Exploiting Different Strategies for the Parallelization of an SMT Solver*

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

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

[2] RWTH Aachen
52056 Aachen, Germany
eab@informatik.rwth-aachen.de

In this paper we present two different parallelization schemes for the SMT solver iSAT, based on (1) the distribution of work by dividing the search space into disjoint parts and exploring them in parallel, thereby exchanging learnt information, and (2) a portfolio approach, where the entire benchmark instance is explored in parallel by several copies of the same solver but using different heuristics to guide the search. We also combine both approaches such that solvers examine disjoint parts of the search space using different heuristics. The main contribution of the paper is to study the performances of different techniques for parallelizing iSAT.

1 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 focus on the evaluation of different parallelization schemes for SAT modulo theories (SMT) solvers.

The *propositional satisfiability problem* (SAT) 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.

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 serves us with benchmarks for this paper.

In the last decade we could observe a massive increase in the *efficiency* of SAT- and SMT-solvers. This success 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 and the usage of watches.

Though SAT- and SMT-solvers are subject of highly active research, less work was done on techniques for their *parallelization*. We can identify two steams in the area mentioned first: One class of parallel solvers, such as [13, 4, 7, 3] splits the search space dynamically into disjoint

*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.

parts and assigns them to the available processes/threads. In contrast, the other class utilizes a competitive approach by combining solvers with different heuristics and/or parameters [10] in a single portfolio. Either way, both approaches make use of exchanging information in order to examine the search space faster.

One of the few available parallel SMT-solvers is a parallel version of Z3 [6], which supports decidable theories like linear arithmetic, and which is based on the competitive approach [18]: multiple solvers with different heuristics explore the state space in parallel.

Another parallel solver is our SMT-solver Picoso [12], which supports the first order theory over the reals extended with transcendental functions. The logic is very powerful, and allows to formalize properties of systems with continuous components, like hybrid systems. The price of this power, however, is the loss of decidability. Picoso is based on the parallelization of the iSAT satisfiability checking algorithm [9], which uses interval arithmetic (cf. e. g., [14]).

Picoso can split the search space on demand into disjoint parts, which are examined in parallel. This approach has already been investigated in [12]. The contribution of this paper is the extension of the solver to support also the competitive approach. Furthermore, we implemented a combination of both parallelization approaches in such a way that the search space is split into disjoint parts, each of which is examined using a different heuristic. We describe the adaptation of the above parallelization schemes for Picoso, and give an experimental evaluation for benchmarks from hybrid systems analysis using bounded model checking (BMC). The rest of the paper is structured as follows: Section 2 introduces the iSAT satisfiability checking algorithm. Section 3 describes Picoso and the implemented parallelization approaches. Section 4 presents our experimental results. We conclude our paper in Section 6.

2 iSAT

The iSAT algorithm [9] checks satisfiability of formulae being the boolean combination of boolean variables and possible 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 [5] and interval constraint propagation (see e. g. , [2]) enriched by enhancements like conflict-driven learning and non-chronological backtracking.

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 clauses, each clause being the disjunction of literals, whereas literals are possibly negated propositions and real constraints). This rewriting applies the Tseitin-transformation [17].

Furthermore, all arithmetic constraints 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 (e. g., $x \cdot y = h_1$) operator expressions over variables and (2) bounds on variables (e. g., $h_2 \leq 2$).

[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 $[0, 1]$.

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, 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 above example 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.

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 having 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 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 a potential solution.

Note that having the possibility of an "unknown"-answer, like the potential solution, is unavoidable due to the undecidability of the logical domain. The source of such potential 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 potential solution.

In [9] a comparison between iSAT and ABSolver [1], 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.

3 Picoso

In this section we present in detail two conceptually different possibilities for efficient parallelization of the SMT solver iSAT and their combination, yielding three different parallel solver versions.

The first one, Picoso$^{\mathrm{S}}$ (S for split), exploits parallelism by splitting the search space on demand into disjoint parts. These parts are examined in parallel by different clients but using the same variable ordering heuristic. Picoso$^{\mathrm{S}}$ achieves load balancing by dynamic work stealing based on the concepts of guiding paths and a volume-based heuristic: If one client gets idle, then it steals a portion of work in form of a still unevaluated subproblem from another busy client.

Picoso$^{\mathrm{AP}}$ (AP for algorithm portfolio) exploits parallelism by competition, thereby combining copies of the sequential iSAT-solver in a static algorithm portfolio. All processes in Picoso$^{\mathrm{AP}}$ solve the whole problem in parallel with different variable ordering heuristics but without problem splitting.

The third version, Picoso$^{\mathrm{S+AP}}$, is the combination of both the competition and the search space splitting strategies. It employs dynamic partitioning of the search space based on work stealing, whereby each client uses a different variable ordering heuristic. In all three versions of Picoso the clients share information in form of conflict clauses.

The implementation of Picoso$^{\mathrm{S}}$, Picoso$^{\mathrm{AP}}$, and Picoso$^{\mathrm{S+AP}}$ follows the well-known master/slave (client) model. There is a parameterizable number of clients performing the search process and a master process, which acts as a coordinator.

Whereas Picoso$^{\mathrm{S}}$ was already investigated in [12], the novelty is the implementation and the evaluation of Picoso$^{\mathrm{AP}}$ and Picoso$^{\mathrm{S+AP}}$. To make the paper self-contained we summarize in the next subsection the implementation of Picoso$^{\mathrm{S}}$.

3.1 Picoso$^{\mathrm{S}}$: Parallelization by Search Space Splitting

In Picoso$^{\mathrm{S}}$ several clients (copies of iSAT) with the same heuristic perform the search process. A master process manages the work distribution and controls the communication of conflict clauses among the clients. Communication is centralized: there is no direct communication between the clients, only between the master and the individual clients.

During the initialization phase all available clients read the same input formula and store the problem clauses in their local database. The first client starts to solve the entire problem. All other clients are idle and send work requests to the master.

When receiving a work request, the master selects a non-idle client (based on volume a heuristic as described in detail in [12]) and asks it for a subproblem that has still to be solved. The selected client determines such a subproblem according to the mechanism described below, 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 has finished solving its current subproblem with the result "unsatisfiable" and is run out of work.

Similarly to the sequential solver, Picoso$^{\mathrm{S}}$ terminates (1) with "unsatisfiable", if all clients are idle, (2) with "satisfiable", if a client was able to find a model, or (3) with "potential solution", if a client found a potential solution.

In the following we describe how the new subproblems are split off.

Problem Splitting The basic idea is to extend the concept of *guiding paths*, first introduced in [19] for parallel SAT-solvers, to the richer and more complex framework of SMT solving. In

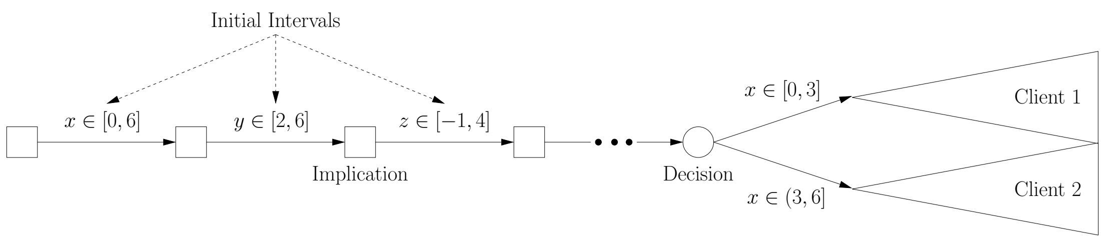


Figure 1: Search space partitioning at interval splitting points in Picoso$^{\mathrm{S}}$

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.

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 whether the subproblem corresponding to the subsequence up to this element combined with the negation of this particular 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 been handled yet. 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 the flag set to false. In case of a conflict, backtracking removes also the undone decisions and implications from the guiding path.

When a client is 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 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 bounds on the guiding path preceding the chosen bound as well as the complement of the chosen bound. Since another client will solve this subproblem, the sending client sets the flag of the selected bound to false. The client receiving the generated subproblem sets all flags in its initial guiding path to false.

To illustrate the generation of subproblems as it is done in Picoso$^{\mathrm{S}}$, 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 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 $x \leq 3$ to false and sends GP'_2 to the master, which forwards it to client 2. In contrast to the boolean case, guiding paths usually 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 message.

3.2 Picoso$^{\mathsf{AP}}$: Algorithm Portfolio Design

When the solver wants to split the box of a variable, a variable ordering heuristic is used to determine that variable. The performance of iSAT on different problem instances depends heavily on the chosen heuristic. Since none of the available heuristics performs well on all classes of problems, executing different heuristics competitively in parallel is a promising approach to make the solver more robust. The combination of different variable ordering heuristics in an algorithm portfolio may result in better performance than the one which can be obtained by a single heuristic.

General Architecture and Communication Structure During the initialization phase each client reads the whole problem and starts the search using a *different* variable ordering heuristic. The first client that finishes the search process determines the termination time of the portfolio. This client sends a finish message to the master and reports the result of the search. When receiving a finish message, the master stops all clients.

Variable Ordering Heuristics We have implemented several decision heuristics for the selection of the next variable whose domain will be split into two sub-intervals. From the supported heuristics we selected the 4 most successful ones to be considered in this paper:

• The *Natural* heuristic uses a static order which does not change during the solution process. The order is determined by the order of variable creation. In our solver, the variables occurring in the declaration are created first, then the auxiliary variables generated during the Tseitin transformation and, at last, the auxiliary variables for arithmetic expressions (see Section 2).

• The *BF+VSIDS* heuristic splits boolean variables first (BF, Boolean First). If there is a choice between boolean variables, the *Variable State Independent Decaying Sum (VSIDS)* heuristic is used to select one of them. *VSIDS* prefers those variables which were recently involved in conflicts. To have a measure for this property, *VSIDS* assigns an activity to each variable. For each conflict, the activities of those variables which played a role in the conflict resolution get increased by an increment value. To put larger weight on recent conflicts, the increment value increases with each conflict. The activities and the increment value are scaled down from time to time to prevent an overflow. For each decision, the variable with the highest activity is selected.

Though the approaches are similar, there is a difference between the *VSIDS* heuristic for iSAT and for SAT-solving. In SAT-solving a decision assigns a value to a variable, and thus the same variable cannot be chosen twice without an intermediate conflict. In iSAT a decision only splits the box of a variable, and the same decision variable can be taken several times without any conflict. Since the activities do not change when no conflict occurs, we want to avoid that the most active variable is selected for all decisions until a conflict happens. Therefore we attach to each variable a counter that keeps track of how often the variable acted as a decision variable since the last conflict, and we put an upper bound on the counter value of decision variables. We reset those counters when a conflict occurs or if all of them exceed the upper bound.

• The *VSIDS+BF* heuristic selects the variable with the highest activity according to the *VSIDS* heuristic. If there are several candidates with the same activity, boolean variables are preferred.

• The static *BMC-Forward* heuristic sorts variables according to the timeframe they belong to. The boxes of variables describing the initial configuration are split first, then those describing the state after one transition and so on.

3.3 Picoso^{S+AP}: Combination of Search Space Splitting and Algorithm Portfolio Design

In order to improve the robustness of the search space division approach, Picoso^{S} has been modified to use the four different heuristics described in Section 3.2. At the beginning of the search we assign to each solver client one fixed heuristic. Then the clients proceed with a traversal of the search space as described in 3.1.

3.4 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 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 in Picoso^{S} , Picoso^{AP} and Picoso^{S+AP} clients share only “short” conflict clauses, having a length less or equal to 6.

4 Experimental Results

The experiments were performed on a machine with four 2.3 GHz AMD processors and 32 GByte of main memory, running Ubuntu 8.04.1x86_64 GNU/Linux. We set a time limit of three hours for all experiments. The communication has been realized using MPICH2, an implementation of the Message Passing Interface standard [15].

Benchmarks. For our measurements, we used a set of seven non-linear and one linear BMC benchmarks. The non-linear ones are (1) a controller for train separation [11], (2) a model of the discrete-continuous behavior of a golf ball on a miniature golf course, (3) a collision avoidance protocol for air traffic management [16], (4) a model of a car parking assistant, (5) an asynchronous arbiter modeled at the circuit level, (6) an oscillator circuit, and (7) a bus protocol, where BMC is applied to check invariants of UsbPhy (Universal Serial Bus). The linear benchmark models an elastic distance control of trains running on the same track [8].

For all benchmarks we created input formulae for the solvers by unrolling the modeled transition systems a certain number of times. Thereby, we selected those successive depths for which all sequential solvers with different heuristics need at least 5 s to compute a result and for which at least one of the solvers (sequential or parallel version) does not exceed the time limit. The final number of benchmark instances is 65. The solver runs for different unrolling depths are independent from each other, i. e., no information (e. g., about conflicts) is shared between different unrolling depths.

Solvers. We compare four different variants of our solver: 1) iSAT is the sequential version which uses only one process and a fixed decision heuristic. 2) Picoso^{S} is a parallel solver with a parameterizable number of client processes. They split the search space into disjoint parts and solve the subproblems in parallel using the same pre-defined decision heuristic. 3) Picoso^{AP} is also a parallel solver with a parameterizable number of processes. All processes in Picoso^{AP} solve the whole problem in parallel applying different heuristics, but without problem splitting. 4) In Picoso^{S+AP} the search space is split dynamically into disjoint parts.

In all parallel versions, namely in Picoso^{AP} , Picoso^{S} and Picoso^{S+AP} , the clients cooperate with each other by exchanging derived conflict clauses.

Experimental Results. For the experimental results we have run the sequential solver iSAT and the parallel solver Picoso^{S} with each of the 4 available heuristics. Furthermore we have

Solver	#Solved instances (out of 65)	$\sum$ Time [s] for all solved instances	Average time over all solved instances	$\sum$ Time [s] for the 31 instances solved by all solvers
$\text{iSAT}_{\text{Natural}}$	49	31806.09	649.10	13152.30
$\text{iSAT}_{\text{BF+VSIDS}}$	38	19497.64	513.10	4753.40
$\text{iSAT}_{\text{VSIDS+BF}}$	46	30762.41	668.75	7340.20
$\text{iSAT}_{\text{BMC-Forward}}$	55	59728.75	1085.98	12078.28
$\text{Picoso}^{\text{S}}{}_{\text{Natural}}$	60	65345.88	1089.10	2151.46
$\text{Picoso}^{\text{S}}{}_{\text{BF+VSIDS}}$	43	29935.43	696.17	2994.33
$\text{Picoso}^{\text{S}}{}_{\text{VSIDS+BF}}$	52	28002.18	538.50	2172.14
$\text{Picoso}^{\text{S}}{}_{\text{BMC-Forward}}$	60	60533.67	1008.89	3305.94
$\text{Picoso}^{\text{S+AP}}$	63	40631.89	644.95	1639.21
$\text{Picoso}^{\text{AP}}$	64	43660.72	682.20	1399.88

Table 1: Experimental results for different solver configurations

run $\text{Picoso}^{\text{AP}}$and $\text{Picoso}^{\text{S+AP}}$. All the parallel solvers were run with four clients and one master process, since only four sufficiently different decision heuristics are available.

Information about the time consumption of the different solver variants is given in Table 1. The first column specifies the solver. The second column shows the number of instances that could be solved before the time limit was exceeded, and the third one gives the sum of the running times for all successfully solved instances (without adding up the time limit for the unsolved instances). The fourth column lists the average running time for the successfully solved instances. There were 31 instances that could be solved by all solvers. To get more comparable data and better impression about the solvers' efficiency, the sum of the running times for those 31 instances is given in the fifth column.

The first observation we make is that Picoso^{S} is superior to iSAT with respect to the number of solved instances. Furthermore, the portfolio solver $\text{Picoso}^{\text{AP}}$ listed in the last line is able to solve more problem instances than the iSAT and Picoso^{S} solvers.

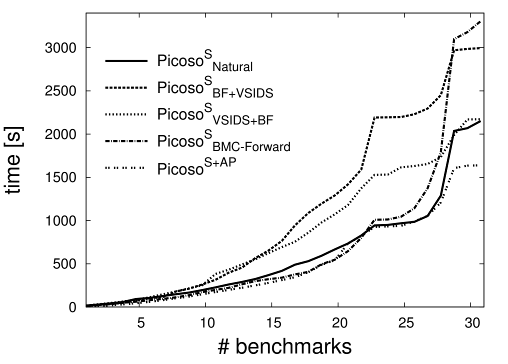
The sum of the running times increases when more instances are solved. This holds also for the average time, since those instances that could not be solved by all solvers are of course the harder problems. As the portfolio solvers solve more of those hard instances, the fact that the average running time of the portfolio solvers is in the lower area of the average running times of the iSAT and Picoso^{S} solvers shows, that the portfolio solvers are actually faster. This conclusion can be seen more directly in the last column, listing the sum of the running times for those 31 instances that could be solved by all solvers.

Figure 2(a) considers only those 31 instances and shows the comparison of Picoso^{S} with the same heuristics for each process and $\text{Picoso}^{\text{S+AP}}$, where the clients apply different heuristics to solve unevaluated subproblems. Thereby the vertical axis (time [s]) gives the sum of the running times for all at the horizontal axis registered instances. As expected, we can observe that $\text{Picoso}^{\text{S+AP}}$ is faster than all other Picoso^{S} versions.

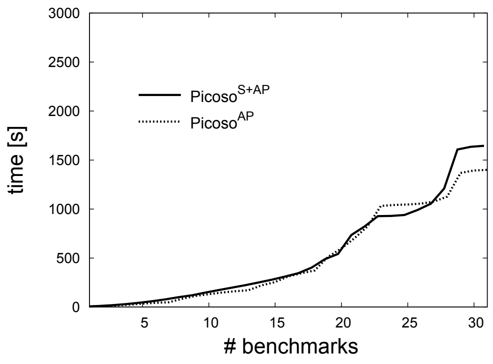
Figure 2(b) visualizes the results, again only for the 31 common instances, for the different portfolio solvers. The figure shows us that $\text{Picoso}^{\text{AP}}$ is more robust than the other solver $\text{Picoso}^{\text{S+AP}}$, and that it achieves the best results. It is able to solve problems faster and, as shown in Table 1, it can solve one more instance.

Table 2 shows the speedups yielded by the different parallel algorithms w.r.t. the sequential iSAT versions. The speedup values are defined as $\frac{\sum_{b \in B} t^{\text{seq}}(b)}{\sum_{b \in B} t^{\text{par}}(b)}$, where B is the set of benchmarks successfully solved by both compared solvers, and while $t^{\text{seq}}(b)$ and $t^{\text{par}}(b)$ denote the running times of the sequential and of the parallel solver for benchmark $b \in B$.

$\text{Picoso}^{\text{AP}}$ provides a good speedup between 3.39 and 9.39 depending on the heuristic chosen for the sequential iSAT. The second best variant, $\text{Picoso}^{\text{S+AP}}$, still provides a good speedup between 2.89 and 8.02. Since we used 4 solver clients and one master, the medium results indicate a linear speedup.



(a) Comparing different Picoso^{S} versions



(b) Comparing different Picoso^{AP} versions

Figure 2: Comparison of Picoso^{S} and Picoso^{AP}

Solver	iSAT$_{\text{Natural}}$	iSAT$_{\text{BF+VSIDS}}$	iSAT$_{\text{VSIDS+BF}}$	iSAT$_{\text{BMC-Forward}}$
Picoso^{S}$_{\text{Natural}}$	6.11	2.20	3.41	5.61
Picoso^{S}$_{\text{BF+VSIDS}}$	4.39	1.58	2.45	4.03
Picoso^{S}$_{\text{VSIDS+BF}}$	6.05	2.18	3.37	5.56
Picoso^{S}$_{\text{BMC-Forward}}$	3.97	1.43	2.22	3.65
Picoso^{S+AP}	8.02	2.89	4.47	7.36
Picoso^{AP}	9.39	3.39	5.24	8.62

Table 2: Speedups of the parallel version compared to the sequential solver

Our results outline that the combination of different heuristics for parallel SMT solvers can dramatically improve the performance and robustness. In our experiments, the algorithm portfolio design solver Picoso^{AP} achieves the best results. There are at least two reasons for its success. Firstly, solvers, especially for the non-linear domain, are very sensitive to changes in the decision heuristics. If one heuristic performs well on a given benchmark, it is better to use this good heuristic for the whole problem, as it is done by Picoso^{AP}, and not to use different heuristics for different subproblems, as it is the case for Picoso^{S+AP}. Secondly, Picoso^{AP} produces less communication overhead compared to the search space splitting variants, since we do not need to spend time for sending and receiving subproblems.

5 Future work

The scalability of the algorithm portfolio approach is limited by the number of available heuristics and/or parameters. Also for the search space division approach, we expect that linear speedup is not possible for a large number of processors due to the communication overhead. In future work we therefore plan to combine the competitive and the search space division approaches in Picoso.

Currently, we have already finished the first prototype for a combination of both approaches. In its implementation we combine several Picoso^{S} solvers in a portfolio. One Picoso^{S} solver manages a number of clients which employ the same decision heuristic. Different Picoso^{S} solvers have different heuristics; together they constitute a portfolio. All Picoso^{S} solvers are able to distribute conflict clauses among each other and maintain their own master, which is responsible for work distribution and lemma exchange. All clients that belong to the same Picoso^{S} solver explore disjoint parts of the search space with the same heuristic. The portfolio terminates as soon as one of the Picoso^{S} solvers has finished the traversal of the search space. First experimental results with this version are very promising.

As another possibility we plan to divide the search space dynamically into disjoint parts and to let the processors solve the consequent parts competitively in parallel with different heuristics. This reduces the communication between processors because the search space is split into fewer parts. But, since the same subproblem is then solved by several solver instances using different heuristics, the guiding path concept has to be generalized in order to split off further subproblems when a set of processors becomes idle.

6 Conclusion

The results in this paper show that an SMT-solver for non-linear arithmetic can be parallelized efficiently using two conceptually different techniques: by search space division and by an algorithm portfolio approach. Thereby the use of different heuristics and lemma exchange is essential to improve the performance and the robustness of the parallel SMT solver.

References

[1] 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)*, pp. 924–929, San Jose, CA, USA, 2007. EDA Consortium.

[2] F. Benhamou and L. Granvilliers. Continuous and interval constraints. In *Handbook of Constraint Programming*, Foundations of Artificial Intelligence, chapter 16, pp. 571–603. Elsevier, Amsterdam, 2006.

[3] W. Chrabakh and R. Wolski. GridSAT: a system for solving satisfiability problems using a computational grid. *Parallel Computing*, 32(9):660–687, 2006.

[4] G. Chu and P. J. Stuckey. PMiniSat – a parallelization of MiniSAT 2.0. Technical report, Department of Computer Science and Software Engineering, University of Melbourne, Australia, Mar. 2008.

[5] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

[6] L. M. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *14th Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, vol. 4963 of *LNCS*, pp. 337–340, 2008.

[7] Y. Feldman, N. Dershowitz, and Z. Hanna. Parallel multithreaded satisfiability solver: Design and implementation. *ENTCS*, 128(3):75–90, 2005.

[8] M. Fränzle and C. Herde. HySAT: An efficient proof engine for bounded model checking of hybrid systems. *Formal Methods in System Design*, 30:179–198, 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*, 1, 2007.

[10] Y. Hamadi, S. Jabbour, and L. Sais. ManySAT: a parallel SAT solver. *Journal on Satisfiability, Boolean Modelling and Computation*, 6:245–262, 2009.

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

[12] N. Kalinnik, T. Schubert, E. Ábrahám, R. Wimmer, and B. Becker. Picoso – A parallel interval constraint solver. In H. R. Arabnia, editor, *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, Las Vegas, NV, USA, July 2009. CSREA Press.

[13] M. D. T. Lewis, T. Schubert, and B. Becker. Multithreaded SAT solving. In *Asia and South Pacific Design Automation Conference*, pp. 926–931. IEEE CS, 2007.

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

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

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

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

[18] C. M. Wintersteiger, Y. Hamadi, and L. M. de Moura. A concurrent portfolio approach to SMT solving. In A. Bouajjani and O. Maler, editors, *21st Int'l Conf. on Computer Aided Verification (CAV)*, vol. 5643 of *LNCS*, pp. 715–720, Grenoble, France, 2009.

[19] H. Zhang, M. P. Bonacina, and J. Hsiang. PSATO: A distributed propositional prover and its application to quasigroup problems. *Journal of Symbolic Computation*, 21(4):543–560, 1996.