Analysing Mini-Sat

CS4402B

JEFFREY WONG

Plan

Overview of Sat Problem

- What is an Sat Problem
- Visual representation
- Solving an Sat Problem

Parallel solving

- Search space splitting
- Clause sharing
- Difficulties that arise

Parallelizing MiniSat - report

- Implementation
- Benefits
- Limitations

Experimentation

Conclusion

What is a SAT problem

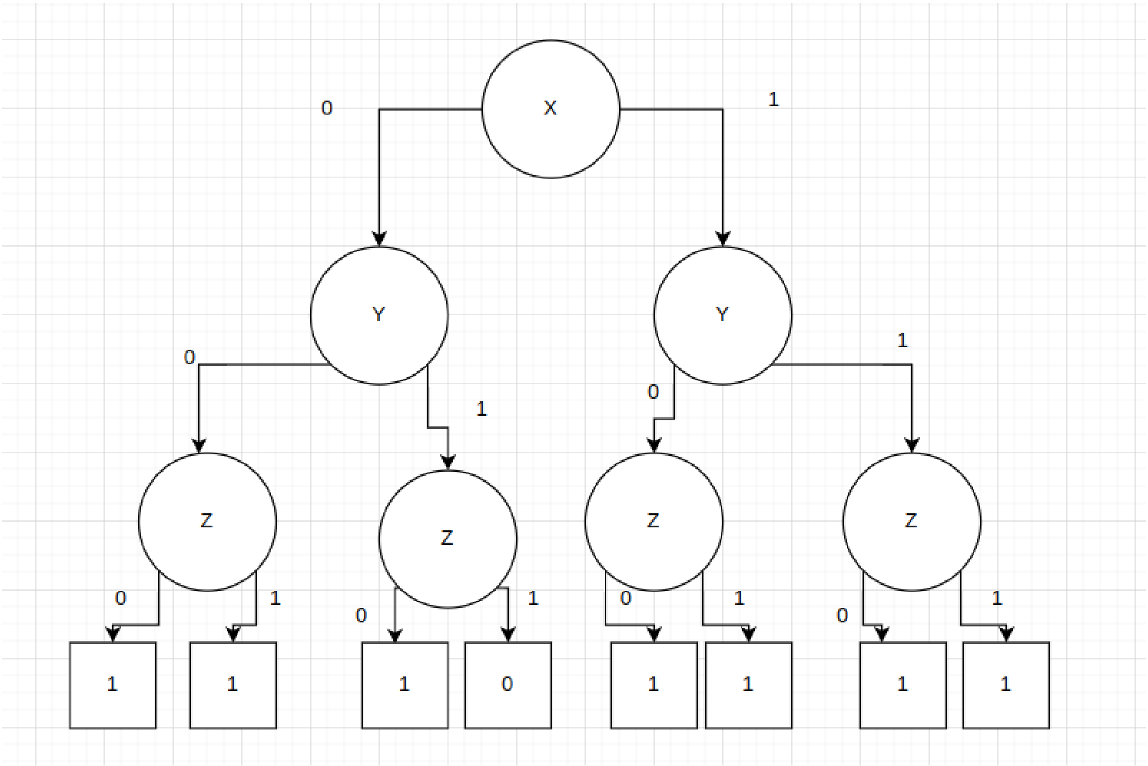
Think back to 2209. We encountered problems such as:

$$(\neg A \vee B) \wedge (\neg A \vee C) \text{ or } (x \vee \neg y \vee z)$$

Given problems in conjunctive normal form (CNF), can we find a set of variables to create a true condition.

Applications

- AI decision making
- Robotics and sensor data.
  - Oil temperature: True
  - Coolant temperature
  - Computer error: False
  - Thus, the Sat problem could be $(\text{Oil} \vee \text{Cool}) \wedge (\neg \text{Computer})$



What is a SAT problem

Visual Representation (x V ¬y V z)

Problems can be visually demonstrated as binary trees. Leafs can take the path of (x,¬x) or (0,1)

Solving a SAT problem

Naive solution, brute force

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

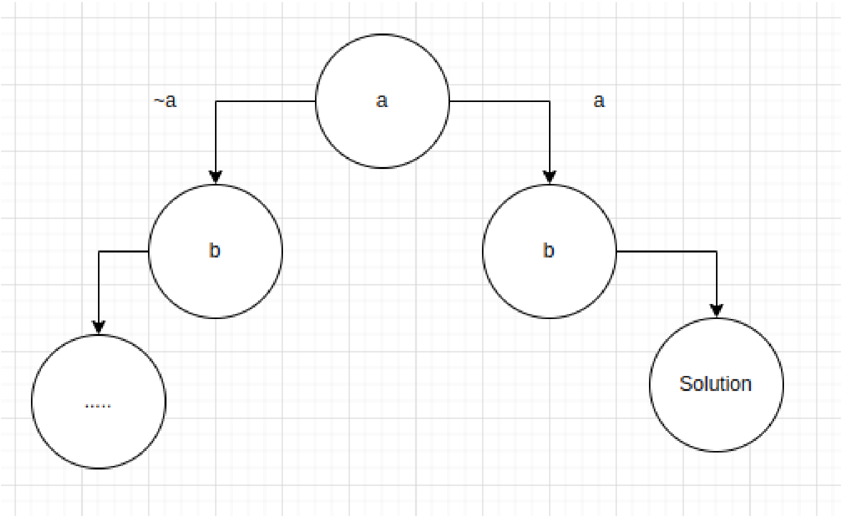
A	B	C	$((\neg A \vee B) \wedge (\neg A \vee C))$
F	F	F	**T**
F	F	T	**T**
F	T	F	**T**
F	T	T	**T**
T	F	F	**F**
T	F	T	**F**
T	T	F	**F**
T	T	T	**T**

$(X \vee \neg Y \vee Z)$

x	y	z	$(x \vee (\neg y \vee z))$
F	F	F	**T**
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**

O(2^n) to attempt every possibility.

Sat Solvers



Simple Backtrack

Like a maze game. Has visited nodes and attempts to visit all nodes to find a solution.

In this example, using DFS we find a solution early on, we can see wasted computing.

¬a will never result in SAT.

$(\neg a \vee b) \wedge (\neg a \vee \neg c)$

Sat Solvers

DPLL – Depth first backtrack search

Branching
- Choosing at random to move forward.

Unit Propagation
- Assigning 0 or 1 to literal.

Backtracking
- Moves back to last point of decision making when a conflict is observed.

Makes use of Boolean constraint propagation to make inferences to cut down search area.

Sat Solvers

CDCL – Conflict Driven Clause Learning

- Takes DPLL and expands upon the idea.
- Advancements over DPLL
  - Jumping is non-chronological
  - Clause learning

Types of Sat Solvers

IN-COMPLETE SOLVERS

Satisfiability or un-satisfiability

COMPLETE SOLVERS

Find a solution or prove no solution exists

Majority of applications of Sat solvers are complete.

DPLL fits into this category

CDCL extends DPLL

MiniSat, being based on CDCL is also complete

Parallelizing Sat

PAST STRATEGIES

"AN OVERVIEW OF PARALLEL SAT SOLVING" - RUBEN MARTINS · VASCO MANQUINHO · INÊS LYNCE

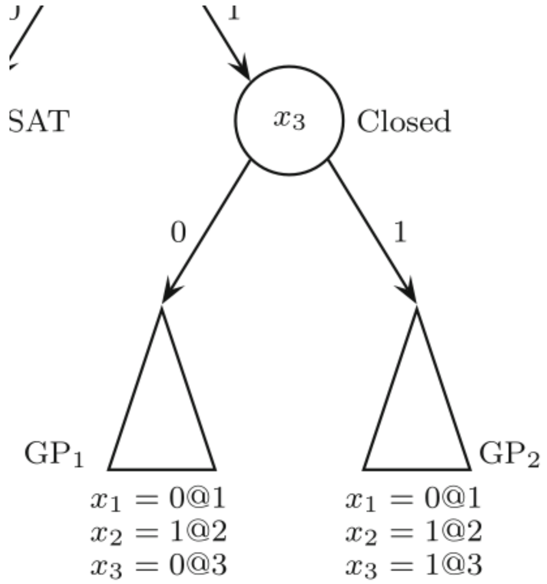
Past Strategies - Search Space Splitting

Makes use of divided paths.

Subspace proof time needed cannot be predicted

Uneven work distribution.

Benefits from work stealing procedures



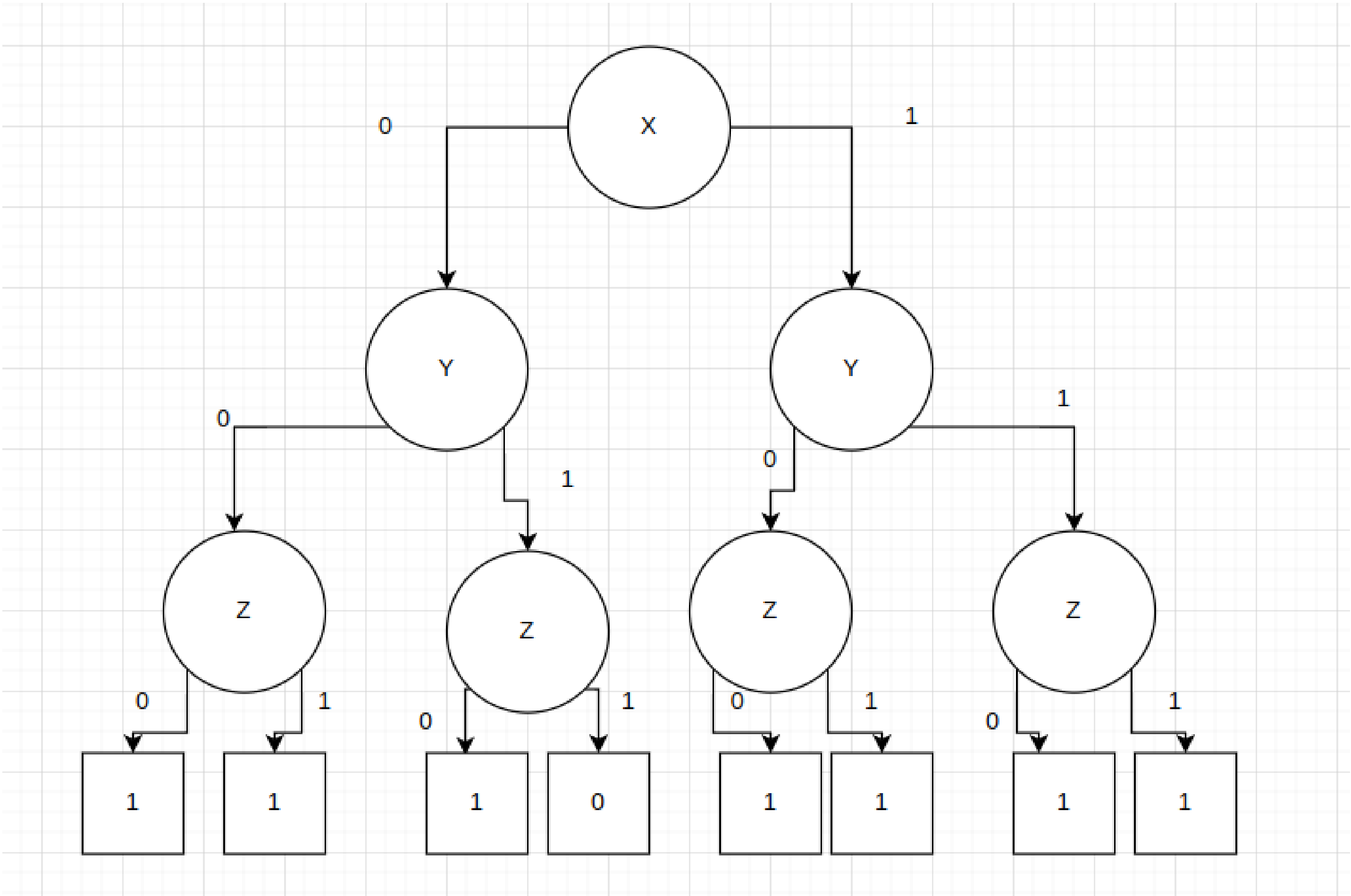
Past Strategies - Search Space Splitting

NAGSAT

Master Process performs full DPLL

Workers will request from master a guiding path

3 possibilities of a worker
- Termination
- Reduction of search space (pruning)
- Solving the problem



Past Strategies - Clause sharing

Share between threads generated conflict clauses

When a thread finds that a solution cannot be achieved, it will generate a conflict clause. This will alert other processes of that subspace that additional work will be unfruitful.

With looking at each branch exponential memory usage becomes a problem.

To get around this, many parallel Sat solvers make us of a limit, I.e., 5 or less literals.

- Only share conflict clauses that have number of literals below that limit.
- This limit may also include limiting conflict clauses based such as restrictions regarding guiding path.

The large variety of possible strategies makes it difficult to find superiority over one another.

What makes miniStat

Improvements made

Conflict minimization

$$
\begin{array}{l}
\textit{strengthenCC}(\textbf{\textit{Clause}}\ \mathrm{C}) \qquad\qquad -\ \textit{C is the conflict clause} \\
\quad \textbf{for each}\ p \in C\ \textbf{do} \\
\qquad \textbf{if}\ (\textit{reason}(\overline{p}) \backslash \{p\}\ \subseteq\ C) \\
\qquad\quad \text{mark p} \\
\quad \text{remove all marked literals in } C
\end{array}
$$

Preprocessing - something to consider

Ability to substantially decrease runtime significantly before actual computing stage. This is done by reducing the input CNF given to the Sat Solver

Ways to cut down CNF

- Get rid of redundancies
  - Internal variables such as MUX'es
  - Equivalent Literals

Parallelizing MiniSat Report

Parallelized design - MiniSat

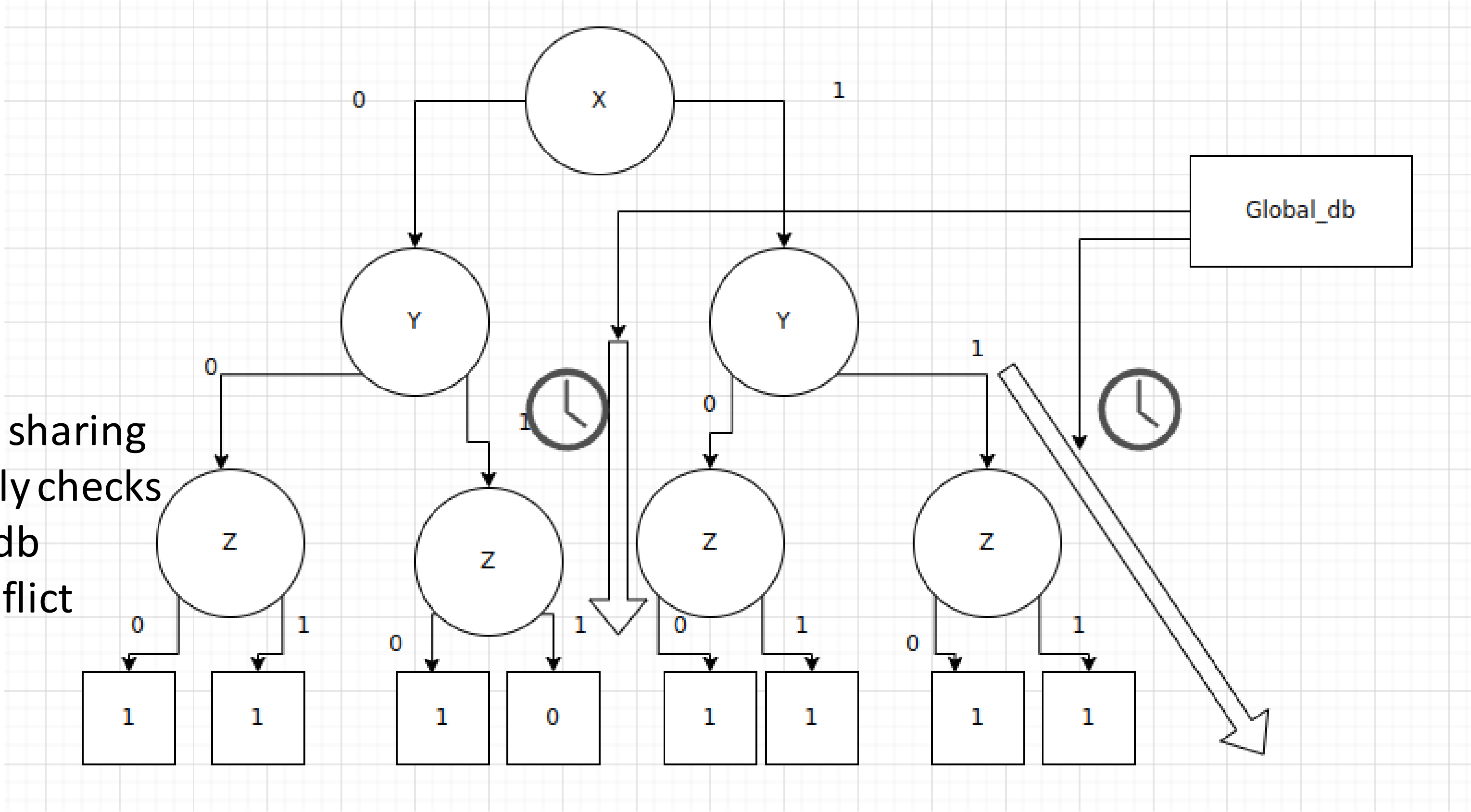
Our overall strategy is to allow the two different assignment of a assumed variable to be explored in parallel.

Convert iterative to recursive. Then parallelize recursive portion

Thread has local data collection and synchronizes with global data collection.

Works off thief stealing work. Must copy in context

Global_DB



Take advantage of sharing
Worker periodically checks back with global_db
Learns of new conflict clauses

Implementation: Iterative to recursive

Changes:

Input parameters (to include depth of search)

Return value (backtracking level)

```
int search(Solver *solver) {
  int blevel, res;
  while(res == UNDEF) {
    confl = propagation(solver);
    if(confl) {
      if(level(solver)==solver->root_level) {
        res = UNSAT;
        break;
      }
      blevel = analyze(solver,confl,&learned);

      expand_DB(solver,learned);
      cancel_until(solver,blevel);
    } else {
      next = select_next_var(solver);
      if(next == UNDEF) {
        res = SAT;
        break;
      }
      assume(solver,lit_neg(next));
    }
```

```
    confl = propagation(solver);
    if(confl) {
      backtrack = true;
      if(level(solver)==solver->root_level) {
        set_res(UNSAT);
        blevel = solver->root_level - 1;
      } else {
        blevel = analyze(solver,confl,&learned);
        expand_DB(solver,learned);
        cancel_until(solver,blevel);
      }
    } else {
      next = select_next_var(solver);
      if(next == UNDEF) {
        set_res(SAT);
        blevel = solver->root_level - 1;
        backtrack = true;
      } else {
        assume(lit_neg(next));
```

Implementation
Recursive to Parallelized

Built on Cilk - 5

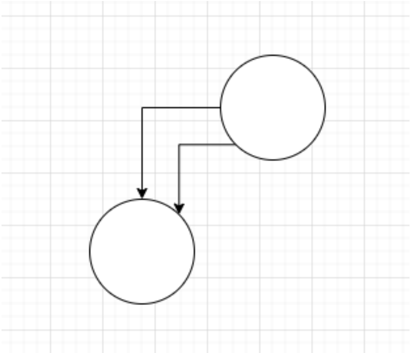
Big idea, allow two different assignment of assumed variable.

Extra parameter assume is used for building context after a theft of work

If one worker finds conflict,
it suggests that, that branch is not viable,
thus terminating the other worker.

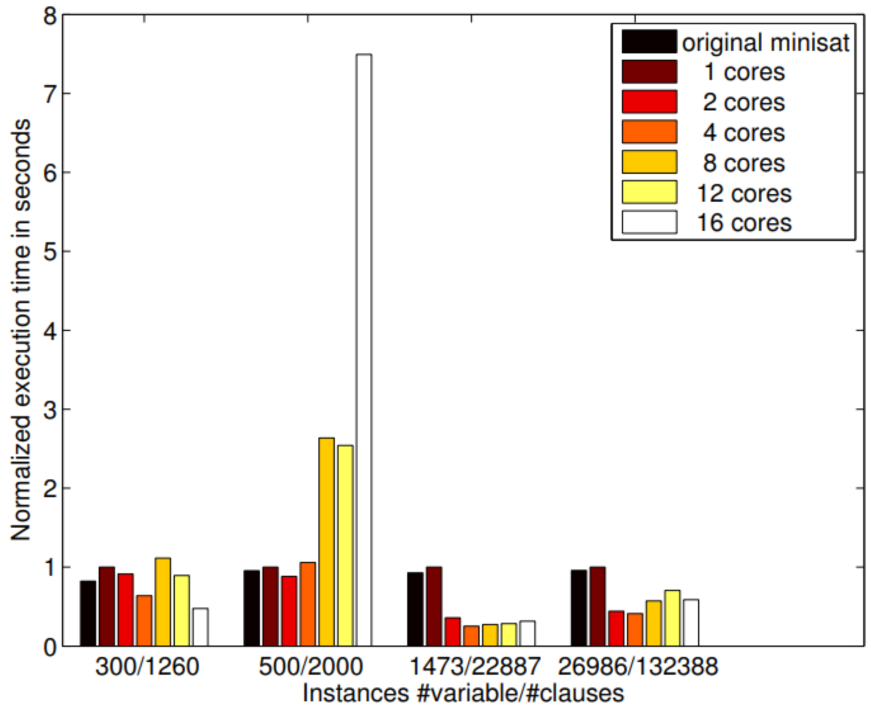
```
int search(Solver *s,int depth,var *assume) {
  int blevel = INT_INF;
  bool backtrack = false;
  var *new_assume = NULL;
  inlet void catch(int b) {
    blevel = min(blevel, b);
    abort;
  }
  while(!backtrack) {
    blevel = INT_INF;
    fetch_from_globalDB(s);
    blevel = process_fetched_clauses(s);
    // ...
    confl = propagation(s);
    if(confl) {
      backtrack = true;
      if(level(s) == s->root_level) {
        set_res(UNSAT);
        blevel = s->root_level - 1;
      } else {
        blevel = analyze(s,confl,&learned);
        post_to_globalDB(learned);
        expand_DB(s,learned);
        cancel_until(s,blevel);
      }
    } else {
      next = select_next_var(s);
      if(next == UNDEF) {
        set_res(SAT);
        blevel = s->root_level - 1;
        backtrack = true;
      } else {
        assume(s,lit_neg(next),assume);
        catch(spawn search(s,depth+1,assume));
        if( !SYNCHED && blevel == INT_INF ) {
          s = get_current_solver();
          replay(s,assume,new_assume,depth);
          // ...
          assume(s,next,new_assume);
          catch(spawn search(s,depth+1,
                             new_assume));
        }
        sync;
        if(blevel == INT_INF) {
           break;
        }
        backtrack = (blevel < depth);
        if(!SYNCHED && !backtrack) {
          s = get_current_solver();
          replay(s,assume,new_assume,depth);
        }
      }
    }
  }
  if(new_assume) free(new_assume);
  return blevel;
}
```

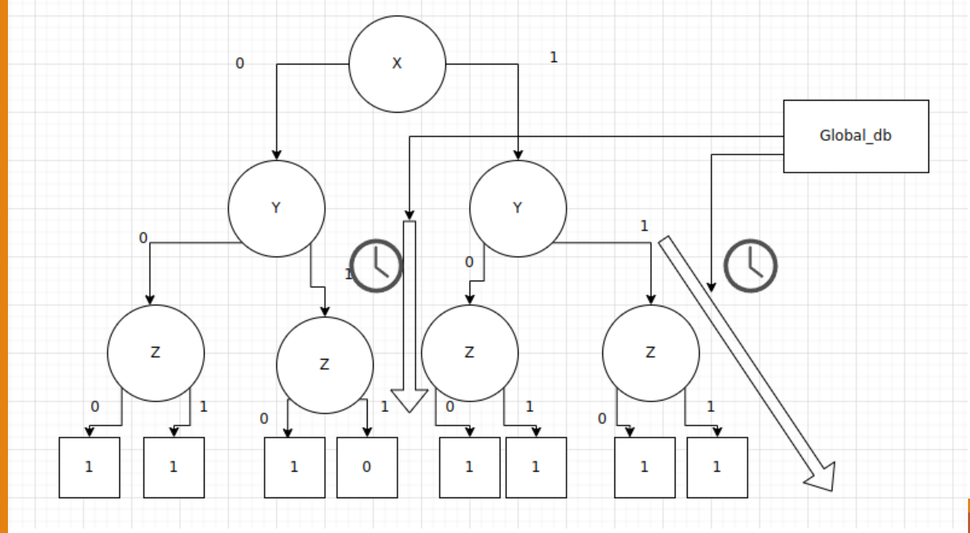
Fetch from global_db



The limitations of parallelized MiniSat Global_DB - Revisited

- Notice the similarity in performance between original minisat and 1 core paralleled minisat
- Additional overhead created by fetch synchronization overhead possibly





Takeaways

Sat problem is not a cookie cutter solution, it is depends on the input data.

Conceptually may not be difficult, but very memory and hardware intensive

Parallelism is limited by memory copying, increasing number of x processors does not directly translate to a x-fold increase