Investigating the use of Modern Heterogeneous CPU-FPGA Architectures on the 3-SAT Problem

Gabriel Coimbra
Universidade Federal Viçosa, UFV
Florestal, Brazil
gabriel.coimbra@ufv.br

Lucas Bragança
Universidade Federal Viçosa, UFV
Florestal, Brazil
lucas.braganca@ufv.br

José Augusto M. Nacif
Universidade Federal Viçosa, UFV
Florestal, Brazil
jnacif@ufv.br

Abstract—**The technology of Field Programmable Gate Array (FPGA) has been very promising in solving problems with low power cost and high performance. In this paper, we investigate how an FPGA can help accelerate traditional CPU/GPU SAT solvers. For this, we used OpenCL, a framework that provides an easy-to-use programming interface across heterogeneous platforms. We were able to design an accelerator that can solve up to 40 variables of a satisfiable instance in 14 ms. Unsatisfiable instances with 20 variables can be solved in 343 ms.**

Index Terms—**SAT, bruteforce, FPGA**

I. Introduction

SAT is a large studied NP-complete problem on Computer Science and Mathematics. It checks if there is a combination of the variable values in a boolean formula that can successfully satisfy it. A formula is represented by a Conjunctive Normal Form (CNF) [1], which is a logic conjunction of clauses. Every clause then holds the literals in a disjunction as the example in 1. This means that at least one literal in each formula must be true for the whole instance to be true. This problem has been largely studied and has numerous varieties of applications in Electronic Design Automation (EDA), Automatic Theorem Proving and Artificial Intelligence [2].

$$(X_0 \lor \neg X_1 \lor \neg X_2) \land (\neg X_1 \lor X_2 \lor X_3) \land (X_1 \lor \neg X_4 \lor \neg X_3) \quad (1)$$

Since the introduction of Davis-Putenam-Longemann-Loveland (DPLL) algorithm [3], which is the base current CPU SAT solvers rely upon, a lot of new techniques and heuristics have been introduced in the basic SAT arsenal. Conflict Driven Clause Learning (CDCL) presented in [5] is now the state-of-the-art of the majority of the solvers. But recently, instead of searching for a better algorithm for increasing the CPU perfomance, many of studies begin to shift the focus to GPUs or FPGAs for accelerating the solving process [6]–[13]. Because the inherently sequential nature of SAT [14], is challenging to build SAT solvers with good performance on FPGAs. We use OpenCL, an alternative to Register Transfer Level (RTL), that is often increases productivity with the cost of lower throughput.

In the paper, we show how a low latency acceleration pipeline is implemented with OpenCL in a heterogeneous CPU-FPGA platform. The algorithm is a simple brute force which relies upon the fast circuitry that it is built in the FPGA for targeting specific problems. For better results within our algorithm we targeted 3-SAT problems, a variation of SAT which every clause has at maximum three literals. As we can successfully turn any SAT in a *equisatisfiable* 3-SAT in polynomial time [15] we take advantage of this property to extend the range of applications of this solver back to the original SAT.

In a recent study, *Yuan et al* [6] claims that the software solver is faster if a small number of variables are left undecided for the FPGA. They've shown that, in some instances using MiniSAT, a golden goal effect is observed: about 25% CPU time is spent on the last 16 variables, and over 45% is spent on the last 20. This is also our main motivation for a brute force algorithm. With an Altera DE2-115 board, they were able to solve fixed instances with 13-variables in linear time as the number of clauses increases. The main difference from our design, is that the FPGA space is saved with the cost of time complexity, since they generate a cell for each possible assignment, and we have a single solving unit.

The remainder of this paper is structured as follows: we first discuss related work in Section II, then we give a brief explanation of our platform and describe the algorithm and our limitations in Section III. In Section IV we describe the results and then we proceed to show some related works and what could be done to improve our design in Section IV.

II. Related Work

Thong et al. [9] made a thorough Ph.D thesis about SAT solving on FPGAs. Their design consists of a multithreaded FPGA accelerator unit. Variables are hardware addresses that do not require any transformation or clause lookup resulting in a more compact and faster memory. The hardware is intended to solve Boolean Constraint Propagation (BCP), generated by in process of solving the formula, since BCP is the most time consuming for the CPU. They use complex data structures for storing SAT, including linked lists and arrays. The architecture consists of a network of Processing Elements (PE) that each one performs a BCP implication.

Yuan et al. [6] aims to extract fixed scale sub-problems with a software solver and brute-force the problem's variables in hardware. They describe two architectures: the hardware side is a conjunct of cells, each one test all clauses for a specific assignment. On the host side, the CPU processes the formula until 13 variables are left undecided and then pushes the partition to the hardware and repeats until the formula is found to be *SAT* or *UNSAT*. Similarly to our implementation, we leverage the communication overhead limiting the data transfer to the formula as input and the result as output. The software continuously sends more instances to be solved in the FPGA and backtracks if a subinstance was found to be unsatisfiable.

Davis et al. [13], present a hardware co-processor for BCP acceleration and also establish architectural requirements that every CPU-FPGA solver should follow to present good performance. Using a popular hypergraph tool hMetis, they ensure that the partitions do not have cross dependencies and dispatch the generated partitions to several inference engines in the FPGA. These engines then performs several implications with the given partitions and tries to detect conflicts. This design is mainly limited by the BRAM capacity of the FPGA, since the inference engines should store the partitions while solving. Instead of OpenCL, all these works use RTL languages such as Verilog or VHDL. We also have to consider their limitations, as example [6] uses a Cyclone IV board which is higher limited than our Arria 10.

These works implement heterogeneous solvers, but they differ on the granularity of the parallelism [18]. *Yuan et al.* test multiple variables in parallel and thus have a variable level parallelism. *Davis et al.* the division of the SAT provides at partition level, since they partition the problem into many sub instances. At last, in *Thong et al.* the design is at task/thread grade, since it launches multiple threads at the same time for a single instance. Our work, like *Yuan et al.* is at variable level and also have a minimal data transfer cost, since the communication is minimal: we send the problem to the FPGA and get back the results. As opposed to *Gulati et al.* [10], [11], we store the problem data continuously in a vector with linear space (in these papers, they store the variables and the clauses in a matrix). And also opposed to *Yuan et al.* [6], our accelerated portion scaling on the FPGA is quadratic to the number of variables, instead of exponential, this would ease the replication of the hardware.

III. Methodology

Our platform, Intel-Altera Heterogeneous Architecture Research Platform (HARP), is an Altera Arria 10 FPGA and an Intel Xeon E5 CPU integrated by a QuickPath Interconnect ™ bus with a bandwith up to 12.4 GB/s [19] . The FPGA has access to the platform RAM memory, the *shared memory* using a DDR4 bus.

We develop our algorithm in OpenCL which is a High Level Synthesis (HLS) language, it provides anyone with a basic knowledge of the internals of the FPGA the ability write an algorithm in a high level language and successfully translate to RTL.

The workflow in Figure 1, follows the steps: using the Altera Offline Compiler we generate a binary bitstream of the OpenCL kernel and then we image this kernel into the FPGA. On the host side, we have a C/C++ program where we share memory containing the problem. We also start multiple kernels, stop and synchronize with board. Emulation capabilities are also available, Altera ™provides an offline emulator tool that is capable of compiling the code of the OpenCL kernel into x86-64 architecture.

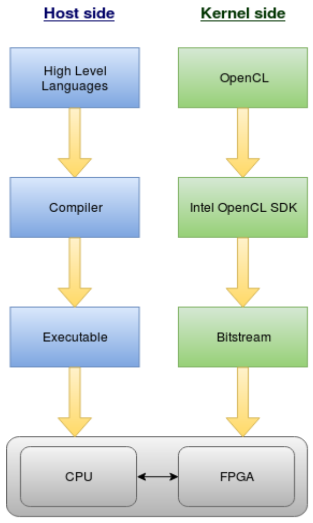

Fig. 1. OpenCL workflow.

Our algorithm relies on the OpenCL 1.0 specification and we synthesized our device Intel® Software Development Kit (SDK) for OpenCL ™.

We store the data in the Conjunctive Normal Form (CNF) format [1] in a vector. The output is the assignment list and the result. If the formula has N variables, then the assignment list has N bits. The result is just one bit long.

After the host shares the data and starts the kernel, the kernel then proceeds to download all the formula from the *shared memory* to it's high speed Block Random Access Memory (BRAM). Then it launches the Algorithm 1.

The algorithm, for each combination of variable values, tests all formula's clauses under the current assignment. For this test, we use a exclusive OR logic operator since if the variable is negative in the formula it cannot also be negative on our assignment list and vice-versa.

In the software solver, the cycle of sending the problem and using the result can be repeated while there is still variables which the values are unknown.

Algorithm 1 SAT Brute force

```
1:  procedure SOLVER($formula$)
2:      $a \leftarrow permutations\ of\ variables$
3:      for each $assignment$ do
4:          $result \leftarrow Evaluate(formula, assignment)$
5:          if $result$ is SAT then
6:              return $SAT$
    return $UNSAT$
    ▷ All clauses tried.
7:  procedure EVALUATE($c, a$)
8:      $i \leftarrow 0$
9:      for each $clause$ in $formula$ do
10:         for $n = 0$ while $n\ less\ than\ 3$ do $n++$
11:             $value \leftarrow current\ variable\ assigment$
12:             if $value \oplus clause[i+n]$ then
13:                 Continue
14:             else return $UNSAT$          ▷ Formula is false.
15:         $i \leftarrow i + 3$                 ▷ Next clause.
    return $SAT$                             ▷ All clauses tested.
```

The algorithm in Figure 1 works as follows, for the generated combination of possible variable values it checks all three literals in the first clause. This is made with a exclusive or logic operation, since if the literal is negated in the clause, it can't be on the assignments. Then, the algorithm proceeds to check every other clause left. With first clause that is not satisfied, the algorithm then advances to another combination.

With this algorithm it becomes clear why the decision to turn SAT into 3-SAT: if the number of literals in a clause is a variable number, we would need a second loop with unknown bounds and our SDK will be unable to pipeline this loop since we don't know it's range. This characteristic is also present on original SAT problems. It is possible for our design to solve loops with dynamic limits, but then it becomes orders of magnitude more slower, since there is no pipelining.

Although our SDK performs many optimizations in our algorithm and is able to archive a high parallelism, we keep the code as simple as possible and use primarily cheap operations to the FPGA, like shifts, adds and logical operations. Even the BRAM access is leveraged by using wires within the circuit when a the algorithm needs a specific value, instead of accessing the memory blocks. With these characteristics and the fact that there is no need for synchronization, we have a very small communication cost in the solving process.

IV. Results and Discussion

In this section we present our device's resource usage and a more deep discussion about our results.

Processing the formula within the shared-memory would be impratical since the high latency access in the *shared memory* bus. We bypass this by storing locally the problem in the FPGA's BRAM when the kernel starts. Because of this choice, the problem size is limited by BRAM memory.

TABLE I
Resource utilization as reported by the SDK for 40 variables solver.

Resource	Utilization
Logic utilization	51%
ALUTs	23%
Dedicated logic registers	28%
Memory blocks	25%
DSP blocks	12%
Compilation Time	140 minutes

As seen in Table I the compilation is very time consuming. This is normal using this SDK for this architecture and does not reflect much the complexity of the solver, since simpler designs like a vector addition example provided in [4] takes about 122 minutes on the same system.

TABLE II
Time results for SAT solving.

SAT?	Clauses	Variables	FPGA Time (μs)
Yes	8	8	4,675.0
Yes	8	50	5,048.04
Yes	16	16	4,452.4
Yes	16	80	9,392.9
Yes	20	20	14,926.0
Yes	20	100	19,558.2
Yes	31	31	14,667.5
Yes	40	40	14,847.8
No	5	40	4,155.0
No	8	40	4,068.1
No	8	200	3,692.5
No	16	80	23,373.3
No	16	200	26,320.2

It is really hard to estimate a time when a formula is proven to be satisfied. Because the proof of the problem can be anywhere between the first and last assignment. One way to analyze the results in Table IV, is using the worst-case scenario as a upper limit of time spent in problems with that size.

The number of variables and clauses were chosen on this principle: is hard to generate a formula with many variables and not a big corresponding number of clauses that is UNSAT. This is true when trying to generate random formulas. The contrary is also true, is hard to generate a formula that is SAT true with many clauses.

For 16 and 20 variables we used 200 clauses (200 is the limit of BRAM), and for 8 and 5 there was no need for this many clauses for the problem be unsatisfiable. For solvable instances, the values were chosen to demonstrate that the solver can be fast when the number of clauses is incremented.

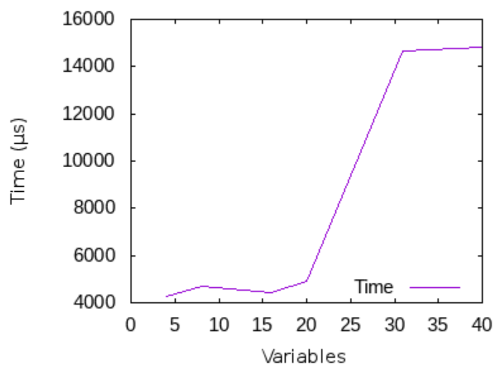

Fig. 2. Results for satisfiable instances.

In Figure IV we see a steadily increase of the time consumed with the number of variables. But this is not always the case, since the solution can be anywhere from the first and last possible combination. A bigger problem can be solved faster than a small one.

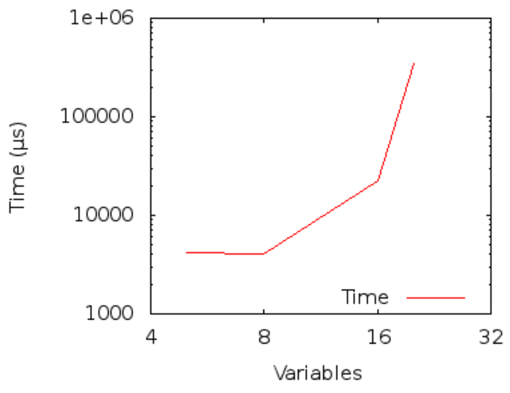

Fig. 3. Results for unsatisfiable instances.

In Figure III, we can see unsolvable formulas takes longer time to solve. This happens when the solver has to check every possible assignment.

In this work, we could understand how a simple unit of acceleration in FPGA can help the SAT solvers. Although the number of variables is limited, using the golden-stage effect on SAT problems [6], we can achieve better performance by dispatching problem partitions whenever necessary. Although this work clarifies a possible acceleration of SAT in FPGAs, more future work is needed to better understand how FPGAs can help in SAT solution.

V. Conclusion and Future work

In this paper we saw how a simple acceleration unit in the FPGA can help SAT solvers. Although the number of variables is limited, using the golden goal effect in SAT problems [6], we can achieve better performance dispatching partitions of the problem on the FPGA. We need future works to investigate more deeply how FPGAs may help in SAT solving.

Acknowledgments

Thanks Intel Research Labs, who gave us access to the CPU-FPGA HARP platform. The Paderborn University who currently hosts the HARP platform. To CNPq and UFV, which is our fund raiser and university, respectively. And also to Cadence Design Systems, Inc, who provided technical and intelligence support.

References

[1] Prestwich, Steven David. ”CNF Encodings.” Handbook of satisfiability 185 (2009): 75-97.

[2] Marques-Silva, Joao. ”Practical applications of boolean satisfiability.” Discrete Event Systems, 2008. WODES 2008. 9th International Workshop on. IEEE, 2008.

[3] Davis, Martin, George Logemann, and Donald Loveland. ”A machine program for theorem-proving.” Communications of the ACM 5.7 (1962): 394-397.

[4] https://www.altera.com/products/design-software/embedded-software-developers/opencl/developer-zone.html

[5] Marques-Silva, João P., and Karem A. Sakallah. ”GRASP: A search algorithm for propositional satisfiability.” IEEE Transactions on Computers 48.5 (1999): 506-521.

[6] Yuan, Zhongda, Yuchun Ma, and Jinian Bian. ”SMPP: Generic SAT Solver over Reconfigurable Hardware Accelerator.” Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE, 2012.

[7] Safar, Mona, et al. ”A reconfigurable, pipelined, conflict directed jumping search sat solver.” Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011. IEEE, 2011.

[8] Thong, Jason, and Nicola Nicolici. ”FPGA acceleration of enhanced boolean constraint propagation for SAT solvers.” Proceedings of the International Conference on Computer-Aided Design. IEEE Press, 2013.

[9] Thong, Jason. FPGA Acceleration of Decision-Based Problems using Heterogeneous Computing. Diss. 2014.

[10] Gulati, Kanupriya, et al. ”Efficient, scalable hardware engine for Boolean satisfiability and unsatisfiable core extraction.” IET Computers & Digital Techniques 2.3 (2008): 214-229.

[11] Waghmode, Mandar, et al. ”An efficient, scalable hardware engine for Boolean satisfiability.” Computer Design, 2006. ICCD 2006. International Conference on. IEEE, 2007.

[12] Kanazawa, Kenji, and Tsutomu Maruyama. ”An FPGA solver for WSAT algorithms.” Field Programmable Logic and Applications, 2005. International Conference on. IEEE, 2005.

[13] Davis, John D., et al. ”A practical reconfigurable hardware accelerator for Boolean satisfiability solvers.” Design Automation Conference, 2008. DAC 2008. 45th ACM/IEEE. IEEE, 2008.

[14] Hölldobler, Steffen, et al. ”A short overview on modern parallel SAT-solvers.” Advanced Computer Science and Information System (ICACSIS), 2011 International Conference on. IEEE, 2011.

[15] Aho, Alfred V., and John E. Hopcroft. The design and analysis of computer algorithms. Pearson Education India, 1974.

[16] Safar, Mona, M. Watheq El-Kharashi, and Ashraf Salem. ”FPGA-based SAT solver.” Electrical and Computer Engineering, 2006. CCECE’06. Canadian Conference on. IEEE, 2006.

[17] Balyo, Tomáš, Marijn JH Heule, and Matti Järvisalo. ”SAT COMPETITION 2017.”

[18] Sohanghpurwala, Ali Asgar, Mohamed W. Hassan, and Peter Athanas. ”Hardware accelerated SAT solvers—A survey.” Journal of Parallel and Distributed Computing 106 (2017): 170-184.

[19] Intel, An. ”Introduction to the intel quickpath interconnect.” White Paper (2009).