Implementing CSAT Local Search on FPGAs

Martin Henz, Edgar Tan, and Roland H.C. Yap

School of Computing, National University of Singapore, Singapore

Abstract. Stochastic local search methods such as GSAT, WalkSAT and their variants have been used successfully at solving propositional satisfiability problems (SAT). The key operation in these local search algorithms is the speed of variable flipping. We present a parallel FPGA designs for CSAT capable of one flip per clock cycle which is achieved by exploiting maximal parallelism and "multi-try" pipelining. Experimental results show that a speedup of two orders of magnitude over software implementations can be achieved.

1 Introduction

Stochastic local search (SLS) methods have been used successfully for solving satisfiability problems (SAT).

The key features of local search SAT algorithms are an initial assignment of variables occurring in the given set of clauses cnf followed by local moves, which flip (invert) the truth value of a chosen variable until a satisfying assignment is found. The instances of the algorithm differ in their choice of initial assignment (`INIT_ASSIGN`) and variable choice (`CHOOSE_FLIP`).

In previous work, we show an implementation of the SLS variant GSAT [HTY01], which achieves one flip per clock cycle. Here, we extend this design methodology and approach to the CSAT variant [GW93].

In order to describe and analyze parallel implementations of `GenSAT` algorithms, we use a notation close to the parallel functional language NESL [BHSZ95] explained in [HTY01], which allows an asymptotic complexity analysis with respect to the two factors that concern hardware implementations. The total size of the design can be measured by an abstraction of the number of gates needed to run a program P, denoted by $g(P)$. The analog of time complexity is the depth of program P, denoted by $d(P)$, the number of time units required for execution of P.

Let V be a set of boolean variables $v_1, v_2, \ldots, v_n$. A SAT problem $\mathcal{C}$ is a propositional formula in conjunctive normal form (CNF) consisting of a conjunction of m clauses. Each clause c_i is a disjunction of one or more literals where each literal is either a variable v_j or its negation $\neg v_j$. A SAT problem is called a k-SAT problem, if each clause has at most k literals. Without loss of generality, we consider only 3-SAT problems in this paper. The formula $\mathcal{C}$ is satisfiable, if there is an assignment of truth values to all variables in V that satisfies all clauses.

2 The CSAT Algorithm

CSAT [GW93] is a variation of GSAT, in which the candidate variables to be flipped are categorized into three groups with decreasing priority: increase in score (downward

M. Glesner, P. Zipf, and M. Renovell (Eds.): FPL 2002, LNCS 2438, pp. 1156–1159, 2002.

Program 1 The CHOOSE_FLIP algorithm for CSAT

```
procedure CHOOSE_FLIP(V, cnf)   returns: variable to be flipped
      U := []; S := []; s := score(cnf, V);
L1 :  for i := 1 to n do /* for all variables */
            s' := score_f(i, cnf, V);
            if s' > s then U := U ∪ [i]
            else if s' = s then S := S ∪ [i]
      end ;
      if U ≠ [] then return CHOOSE(U)
      else if S ≠ [] then return CHOOSE(S)
      else return CHOOSE(V)
end
```

moves), no change in score (sideways moves), and decrease in score. Program 1 shows a sequential implementation; the choice between moves of the same group is done by the macro CHOOSE, which randomly selects an element of a given sequence. The function $score(cnf, V)$ returns the number of clauses satisfied by variable assignment V and $score_{\mathbf{f}}(i, cnf, V)$ the number of satisfied clauses if variable i is flipped.

Our goal is to exploit the parallelism available in a hardware implementation. There is a trade-off between the degree of parallelism in the implementation and the hardware resource requirements. The time complexity of the sequential implementation of CHOOSE_FLIP for 3-SAT is $\mathcal{O}(nm)$. A direct parallel implementation with the sequential loop *L1* in Program 1 with parallelization of $score_{\mathbf{f}}$ by evaluating all clauses in parallel gives a time complexity of CHOOSE_FLIP of $\mathcal{O}(n\ log\ m)$ for 3-SAT. A fully parallel implementation can achieve a time complexity of $\mathcal{O}(log\ m)$ at the expense of replicating the scoring circuit n times.

A more realistic hardware implementation hinges on two key observations: (i) the selection of the flip variable can be done on the basis of relative contribution to the score of that variable when flipped; and (ii) the number of clauses which will be affected by a change to one variable is small, typically much less than m. Program 2 shows the resulting CSAT hardware design written in our notation.

This implementation assumes 3-SAT clauses where only the clauses relevant to a variable j are considered in $score$. Furthermore, the incremental score evaluation pre-computes for each of the two possibilities of a variable flip i, a simplified set of clauses. The optimization uses the notation $\mathtt{EVAL}_j^{\mathcal{C}(i^+)}(V)$ to denote the evaluation of a new j-th 2-SAT clause formed by deleting variable i from an original clause where variable i occurs positively (respectively $\mathtt{EVAL}_j^{\mathcal{C}(i^-)}(V)$ where i is negative). RECEIVE_INITIAL_ASSIGNMENT and SEND_ASSIGNMENT perform the data transfer from and to the external software environment, respectively.

The incremental score computation is done in *s1* and *s2*. The following optimization (not feasible for GSAT), allows to reuse the circuitry for computing the score increments for negative and positive variables. The *Diff* array gives the incremental score for flipping from $0 \rightarrow 1$, and the incremental score for the reverse flip needs a negation. Since CSAT only needs to consider a positive incremental score for downward moves, there are only two cases to consider for a downward move for variable i: (i) $V[i] = 0$ and

Program 2 Instance Specific Implementation of the CSAT Algorithm

```
program MAIN
        V := RECEIVE_INITIAL_ASSIGNMENT();
        for i := 1 to maxflips do
s1          (if SATISFIED(V) then break ||
s1          Dyn0 := {SUM({EVAL_j^{C(i+)}(V) : j ∈ [1...|C(i+)|]}) : i ∈ [1...n]} ||
s1          Dyn1 := {SUM({EVAL_j^{C(i-)}(V) : j ∈ [1...|C(i-)|]}) : i ∈ [1...n]});
s2          Diff := {Dyn1[i] − Dyn0[i] + Static[i] : i ∈ [1...n]};
s3          (Side := {Diff[i] = 0 : i ∈ [1...n]} ||
s3          Down := {Diff[i] ≠ 0 ∧ ISNEG(Diff[i]) = V[i] : i ∈ [1...n]});
s4          if SUM(Down) > 0 then FlipVars := Down
s4          else if SUM(Side) > 0 then FlipVars := Side
s4          else FlipVars := {1 : i ∈ [1...n]};
s5          v := CHOOSE_ONE(FlipVars);
s6          V[v] := ¬V[v]
        end ;
        SEND_ASSIGNMENT(V)
```

$Diff[i] > 0$ for a $0 \rightarrow 1$ flip; (ii) $V[i] = 1$ and $Diff[i] < 0$ for a $1 \rightarrow 0$ flip (note that $Diff[i]$ is negative for variables whose value is currently 1). This is done in *s3* where $\texttt{ISNEG}(x)$ is a macro returning 1 when x is negative and otherwise 0. Finally, to achieve a one flip design, we use multi-try pipelining [HTY01] with pipeline stages labeled as *s1* to *s6*. Assuming each stage is executed in one cycle, this implementation yields a depth complexity of $\mathcal{O}(log\ m + log\ n)$ and gate complexity of $\mathcal{O}(m + n)$ for 3-SAT.

3 Performance Evaluation

The actual FPGA implementation of CSAT (Program 2) was obtained using Handel-C programs,which is compiled into a netlist and then turned into a FPGA bitmap file with Xilinx Foundation Express. The target hardware used was a prototyping board developed by Celoxica Inc, the RC-1000PP board which contains a XCV1000E FPGA from Xilinx with 1.5 million system gates grouped into 6144 CLBs of two slices each.

All designs run at one flip per clock cycle and exploit parallel clause evaluation, parallel score computation and multi-try pipelining. In order to compare the efficiency of FPGA implementations, we measure their *flip rate*, i.e. the number of flips per second (flips/s). Column 4 of Table 1 shows the flip rate of the GSAT41 software by Selman and Kautz running CSAT on a Pentium II-400MHZ machine. The FPGA implementations of CSAT and GSAT are all run at a clock speed of 20MHz, which corresponds to a flip rate of 20,000 K flips/s. Column 5 shows that the FPGA implementation is two orders of magnitude faster than software.

In [HTY01], we developed a performance measure for hardware SAT implementations that takes hardware cost into account. This measure, called *flip density*, divides the flip rate by the number of FPGA slices (*fps/sl*). The last four columns in Table 1 shows that the presented CSAT implementation improves over the GSAT implementation of [HTY01] by a factor of 1.5 with respect to this performance measure.

Table 1 Flip Rate Speedup and Flip Density for Hardware Implementations of CSAT using Selected Benchmarks from SATLIB [SAT]

SAT Problem	Var (n)	Clause (m)	Software K *fps*	Speedup factor	GSAT slices	CSAT slices	GSAT *fps/sl*	CSAT *fps/sl*
uf20-01	20	91	52.7	393	1490	1019	13832	20332
uf50-01	50	218	72.8	287	3170	2062	6521	10131
uf100-01	100	430	71.6	291	5918	3860	3484	5391
aim-50-1_6-yes1-1	50	80	138.0	151	1818	1110	11374	18795
aim-50-3_4-yes1-1	50	170	74.3	281	2464	1667	8386	12521
aim-50-6_0-yes1-1	50	300	40.6	514	3506	2600	5897	8022
aim-100-1_6-yes1-1	100	160	138.4	151	3480	2018	5932	10331
aim-100-3_4-yes1-1	100	340	72.7	287	4712	3080	4381	6773
flat30-1	90	300	96.7	216	3515	2141	5883	9744
rti_k3_n100_m429_0	100	429	71.2	293	5904	3857	3494	5401
bms_k3_n100_m429_0	100	429	107.0	195	4766	2812	4332	7409

4 Conclusion

We show how to achieve a one flip per clock cycle design for the SLS solver CSAT. This design has comparable asymptotic depth and gate complexity to the one flip GSAT design in [HTY01] but improves on the flip density. Thus CSAT has the potential of a smaller implementation compared with GSAT.

The results here also demonstrate the potential hardware acceleration afforded by an FPGA implementation. The software results were taken on a Pentium II running at 400MHz, while the FPGA was only clocked at 20MHz. In this setting, we observe a speed-up of two orders of magnitude. Current work—beyond the scope of this paper—achieves one flip per clock cycle reconfigurable SLS implementations at the expense of a reduction in flip density. Details of the FPGA implementations with the SLS WalkSAT [SKC94] algorithm are reported in [Tan01].

References

[BHSZ95] Guy Blelloch, Jonathan Hardwick, Jay Sipelstein, and Marco Zagha. NESL user's manual, version 3.1. Technical Report CMU-CS-95-169, Carnegie Mellon University, Pittsburgh, PA, 1995.

[GW93] Ian P. Gent and Toby Walsh. Towards an understanding of hill-climbing procedures for SAT. In *Proceedings of AAAI-93*, pages 28–33, 1993.

[HTY01] Martin Henz, Edgar Tan, and Roland Yap. One flip per clock cycle. In Toby Walsh, editor, *Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 509–523, Cyprus, 2001. Springer-Verlag, Berlin.

[SAT] SATLIB – The Satisfiability Library, `http://www.satlib.org`.

[SKC94] B. Selman, H. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of AAAI-94*, pages 337–343, 1994.

[Tan01] Edgar Tan. *Local Search Algorithms for SAT on FPGAs*. Master's thesis, School of Computing, National University of Singapore, 2001.