Picoso – A Parallel Interval Constraint Solver*

Natalia Kalinnik[1] Tobias Schubert[1] Erika Ábrahám[2] Ralf Wimmer[1] Bernd Becker[1]

[1] Faculty of Engineering,
Albert-Ludwigs-University Freiburg,
79110 Freiburg im Breisgau, Germany
{kalinnik | schubert | wimmer | becker}@informatik.uni-freiburg.de

[2] Chair of Computer Science II,
RWTH Aachen,
52056 Aachen, Germany
eab@informatik.rwth-aachen.de

***Abstract*—This paper describes the parallel interval constraint solver Picoso, which can decide (a subclass of) boolean combinations of linear and non-linear constraints. Picoso follows a master/client model based on message passing, making it suitable for any kind of workstation cluster as well as for multi-processor machines. To run several clients in parallel, an efficient work stealing mechanism has been integrated, dividing the overall search space into disjoint parts. Additionally, to prevent the clients from running into identical conflicts, information about conflicts in form of conflict clauses is exchanged among the clients. Performance measurements, using four clients to solve a number of benchmark problems, show that Picoso yields (almost) linear speedup compared to the sequential interval constraint solver iSAT, on which the clients of Picoso are based.**

I. Introduction

Recent trends in hardware design towards multi-core and multi-processor systems, computer clusters, and supercomputers call for the development of dedicated parallel algorithms in order to exploit the full potential of these architectures. In this paper we describe a parallel satisfiability modulo theories architecture and its implementation, and present some results.

The *propositional satisfiability problem* poses the question if a propositional formula is satisfiable, i. e., if there is an assignment mapping values to the variables in the formula such that the formula evaluates to true. *SAT-solvers* are devoted to solve such questions. One of the strongest industrial application field of SAT-solving is in circuit verification.

Extending the propositional logic by embedding some theories, e. g., equalities, uninterpreted functions, or theories over the reals, results in powerful logics and leads to the *satisfiability modulo theories* (SMT) problem. *SMT-solvers* find applications in several verification domains, for example in bounded model checking of hybrid systems, which is the focus of our interest and serves us with benchmarks for this paper.

*This work was partly supported by the German Research Council (DFG) as part of the Transregional Collaborative Research Center "Automatic Verification and Analysis of Complex Systems" (SFB/TR 14 AVACS). See www.avacs.org for more information.

In the last decade we could observe a massive increase in the *efficiency* of SAT- and SMT-solvers. This success, which led to industrial acceptance and employment, was mostly due to new efficient heuristics for and optimizations of the *sequential* SAT-solving algorithms. The most prominent examples are conflict-driven non-chronological backtracking, the usage of watches, and the VSIDS variable selection heuristics.

Another research direction for increasing efficiency is the *parallelization* of satisfiability checking algorithms, which is a non-trivial challenging problem. In the domain of propositional satisfiability solving, parallel SAT algorithms can be traced back to at least 1994, when Böhm and Speckenmeyer presented the first parallel implementation of a DPLL procedure for a transputer system consisting of up to 256 processors [1]. During the past decade, a number of more advanced implementations of parallel SAT-solvers have been developed, e. g. [2], [3], [4]. However, we are not aware of any published parallel SMT-solver. In this paper we introduce such a parallel SMT-solver for the first order theory of the reals extended with transcendental functions. This is a very powerful logic, and allows to formalize properties of systems with continuous components, like this is the case for hybrid systems.

Hybrid systems are systems with combined discrete and continuous dynamics. An often cited simple example is a thermostat, which switches a heater on and off (building the discrete part), thereby keeping the temperature of a room (the continuous component) between two threshold values. The complexity of hybrid systems and their occurrence in safety-critical applications pose a challenge in verification.

One of the most successful methods in formal verification is *bounded model checking* (BMC) [5], a powerful technique to refute safety properties, i. e., to *detect errors* in systems. Given a system and a (safety or liveness) property, BMC searches for an execution of increasing length, which violates the property. The existence of such an execution gets encoded as a formula, which is checked for satisfiability by an appropriate solver for the underlying logic.

BMC, originally developed for discrete systems, has also

been adopted to hybrid systems [6], [7], [8]. As the behavior of a hybrid system can be non-linear and non-deterministic, the resulting BMC formula is a boolean combination of rich arithmetic constraints over the reals (involving transcendental functions like $\sin$, $\log$, ...). Such BMC formulae are usually difficult and time-consuming to solve. Thereby, the efficiency of the BMC procedure heavily depends on the efficiency of the underlying solver. Besides further optimizing the (sequential) cores of such solvers, parallelization is a promising way to reduce the computation time.

Note that the satisfiability question of this problem class is undecidable. To nevertheless be able to handle problems from this logical domain, the authors of [9] proposed the iSAT algorithm, which uses interval arithmetic (cf. e. g., [10]).

In this paper we present the parallel interval constraint solver Picoso. To our best knowledge, Picoso is currently the only parallel solver for the undecidable domain of non-linear constraints involving transcendental functions. The algorithmic core of Picoso is based on the above-mentioned interval constraint solver iSAT and has been extended by an adaptation of the classical master/client model. The basic idea of our parallelization is to decompose the search space on demand into disjoint parts, which are then solved by the involved clients (copies of iSAT) in parallel. Additionally, distribution of conflict clauses among the clients has been integrated, too. This allows the clients to directly benefit from information others have learnt about the given benchmark problem and in particular prevents the clients from running into identical conflicts. The complete communication has been carried out using MPICH2 [11], an implementation of the Message Passing Interface (MPI) standard [12].

The remainder of the paper is structured as follows: Section II introduces the logical domain and the basic iSAT satisfiability checking algorithm. Section III describes the parallelization of iSAT, resulting in our parallel solver Picoso. After presenting benchmark results in Section IV, we conclude with a summary of the work done so far.

II. iSAT

The iSAT algorithm [9] checks the satisfiability of formulae being the boolean combination of boolean variables and non-linear (including transcendental) arithmetic constraints over bounded reals and integers. This algorithm tightly integrates SAT solving based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [13] and interval constraint propagation (see e. g. [14]) enriched by enhancements like conflict-driven learning and non-chronological backtracking.

A. Syntax and semantics of iSAT formulae

The input language of iSAT consists of arbitrary boolean combinations of boolean variables (propositions) and non-linear arithmetic constraints over the reals and integers. For simplicity, we restrict ourselves in this paper to the real domain.[1]

By the front-end of our constraint solver, these formulae are rewritten into equi-satisfiable formulae in conjunctive normal form (CNF, a conjunction of disjunctions). This rewriting applies the Tseitin-transformation [15], which can be done in linear time on the cost of new auxiliary variables. Also the size of the resulting CNF is linear in the number of operators in the input formula.

Furthermore, all arithmetic constraints, which are (in)equations, get decomposed into simpler (in)equations containing at most 3 variables. This happens by introducing additional auxiliary variables for all inner nodes of the constraint expressions. For example, $\sin(x \cdot y) \leq 2$ gets decomposed into $x \cdot y = h_1 \wedge \sin h_1 = h_2 \wedge h_2 \leq 2$, where h_1 and h_2 are fresh auxiliary variables. After this transformation the resulting equi-satisfiable formula contains (1) equations between a variable and a unary (e. g., $\sin h_1 = h_2$) or binary operator expression over variables (e. g., $x \cdot y = h_1$) and (2) bounds on variables (e. g., $h_2 \leq 2$). We benefit from this transformation in the form of simple interval propagation rules, see Section II-B.

Thus, the abstract syntax of formulae is defined as follows:

$$\begin{array}{rcl} \mathit{formula} & ::= & \mathit{clause} \mid \mathit{formula} \wedge \mathit{clause} \\ \mathit{clause} & ::= & \mathit{atom} \mid \mathit{clause} \vee \mathit{atom} \\ \mathit{atom} & ::= & \mathit{bound} \mid \mathit{equation} \\ \mathit{bound} & ::= & \mathit{var}\ \mathit{comp\_op}\ \mathit{rational\_const} \\ \mathit{equation} & ::= & \mathit{pair} \mid \mathit{triplet} \\ \mathit{pair} & ::= & \mathit{var} = \mathit{unary\_op}\ \mathit{var} \\ \mathit{triplet} & ::= & \mathit{var} = \mathit{var}\ \mathit{binary\_op}\ \mathit{var} \end{array}$$

where var is a real-valued variable, $\mathit{comp\_op}$ is one of $<$, $\leq$, $=$, $\geq$, or $>$, $\mathit{unary\_op}$ and $\mathit{binary\_op}$ are unary and binary operation symbols, respectively, and *rational_const* ranges over the rational numbers.

The semantics of formulae is straightforward. Let *Var* be the set of variables. A *valuation* $\beta : \mathit{Var} \to \mathbb{R}$ satisfies a constraint formula iff the formula resulting from replacing each variable x by its value $\beta(x)$ evaluates to true. A formula is satisfiable iff there is a satisfying valuation. Otherwise it is called unsatisfiable. Note that a formula in CNF is satisfied by a valuation iff all of its clauses are satisfied, i. e., iff at least one atom in each clause is satisfied.

B. The iSAT algorithm

Propositional SAT-solvers basically execute a loop consisting of (1) making a *decision*, (2) *propagating* the decision, and (3) if a conflict occurred, *resolving the conflict*.

Let us explain the SAT-procedure on an example. Assume a problem consisting of two clauses $(x \vee y) \wedge (x \vee \overline{y})$, where $\overline{y}$ stands for the negation of y. A solver could *decide* to search first for solutions with x being false. Using the clause $(x \vee y)$, the solver would *propagate* that, under the current decision,

[1] Since we assume that the variable domains are all restricted by both upper and lower bounds, the integer case reduces to finite domains, and is therefore decidable. Note that boolean variables can be represented as integers with domain $[0, 1]$.

y must be true. Similarly, with $(x \vee \overline{y})$ it would propagate that y should be false, leading to a *conflict*. Modern solvers use a resolution-based conflict analysis procedure to *resolve a conflict*. The analysis yields a new clause called a *conflict clause*, that gets added to the formula in order to prevent the solver from running into the same conflict again. In the example above the solver would undo its decision about x and learn the conflict clause (x), which now leads the search into the other half of the search space.

Since the domain is finite, the method is complete, i. e., either the solver finds a conflict-free assignment, or after having tried all necessary cases, it concludes that the formula is unsatisfiable.

Now, let us switch from the propositional to the real domain. Usually, SMT-solvers adapt the SAT-solving mechanism and combine it with other decision procedures for the given theory. However, the logic we deal with is undecidable, so there is no (complete) decision procedure we could embed.

Instead of valuations, the iSAT algorithm manipulates interval valuations, assigning an (open or closed, possibly also point) *interval* from the real domain, a so-called *box*, to each variable, within which it currently searches for a satisfying solution. Initially, all variables are bounded by an *initial box*. A *decision* in iSAT consists of *splitting* a box into two halves, and deciding in which one to search first. *Propagation* can, similarly to the propositional case, restrict the possible variable values under the current decisions, leading to smaller boxes. A *conflict* occurs when the box of a variable becomes empty. Similarly to the propositional case, iSAT learns a *conflict clause* that prevents the solver from future conflicts with the same "reason".

Let us again use a simple example for illustration. Let $(x = y + z \vee x > 8.0)$ be a clause and let the current boxes for x, y, and z be $[-5.0, 10.0]$, $[3.2, 3.8]$, and $[4.2, 5.0]$, respectively.

Assume that the solver *decides* to split the box of x at 8.0 and to search first for solutions from $[-5.0, 8.0]$. By propagation, iSAT would recognize that the literal $x > 8.0$ cannot be satisfied by any values from the box of x, and thus the literal $x = y + z$ must be fulfilled in order to satisfy the clause under the current decisions.

From $x = y + z$ we know that the value of x must be in the interval of $y + z$, i. e., in $[3.2 + 4.2, 3.8 + 5.0] = [7.4, 8.8]$. By the intersection with the current box $[-5.0, 8.0]$ of x, we can compute a new, tighter interval for x without the loss of any possible solutions, namely $[7.4, 8.0]$. We can further deduce from the redirection $z = x - y$ that z must be in $[7.4-3.8, 8.0-3.2] = [3.6, 4.8]$ yielding the new box $[4.2, 4.8]$ for z. Such deduced new interval bounds may trigger further propagation steps.

For efficiency and termination reasons, we avoid long interval propagation chains with just negligible progress: the propagation stops if a fixed point is reached or if the progress of the newly deduced intervals becomes negligible.

Due to undecidability, iSAT terminates with one of three possible answers: (1) it reports satisfiability if it finds an interval valuation (which can also consist of point intervals) such that all values from all intervals satisfy the formula, or (2) it reports unsatisfiability if all branches lead to conflicts, or (3) it reports a valuation containing an approximate solution.

Note that having the possibility of an "unknown"-answer, like the approximate solution, is unavoidable due to the undecidability of the logical domain. The source of such approximate solutions is the following: In general, equations like $x = y \cdot z$ can only be satisfied by point intervals. However, reaching such point intervals by propagation cannot be guaranteed for continuous domains. One option to mitigate this problem is to stop the search when all intervals have a width smaller than a certain threshold, the so-called *minimal splitting width*, and to return the found approximation of a solution.

In [9] a comparison between iSAT and ABSolver [16], the only other SMT-based solver addressing the domain of boolean combinations of non-linear arithmetic constraints over the reals, is given. The results clearly outline, that iSAT yields orders of magnitude of speedup compared to ABSolver, making it the number one choice for the algorithmic core of a parallel solver, tackling the particular domain considered here.

III. PICOSO

In this section we describe the design of Picoso by introducing its communication scheme and the work distribution mechanism. Furthermore, we discuss the integration of lemma exchange.

A. General architecture and communication structure

The implementation of the distributed solver Picoso follows the well-known master/client model. There are several clients (copies of iSAT), performing the search process, and a master process, acting as the coordinator by controlling the work distribution and the sharing of conflict clauses among the clients. Communication takes place only between the master and individual clients. Currently, there is no direct communication between clients.

During the initialization phase all available clients read the same input formula and store the problem clauses in their local database. Then the first client determines all implications forced by the initial intervals of the variables (also specified in the input formula) and starts solving the entire problem. All other clients are idle and send requests for work to the master.

When receiving a request for work, the master selects a non-idle client and asks it for a subproblem still to be solved. The selected client determines such a subproblem according to the mechanism described in Section III-B, and sends it to the master which transfers it to the idle client. During the search process, this kind of work stealing is performed whenever a client is ready with solving its previous subproblem.

Similarly to the sequential solver, Picoso terminates with (1) UNSAT, if all clients are idle, or with (2) SAT, if a client was able to find a model, or with (3) APPROXIMATE SOLUTION, if one of the clients has found an approximate solution.

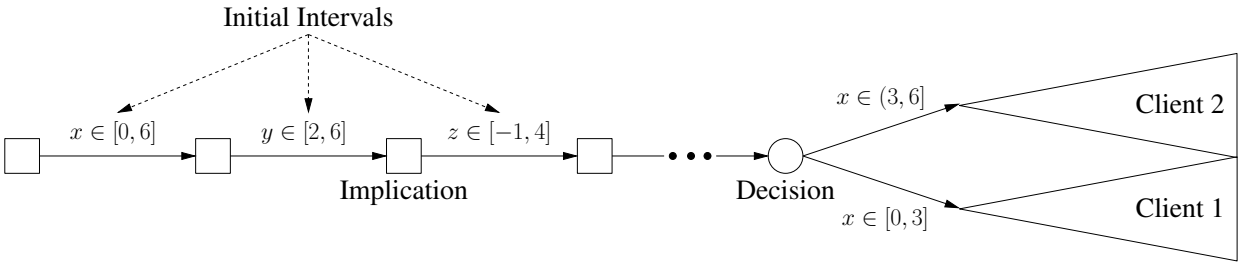

Fig. 1. Search space partitioning at interval splitting points in Picoso

In more detail, the task responsibilities of the master are as follows:

- to start/stop the clients,
- to receive work requests from idle clients,
- to select a running client for providing an unevaluated subproblem,
- to receive subproblems from non-idle clients and to forward them to idle clients,
- to receive/forward conflict clauses from/to clients, and
- to receive the results from the clients.

Likewise, the tasks of the client processes are as follows:

- to receive and solve a new subproblem,
- to return an unevaluated part of the client's current subproblem (if asked for by the master),
- to forward *well-suited* conflict clauses to the master process,
- to receive conflict clauses from the master and to insert them into the client's local clause database, and
- to send the results to the master process.

As mentioned in the introduction, the communication has been realized using MPICH2 [11], an implementation of the Message Passing Interface standard [12]. According to the master/slave model sketched above, all communication tasks are encoded as *messages* and sent/received using `MPI_Send` and `MPI_Recv` respectively.

B. Problem splitting from the client's perspective

As explained above, the clients work in parallel, and each time a client is ready with its work (and did not find any solution (or approximate solution)), it asks for a new subproblem to solve. In the following we describe how those subproblems are determined.

The basic idea is to split the search space on demand into disjoint parts, which are then treated in parallel by different processes. For this purpose, we extended the concept of *guiding paths*, first introduced in [17] for parallel SAT-solvers, to the richer and more complex framework of SMT solving.

In our setting, a guiding path describes the current search process of a client solver, extended with some information about which subproblems still need to be solved. If the master asks a client for a subproblem, the client remembers with the help of these flags which subproblem it gave away, and thus does not need to search through.

More formally, a guiding path of a solver is a sequence of bounds, consisting of all decisions and propagated implications in chronological order with a flag attached to each sequence element. The flag of a sequence element stores if the subproblem corresponding to the subsequence up to this element and the negation of this element still needs to be checked. A new decision appends the chosen bound to the guiding path with its flag set to true, since the decision is a box split, and the other half of the split box has not yet been handled. Bounds which are implied by propagation are consequences of earlier decisions. Consequently, the combination of earlier decisions with the negation of an implied bound is conflicting and does not need to be checked: they are appended to the guiding path with a flag set to false. In case of a conflict, backtracking removes also the undone decisions and implications from the guiding path.

When a client gets asked by the master for a subproblem, it is in principle free to pick any bound with a true flag from its own guiding path to generate a new and unevaluated subproblem. However, for different reasons it is a good policy to select the top-most one (but other choices would also be conceivable). The search space division is performed by returning to the master the subproblem consisting of all the bounds on the guiding path preceding the chosen bound as well as the complement of the chosen bound. Since this subproblem now gets solved by another client, the sending client sets the flag of the selected bound to false. The client which receives the generated subproblem sets all flags on its initial guiding path to false.

Besides the exchange of conflict clauses (see Section III-D) and the generation of new subproblems as described above, we also modified the core iSAT algorithm to be able to start the search process at any point within the search space specified by an initial guiding path. During the search, the clients periodically check (per default after each third decision) for incoming messages from the master, using the `MPI_Iprobe` command.

To illustrate the generation of subproblems as it is done in Picoso, assume we have two clients and a formula, consisting of three real-valued variables x, y, and z with initial intervals $x \in [0,6]$, $y \in [2,6]$, and $z \in [-1,4]$. Figure 1 shows this scenario. After determining the implications forced by the initial intervals for x, y, and z, the first client starts the search process by making a decision, e. g., splitting the interval of x

into $x \in [0,3]$ and $x \in (3,6]$, and deciding to evaluate the branch $x \in [0,3]$ first. This decision can be represented by the simple bound $x \leq 3$ with flag true. As a consequence, the guiding path of the first client is as follows:

$$\begin{aligned} GP_1 = [\ & (x \geq 0, \texttt{false}), (x \leq 6, \texttt{false}), \\ & (y \geq 2, \texttt{false}), (y \leq 6, \texttt{false}), \\ & (z \geq -1, \texttt{false}), (z \leq 4, \texttt{false}), \\ & (x \leq 3, \texttt{true})\,] \end{aligned}$$

Client 2 is still idle and sends a work request to the master process, which asks client 1 to provide an unevaluated subproblem. As described before, client 1 picks all bounds preceding the first simple bound with a true flag ($x \leq 3$) and the complement of that particular bound ($x > 3$) to specify a new subproblem:

$$\begin{aligned} GP_2 = [\ & (x \geq 0, \texttt{false}), (x \leq 6, \texttt{false}), \\ & (y \geq 2, \texttt{false}), (y \leq 6, \texttt{false}), \\ & (z \geq -1, \texttt{false}), (z \leq 4, \texttt{false}), \\ & (x > 3, \texttt{false})\,] \end{aligned}$$

As can be seen, GP_2 contains the two bounds $x \geq 0$ and $x > 3$. Since the latter one is stronger than the first one, $x \geq 0$ can be removed from the guiding path, resulting in:

$$\begin{aligned} GP_2' = [\ & (x \leq 6, \texttt{false}), \\ & (y \geq 2, \texttt{false}), (y \leq 6, \texttt{false}), \\ & (z \geq -1, \texttt{false}), (z \leq 4, \texttt{false}), \\ & (x > 3, \texttt{false})\,] \end{aligned}$$

Finally, client 1 sets the flag of its bound $x \leq 3$ to false and sends GP_2' to the master, which forwards it to client 2. Unlike in the boolean case, guiding paths may contain atoms which are implied by other atoms on the guiding path (like $x \geq 0$ and $x > 3$ in the example above). They are redundant and therefore removed automatically by the ceding client in order to reduce the size of the messages.

C. Work distribution from the master's perspective

When designing a parallel algorithm it is important to evenly distribute the workload among the processes in order to keep the communication as low as possible. In general the size of the search tree and moreover the computation time for its exploration are unpredictable. Therefore, in Picoso the work load is distributed dynamically among clients at runtime. To assure the most efficient use of the available computation power provided by the distributed architecture, a dynamic work stealing policy, similar to [4], [18], has been implemented.

In Picoso this works as follows: Suppose one client finished exploring its current subproblem and terminates with UNSAT. This client can be re-used for the exploration of the next subproblem and because of this it sends a work request to the master. Subsequently, the task of the master process is to determine a client that should split its current fraction of the search space and provide an unevaluated subproblem.

Usually, there are several active clients and the master is free to pick out any of them for the problem distribution. An essential criterion for the client selection is the current work load of a client. The bounds of a guiding path represent an n-dimensional box. For each client the master computes the volume of the box, formed by the client's guiding path. Larger guiding path boxes presumable correspond to larger portions of the search space. Motivated by this fact, the master asks the client with the maximum volume to split its current subproblem.

D. Sharing of conflict clauses

In sequential SAT solving learning plays a very important role to prune the search space. Generating and recording information in form of conflict clauses (also referred to as *lemmas*) prevents the solver from attempting the same assignments and running into the same conflicts again. It may also prevent the solver from visiting those parts of the search space that can be inferred to contain no satisfying assignment. Thus, exchanging lemmas generated by one client with the other clients can help all clients to prune the unexplored search space faster and by this to improve the overall parallel solver performance. Nevertheless, exchanging conflict clauses is associated with a communication overhead, therefore it is very important to decide which conflict clauses are worthwhile to share.

If the mechanism of sharing learnt clauses is switched off, then lemmas are exchanged only in a natural way: If one client finds out that its subproblem is unsatisfiable, then the clause database is preserved for the next solver run.

When sharing is active, then the clauses to be exchanged are selected according to their lengths. In Picoso, clients share only "short" conflict clauses in order to minimize the communication overhead. Furthermore, short clauses are expected to be more helpful in generating further implications. Based on experimental evaluation we select learnt clauses having a length less or equal to 6. To reduce the communication costs all selected lemmas are bundled up in a transfer buffer with length 20. A bundle of conflict clauses is communicated to the master as soon as the corresponding buffer is filled completely. The master process gets the conflict clauses and distributes them among the available clients. The clients analyze received learnt clauses immediately and integrate them in their own clause database. This ensures that a client becomes promptly aware that its current subproblem is unsatisfiable if a received conflict clause is violated.

E. Other approaches and planned work

In the previous sections we introduced the parallel algorithm of Picoso. The underlying master/client MPI architecture is one of several possible solutions for the communication between different solvers. In order to optimize our solver, in the future we will implement and compare different approaches.

In the master/client approach all clients send their conflict clauses to a single master, which distributes them. Another possibility would be a direct client-to-client communication of that information. It is not clear which approach is better. On the one hand, in the master/client setting the clients do not have to invest much time in communication. On the other

hand, the main communication overhead is left to the master. This might be a bottleneck when having a large number of clients.

Instead of (or additionally to) MPI-based communication we could also consider a multi-threaded solution. Threads have access to a shared memory, thus parts of the communication become superfluous. However, synchronization is expensive, thus also here the question, which solution is better, remains open.

To delegate work, in the current implementation the search space is divided into disjoint parts, that are searched through by the different solvers. However, a "good" partitioning of the search space is hard to determine. For example in the case of an unsatisfiable formula, a division based on the variables which occur in the *core* of unsatisfiability, i. e., which cause unsatisfiability, would probably be advantageous. However, we do not know those variables in advance. As future work we will try different search space splitting heuristics.

IV. Experimental Results

In this section we compare the performance between the sequential solver iSAT and our parallel solver Picoso. The experiments were performed on a machine with four 2.3 GHz AMD processors and 64 GByte of physical memory, running Ubuntu 8.04.1x86_64 GNU/Linux.

For our measurements, we used a set of four non-linear and one linear BMC benchmarks. The non-linear ones are (1) a controller for train separation implementing a "moving block" interlocking scheme of the forthcoming European Train Control System Level 3 [19] (*etcs_train*), (2) a model of the discrete-continuous behavior of a golf ball on a minigolf course (*minigolf*), (3) a conflict resolution protocol for air traffic management aiming at avoiding collisions between airplanes, which was specified by Tomlin [20] (*tomlin*), and (4) a model of a car parking assistant (*parking*). The linear benchmark describes an elastic approach to distance control of trains running on the same track [8] (*train_distance*). Here, trains can accelerate or decelerate freely if they do not violate their mutual safety envelopes. An automatic speed control takes authority over a train if another train gets close, thereby controlling acceleration proportional (within physical limits) to the front and/or back proximity of the neighboring trains.

For all benchmarks we created input formulae for the two solvers by unrolling the modelled systems a certain number of times, which is given in the column denoted by "depth". The solver runs are independent from each other, i. e., no information (e. g. about conflicts) is kept between different unrolling depths of the same system. Thereby, we selected those depths for which the sequential solver needs at least 60 s to compute a result and for which at least one of the two solvers does not exceed the time limit. For *tomlin* and *parking* we only used one unrolling depth because, for all other depths, the formula is unsatisfiable and both solvers terminate within a few seconds.

All experiments in Table I were run with four clients and one master process. The clients are allowed to share learnt clauses with length less or equal to 6. The length of the transfer buffer for conflict clauses was set to 20. The computation times for Picoso are the arithmetical mean of five individual runs. We set a time-out limit of 30 000 s for both the sequential and the parallel solver. In the last column we give the speedup of the parallel version compared to the sequential one.

TABLE I
Experimental Results

Benchmark	depth	iSAT time [s]	Picoso time [s]	result	speedup
minigolf	8	80.32	28.60	unsat	2.79
minigolf	9	305.44	123.00	unsat	2.47
minigolf	10	2005.29	944.60	unsat	2.12
minigolf	11	10932.27	4176.00	unsat	2.61
minigolf	12	–time out–	–time out–	unknown	—
etcs_train	37	80.00	23.80	unsat	3.36
etcs_train	40	127.55	38.20	unsat	3.32
etcs_train	41	105.74	43.40	unsat	2.41
etcs_train	42	144.46	47.00	unsat	3.06
etcs_train	43	151.22	70.00	unsat	2.15
etcs_train	44	97.58	66.60	unsat	1.46
etcs_train	45	177.47	63.00	unsat	2.80
etcs_train	46	393.86	42.80	unsat	9.18
etcs_train	47	184.81	95.60	unsat	1.93
etcs_train	48	253.50	93.60	unsat	2.70
etcs_train	49	311.49	92.60	unsat	3.35
etcs_train	50	597.37	82.60	unsat	7.22
etcs_train	51	313.75	103.20	unsat	3.03
etcs_train	52	683.84	150.80	unsat	4.52
etcs_train	53	329.67	90.20	unsat	3.65
etcs_train	54	367.00	142.00	unsat	2.58
etcs_train	55	703.12	245.60	unsat	2.86
etcs_train	56	678.10	249.40	unsat	2.71
etcs_train	57	590.33	210.80	unsat	2.80
etcs_train	58	411.31	323.80	unsat	1.27
etcs_train	59	1923.10	537.60	unsat	3.58
etcs_train	60	4716.36	1462.00	unsat	3.22
etcs_train	61	–time out–	25568.00	approx. sol.	>1.17
train_distance	8	85.94	12.53	unsat	7.08
train_distance	9	158.50	28.20	unsat	5.64
train_distance	11	149.04	46.40	unsat	3.21
train_distance	12	2290.81	115.00	unsat	19.91
train_distance	13	642.09	137.60	unsat	4.68
train_distance	14	3012.74	592.00	unsat	5.08
train_distance	15	22904.00	1798.80	unsat	12.73
train_distance	16	–time out–	12193.00	unsat	>2.46
train_distance	17	–time out–	21471.00	unsat	>1.39
train_distance	18	–time out–	–time out–	unknown	—
tomlin	3	212.00	197.40	approx. sol.	1.07
parking	53	108.00	33.20	approx. sol.	3.25

For *all* instances we can observe a significant speedup. The average speedup of all benchmarks is > 3.8 which is linear in the number of clients. Remarkably, for some of the *train_distance* instances, the speedup is super-linear. This is due to the different evaluation order compared to the sequential solver and the conflict clause exchange: Assume that one clients computes a conflict clause c in a part of the search space which would be considered by the sequential solver only towards the end of the search process. If c prunes a considerable part of the overall search space, this obviously leads to a super-linear speedup.

Moreover, the parallel solver is able to solve some instance for which the sequential solver fails due to the time limit.

V. Conclusion

In this paper we presented Picoso, which—to the best of our knowledge—is the first published parallel SMT-solver for boolean combinations of linear and non-linear arithmetic constraints.

The parallelization is based on the classical master/client model and can readily be adapted to other SMT engines. The algorithmic core of the clients is formed by the iSAT algorithm which merges interval constraint propagation with boolean SAT solving techniques to decide the satisfiability of such formulae. To distribute the workload evenly among the clients, it employs dynamic work stealing based on guiding paths and exchanges information in form of conflict clauses. To reduce the communication overhead, only conflict clauses of a bounded length are distributed.

The experimental results demonstrate that Picoso yields a (nearly) linear speedup for both linear and non-linear benchmarks and thus provides an excellent basis for further research in this area. This will include improvement of partitioning strategies, integration of shared memory concepts and application specific parallelism.

References

[1] M. Böhm and E. Speckenmeyer, "A fast parallel SAT-solver – efficient workload balancing," *Annals of Mathematics and Artificial Intelligence*, vol. 17, no. 3-4, pp. 381–400, 1996.

[2] M. D. T. Lewis, T. Schubert, and B. Becker, "Multithreaded SAT solving," in *Asia and South Pacific Design Automation Conference (ASPDAC)*. IEEE Computer Society, 2007, pp. 926–931.

[3] Y. Feldman, N. Dershowitz, and Z. Hanna, "Parallel multithreaded satisfiability solver: Design and implementation," *Electronic Notes in Theoretical Computer Science*, vol. 128, no. 3, pp. 75–90, 2005.

[4] T. Schubert, M. D. T. Lewis, and B. Becker, "PaMira – A parallel SAT solver with knowledge sharing," in *6th Int'l Workshop on Microprocessor Test and Verification (MTV 2005)*. IEEE Computer Society, 2005, pp. 29–36.

[5] E. M. Clarke, A. Biere, R. Raimi, and Y. Zhu, "Bounded model checking using satisfiability solving," *Formal Methods in System Design*, vol. 19, no. 1, pp. 7–34, 2001.

[6] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani, "Verifying industrial hybrid systems with MathSAT," in *Int'l Workshop on Bounded Model Checking (BMC'04)*, ser. Electronic Notes in Theoretical Computer Science, vol. 119, no. 2, 2005, pp. 17–32.

[7] E. Ábrahám, B. Becker, F. Klaedke, and M. Steffen, "Optimizing bounded model checking for linear hybrid systems," in *Int'l Workshop on Verification, Model Checking, and Abstract Interpretation (VMCAI)*, ser. Lecture Notes in Computer Science, vol. 3385. Springer-Verlag, 2005, pp. 396–412.

[8] M. Fränzle and C. Herde, "HySAT: An efficient proof engine for bounded model checking of hybrid systems," *Formal Methods in System Design*, vol. 30, no. 3, pp. 179–198, 2007.

[9] M. Fränzle, C. Herde, T. Teige, S. Ratschan, and T. Schubert, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *Journal on Satisfiability, Boolean Modeling, and Computation*, vol. 1, 2007.

[10] R. E. Moore, *Interval Analysis*. Englewood Cliffs, NJ, USA: Prentice Hall, 1966.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A high-performance, portable implementation of the MPI message passing interface standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, Sept. 1996.

[12] M. Snir, S. Otto, D. Walker, J. Dongarra, and S. Huss-Lederman, *MPI: The Complete Reference*. MIT Press, 1995.

[13] M. Davis, G. Logemann, and D. Loveland, "A machine program for theorem proving," *Communications of the ACM*, vol. 5, pp. 394–397, 1962.

[14] F. Benhamou and L. Granvilliers, "Continuous and interval constraints," in *Handbook of Constraint Programming*, ser. Foundations of Artificial Intelligence. Elsevier Science Publishers, 2006, ch. 16, pp. 571–603.

[15] G. S. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pp. 115–125, 1970.

[16] A. Bauer, M. Pister, and M. Tautschnig, "Tool-support for the analysis of hybrid systems and models," in *Int'l Conf. on Design, Automation, and Test in Europe (DATE)*. San Jose, CA, USA: EDA Consortium, 2007, pp. 924–929.

[17] H. Zhang, M. P. Bonacina, and J. Hsiang, "PSATO: A distributed propositional prover and its application to quasigroup problems," *Journal of Symbolic Computation*, vol. 21, no. 4, pp. 543–560, 1996.

[18] B. Jurkowiak, C.-M. Li, and G. Utard, "Parallelizing SATZ using dynamic workload balancing," in *Workshop on Theory and Applications of Satisfiability Testing (SAT 2001)*, vol. 9. Elsevier Science Publishers, June 2001.

[19] C. Herde, A. Eggers, M. Fränzle, and T. Teige, "Analysis of hybrid systems using HySAT," in *Int'l Conf. on Systems (ICONS)*. IEEE Computer Society, 2008, pp. 196–201.

[20] C. Tomlin and S. Sastry, "Conflict resolution for air traffic management: A study in multi-agent hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, pp. 509–521, 1998.