A Distributed Genetic Algorithm Solution to the Boolean Satisfiability Problem

M S Hasan[1,2], B P Amavasai[1] and J R Travis[1]
[1]*Microsystems & Machine Vision Laboratory, Sheffield Hallam University, UK*
[2]*Faculty of Computing, Engineering and Technology, Staffordshire University, UK*
m.s.hasan@staffs.ac.uk, b.p.amavasai@shu.ac.uk, j.r.travis@shu.ac.uk

Abstract

This paper attempts to improve the solution of the NP complete Boolean Satisfiability (BSAT) problem by partitioning the task into three sub-tasks and distributing them over an experimental 3-node Distributed Computing System (DCS). A genetic algorithm (GA) has been used to consider multiple feasible solutions. The GA based algorithm is applied to the standard BSAT benchmarks on a single computer and on DCS configuration using non-optimised and optimised executables. The task is coarsely partitioned and distributed over the DCS using the Simple Object Access Protocol (SOAP) technology. The results reveal that the DCS enabled solution exhibits better performance than a single computer configuration for non-optimised GA code. However, no clear correlation could be identified between the single computer and the DCS for the optimised version of the GA search. The main contribution of this investigation is the design of a GA based solution to the BSAT problem for DCS.

1. Introduction

Given a Boolean function *F* in product-of-sum representation, then Boolean satisfiability (BSAT) problem is defined as finding an assignment to the variables such that *F* evaluates to TRUE. An "instance" is satisfied when the Boolean expression is TRUE for some assignment to the variables [1]. Otherwise it is said to be "unsatisfiable". The BSAT problem [2] is of crucial importance in many fields, for instance, artificial intelligence [3], hardware design [4], [5] *etc*. Due to the complexity of the problem, it can take several years to obtain a solution using the current fastest computer, even for $N = 50$ [6]. The basic idea of the paper is to provide a solution to the BSAT problem using DCS of general purpose computers [7] and genetic algorithm (GA) [8].

DCS refers to a computing environment where the resources and tasks are dynamically shared by the computing nodes to meet the load/demand requirement of the system [9]. On the other hand, Genetic algorithms (GA) attempt to solve complex problems by modelling Darwin's theory of evolution where solutions of a particular problem are allowed to evolve over time. GAs are widely used for optimised searching [10]. Suitablity of GA fitness functions can be found in [11].

This paper is organised as follows. Section 2 reviews previous works, section 3 explains the GA BSAT search algorithm, section 4 presents and discusses all results and finally section 5 draws some conclusions.

2. Previous works

A parallel algorithm MP_SAT has been proposed in [12] that makes use of fine grain parallelisms in the clause and variable operations. It speeds up the SAT solver performance by exploiting efficient single processor SAT algorithms like Chaff's [13] solver on an integrated processor Multiple-Instruction-Multiple-Data (MIMD) stream architecture connected to DRAM storage as shown in figure 1 [12]. A fuzzy-genetic approach to the BSAT problem is presented in [6] that makes use of fuzzy logic [14], [15] to assign a fitness measure to chromosomes (feasible solutions) in the search space. The original binary domain {0, 1} is mapped into a continuous fitness domain [0, 1] by fuzzy logic. The GA is used to optimise the solution in the continuous domain and finally the derived solution is converted back (decoded) to the Boolean format.

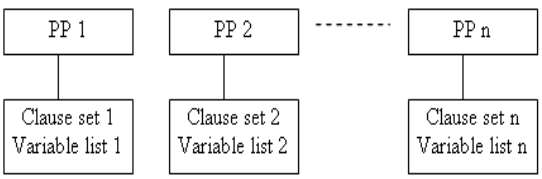

Figure 1. Task partitioning among several processors.

3. The proposed GA BSAT algorithm

Crossover and mutation operations are applied to the probable multiple solutions to improve them and to generate a final solution. For simplicity and speed, the fitness function returns an integer that is the number of the clauses satisfied by a solution. The algorithm stops when a solution is found that satisfies all the clauses or a predefined number of generations (iterations) have been executed. Figure 2 shows the flowchart of the proposed GA BSAT algorithm. The initial chromosomes are generated with random values. Mutation is applied after every 100 (hundred) generations.

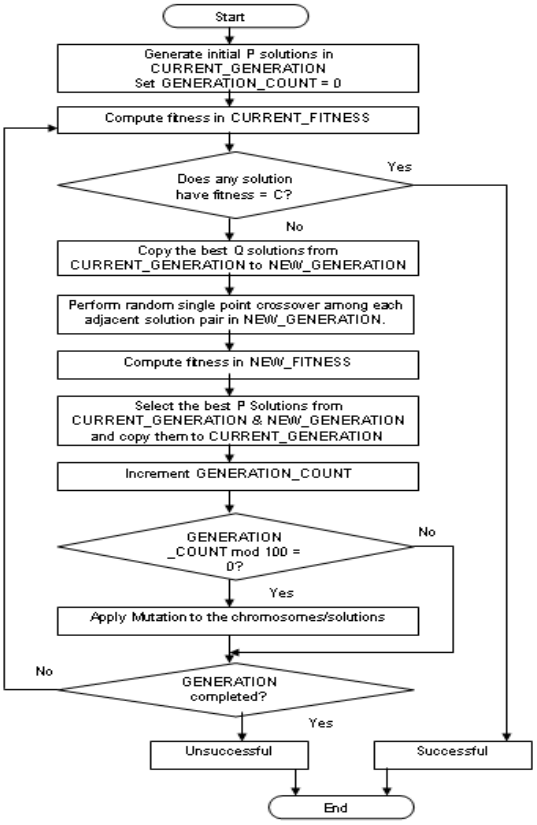

Figure 2. Flowchart representation of the proposed GA BSAT algorithm

3.1. Data structure

For the next few sections, we defined the following terms

- *V*: number of variables in the Boolean expression/function
- *C*: Number of clauses in the Boolean expression/function
- *P*: Size of /number of solutions in current generation
- *Q*: Size of /number of solutions in next generation
- CURRENT_GENERATION: a *P*x*V* matrix that stores the current *P* probable solutions where *P* is the size of population. CURRENT_GENERATION[11] is the *k*-th solution.
- NEW_GENERATION: a *Q*x*V* matrix that stores the new probable solutions after cross-over and *Q* <= *P*.
- *GENERATION*: Number of generations the algorithm is applied to the benchmark.
- CURRENT_FITNESS: a *P*x1 matrix to store fitness of CURRENT_GENERATION solutions.
- NEW_FITNESS: a *Q*x1 matrix to store fitness of NEW_GENERATION solutions.
- EXPRESSION: A matrix that stores the Boolean instance. Row *k* stores the *k*-th clause.

3.2. Partitioning the GA BSAT algorithm

The number of generations to execute is coarsely partitioned into three sub-generations and each of the three computers of the DCS executes one sub-generation. Each computer is able to execute its own sub-generations without depending on the others. The GA BSAT algorithm aborts/discards the other two sub-generations whenever one computer obtains a solution. The partitioning is depicted in figure 3.

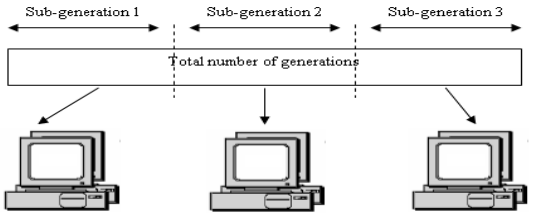

Figure 3. Parallelisation of GA based BSAT algorithm onto 3-node DCS computers.

4. Results and discussion

50 random instances are selected from the benchmark suit uf20-91 [16] and for each of these, 30 readings were taken to obtain statistically interpretable data.

The GA BSAT search method is applied for population size of 20 and 10,000 generations. Since each expression has 20 variables, 20 chromosomes/solutions are considered. It has been observed that for population sizes larger than 20, *e.g.* 40 or 60 and for generations higher than 10,000 there is no significant improvement in results. The algorithms was compiled using the GNU C compiler version 3.3.2, first with optimisations switched off and then with optimisations set to level O3.

4.1 Single computer configuration: Non-optimised vs. O3 optimised machine code

For each instance, the executions that found a solution are considered. The average of these successful search times are plotted in figure 4. For most of the instances, O3 optimised code exhibits a lower run time and is always less than 250 ms.

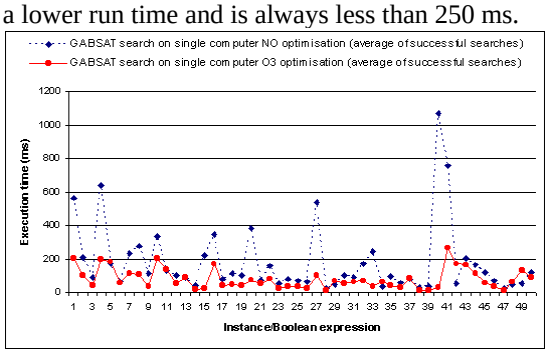

Figure 4. Run time of GA BSAT search on single computer for non-optimised and O3 optimised code.

4.2 The 3-node DCS configuration: Non-optimised vs. O3 optimised machine code

No straightforward relationship is found between non-optimised and O3 optimised code from figure 5 that plots average time of successful searches. O3 optimised code exhibits lower run time for more that 35 instances and maximum run time stays below 400 ms.

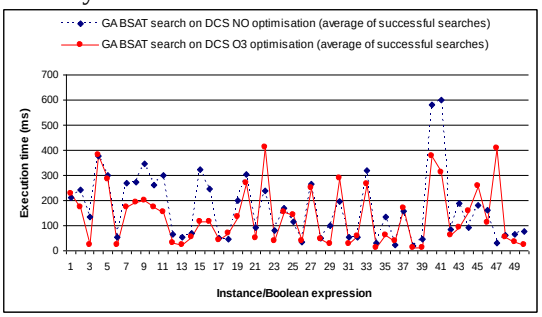

Figure 5. Run time of GA BSAT search on the DCS for non-optimised and O3 optimised code.

4.3 GA BSAT search on a single computer vs. on the DCS

Figure 6 and 7 show that the maximum run time for successful search is higher for non-optimised code (≈1100ms) than that of O3 optimised code (≈600ms). But, no clear correlation can be identified between the single computer and the DCS approach.

4.3.1. Non-optimised machine code. In general, the GA BSAT algorithm showed superior performance on the DCS implementation. Figure 6 shows the scenario that in 400 ms time, the DCS found solutions for 48 instances, whereas the single computer found solutions for 45 instances.

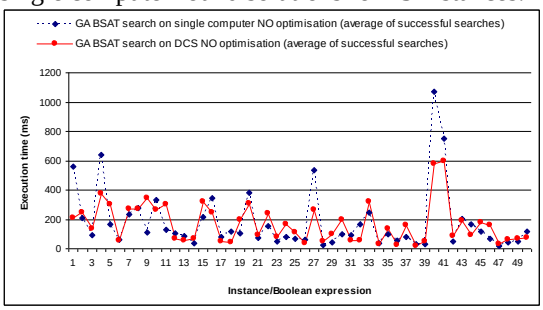

Figure 6. Run time of GA BSAT search on single computer and the DCS for non-optimised code.

4.3.2. O3 optimised machine code. For the O3 optimised code, the single computer approach exhibits better run time than the DCS implementation. Figure 7 shows that for most of the instances, successful search run times lie within 200 ms for GA BSAT on single computer.

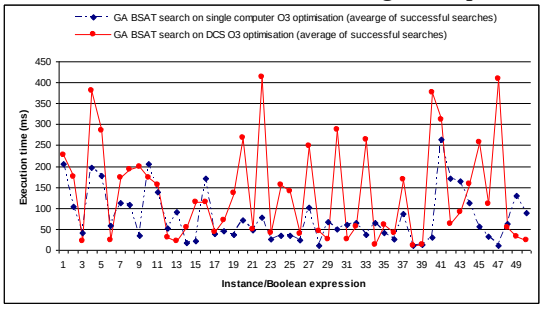

Figure 7. Run time of GA BSAT search on single computer and the DCS for O3 optimised code.

4.4 Maximum run time for unsuccessful search

Figure 8 and 9 show the comparison of maximum run time (unsuccessful search) for non-optimised and O3 optimised machine code, respectively. These two figures depict that the sub-generations should take approximately 1/3rd time of the entire generation to execute in case of unsuccessful search.

5. Conclusion

In this paper, a GA approach to solving the Boolean Satisfiability Problem has been presented. The problem is coarsely partitioned so that it may be easily distributed on a 3-node DCS of computers. Well-known standardised protocols (SOAP) [17] are used to distribute the problem.

The GA BSAT search algorithm has demonstrated diverse results for different cases. These are listed below.

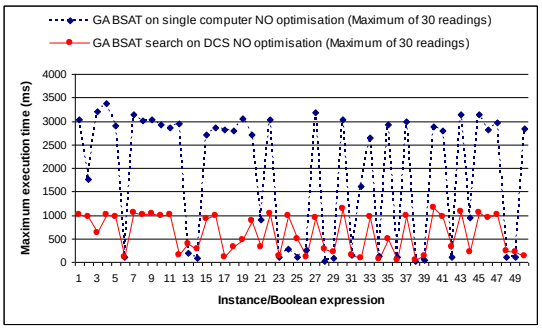

Figure 8. Maximum run time of GA BSAT search on single computer and the DCS for non-optimised code.

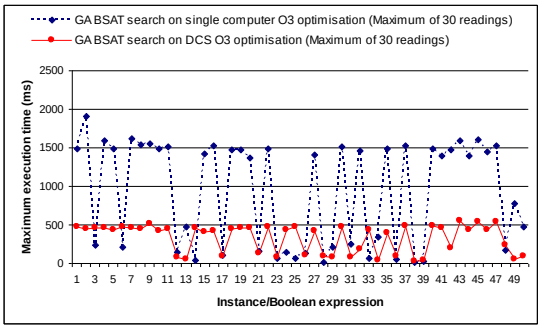

Figure 9. Maximum run time of GA BSAT search on single computer and the DCS for O3 optimised code.

Single computer: non-optimised vs. optimised machine code: For most of the instances, O3 optimised code takes less time to execute.

DCS: non-optimised vs. optimised machine code: It was obvious that O3 optimised code exhibited lower run time for more that 35 instances.

Non-optimised machine code: single computer vs. DCS: In general, the GA BSAT algorithm showed superior performance on the DCS by finding solutions for more instances than single computer.

O3 optimised machine code: single computer vs. DCS: The DCS demonstrated a worse run time than single computer.

Acknowledgement

We wish to acknowledge the support of Dr Abu Saleh Jabir, Senior Lecturer, Oxford Brookes University, UK for providing SAT problem material and suggestions for this research.

6. Reference

[1] Iouliia Skliarova, António B. Ferrari, “A SAT Solver Using Software and Reconfigurable Hardware”, Proceedings of the 2002 Design, Automation and Test in Europe Conference and Exhibition (DATE 2002).

[2] Michael R. Garey, David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W.H. Freeman & Company, San Francisco, CA: Freeman, 1979, ISBN: 0716710455.

[3] L. Zhang, C. Madigan, M. Moskewicz and S. Malik, "Efficient Conflict Driven Learning in a Boolean Satisfiability Solver", Proceedings of International Conference on Computer Aided Design (ICCAD2001), San Jose, CA, Nov. 2001.

[4] M. Velev, and R. Bryant, “Effective Use of Boolean Satisfiability Procedures in the Formal Verification of Superscalar and VLIW Microprocessors” Proceedings of the Design Automation Conference, July 2001.

[5] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu. “Symbolic Model Checking without BDDs” Tools and Algorithms for the Analysis and Construction of Systems (TACAS'99), number 1579 in LNCS. Springer-Verlag, 1999.

[6] Witold Pedrycz, Giancarlo Succi, and Ofer Shai, “Genetic–Fuzzy Approach to the Boolean Satisfiability Problem”, IEEE transactions on evolutionary computation, vol. 6, no. 5, October 2002.

[7] I. Foster, C. Kesselman, S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", International J. Supercomputer Applications, 15(3), 2001.

[8] Bill P. Buckles and Frederick E. Petry, *Genetic algorithms*, IEEE Computer Society Press, CA, USA, 1992, ISBN 0818629355.

[9] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Open Grid Service Infrastructure WG, Global Grid Forum, June 22, 2002.

[10] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.

[11] K. A. De Jong and W. M. Spears, “Using genetic algorithms to solve NP-complete problems” Proceedings of 3rd International Conference on Genetic Algorithms, San Mateo, CA: Morgan Kaufmann, 1989, pp. 124–132.

[12] Ying Zhao, Sharad Malik, Matthew W. Moskewicz, Conor F. Madigan, “Accelerating Boolean Satisfiability through Application Specific Processing”, Proceedings of International Symposium on Systems Synthesis (ISSS2001), Montréal, Québec, Canada, Sep, 2001, pp. 244-249.

[13] M.W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, “Chaff: Engineering an efficient SAT solver” in Proceedings of the 38th ACM/IEEE Design Automation Conference, Las Vegas, Nevada, June 2001, pp. 530–535.

[14] Zadeh, L.A., "Fuzzy Sets", Information and Control, Vol. 8, No. 3, June 1965, pp. 338-353.

[15] Bellman, R.E., and Zadeh L.A., “Decision making in a fuzzy environment”, Management Science, Vol. 17B, No. 4, 1970, pp. 141-64.

[16] http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html - Last accessed in Aug 2004.

[17] http://gsoap2.sourceforge.net/