Parallelizing MiniSAT

CS 4402B

Riley Gavigan

April 3, 2024

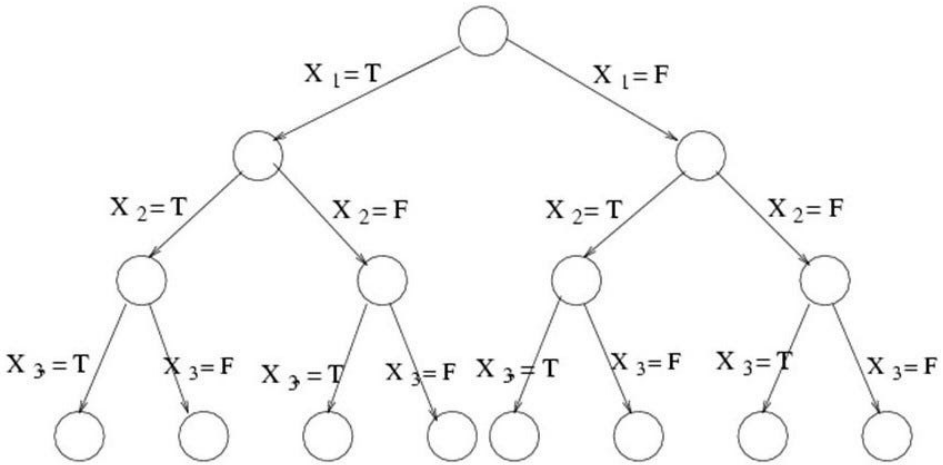
The Boolean Satisfiability (SAT) Problem

Classic NP-Complete problem. Consists of Conjunctive Normal Form (CNF) problems that we want to attempt to satisfy as True.
CNF: conjunctions (ANDs) of clauses, which are disjunctions (ORs) of literals (variables).

Example: $(A \vee B) \wedge (A \vee C) \wedge (A \vee \neg B \vee C)$

Applications: Electronic Design Automation, Program Verification, Constraint Solving in AI

Visualization



Example: This is a visual representation of the decision tree for the clause $(X_1 \vee X_2 \vee X_3)$

Growth: 2^n, where $n = \#variables$ (for 1 clause)

Brute Force SAT Solving

$$(X_1 \vee X_2) \wedge (\neg X_2 \vee X_3)$$

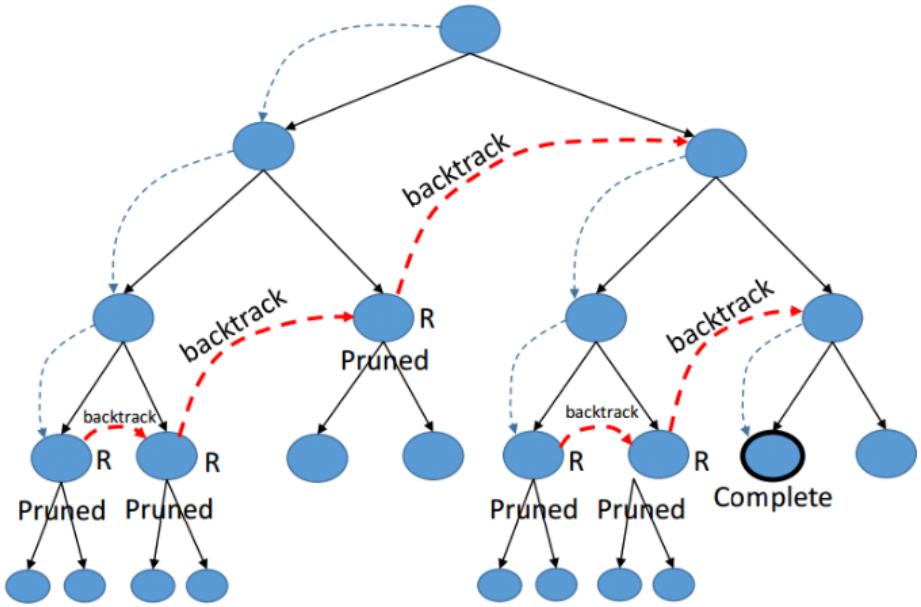
X_1	X_1	X_3	$(X_1 \vee X_2) \wedge (\neg X_2 \vee X_3)$
F	F	F	F
F	F	T	T
F	T	F	F
F	T	T	T
T	F	F	T
T	F	T	T
T	T	F	T
T	T	T	T

Here, and in many other cases, we can find optimizations. In this case, it is evident that no matter what value X_2 takes, if either X_1, X_3 or both are T the result is T.

Simplified: $X_1 \vee X_3$

Backtracking

Backtracking involves depth-first search: if we make assumptions about the value of literals and are unable to find a solution, we backtrack to test out other combinations.



When a solution is found, we can stop back-tracking and return true. In the worst case, we back-track to test out every combination before returning false (no solution).

Serial SAT Solver Optimizations

1. Restarting: Periodically resetting search process to improve efficiency. Can be based on factors such as number of conflicts.
2. Conflict-Driven Backtracking: When conflicts are encountered, the cause is identified and the SAT solver learns from it by adding a new clause to the formula. It "learns" a clause to repeat mistakes.
3. Dynamic Variable Ordering: Prioritizing variables during the search process to prune the search space by finding more conflicts. Adjusts priority based on previous behaviour such as involvement in conflicts.

Previous Dynamic SAT Solver Approaches

1. Guided Path: Divides search space into disjoint subspaces. Does this by branching on each "assumption" variable encountered to search these subspaces in parallel. Used by most parallel solvers.
2. "Hope it Sticks" (name I made up): Running multiple independent serial solvers with unique "tuning parameters", terminating when one of them finds a result. Serial solvers can make use of serial optimizations.

MiniSAT

Lightweight and highly efficient SAT solver
Utilizes conflict-driven backtracking and dynamic variable ordering

Utilizes several data structures for efficiency:

- assume (log of assumptions)
- trail (log of variable assignments)
- clause_DB (list of clauses)
- assign (current assignments)
- reason (reasoning for assignments)
- order (variable order sorted by activity)
- activity (variable activities)

Methods for Parallelizing MiniSAT

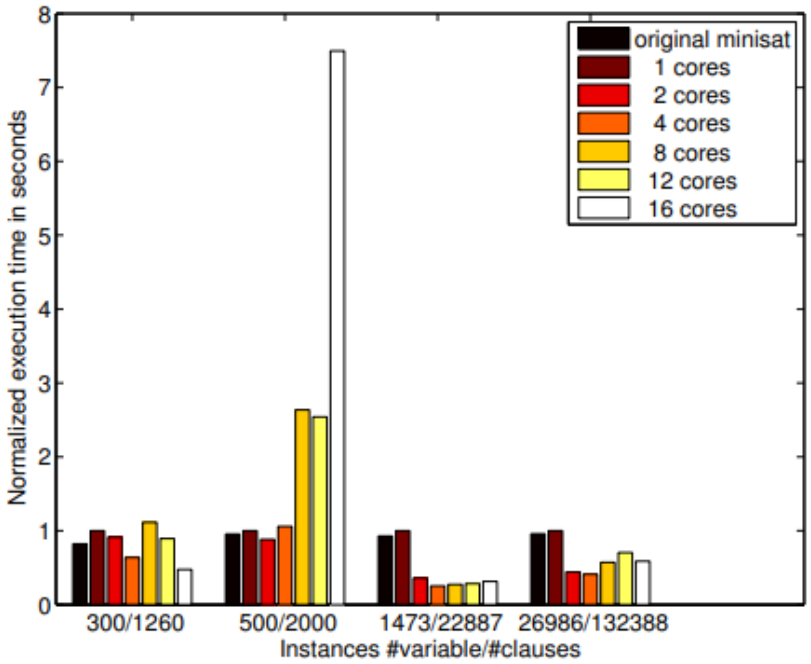
- Converting the iterative MiniSAT implementation to a recursive implementation that can be parallelized
- Allow the 2 branches (0/1) of a given node to be searched in parallel
- New global_DB data structure to allow learning to occur: workers add to and fetch clauses from this periodically
- Workers have local clause_DB structure. All other structures employ some optimizations to avoid expensive copying.

Parallel MiniSAT Implementation

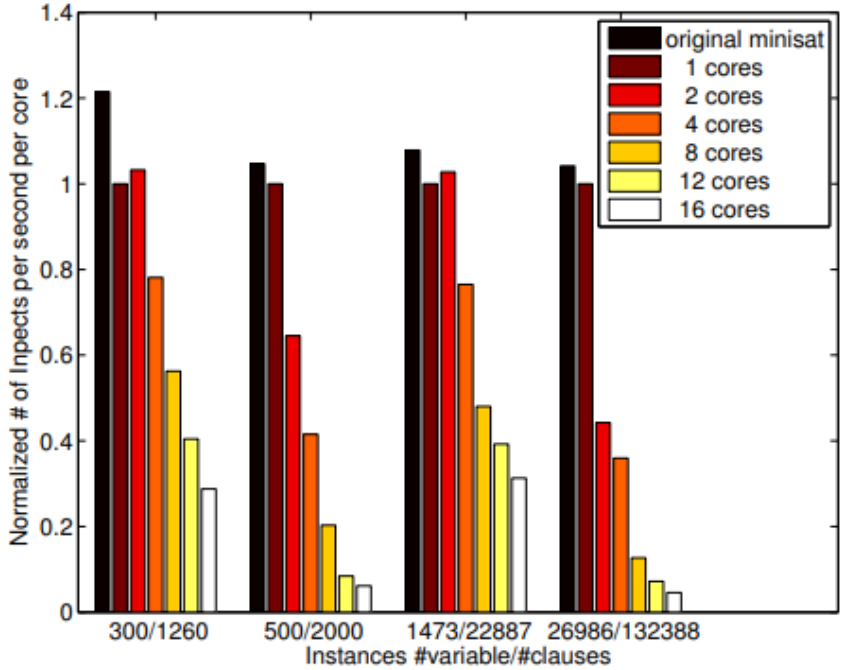
```
1  int search(Solver *s,int depth,var *assume) {
2    int blevel = INT_INF;
3    bool backtrack = false;
4    var *new_assume = NULL;
5    inlet void catch(int b) {
6      blevel = min(blevel, b);
7      abort;
8    }
9    while(!backtrack) {
10     blevel = INT_INF;
11     fetch_from_globalDB(s);
12     blevel = process_fetched_clauses(s);
13     // ...
14     confl = propagation(s);
15     if(confl) {
16       backtrack = true;
17       if(level(s) == s->root_level) {
18         set_res(UNSAT);
19         blevel = s->root_level - 1;
20       } else {
21         blevel = analyze(s,confl,&learned);
22         post_to_globalDB(learned);
23         expand_DB(s,learned);
24         cancel_until(s,blevel);
25       }
26     } else {
27       next = select_next_var(s);
28       if(next == UNDEF) {
29         set_res(SAT);
30         blevel = s->root_level - 1;
31         backtrack = true;
32       } else {
33         assume(s,lit_neg(next),assume);
34         catch(spawn search(s,depth+1,assume));
35         if( !SYNCHED && blevel == INT_INF ) {
36           s = get_current_solver();
37           replay(s,assume,new_assume,depth);
38           // ...
39           assume(s,next,new_assume);
40           catch(spawn search(s,depth+1,
41                              new_assume));
42         }
43         sync;
44         if(blevel == INT_INF) {
45           break;
46         }
47         backtrack = (blevel < depth);
48         if(!SYNCHED && !backtrack) {
49           s = get_current_solver();
50           replay(s,assume,new_assume,depth);
51         }
52       }
53     }
54   }
55   if(new_assume) free(new_assume);
56   return blevel;
57 }
```

Top-level that breaks and returns when solver needs to backtrack. Workers keep track of Solver state, depth, and assumptions.
Fetches from global_DB each loop iteration.

Results



(a) Execution time. We can see that sometimes the time improves with cores but not always, sometimes it is much worse.



(b) Unit throughput. We can see it trend downwards consistently as core # increases.

Limitations

- Execution time does not scale with an increase in processor cores in the parallel implementation. Sometimes it is worse than serial MiniSAT
- There is additional overhead such as replaying assume and fetching / processing clauses from global_DB as processor count increases
- Memory usage grows linearly with number of processes and heavily limits speed-up

Conclusions and Takeaways

1. We don't know if an efficient parallel solver for the SAT problem can be completely deterministic, but this paper hopes to move toward that direction.
2. It is currently unclear how to get around issues pertaining to synchronization needs for learned clauses and watched literals. This is open as a way to optimize Parallel MiniSAT.
3. How Parallel MiniSAT can be made more deterministic is an open problem.