SAT Solver

Frederik Hennecke, Pascal Brockmann

01.07.2024

Problem Question

How can we use MPI to parallelize SAT solvers?

Problem Description

- The Boolean Satisfiability Problem (SAT)

Problem Description

- The Boolean Satisfiability Problem (SAT)
- Problem: How to determine whether an assignment of truth values exists in a given boolean formula such that the entire formula evaluates to true?

Problem Description

- The Boolean Satisfiability Problem (SAT)
- Problem: How to determine whether an assignment of truth values exists in a given boolean formula such that the entire formula evaluates to true?
- Example:

$$(x_1 \lor \neg x_2) \land (\neg x_1 \lor x_2 \lor x_3) \land (\neg x_1 \lor x_4)$$

Problem Description

- The Boolean Satisfiability Problem (SAT)
- Problem: How to determine whether an assignment of truth values exists in a given boolean formula such that the entire formula evaluates to true?
- Example:

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

- Problem is NP-Complete

Library

- Can be compiled with CMake and Make
- Executed with *./SATMPI -a algorithm -f filepath*
- Reads in files in DIMACS format:

```
c this is a comment
p cnf 4 3
 -1 -2  0
-1  2  3  0
-1  4  0
```

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

Overview: Algorithms

- Bruteforce
- Davis-Putnam algorithm
- Davis–Putnam–Logemann–Loveland (DPLL)
- Conflict-driven clause learning (CDCL)

Algorithms: Bruteforce

1. Assign values to literals

Algorithms: Bruteforce

1. Assign values to literals
2. Check if the assignment solves the formula

Algorithms: Bruteforce

1. Assign values to literals
2. Check if the assignment solves the formula
3. Yes: The Formula is solvable, return true

Algorithms: Bruteforce

1. Assign values to literals
2. Check if the assignment solves the formula
3. Yes: The Formula is solvable, return true
4. No:
    1. Assignments left: Go to step two
    2. No assignments left: return false

Algorithms: Bruteforce

1. Assign values to literals
2. Check if the assignment solves the formula
3. Yes: The Formula is solvable, return true
4. No:
    1. Assignments left: Go to step two
    2. No assignments left: return false
5. Worst case: $O(2^n)$satisfiability checks because we have 2^n different binary numbers

Algorithms: Bruteforce Example

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

a	b	$((a$	$\vee$	$(\neg$	$b))$	$\wedge$	$((\neg$	$a)$	$\vee$	$b))$
0	0	0	1	1	0	1	1	0	1	0
1	0	1	1	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	1
1	1	1	1	0	1	1	0	1	1	1

The assignment can be seen as a binary number, which can be increased by 1 in each step.

Algorithms: Bruteforce Parallel

- Algorithm: Split the binary numbers into N parts

Algorithms: Bruteforce Parallel

- Algorithm: Split the binary numbers into N parts
- Worst case: $O(2^n/N)$ satisfiability checks because we have 2^n different binary numbers which get split into N parts

Algorithms: Davis-Putnam

1. Multiple satisfiability-checks

Algorithms: Davis-Putnam

1. Multiple satisfiability-checks
2. Choose literal

Algorithms: Davis-Putnam

1. Multiple satisfiability-checks
2. Choose literal
3. Create new clauses
   - Example: chosen literal a

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

Algorithms: Davis-Putnam

1. Multiple satisfiability-checks
2. Choose literal
3. Create new clauses
   - Example: chosen literal a

$$(a \lor \neg b \lor c) \land (\neg a \lor b \lor d)$$
$$\Downarrow$$
$$(\neg b \lor c \lor b \lor d)$$

4. Remove clauses containing chosen literal

Algorithms: Davis-Putnam

1. Multiple satisfiability-checks
2. Choose literal
3. Create new clauses
   - Example: chosen literal a

$$(a \lor \neg b \lor c) \land (\neg a \lor b \lor d)$$
$$\Downarrow$$
$$(\neg b \lor c \lor b \lor d)$$

4. Remove clauses containing chosen literal
5. Repeat

Algorithms: DP Parallel

- Not implemented yet

Algorithms: DP Parallel

- Not implemented yet
- How to parallelize:
  - Parallelize satisfiability-checks for clauses
  - Choose multiple different literals at the beginning

Algorithms: Davis-Putnam-Logemann-Loveland

- Basic idea similar to bruteforce

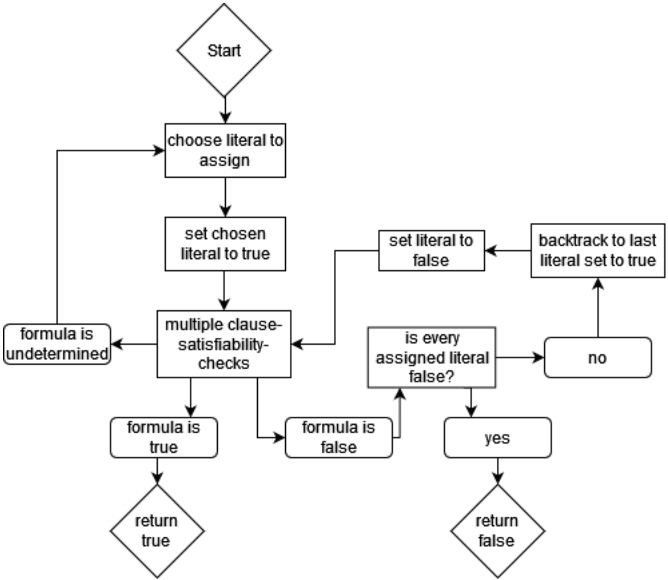
Algorithms: Davis-Putnam-Logemann-Loveland

- Basic idea similar to bruteforce
- But literals will be assigned one after another

Algorithms: Davis-Putnam-Logemann-Loveland

- Basic idea similar to bruteforce
- But literals will be assigned one after another
- Backtracking

Algorithms: DPLL-procedure



Algorithms: DPLL Parallel

- Already implemented

Algorithms: DPLL Parallel

- Already implemented
- Each process starts with assigned literals

Algorithms: DPLL Parallel

- Already implemented
- Each process starts with assigned literals
- Example: 2 processes
  - The First process starts with the first variable set to true
  - The Second process starts with the first variable set to false

Algorithms: CDCL

- Extends the DPLL algorithm:

Algorithms: CDCL

- Extends the DPLL algorithm:
- Conflict Analysis: When a conflict (i.e., a clause that cannot be satisfied) is detected, it identifies a set of decisions (variable assignments) that led to the conflict.

Algorithms: CDCL

- Extends the DPLL algorithm:
- Conflict Analysis: When a conflict (i.e., a clause that cannot be satisfied) is detected, it identifies a set of decisions (variable assignments) that led to the conflict.
- Clause Learning: Derives a new clause that prevents the same conflict from occurring in the future. This clause is added to the formula, reducing the search space.

Algorithms: CDCL

- Extends the DPLL algorithm:
- Conflict Analysis: When a conflict (i.e., a clause that cannot be satisfied) is detected, it identifies a set of decisions (variable assignments) that led to the conflict.
- Clause Learning: Derives a new clause that prevents the same conflict from occurring in the future. This clause is added to the formula, reducing the search space.
- Backjumping: CDCL uses non-chronological backjumping; it jumps back directly to the most recent decision point that is relevant to the conflict.

Algorithms: CDCL

- Extends the DPLL algorithm:
- Conflict Analysis: When a conflict (i.e., a clause that cannot be satisfied) is detected, it identifies a set of decisions (variable assignments) that led to the conflict.
- Clause Learning: Derives a new clause that prevents the same conflict from occurring in the future. This clause is added to the formula, reducing the search space.
- Backjumping: CDCL uses non-chronological backjumping; it jumps back directly to the most recent decision point that is relevant to the conflict.
- Decision Heuristics: tries to select literals that are more likely to lead to conflicts.

Algorithms: CDCL

- Extends the DPLL algorithm:
- Conflict Analysis: When a conflict (i.e., a clause that cannot be satisfied) is detected, it identifies a set of decisions (variable assignments) that led to the conflict.
- Clause Learning: Derives a new clause that prevents the same conflict from occurring in the future. This clause is added to the formula, reducing the search space.
- Backjumping: CDCL uses non-chronological backjumping; it jumps back directly to the most recent decision point that is relevant to the conflict.
- Decision Heuristics: tries to select literals that are more likely to lead to conflicts.
- Restarts: Periodically restarts the search with the learned clauses retained. This helps to escape from difficult regions of the search space, and *often* leads to faster convergence.

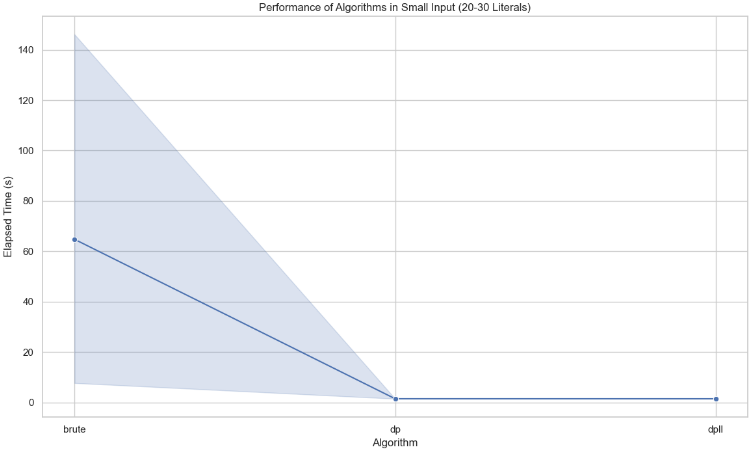
Algorithms: CDCL

- Sequential CDCL is implemented but still quite slow. Needs more time.

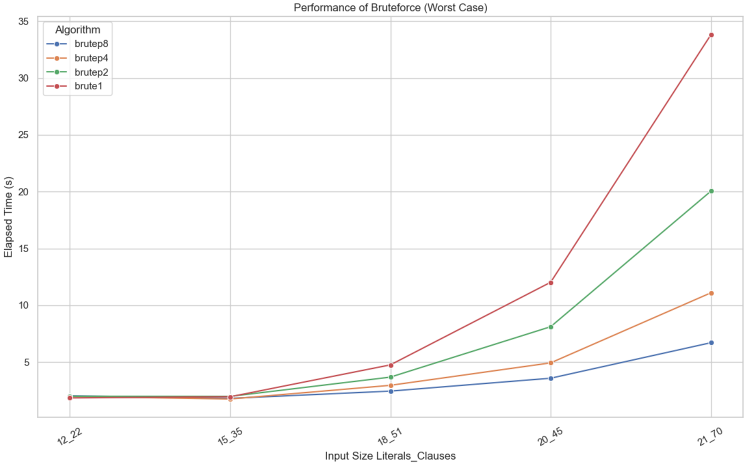
Algorithms: CDCL

- Sequential CDCL is implemented but still quite slow. Needs more time.
- Idea for parallelization:
  - Partition the search space into disjoint segments and assign each segment to a different process.
  - Share learned clauses among processes

Performance Analysis



Performance Analysis



Performance Analysis

Next Steps

- DP Parallel

Next Steps

- DP Parallel
- improve CDCL

Next Steps

- DP Parallel
- improve CDCL
- CDCL Parallel

Next Steps

- DP Parallel
- improve CDCL
- CDCL Parallel
- In-depth Performance Analysis

Conclusion

- Several algorithms are already implemented

Conclusion

- Several algorithms are already implemented
- The parallel algorithms are almost always faster than their sequential counterparts

The End

Questions?