Feasibility Study on the SAT Solver on DataFlow Architecture

Zivojin Sustran, Zoran Ognjanovic, Milan Todorović and Veljko Milutinovic

Abstract — The SAT problem is the first known example of NP- complete problems. This means that there is no known efficient algorithm for solving this problem, and it is mostly believed, yet still not proven, that such algorithm doesn't exist.

This research tries to explore possibility for implementing a SAT solver on the DataFlow architecture. High degree of independency between calculations, used in solving SAT problems, is a reason for possible speed-up on the DataFlow architecture.

Index Terms — SAT problem, DPLL procedure, SAT solvers, DataFlow

I. Introduction

Boolean satisfiability, or SAT for short, is a problem of determining whether there is an assignment for the variables (valuation) for a given propositional formula, that makes the formula true. This problem is one of the most famous NP-complete problems, which was independently shown by Cook [1] and Levin [3]. The consequence of this is that, unless $P = NP$, all complete algorithms for solving SAT problem require exponential time in the worst-case scenario. Despite this fact, modern SAT solvers are capable of handling large problem instances due to improvements made to them in the last 15 years. This allows their application to a wide range of practical problems [4], such as hardware and software verification, constraint satisfaction, planning, etc. These problems can be solved by translating into SAT, and although translating a problem to SAT usually produces larger representation than the original, it is often more efficient to do that and to use modern SAT solvers, than to use solvers specialized for the problem's initial representation [5].

SAT solvers can be divided into two groups, complete and stochastic. For a given propositional formula, complete SAT solvers guarantee to either find valuation that satisfies the formula or to determine that such valuation does not exist. On the other hand, stochastic solvers can prove that formula is satisfiable, often faster than complete solvers, but they can't prove that formula is unsatisfiable [7].

II. DPLL procedure

Most modern complete SAT solvers are based on DPLL (Davis-Putnam-Logemann-Loveland) procedure [2]. The procedure uses backtrack search and some additional rules for increasing efficiency. As input, the procedure expects formula in conjunctive normal form, and as output gives YES if formula is satisfiable and NO otherwise. As conjunction and disjunction are commutative and associative, the order of clauses in an input formula is not important and formulas can be represented as a set of clauses which can be, similarly, represented as sets of literals (variable or negation of variable). The procedure assumes that empty set of clauses (empty formula) is satisfiable, and that empty set of literals (i.e., empty clause) is unsatisfiable, which makes the formula that contains such clause unsatisfiable.

The main step of the procedure - branching on literals- consists of selecting a literal l that occurs in the formula F. Then l is replaced with $\top$. After that, the new formula is simplified, and it's satisfiability is examined recursively. If it is proven to be satisfiable, the starting formula F is also satisfiable. Otherwise, l is replaced with $\bot$ in F, and the process is repeated. If this formula is proven to be satisfiable, F is also satisfiable, but if it is proven to be unsatisfiable, then the original formula F is also unsatisfiable. Although the strategy for selection of literals has no impact on procedure's correctness, it has great impact on its efficiency.

Two additional rules, that are not required for completeness of procedure, are used to increase efficiency of this search. The first rule is called unit clause, and it is based on the fact that if a clause in the formula consists of only one literal, that literal must be true if the formula is to be satisfied. The second rule, pure literal, is based on the fact that if literal occurs in a formula, and its negation doesn't, then all clauses that contain the literal can be erased without affecting the affect satisfiability of the formula. Both of these rules eliminate the need for branching on those literals. The original DPLL procedure is shown in Fig. 1.

Most of the modern, state-of-the-art, SAT solvers, like PICOSAT, MiniSAT, SATO, CHAFF and others, use DPLL procedure with many additional features that are introduced to increase performance. In the original DPLL procedure, argument is a formula that is modified through recursive calls. This is inefficient for larger instances, so the first feature would be to maintain the current partial valuation, to keep the formula in the original form and only check its value against the current valuation. Also, the procedure is usually implemented in an iterative way rather than recursive, which reduces memory usage [5].

III. Problem Statement

The main goal of this research is to create a prototype SAT solver using the Maxeler DataFlow Engine MPC-N40 [10].

This work was supported by the Serbian Ministry of Sciences, under the project III 44006.

Z. Sustran is with University of Belgrade, Serbia (e-mail: zika@etf.bg.ac.rs).
M. Todorović is with the Mathematical Institute is with Serbian Academy of Sciences and Arts, Serbia.
V. Milutinovic is with University of Belgrade, Serbia (e-mail: vm@etf.bg.ac.rs).

```
function DPLL (F :formula)
begin
  if F is empty then return YES;
  replace all ¬⊥ with ⊤ and ¬⊤ with ⊥ in F;
  remove all ⊥ from F;
  if F contains empty clause then return NO;
  if ∃ clause c s.t. ⊤ ∈ c or l, ¬l ∈ c
    then return DPLL(F \ {c});
  if ∃ unit clause {l} in F
    then return DPLL(F [l → ⊤]);
  if ∃ pure literal l in F
    then return DPLL(F [l → ⊤]);
  begin
    select a literal l from F;
    if DPLL(F [l → ⊤]) = YES then return YES;
    else return DPLL(F [l → ⊥]);
  end
end
```

Fig. 1. DPLL procedure.

Existing SAT solvers are implemented for general purpose processors with ControlFlow architecture. ControlFlow processors are usually optimized for integer instructions. Calculating Boolean expressions using integer instructions is slow. The idea is to transfer Boolean expressions to the MPC-N40 and accelerate their evaluation.

The prototype solver will show how much acceleration can be accomplished using DataFlow architecture for certain set of SAT problems. The set of SAT problems solvable by the prototype solver contains SAT problems with limited number of variables. Upper bound of the number of variables is determined by hardware limits.

Determining of how much speed-up is possible with DataFlow architecture is important because it will give us insight whether the prototype solution is possible path toward creation of a useful industry SAT solver on DataFlow architecture. If speed-up is big, it is probably possible to make the useful industry SAT solver by reducing some constraints used in the prototype solver. Reducing some constraints will reduce speed-up of the prototype solver, but if that reduction is smaller than initial speed-up feasible speed-up solution is possible.

In the near future DataFlow architecture will be developed faster than ControlFlow architecture [12] and its hardware will have much more capabilities. Currently clock speed used in hardware is 200 MHz. Hardware with much higher clock speed was made available at the market recently [11]. Use of this hardware for implementing DataFlow architecture can additionally accelerate SAT solver.

IV. The Proposed Solution

The proposed solution tries to solve a SAT problem using a simple algorithm. The simple algorithm for every possible assignment of input variables evaluates the given propositional formula. When evaluates the given propositional formula to true, it returns the variable assignments. Number of calculation steps exponentially depend on the number of input variables. This is also true for all existing complete solvers in the worst case.

Because there is no true data dependency in calculations for one possible assignment of input variables on DataFlow architecture, they can be successfully pipelined and results can be output each clock cycle [13]. Calculation of one possible assignment of input variables on the ControlFlow architecture requires several machine instructions. Accounting for latency generated by cache misses, required time for one calculation can be measured in thousands of clock cycles in the worst case scenario. This is relatively big advantage for a SAT solver on the DataFlow architecture. Since calculations are independent, both architectures can speed them up by means of parallel execution. On ControlFlow architecture solver can be parallelized by the number of executions determined by processor threading capabilities. On DataFlow architecture solver can be parallelized by the number of executions determined by hardware used for implementation. The number of execution on DataFlow architecture is the same or greater than the number of execution on ControlFlow architecture, if hardware used for both architectures is in the same price range.

Best way to determine whether proposed solution is better than the existing solutions is to implement them and empirically compare them. Comparison is done on different hardware used for both architectures, keeping hardware in the same price range. Comparison is done for a set of SAT problems that can be solved on proposed solutions. Since the set of SAT problems is relatively large, only the worst case scenario is considered. This will give the upper bound for time of execution on both architectures.

V. The Details of Proposed Solution

The proposed solution is made for a simple SAT solving algorithm. The simple algorithm can be divided into two phases: generation of input variables and calculation of a Boolean propositional formula. Those two phases are directly implemented on DataFlow architecture. In order to make execution as fast as possible, multiple calculation units are inserted into second phases. Generation phase distributes input variables to all parallel calculation units. Since every calculation unit generates a result every clock cycle, the third phase is incorporated into a simple algorithm for processing of results. Transfer of results from DataFlow engine to host requires data in specific format, which is different from format used for calculation. This conversion makes the forth and final phase incorporated into a simple algorithm. The complete visualization of algorithm is presented in Fig. 2. Detailed explanation of each phase is in the following subsections.

A. Generation Phase

Generation phase is used to deliver input variables to the calculation phase. Since all input data are generated on DataFlow engine, there is no slow transfer operation of data from host. This means that calculations can begin as soon as DataFlow engine is initialized.

Every possible variables assignment has to be generated. For this purpose a simple unsigned integer counter is used. The counter is initialized to zero and on each clock cycle it is incremented. Every bit of the counter value represents one

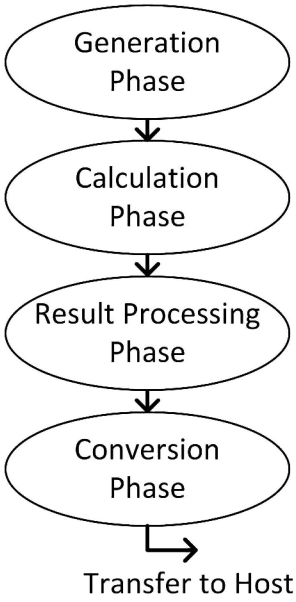

Fig. 2. The simple algorithm for solving SAT problem.

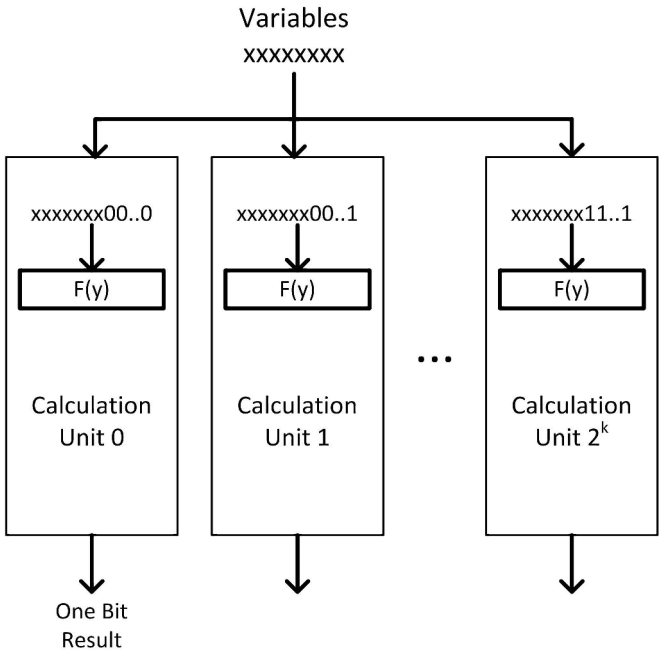

Fig. 3. Calculation phase.

variable. When counter reaches maximum value, every possible combination of input variables is generated. The width of counter value is lower than the number of values in a Boolean propositional formula. Since there are multiple calculation units, each unit has its own private set of assignments for several variables that are not delivered by the generation phase. The delivery of counter value to multiple calculation units can become a problem, if number of calculation units becomes large. The problem is in complicated routing of wires needed to deliver the counter value to each calculation unit. One solution to this problem is to use multiple counters in generation phase. This reduces routing problem with use of redundant counters. The redundant counters do not represent a problem, if relatively small number of counters is used. In this solution the number of counters is kept small, by creating counter for a set of calculation units for which routing is not a problem.

B. Calculation Phase

The calculation phase is used to calculate a given Boolean propositional formula. The formula is directly implemented on DataFlow engine. For different formula recompilation source code is necessary. Compilation can take a lot of time. Calculation of the formula is pipelined so result is produced each clock cycle, independent of the complexity of the formula. There is no true data dependency between different calculations, so the pipeline, after the initial fill phase, is always kept full, till the end of all calculations. This means that, there is no need for stall and there will not be bubbles in the pipeline. The pipeline is maximally utilized all the time.

Calculation phase is divided into multiple calculation units, as illustrated in Fig. 3. Every calculation unit in parallel calculates the given formula and outputs the result. The result is one bit representing whether the formula is satisfied for the set of input variables. The number of calculation units is power of two. That makes easier job distribution over different units. If the number of variable is n and the number of units is 2^k, the generation phase generates m - k global variables and each calculation unit has k private variables. Global variables are the same for each calculation unit. Private variables for each calculation units are unique. The calculation unit output result using both global and private variables. The result is sent to the result processing phase.

C. Result Processing Phase

The Result Processing Phase receives each clock cycle 2^k results. The phase's job is to determine whether there is one result with value one (not important which one). Results processing can be presented as a binary tree of logical OR gates, as depicted in Fig. 4. The results are inserted into tree leaves and the tree root gives answer whether the formula is satisfiable for the input variables assignment. If the formula is satisfiable, the variable assignment is output to the conversion phase.

Result processing phase determines maximum level of parallelization. Routing of wires needed to implement phase function is the most complex task that determines whether the solver is possible to implement on DataFlow engine. Routing complexity is determined by the number of results entering the phase each clock cycle. If routing complexity can be circumvented, the maximum level of parallelization will be determined by the maximum number of the calculation units that can be placed on DataFlow engine.

D. Conversion Phase

The conversion phase receives variables from the result processing phase, that satisfies the formula. Variables are needed to be converted to different format, so they can be transferred to the host. Each variable is transformed to the

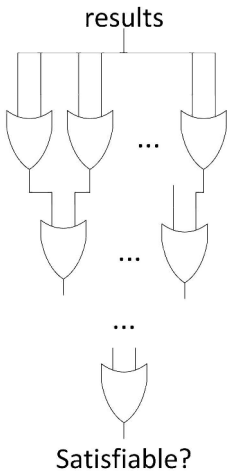

Fig. 4. Result processing phase.

size of one machine word. Machine word is much bigger than one bit thus unnecessary data is being transferred to the host. Time to transfer data linearly depend on the number of variables in the formula, so this latency in not dominant one and further optimization of transfer in not necessary. If there is no assignment that satisfies the formula, nothing is being transferred to the host. When this situation is detected on the host, the formula is declared unsatisfiable.

VI. Analysis of Proposed Solution

Proposed solution is implemented for specified DataFlow engine and existing solution is implemented on ControlFlow processor. Both versions are compared for worst case scenario. Details of comparison are in following subsections.

A. Conditions and Assumptions

Existing solution is implemented on two high-end server processors with use of 16 threads. Each thread has hardware support for execution and does not share processing resources with other processors. The algorithm is implemented to the best of author knowledge.

Proposed solution is implemented on the Maxeler DataFlow Engine MPC-N40. The number of computation units is 2^{11}. There is space on the DataFlow engine for more computation units, but it is impossible to connect them due to routing problem discussed in previous sections. The set of solvable SAT problems is limited by maximum number of variables, 42. Raising the number of variables will slow down calculations by factor of two for each new variable.

Time needed for execution on DataFlow engine can be predicted mathematically. Each computation unit will get 2^{31} different inputs variables assignment, so it will take at least same number of clock cycles to output result in the worst case scenario. Time to execute will be at least 11s. Time to start DataFlow engine and time to transfer data to the host is not included in the previous analysis because it can be considered constant. Time needed for execution on ControlFlow processor will not be predicted here.

B. Result of the Analysis

Execution of the existing SAT solver on ControlFlow architecture it timed to be in the range of 40000s. Execution of the prototype SAT solver on DataFlow architecture is in the range of 30s. The speed-up accomplished by using DataFlow architecture is over one thousand.

VII. Conclusion

SAT problem is well known, and one of the most important NP-complete problem. In the last two decades, considerable progress, both practical and theoretical, has been made in the study of this problem. Many SAT solvers have been developed that can solve challenging satisfiability problems [6]. This allows encoding of different problems from various domains into SAT: hardware verification and fault diagnostic, planning in artificial intelligence, machine-shop scheduling, haplotyping in bioinformatics and graph coloring [4], [9]. On the theoretical side, techniques developed for SAT solvers can be adapted and extended for usage in solvers for other, similar, problems.

The implemented prototype SAT solver greatly overcomes the current implementations of SAT solvers on ControlFlow architecture, for certain set of SAT problems. This leads us to say that research in this direction might be fruitful and successful.

There are two problems that arise from previous discussion. First is programmability of the prototype solver. The prototype solver is made only for one Boolean propositional formula. Creation of such prototype requires time not included into this analysis. There is a possibility to send configuration data to DataFlow engine during initialization to modify formula being calculated. This will additionally complicate the implementation and will slow down the solver. If this slowdown is less than thousand times, better SAT solver will be created than on the existing solutions. Second problem is the number of maximum variables in the formula. The maximum number is too small for use in any practical situations. There are no indications whether the maximum number of variables can be higher. If there is no possibility for that, maybe some hybrid solution can be made, utilizing the best from both worlds.

References

[1] Stephen Cook, "The Complexity of Theorem-Proving Procedures," *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pp. 151-158 (1971)

[2] Martin Davis, George Logemann, Donald Loveland, "A machine program for theorem-proving," *Communications of the ACM*, 1962, Vol.9, Num.7, pp. 394-397

[3] Leonid Levin, "Universal Sequential Search Problems," *Problems of Information Transmission*, 1973, Vol.9, Iss 3, pp. 256-266 (1973)

[4] Marques-Silva, J., "Practical Applications of Boolean Satisfiability," *International Workshop on Discrete Event Systems*, WODES 2008, pp. 74-80

[5] Carla Gomes, Henry Kautz, Ashish Sabharwal, Bart Selman, "Satisfiability Solvers," *Handbook of Knowledge Representation*, Elsevier B.V., pp. 89-122

[6] Holger H. Hoos, Thomas Sttzle, "SATLIB: An Online Resource for Research on SAT" *In: I.P.Gent, H.v.Maaren, T.Walsh, editors, SAT 2000*, pp.283-292, IOS Press, 2000.

[7] Mladen Nikolić, "Metodologija izbora pogodnih vrednosti parametara SAT rešavača," *Magistarska teza*, Matematički Fakultet, Univerzitet u Beogradu, 2008.

[8] Filip Marić, Predrag Janičic, "Formalization of abstract state transition systems for SAT," *Logical Methods in Computer Science* Vol. 7 (3:19) 2011, pp. 1-37

[9] Ian Gent, Toby Walsh, "The search for satisfaction," *Department of Computer Science University of Strathclyde*, Scotland (1999).

[10] Maxeler Techologies. *http://www.maxeler.com/products/mpc-nseries/,* April 2013

[11] Achronix Semiconductor Corporation. "Speedster22iHD FPGA Family." *http://www.achronix.com/wp-content/uploads/docs/* Speedster22iHD FPGA Family DS004.pdf, April 2013

[12] Flynn, M., Mencer, O., Milutinovic, V., Rakocevic, G., Stenstrom, P., Trobec, R., Valero, M., "Moving from petaflops to petadata", *Communications of the ACM*, ACM, New York, NY, USA, Volume 56, Issue 5, May 2013, pp. 39-42.

[13] -, "Multiscale Dataflow Programming," *Maxeler Techologies*, Version 2012.2, December 2012