, and job control in Distributed Resource Management Systems (DRMS) in a standard way.

3. **Framework for Parallel SAT solving**

DRMAA controls running of jobs on a cluster, while job creation , Job submission and job monitoring on the cluster is done by the core of our framework. The framework is also responsible for the splitting of the search tree and generation of necessary knowledge base that helps to solve a SAT instance quickly .The framework is flexible enough to work with any available SAT solver and many batch schedulers. Following is the diagram of layered architecture of our framework

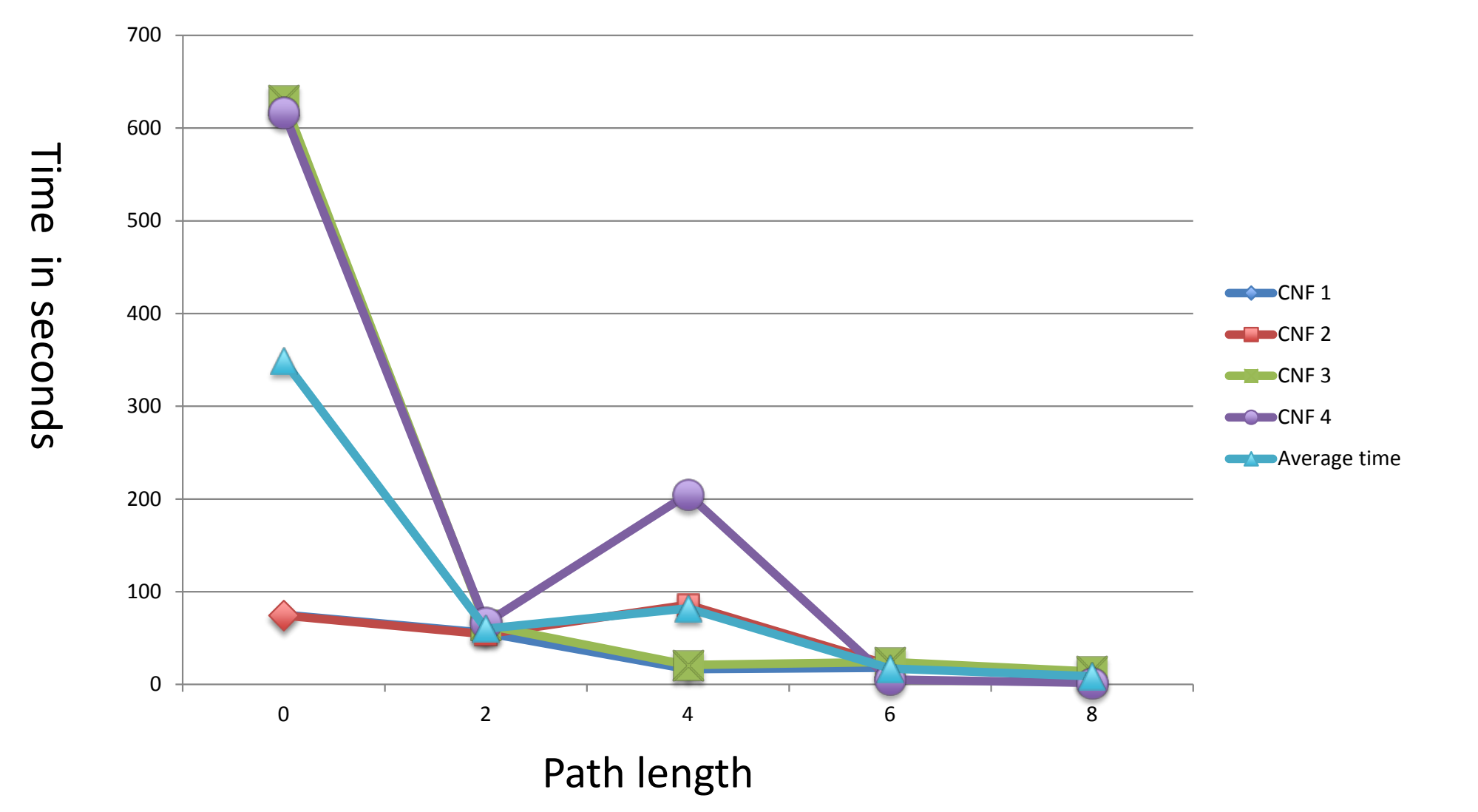

miniSAT
tsat
Crypto miniSat
Any other Solver
Framework for Parallel SAT solving
DRMAA
DRMS(Oracle Grid Engine, PBS, torque, codor etc.)
Operating System

The layered architecture of the Framework

Results and Statistics

With dynamic and static splitting we have got interesting results, following table and chart indicate the results we got from the Static Splitting of a search tree on four different SAT instances .With Dynamic splitting we got good preliminary results as well and we were able to solve the SAT instance using less resources. miniSat is used as base solver for our experiments, but we can use any SAT solver with our framework.

	CPU used				
	1	4	16	64	256
CNF 1	75.4	55.6	16.5	18.2	11.3
CNF 2	74.3	54.3	85.9	20.6	8.7
CNF 3	630	64	20.9	24.18	13.9
CNF 4	617	66.4	205	5.3	1.64
Average time	349.175	60.075	82.075	17.07	8.885



Conclusion and Future Work

In this poster we have presented our SAT solving framework. The framework is batch scheduler independent and it can run with any batch scheduler and with any off-the-shelf SAT solver without modifying the original code of the SAT solver. The presented framework make efficient use of cluster resources and can solve the SAT instances very efficiently.

We have got good results with static splitting, but we also have identified few shortcomings with static splitting of search tree, Although Our initial results with static splitting are encouraging but in future we will be focusing on the dynamic splitting of search tree to mitigate the limitations of static splitting.

Acknowledgements

We thank the following organizations for providing the finical support to carry out our research work
- **NBIF(New Brunswick Innovation Fund)**
- **NSERC (Natural Sciences and Engineering Research Council of Canada)**
- **University of New Brunswick**

We would also like to acknowledge the support from **ACEnet** for providing the computing resources and infrastructure.