The 3-*SAT* Decision Problem
Towards a Parallel Search Implementation

Team Satisfaction

Christopher Wood, Eitan Romanoff, Ankur Bajoria

May 4, 2013

Agenda

Boolean Satisfiability

Boolean satisfiability is an *NP*-complete decision problem defined as:

$$SAT : \phi \to \{YES, NO\}$$

Input: Boolean formula ϕ_n on n variables.

$$\phi_5 = (x_1 \vee x_2 \vee \neg x_3) \vee \neg x_4 \wedge x_5 \wedge (\neg x_2 \vee x_3 \wedge (x_4 \wedge x_1))$$

Output: *YES* if there exists a truth assignment to the variables in ϕ_n such that it evaluates to true, *NO* otherwise.

$$\phi_n \text{ is satisfiable} \Leftrightarrow SAT(\phi_n) = YES$$

3-*SAT* $\in$ *NP*

- A special case of *SAT* that fixes the format of ϕ_n.
- Each input formula is in 3-*CNF* form:
    - The conjunction (Boolean AND) of arbitrarily many clauses, where each clause is the disjunction (Boolean OR) of exactly three literals (a Boolean variable or its negation).

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

- *SAT* reduces to 3-*SAT*, so 3-*SAT* $\in$ *NP*.

Exhaustive Search for 3-*SAT*

Input: 3-*CNF* formula ϕ_n on n variables, **Output:** YES or NO

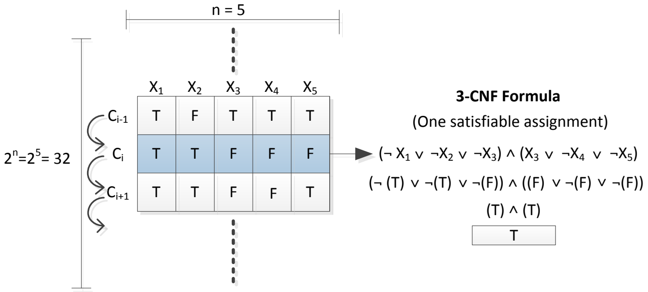
$C \leftarrow FALSE^n$ (array of n False values, the initial configuration)
for $i = 0 \rightarrow 2^n - 1$ **do**
    $SAT \leftarrow TRUE$
    for all $clause \in \phi_n$ **do**
        if $evaluate(clause, C) = FALSE$ **then**
            $SAT \leftarrow FALSE$
        end if
    end for
    if $SAT = TRUE$ **then return** YES
    $C \leftarrow nextConfig(C)$
end for
return NO

Exhaustive Search for 3-*SAT* - A *Very Satisfiable* Example!

Input: $\phi_5 = (\neg X_1 \vee \neg X_2 \vee \neg X_3) \wedge (X_3 \vee \neg X_4 \vee \neg X_5)$

Output: Yes

Note: no early termination once a satisfiable solution is found



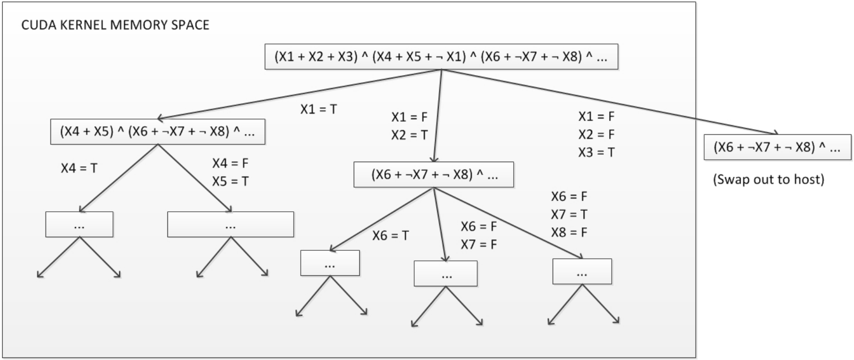
Evaluating a Clause

- Evaluating a clause depends on how ϕ_n and the variable truth assignments are stored.
  - `boolean[] variables` for variable assignments and `Literal[][3] formula` for ϕ_n.
  - A `Literal` has a variable ID and negated flag

```
for (int c = 0; c < numClauses; c++) {
    boolean clauseValue = false;
    for (int l = 0; l < 3 && clauseValue == false; l++) {
        if (formula[c][l].negated == true && !variables[formula[c][l].id])
            clauseValue = true;
        else if (formula[c][l].negated == false && variables[formula[c][l].id])
            clauseValue = true;
    }
    // Check the value of the clause now...
}
```

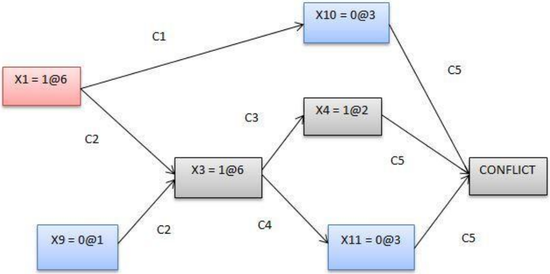
Recursive Divide and Conquer with CUDA

- TODO: overview



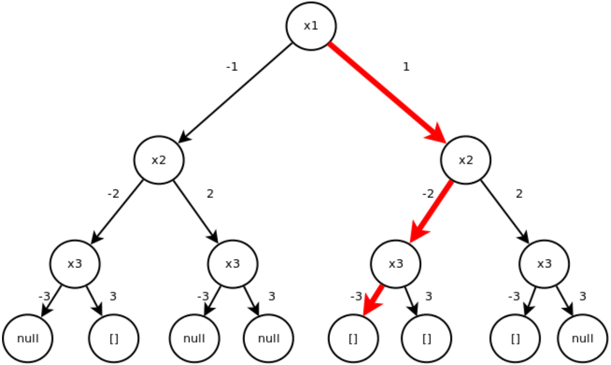
Advanced Clause Learning and Sharing with ManySAT

- TODO: overview



Intelligent Literal Decisions and Advanced Data Structures for DPLL

- TODO: overview



(x1 ^ ~x2 ^ ~x3)

Parallel Implementation Design Goals

Our design goals included:

- Evenly divide the configuration space among different threads
- Minimize (or remove) conlficts for shared variables

Our solution consists of:

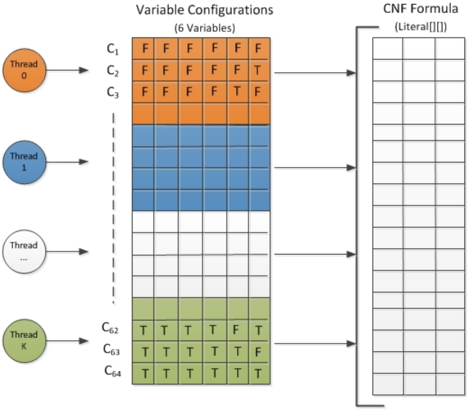
-

The Parallel Transformation

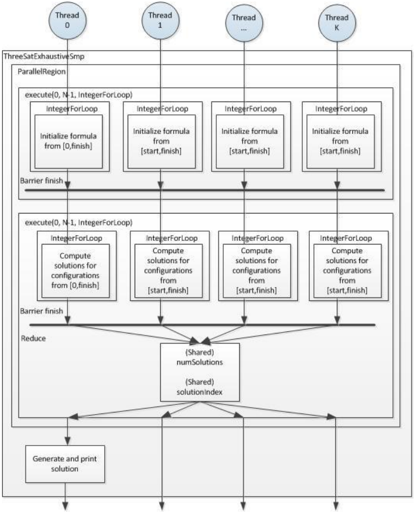
Managing variable access

Sequential Variable	Parallel Variable	Shared?
Literal[][]		
configuration[]		
numSatisfiable		

Computation Partition Strategy



Thread Synchronization

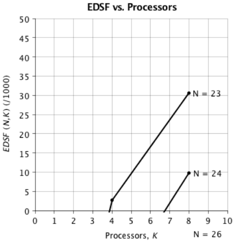


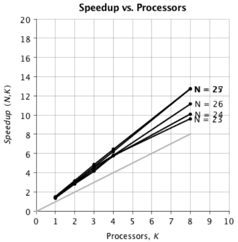
The Parallel Solver

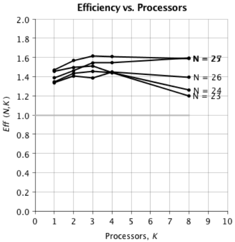
Demo time!

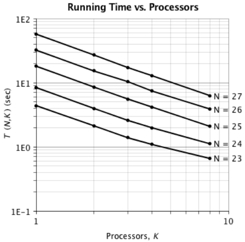
$$\phi_5 = (\neg X_1 \vee \neg X_2 \vee \neg X_3) \wedge (X_3 \vee \neg X_4 \vee \neg X_5)$$

Speedup Metrics (Exhaustive)

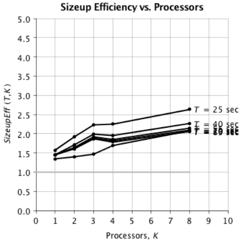


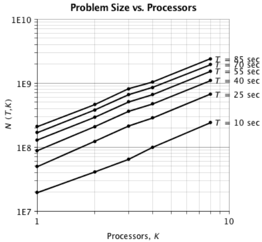


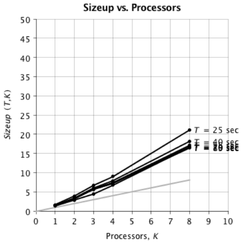


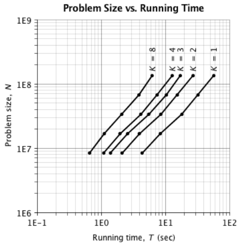


Sizeup Metrics (Exhaustive)









Performance Observations

- We achieved *superlinear* speedups and sizeups using a guided schdule with varying the number of variables
- Exaustive 3-SAT problems have implicit unbalanced loads
  - A guided schedule yielded the best results for speedup and sizeup
  - A dynamic schedule caused a *significant* amount of overhead
- Our parallel programs achieve better performance when the number of variables is varied:
  - The problem size $N = f(N_v, N_c) = 2^{N_v} \times N_c$

Lessons Learned

- If the problem size is a function of *multiple* variables, experiments should only change one of such variables to gather valid performance data
- A guided schedule yielded the most balanced load for our computation partition strategy
- TODO
- TODO

Future Work

- Implement more advanced heuristics for literal selection
- Strive for wider splits of the configuration search space tree among multiple processes
- Experiment with different data structures to see what's the most optimal

Questions?

Fire away!