Understanding SMT solvers

Exploring Parallelization in Floating-Point Problems

Responsible Professor: Soham Chakraborty
Supervisor: Dennis Sprokholt

Tristan Schmidt
tcrschmidt@tudelft.nl

What optimizations are possible by solving floating-point SMT problems in parallel?

1. Introduction

- The solving of floating-point SMT problems is of great use in software verification.
- Many solvers exist, but all have their own strengths, weaknesses and approach when it comes to solving a problem. Solvers usually have a subset of problems where they perform well.
- Parallelisation is a simple optimisation that is possible on many CPUs. It is sometimes used internally, but not often between different solvers.

2. Background

- SMT (Satisfiable Modulo Theories) problems question whether a mathematical formula is satisfiable, that being, that is, if there is an arrangement of values such that the said formula is true.
- SMT solvers try to find solutions for these problems. The most common approach to solve these problems is to transform equations of a specific data type into a boolean satisfiability problem.
- SMT problems involve many different data types, including floating-point, and are either satisfiable or unsatisfiable.

A simple example (written in the SMT-LIB specification) of an unsatisfiable problem: Find a variable x that is not equal to itself:

```
(declare-fun x () (_ FloatingPoint 11 53))
(assert (not (= x x)))
(check-sat)
```

3. Floating-Point Solvers

- CVC5 and Z3: Use bit-blasting, which transforms every floating-point operation of an equation, into its equivalent bit vector operation. It eagerly converts to bit vector equation to a propositional formula. Both use CDCL (Conflict Driven Clause Learning) to find a solution.
- Bitwuzla: Solver for specifically bit vector and floating-point problems. Supports bit-blasting, as well as propagation based local-search.
- GoSAT: Converts floating-point problems to a real number equation, from which it tries to minimize the output of the equation to find a solution using mathematical optimization.

4. Parallel Solver

- The four mentioned solvers are ran in parallel to optimize solving speed.
- Six diverse benchmarks are ran from the 2018 SMT-COMP.

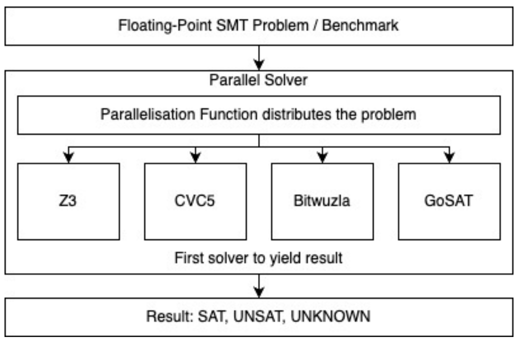

Fig 1. The parallel solver

4. Results

Model	griggio	schanda	ramalho	vector	automizer2019	wintersteiger	Speedup
bitwuzlaprop	7923.54s	1461.23s	1561.95s	**13.32s**	**2.34s**	x	2.30/2.30
cvc5	4985.38s	370.45s	1181.67s	15.90s	2.68s	**3817.71s**	1.35/1.35
gosat	775.48s	~~2580.75s~~	~~2160.00s~~	~~5460.00s~~	~~1440.00s~~	~~1198390.46s~~	29.65/1.37
bitwuzla	4697.98s	200.75s	**1087.13s**	14.19s	2.39s	3839.68s	1.14/1.14
z3	9394.44s	330.98s	1203.06s	18.33s	427.09s	3864.78s	3.52/3.52
parallel	**564.73s**	**166.95s**	1158.85s	27.71s	2.92s	6770.50s	1.00/1.00
Speedup	7.49/7.49	3.72/2.61	1.20/1.08	1.79/0.55	8.45/3.06	2.38/0.57	

Fig 2. Time spent solving for each benchmark

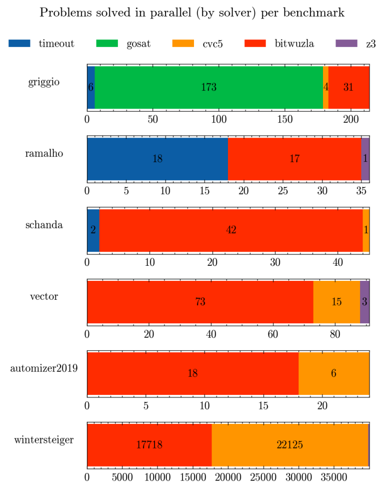

Fig 3. Amount of problems solved in parallel

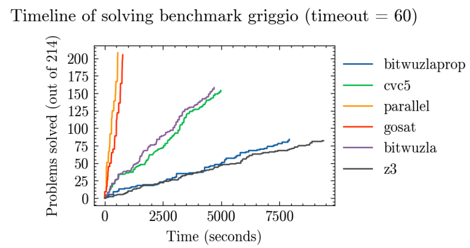

Fig 4. Results of the griggio benchmark

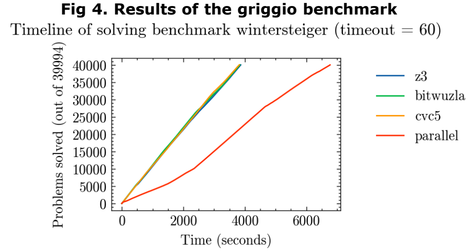

Fig 5. Results of the wintersteiger benchmark

5. Conclusion

- Running solvers in parallel provides a significant speedup when solving more complex problems.
- Parallel solving does not speed up the solving of small problems, it rather provides extra overhead.
- Although most benchmarks are dominated by one solver, other solver still provide a significant reduction in solving time.
- In the future, more solvers could be ran in parallel.