MPIDepQBF: Towards Parallel QBF Solving without Knowledge Sharing*

Charles Jordan[1], Lukasz Kaiser[2], Florian Lonsing[3], and Martina Seidl[4]

[1] Division of Computer Science, Hokkaido University, Japan
[2] LIAFA, CNRS & Université Paris Diderot***
[3] Knowledge-Based Systems Group, TU Wien, Austria
[4] Institute for Formal Models and Verification, JKU Linz, Austria

Abstract. Inspired by recent work on parallel SAT solving, we present a lightweight approach for solving *quantified Boolean formulas (QBFs)* in parallel. In particular, our approach uses a sequential state-of-the-art QBF solver to evaluate subformulas in working processes. It abstains from globally exchanging information between the workers, but keeps learnt information only locally. To this end, we equipped the state-of-the-art QBF solver DepQBF with assumption-based reasoning and integrated it in our novel solver MPIDepQBF as backend solver. Extensive experiments on standard computers as well as on the supercomputer Tsubame show the impact of our approach.

1 Introduction

Recently, there has been much progress in solvers for *quantified Boolean formulas* (QBF) [4, 7]. The quest for QBF solvers is motivated by the vision that QBF solvers become powerful general purpose reasoning engines for PSPACE problems in the same way as SAT solvers are for problems in NP. Then, many interesting application problems that have compact QBF encodings but (assuming NP $\neq$ PSPACE) no compact SAT encodings could be handled by QBF solvers [1].

Most of the recent advances in QBF solving are realized within sequential systems (e.g., [8, 10, 11, 14]), thus not taking advantage of the parallel computing resources provided by modern computer architectures. Although some dedicated parallel QBF solver implementations have been presented [6, 12, 13, 18], to the best of our knowledge none is actively maintained and publicly available. Additionally, the usual focus on sharing information between different working processes can be influential to the solver's overall performance [17] due to restrictions of bandwidth.

In this paper, we propose solving QBFs without global information sharing and present MPIDepQBF[5], a parallel solver for quantified Boolean formulas based

*** Currently at Google Inc.

* Partially supported by the Austrian Science Fund (FWF) under grants S11408-N23 and S11409-N23, by the Wiener Wissenschafts-, Forschungs- und Technologiefonds (WWTF) under grant ICT10-018, and by the Japan Society for the Promotion of Science (JSPS) as KAKENHI No. 25106501.

[5] Open-source, available via SVN (rev. 1928) at `http://toss.sf.net/develop.html`

on a new version of the sequential solver DepQBF. The rest of the paper is organized as follows. First, we review related work in Section 2. Then we describe the basic algorithm of our approach in Section 3. In Section 4, we evaluate the performance of our solver on standard benchmark instances using Tsubame2.5[6]. In Section 5 we conclude with an outlook to future work.

2 Related Work

In the year 2000, work on parallel QBF solvers began with PQSolve, a parallel version of QSolve [6], which implements techniques like quantifier inversion or trivial truth. However, much progress has since been made in QBF solving, in particular clause and cube sharing have been introduced. More recently, PAQuBE [13], QMiraXT [12], and the approach of Da Mota et al. [18] have been presented. However, to the best of our knowledge, none of these tools is publicly available and all seem to be no longer developed.

QMiraXT uses a shared memory architecture for sharing clauses which increases flexibility, but has scalability issues. Da Mota et al. split non-prenex non-CNF formulas into QBFs with free variables which are rewritten to propositional formulas by the workers. The master collects these propositional formulas from which the final result is obtained. PAQuBE was developed as a parallel version of QuBE and supports sharing learnt information between workers. It is the solver most closely related to our approach. However, our solver uses a different approach to generate subproblems. For quantified constraint satisfaction problems (QCSP), a problem-partition approach has been presented by Vautard [20] et al. where parallism seems to be very beneficial. Inspired by the recent success of Treengeling [2] in SAT, we allow each worker short (but increasing) timeouts to solve particular subproblems. If a worker solves its subproblem, the result is combined with previous results to determine what portion of the search tree is completed. Otherwise, we either set more variables and distribute the resulting problems or allow the worker to continue with a longer timeout (depending on the number of free workers, subproblems not yet assigned to workers and parameters). Each worker retains information learnt from its previous subproblems.

3 The Architecture of MPIDepQBF

MPIDepQBF is a QBF solver that accepts input in the standard `qdimacs` format, i.e., QBFs $\psi = Q_1 B_1 Q_2 B_2 \dots Q_n B_n . \phi$ in prenex conjunctive normal form (PCNF), where the formula ϕ is a conjunction of clauses. The quantifier prefix $Q_1 B_1 Q_2 B_2 \dots Q_n B_n$ is a sequence of quantified blocks B_i of variables where $Q_i \in \{\forall, \exists\}$ and adjacent blocks are quantified differently, i.e. $Q_i \neq Q_{i+1}$.

MPIDepQBF has one master process to coordinate arbitrarily many worker processes via MPI. The workers use the sequential QBF solver DepQBF [14], which we extended to allow solving under assumptions. No other modifications

[6] We are grateful to the ELC project for providing access to Tsubame.

were necessary. The workers obtain the input formula only once, at the start. This is the only information shared, otherwise they are completely agnostic of the global solving process. When a worker process is idle, the master process supplies it with assumptions and a timeout. Then, the worker process tries to solve the formula under these assumptions. The result is communicated to the master process which adapts the time limit or selects another set of assumptions, as described below. If the final value of the formula has been determined, the worker processes are stopped. Details on master and worker processes follow.

3.1 Master Process

The master process generates a stream of subproblems, represented by partial variable assignments and timeouts, and distributes them to the worker processes. First, starting from the formula $Q_1 B_1 Q_2 B_2 \dots Q_n B_n . \phi$, we sort each B_i according to the number of occurrences of each variable in ϕ and concatenate these sorted lists. This determines the order in which variables will be set.

Next we build a search tree. The tree has 3 kinds of leaves: sat, unsat, and open, where each open leaf contains a variable assignment and a timeout. We start with a fully-balanced binary tree that has only open leaves with the starting timeout (by default 0.1s). The number of leaves is initially the highest power of 2 smaller than the number of available MPI worker processes times a busy-factor (we use 1.25 by default). Each open leaf contains an assignment of the variables: We assign the variables one-by-one in the order determined in the first step. For each open leaf, we send the subproblem (determined by the timeout and the assignment) to a free worker process.

When a worker returns a result, the master process merges it with the current search tree. If the result is sat or unsat, then the open leaf gets replaced by the result, and the tree is simplified as shown in Procedure simplify. If the result is a timeout, then what happens depends on how many free MPI worker processes are available. If the MPI queue is full (except for the one process that just returned), then we multiply the timeout by a timeout-factor (1.4 by default) and send the same case back to the worker process with the new, longer timeout. If there are fewer open leaves than the number of MPI worker processes times the busy-factor, then we replace the open leaf by, again, a fully-balanced binary sub-tree with assignments that prolong the previous one. This is repeated until the whole search tree is simplified to sat or unsat, when the problem is solved.

3.2 Worker Processes: Search-Based Solving under Assumptions

A worker process runs an instance of the QBF solver DepQBF which is initialized with the complete input formula. We extended the API of DepQBF to allow for assumptions as input. These assumptions can be regarded as assignments to variables which are fixed in the current run. All the learnt information is shared over different runs. Note that this information is only kept locally, so no exchange between the different worker processes is realized. QBF solving with

Procedure simplify(t)

if $t = Branch(\exists, v, t_1, t_2)$ *and* `simplify`$(t_1) =$ *sat* **then** sat;
if $t = Branch(\exists, v, t_1, t_2)$ *and* `simplify`$(t_2) =$ *sat* **then** sat;
if $t = Branch(\exists, v, t_1, t_2)$ *and* `simplify`$(t_1) =$ `simplify`$(t_2) =$ *unsat* **then** unsat;
if $t = Branch(\forall, v, t_1, t_2)$ *and* `simplify`$(t_1) =$ *unsat* **then** unsat;
if $t = Branch(\forall, v, t_1, t_2)$ *and* `simplify`$(t_2) =$ *unsat* **then** unsat;
if $t = Branch(\forall, v, t_1, t_2)$ *and* `simplify`$(t_1) =$ `simplify`$(t_2) =$ *sat* **then** sat;
else if $t = Branch(Q, v, t_1, t_2)$ **then** $Branch(Q, v,$ `simplify`$(t_1),$ `simplify`$(t_2))$;

assumptions has been applied in the context of an incremental approach to QBF-based bounded model checking of partial designs [16]. In this section, we describe the handling of assumptions in DepQBF. Assumptions are used in MPIDepQBF to split the search space and thus generate subproblems for the worker processes.

DepQBF [14] is a search-based QBF solver with conflict-driven clause learning and solution-driven cube learning [9, 21]. In this approach, called *QCDCL* [15], backtracking search is combined with the dynamic generation of new clauses and cubes. Given a QBF, variables are assigned successively until either all clauses of the QBF are satisfied or one clause is falsified under the current assignment. Depending on the assignment, clauses are derived by Q-resolution [5] and cubes are derived by the model generation rule and term resolution [9]. The newly derived clauses and cubes are added to the formula as part of separate sets of learnt clauses and learnt cubes to prune the search-space. The parallel variant MPIDepQBF of DepQBF applies QCDCL where the given PCNF is solved under a set of predefined variable assignments, called *assumptions*.

A set of *assumptions* $A := \{l_1, \ldots, l_n\}$ is a set of literals of variables such that $var(l_i) \in B_1$ for all literals $l_i \in A$. The variables of literals in A are from the first block B_1 of ψ. Each literal $l_i \in A$ represents an assignment to the variable $var(l_i)$. Positive literals $l_i = var(l_i)$ and negative literals $l_i = \neg var(l_i)$ represent the assignment of *true* and *false* to the variable $var(l_i)$, respectively. The *PCNF* $\psi[A]$ *under the assumptions* A is obtained from ψ by deleting the clauses which are satisfied under the assignments represented by A, deleting literals from clauses in ψ which are falsified, and deleting superfluous quantifiers from the quantifier prefix of ψ.

In MPIDepQBF, subproblems for the worker processes are generated by applying the definition of assumptions recursively to the formula $\psi[A]$ under some set A of assumptions. If A assigns all variables from the first block B_1 of ψ, then the quantifier prefix of $\psi[A]$ has the form $Q_2B_2 \ldots Q_nB_n$. Since B_2 is now the leftmost block in $\psi[A]$, variables from B_2 can be assigned in some other set A' of assumptions with respect to $\psi[A]$.

The following properties follow from the semantics of QBF. Given an instance ψ in PCNF, if $Q_1 = \exists$ and $\psi[A]$ is satisfiable then ψ is also satisfiable since the variables of the literals in A are all from the first block B_1. If $Q_1 = \forall$ and $\psi[A]$ is unsatisfiable then ψ is also unsatisfiable. These properties are checked in Procedure simplify presented in Section 3.1.

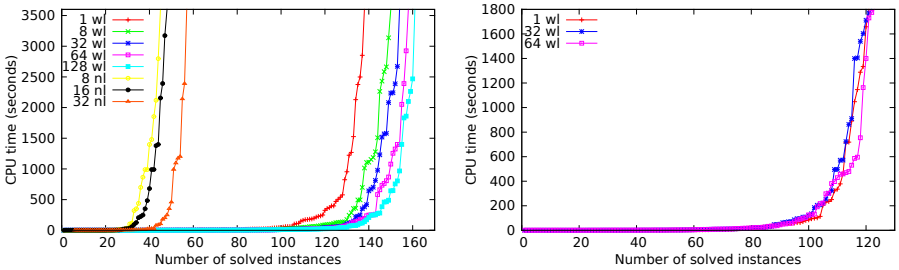

Fig. 1: Cactus plots on eval12r2-bloqqer (with (wl)/without (nl) learning) and eval12r2

Similar to SAT solving under assumptions, assumptions in QCDCL-based QBF solving can be modeled on a syntactic level by adding unit clauses for existential literals in A and unit cubes for universal literals. This is the "clause-based multiple instances (CM)" approach in the terminology of [19].

Alternatively, assumption based reasoning can be realized on the semantic level. Here, the variables of the literals in A are assigned as special *decision variables* (also called *branching variables*) in ψ before the solver makes any assignments to variables not in A. The backtracking procedure of QCDCL has to be modified to guarantee that the assignments represented by A are never retracted. This is the "literal-based single instance (LS)" approach [19]. As in SAT solving, the advantage of the LS approach compared to CM is that all the learnt clauses and cubes can be kept across different calls of the solver with different sets of assumptions. Since assumptions are assigned as decision variables, the learning procedure of QCDCL can never generate clauses and cubes by resolving on variables in A. Because of this advantage, we implemented the semantic LS approach in DepQBF. For the application in MPIDepQBF, we applied a sophisticated analysis of variable dependencies based on the *standard dependency scheme* implemented in DepQBF [14].

4 Evaluation

We evaluated the performance of MPIDepQBF as the number of cores is increased. To this end, we used the `eval2012r2` and `eval12r2-bloqqer` benchmark sets[7], where the latter results from the former by applying the preprocessor Bloqqer [3]. We run our experiments on a small portion of the supercomputer Tsubame. In particular, we used the 'V' queue running on qemu virtual machines with eight physical 2.93 GHz Xeon 5670 cores and 30 GB memory per node.

The left-hand side of Figure 1 shows the performance of MPIDepQBF with various numbers of cores on instances from `eval12r2-bloqqer` with (wl) and without learning (nl) using a timeout of one hour. Whereas with one core only 139 formulas are solved, with 128 cores 160 formulas can be solved. A detailed comparison of the runtimes with different numbers of cores is shown in Figure 2. Besides the number of solved formulas as well as the average runtime for solving

[7] See `http://www.kr.tuwien.ac.at/events/qbfgallery2013/results.html`

# cores (x)	*# solved* both x/128	*avg time (s)* x cores	*avg time (s)* 128 cores
1	137	168.11	62.26
8	148	180.64	64.03
16	149	154.44	76.26
32	151	163.74	79.46
64	155	122.96	98.47

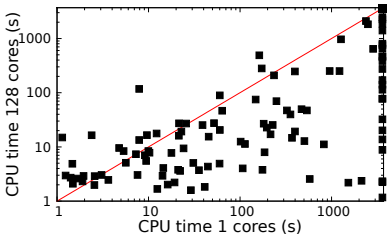

Fig. 2: Runtime comparison: (a) runtimes of x vs. 128 cores for the formulas solved with both x and 128 cores, (b) scatter plot for runtimes of 1 vs. 128 cores

these formulas, we also show the average runtime for solving exactly the same set of formulas when using 128 cores. In additional experiments, we experienced that disabling learning drastically decreases the number of solved formulas. Although workers in MPIDepQBF do not share learnt information, this demonstrates the importance of each worker retaining its learnt information.

The right-hand side of Figure 1 shows performance on non-preprocessed instances. Scaling is limited or non-existent, suggesting the importance of preprocessing for MPIDepQBF. In particular, preprocessing breaks very long clauses into shorter clauses, which is important for memory efficiency in MPIDepQBF.

Besides the experiments on Tsubame, we also evaluated MPIDepQBF on a 12-core 2.4 GHz Intel Xeon 5645 with 96 GB RAM. The results were similar to those above. In particular, with one core 127, with two cores 130 formulas, with four cores 137, with eight cores 139 and with twelve cores 140 formulas could be solved within a timelimit of 600 seconds.

5 Conclusion

We presented MPIDepQBF, a parallel QBF solver based on a search-space splitting approach. A master process generates subproblems and distributes them to arbitrarily-many worker processes, as described above in Section 3. Workers do not exchange information with each other, but keep learnt information locally. To this end, we extended DepQBF with support for assumption-based reasoning. Initial experiments show that more processing power results in a gain of performance, especially when dozens of cores are available. For us, this is surprising given that individual workers do not share learnt information with other workers and gives several possibilities for future development using sophisticated information sharing. In addition, the memory-efficient nature of DepQBF makes usage of many-core, memory-constrained coprocessors promising. More experiments are required to systematically characterize and understand the impact of clause and cube sharing as done in previous works (e.g., [17]).

Our main motivation for working on parallel QBF solving is the desire to solve hard QBF instances stemming from applications. Our hope is that larger systems and parallel solvers will allow us to solve instances beyond the reach of current solvers. That is, while we have challenging instances and access to large systems, we did not have solvers that can utilize these systems.

MPIDepQBF is available at `http://toss.sf.net/develop.html` via SVN.

References

1. Benedetti, M., Mangassarian, H.: QBF-based formal verification: Experience and perspectives. Journal on Satisfiability, Boolean Modeling and Computation 5(1-4), 133–191 (2008)
2. Biere, A.: Lingeling, Plingeling and Treengeling Entering the SAT Competition 2013. In: Proc. of the SAT Competition 2013. Dep. of Computer Science Series of Publications B, University of Helsinki, vol. B-2013-1, pp. 51–52 (2013)
3. Biere, A., Lonsing, F., Seidl, M.: Blocked Clause Elimination for QBF. In: Proc. of the 23rd Int. Conf. on Automatic Deduction (CADE 2011). LNCS, vol. 6803, pp. 101–115. Springer (2011)
4. Büning, H.K., Bubeck, U.: Theory of Quantified Boolean Formulas. In: Handbook of Satisfiability, pp. 735–760. IOS Press (2009)
5. Büning, H.K., Karpinski, M., Flögel, A.: Resolution for Quantified Boolean Formulas. Information and Computation 117(1), 12–18 (1995)
6. Feldmann, R., Monien, B., Schamberger, S.: A Distributed Algorithm to Evaluate Quantified Boolean Formulae. In: Proc. of the 17th Nat. Conference on Artificial Intelligence (AAAI 2000). pp. 285–290. AAAI (2000)
7. Giunchiglia, E., Marin, P., Narizzano, M.: Reasoning with Quantified Boolean Formulas. In: Handbook of Satisfiability, pp. 761–780. IOS Press (2009)
8. Giunchiglia, E., Marin, P., Narizzano, M.: QuBE7.0. Journal on Satisfiability, Boolean Modeling and Computation 7(2-3), 83–88 (2010)
9. Giunchiglia, E., Narizzano, M., Tacchella, A.: Clause/Term Resolution and Learning in the Evaluation of Quantified Boolean Formulas. Journal of Artificial Intelligence Research 26(1), 371–416 (2006)
10. Goultiaeva, A., Bacchus, F.: Recovering and Utilizing Partial Duality in QBF. In: Proc. of the Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2013). LNCS, vol. 7962, pp. 83–99. Springer (2013)
11. Janota, M., Klieber, W., Marques-Silva, J., Clarke, E.: Solving QBF with Counterexample Guided Refinement. In: Proc. of the Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2012). LNCS, vol. 7317, pp. 114–128. Springer (2012)
12. Lewis, M., Schubert, T., Becker, B.: QMiraXT – a multithreaded QBF solver. In: Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV) (2009)
13. Lewis, M., Schubert, T., Becker, B., Marin, P., Narizzano, M., Giunchiglia, E.: Parallel QBF Solving with Advanced Knowledge Sharing. Fundamenta Informaticae 107(2-3), 139–166 (2011)
14. Lonsing, F., Biere, A.: DepQBF: A dependency-aware QBF solver. Journal on Satisfiability, Boolean Modeling and Computation 7, 71–76 (2010)
15. Lonsing, F., Egly, U., Gelder, A.V.: Efficient Clause Learning for Quantified Boolean Formulas via QBF Pseudo Unit Propagation. In: Proc. of the Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2013). LNCS, vol. 7962, pp. 100–115 (2013)
16. Marin, P., Miller, C., Lewis, M., Becker, B.: Verification of partial designs using incremental QBF solving. In: Proc. of the Int. Conf on Design, Automation & Test in Europe (DATE 2012). pp. 623–628. IEEE (2012)
17. Marin, P., Narizzano, M., Giunchiglia, E., Lewis, M.D.T., Schubert, T., Becker, B.: Comparison of knowledge sharing strategies in a parallel QBF solver. In: Proc. of the Int. Conf. on High Performance Computing & Simulation (HPCS 2009). pp. 161–167. IEEE (2009)

18. Mota, B.D., Nicolas, P., Stéphan, I.: A new parallel architecture for QBF tools. In: Proc. of the Int. Conf. on High Performance Computing and Simulation (HPCS 2010). pp. 324–330. IEEE (2010)
19. Nadel, A., Ryvchin, V.: Efficient SAT solving under assumptions. In: Proc. of the Int. Conf. on Theory and Applications of Satisfiability Testing (SAT 2012). LNCS, vol. 7317. Springer (2012)
20. Vautard, J., Lallouet, A., Hamadi, Y.: A parallel solving algorithm for quantified constraints problems. In: Proc. of the 22nd IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2010). pp. 271–274. IEEE Computer Society (2010)
21. Zhang, L., Malik, S.: Towards a Symmetric Treatment of Satisfaction and Conflicts in Quantified Boolean Formula Evaluation. In: Int. Conf. on Principles and Practice of Constraint Programming (CP 2002). LNCS, vol. 2470, pp. 200–215. Springer (2002)