Distributed Local Resolution of Boolean Equation Systems

Christophe Joubert and Radu Mateescu
INRIA Rhône-Alpes / VASY
655, av. de l'Europe, F-38330 Montbonnot St Martin, France
E-mail: {Christophe.Joubert,Radu.Mateescu}@inrialpes.fr

Abstract

Boolean Equation Systems (BESs) allow to represent various problems encountered in the area of propositional logic programming and verification of concurrent systems. Several sequential algorithms for global and local BES resolution have been proposed so far, mainly in the field of verification; however, these algorithms do not scale up satisfactorily as the size of BESs increases. In this paper, we propose a distributed algorithm, called DSOLVE, which performs the local resolution of a BES using a set of machines connected by a network. Our experiments for solving large BESs using clusters of PCs show linear speedups and a scalable behaviour of DSOLVE w.r.t. its sequential counterpart.

1. Introduction

Boolean Equation Systems (BESs) [10] are sets of fixed point equations having boolean variables in their left-hand sides and boolean formulas (built using disjunctive and conjunctive operators) in their right-hand sides. BESs provide a useful representation for various problems encountered in different frameworks, such as *propositional logic programming* (satisfiability of Horn clauses [9]) and *verification of concurrent systems* (equivalence checking [2], model checking [1], and partial order reduction [13]). There are essentially two ways to perform the resolution of a BES: either *globally* (the values of all variables must be computed), which requires the construction of the whole BES before resolution, or *locally* (the value of one variable of interest must be computed), which allows an incremental construction of the BES during resolution. As the size of BESs increases, local resolution scales up better w.r.t. memory consumption, since only the BES portion necessary for solving the variable of interest must be stored.

During the last decade, several sequential algorithms were proposed for global [3, 5, 1] or local [1, 14, 11] BES resolution, some of them being optimized to reduce the memory consumption for BESs with particular forms [11]. All these algorithms have a linear worst-case complexity in the size of the BES (number of variables and operators). However, for solving large BESs, such as those produced from verification problems involving industrial-scale concurrent systems (which may yield BESs with hundreds of millions of variables), the memory of currently available sequential computers is no longer sufficient. A natural way for scaling up the capabilities of BES resolution is to use parallel computers, which can increase memory and CPU resources by one or two orders of magnitude.

In this paper, we propose a distributed algorithm, named DSOLVE, for the local resolution of BESs on standard loosely-coupled parallel computers, such as clusters of PCs or networks of workstations (NOWs). DSOLVE was designed in terms of *boolean graphs* (dependency graphs between boolean variables), which were also used for designing several sequential BES local resolution algorithms [1, 11]. To ensure genericity and portability, DSOLVE was implemented as an application-independent C library (which can be used as back-end for solving specific problems represented in terms of BES resolution), using standard communication mechanisms (TCP/IP sockets). An extensive set of experiments performed on very large BESs with various forms showed that DSOLVE behaves extremely well w.r.t. its sequential counterpart given in [11].

Related work. The only distributed resolution algorithm we are aware of [4] is specialized for model checking properties expressed in a fragment of the modal μ-calculus. This algorithm operates on *game graphs*, whose vertices (resp. edges) represent configurations (resp. moves) in two-player games; however, it could be rephrased in terms of boolean graphs. The experimental results reported in [4] show effective speedups for μ-calculus formulas of particular forms, but do not give sufficient insight on the behaviour of the algorithm when applied to problems other than model checking.

Outline of the paper. Section 2 defines BESs and their associated boolean graphs used to devise resolution algorithms. Section 3 presents the DSOLVE algorithm and Section 4 gives experimental results illustrating its behaviour on large BESs. Finally, Section 5 gives some concluding remarks and outlines directions of future work.

2. Boolean Equation Systems

A Boolean Equation System (BES) [10] defined over a set $\mathcal{X}$ of boolean variables is a set $\{x_i = op_i X_i\}_{1 \le i \le n}$ of fixed point equations, where $x_i \in \mathcal{X}$, $X_i \subseteq \mathcal{X}$, and $op_i \in \{\vee, \wedge\}$. The right-hand sides of equations denote pure disjunctive or conjunctive formulas defined over a subset of variables: for instance, $\vee\{x_1, x_2, x_3\}$ is an abbreviation for $x_1 \vee x_2 \vee x_3$. The boolean constant T (resp. F) is represented as $\wedge\emptyset$ (resp. $\vee\emptyset$). This class of BESs, called *simple* BESs [3], is convenient for devising efficient resolution algorithms, while still being general: any BES containing combinations of disjunctions and conjunctions in the right-hand sides of equations can be made simple (at the price of a linear increase in size) by factoring pure disjunctive and conjunctive subformulas using additional variables and equations. Given a variable assignment $\delta = [b_1/x_1, ..., b_n/x_n]$, which associates boolean values to variables, the semantics of a boolean formula $op_i\{x_{i1}, ..., x_{ik}\}$ w.r.t. δ is defined as $[\![op_i\{x_{i1}, ..., x_{ik}\}]\!]\,\delta = \delta(x_{i1})\ op_i \cdots op_i\ \delta(x_{ik})$. The semantics $[\![\{x_i = op_i X_i\}_{1 \le i \le n}]\!]$ of a BES is the least fixed point of the associated functional $\Phi : \mathbb{B}^n \to \mathbb{B}^n$ defined as $\Phi(b_1, ..., b_n) = \langle [\![op_i X_i]\!]\,[b_1/x_1, ..., b_n/x_n] \rangle_{1 \le i \le n}$ (the absence of negations in boolean formulas ensures the monotonicity of Φ and the existence of its least fixed point).

Resolution algorithms can be devised in a more intuitive way by using *boolean graphs* [1]. Given a BES $\{x_i = op_i X_i\}_{1 \le i \le n}$, its associated boolean graph $G = (V, E, L)$ is defined as follows: $V = \{x_1, ..., x_n\}$ is the set of vertices (boolean variables), $E = \{(x_i, x_j) \mid x_j \in X_i\}$ is the set of edges (dependencies between boolean variables), and $L : V \to \{\vee, \wedge\}$, $L(x_i) = op_i$ is the vertex labeling (each vertex is labeled by the boolean operator in the right-hand side of the corresponding equation). It was shown in [1, 14] that solving a BES locally (i.e., computing the value of a variable x_k) amounts at finding a subgraph (V', E', L') containing x_k and a marking $m : V' \to \mathbb{B}$ which are both *stable* (for each $y \in V'$, $m(y)$ corresponds to the BES solution) and *closed* (for each $y \in V'$, $m(y)$ does not depend on vertices outside V').

Figure 1 shows a BES, its corresponding boolean graph, and a subgraph (enclosed in the grey area) computed by a local resolution algorithm (black and white vertices denote T and F variables, respectively). Such an algorithm consists typically of a forward exploration of the boolean graph starting at a variable of interest x_k, intertwined with a backward propagation of stable variables (whose values are T), which stops either when x_k becomes stable, or the whole portion of the graph reachable from x_k has been explored. Various exploration strategies (e.g., depth-first [1, 11], breadth-first [11], chaotic [14], etc.) can be used, yielding resolution algorithms with a linear time and space worst-case complexity. However, these sequential algorithms and the resources of currently available computers do not scale up for large BESs (containing over 10^8 variables, see Section 4), and therefore distributed solutions become necessary.

$$\begin{cases} x_1 = x_2 \vee x_3 \\ x_2 = x_4 \vee x_5 \\ x_3 = x_2 \wedge x_5 \wedge x_6 \\ x_4 = x_4 \wedge x_5 \\ x_5 = \mathsf{T} \\ x_6 = x_3 \vee x_1 \end{cases}$$

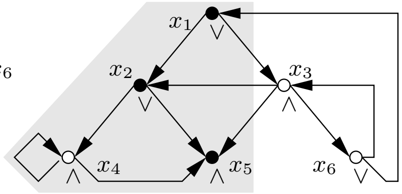


Figure 1. A BES, its boolean graph, and the result of a local resolution for x1

3. Distributed Local Resolution

Hereafter, we consider that our parallel machine is composed of P computers, numbered from 0 to $P - 1$, also called *nodes*, each having its own processor and memory.

Our data allocation is done using a static hash function $h : V \to [0, P\text{-}1]$, which distributes boolean variables equally among nodes. Also, we assume a strongly connected network topology between nodes, since each node will work and communicate symmetrically with every other node. In addition to the P processes performing the distributed BES resolution, we introduce a special *coordinator* process, which provides a user interface for BES resolution (configuration, launching, collection of statistical data, termination detection). Although its algorithm (omitted here due to space limitations) is independent from the BES resolution, the coordinator is in a privileged position, since it has a global view of the computation. In particular, this enables distributed termination detection (DTD) of BES resolution using as many messages as traditional asymmetric DTD mechanisms [12].

In BES resolution, each node is responsible for exploring a part of the boolean graph (see Figure 2). The basic tasks are the forward exploration of edges in the boolean graph (expansion) and the back-propagation of stable variables (stabilization). The tasks executed on a node are independent and asynchronous w.r.t. those executed on other nodes, and therefore can be parallelized efficiently. The use of a static hash function h ensures a good load balancing, but does not necessarily preserve data locality, i.e., does not minimize dependencies between boolean variables as-

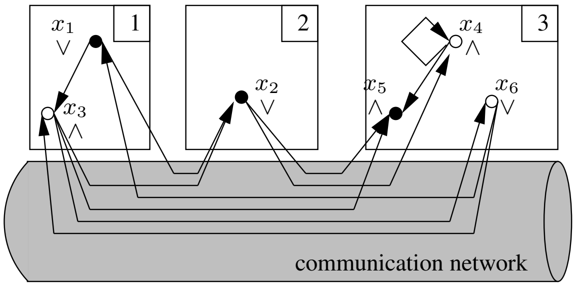


Figure 2. The result of a local distributed resolution for x1 with 3 nodes

signed to different nodes, which induce exchanges of messages (a dynamic load balancing scheme would improve locality, but at the cost of additional messages and synchronization points). To achieve a good overlapping between communications and computations, we use non-blocking asynchronous communication, which avoids synchronization points, and we statically define a fine-grained priority between both activities.

Figure 3 presents the DSOLVE distributed algorithm for local BES resolution. The main function DSOLVE takes as input the variable of interest x, the boolean graph (V, E, L) corresponding to the BES, the static hash function h, the index i of the current node ($i \in [0, P\text{-}1]$), and it returns the boolean value of x. Two auxiliary procedures EXPAND and STABILIZE take boolean variables as inputs, and update the data structures used for expansion and stabilization, respectively. After a phase of initialization of local data structures (lines 2-8), the function loops until termination. Each loop computes one of the three following tasks, whose priorities are given by the cascading if-then-else: *stabilization* of variables by back-propagation (lines 10-18), *expansion* of unstable variables (lines 19-24), and *reception* of messages denoting requests for stabilization or expansion (lines 25-33). In the sequel, we describe each of these tasks in detail.

Initialization. Several local data structures are needed to either compute or store boolean variables and the information attached to them. A counter $c(y)$ indicates the number of unstable successors of variable y that must be stabilized in order to stabilize y itself. For a variable y, $c(y)$ is initialized with its number of successors $|E(y)|$ if y is an $\wedge$-variable, or with 1 otherwise (lines 3-4). A variable y is *stable* (meaning its value is T) when $c(y)$ = 0. A set $d(y) \subseteq V$ contains the predecessor variables of y already encountered (backward dependencies). This information will be used when propagating values of stabilized variables. Three sets are used by node i for exploring the boolean graph: the *search set* $S_i \subseteq V$ stores all boolean variables x_i visited by node i

```
 1: function DSolve(x, (V, E, L), h, i) : 𝔹 is
 2:   if h(x) = i then
 3:     if L(x) = ∧ then c(x) := |E(x)|
 4:     else c(x) := 1 endif;
 5:     d(x) := ∅; W_i := {x}; S_i := {x}; B_i := ∅
 6:   else
 7:     W_i := ∅; S_i := ∅; B_i := ∅
 8:   endif;
 9:   while NotTerminated do
10:     if B_i ≠ ∅ then
11:       while B_i ≠ ∅ do
12:         x_i := choose(B_i);  B_i := B_i \ {x_i};
13:         forall w_i ∈ d(x_i) do
14:           if h(w_i) = i then Stabilize(w_i)
15:           else Send(h(w_i), Evl(w_i)) endif
16:         endfor;
17:         d(x_i) := ∅
18:       endwhile
19:     elsif W_i ≠ ∅ then
20:       x_i := choose(W_i);  W_i := W_i \ {x_i};
21:       forall y_i ∈ E(x_i) do
22:         if h(y_i) = i then Expand(x_i, y_i)
23:         else Send(h(y_i), Exp(x_i, y_i)) endif
24:       endfor
25:     else
26:       Receive(msg_i, sender_i);
27:       case msg_i is
28:         Evl(x_i) → if c(x_i) ≠ 0 then
29:                      Stabilize(x_i)
30:                    endif
31:         Exp(x_{sender_i}, y_i) →
32:                    Expand(x_{sender_i}, y_i)
33:       endcase
34:     endif
35:   endwhile;
36:   return c(x) = 0
37: end

38: procedure Expand(x_i, y_i) is
39:   if y_i ∉ S_i then
40:     S_i := S_i ∪ {y_i};  d(y_i) := ∅;
41:     if L(y_i) = ∧ then c(y_i) := |E(y_i)|
42:     else c(y_i) := 1 endif;
43:     if c(y_i) ≠ 0 then W_i := W_i ∪ {y_i} endif
44:   endif;
45:   if c(y_i) = 0 then
46:     if h(x_i) = i then Stabilize(x_i)
47:     else Send(h(x_i), Evl(x_i)) endif
48:   else
49:     d(y_i) := d(y_i) ∪ {x_i}
50:   endif
51: end

52: procedure Stabilize(w_i) is
53:   c(w_i) := c(w_i) − 1;
54:   if c(w_i) = 0 then B_i := B_i ∪ {w_i} endif
55: end
```

Figure 3. Distributed local resolution of a BES using its boolean graph

$(h(x_i) = i)$, the *working set* $W_i \subseteq S_i$ stores all the variables waiting to be expanded, and the *backward stabilization set* $B_i \subseteq S_i \setminus W_i$ stores all stable variables to be propagated along backward dependencies. The overall resolution process is started by the *initiator* node with index $i = h(x)$, which is responsible for handling the variable of interest x (lines 2-5).

Expansion. The forward exploration of the boolean graph consists in expanding new local boolean variables. A variable x_i is extracted from the working set W_i (line 20), which contains only unstable variables ($\forall x_i \in W_i \cdot c(x_i) \neq 0$). The outgoing edges of x_i and corresponding successor variables y_i are then computed (line 21). If the hash function h determines that y_i must be handled by a remote node of index $h(y_i)$ (line 23), then an expansion message $Exp(x_i, y_i)$ is sent to that node. Otherwise (line 22), local data structures are updated by calling procedure EXPAND(x_i,y_i), whose call invariant is: $c(x_i) \neq 0 \wedge x_i \in V \wedge y_i \in V \wedge h(y_i) = i$. If y_i has not been already visited, it is added to the set of visited variables S_i (lines 39-44) and its counter $c(y_i)$ is set up as stated above (lines 41-42). If y_i is a $\wedge$-sink or was previously stabilized (lines 45-47), its value (which is T) is propagated to local predecessors of y_i by calling procedure STABILIZE(x_i), and to remote predecessors by sending a stabilization message $Evl(x_i)$. Otherwise, backward dependencies $d(y_i)$ are updated with variable x_i (line 49).

Stabilization. The backward exploration of the boolean graph consists in propagating the values of stabilized variables along backward dependencies. For each stable variable $x_i \in B_i$, its value (which is T) is propagated to each predecessor variable $w_i \in d(x_i)$ (lines 11-18). If w_i is stored on a remote node, a stabilization message $Evl(w_i)$ is sent to the node of index $h(w_i)$ (line 15). Otherwise, local data structures are updated by calling procedure STABILIZE(w_i), whose call invariant is: $c(w_i) \neq 0 \wedge w_i \in S_i$. The counter of unstable successors $c(w_i)$ is decremented (line 53) and, if w_i becomes stable, it is added to the set of stable variables B_i (line 54). Finally, stabilization eliminates dependencies between boolean variables (line 17) to reduce memory consumption.

Reception. Symmetrically to the transmission of messages during stabilization and expansion (lines 15 and 23), there is an activity handling the reception of messages. Depending on the message label (line 27), the received edge is processed with procedure STABILIZE (line 29) or EXPAND (line 32).

Distributed termination detection. The variable NOTTERMINATED is set to F when distributed termination of the BES resolution is detected. Conditions of termination are: either the variable of interest x has been stabilized ($c(x) = 0$) during backward propagation (line 53), or the boolean graph has been completely explored, i.e., all local working sets of variables are empty ($\forall i \in [0..P\text{-}1] \cdot W_i = B_i = \emptyset$) and no more messages are transiting through the network. The first condition is detected by the *initiator* node, whose index is $h(x)$, when back propagating values up to x. The second condition involves a DTD. Our DTD algorithm is based on two broadcast waves of global inactivity detection, between the coordinator and the resolution processes.

Complexity. Our distributed algorithm is based on the theory of boolean graphs underlying the sequential algorithms [1, 14]. It is composed of two intertwined graph traversals (forward and backward), whose worst-case time complexity is $O(|V|+|E|)$. The same bound applies for memory complexity, because of the dependencies $d(y)$ stored during graph exploration. The message complexity is $O(|E|)$, the worst-case being obtained with two messages (expansion and stabilization) exchanged per edge. Theoretically, our termination detection algorithm has a complexity $O(|E|)$, but practically it reveals to be very efficient with only 0.01% of total exchanged messages used for termination detection.

4. Implementation and Experiments

Our implementation of DSOLVE (7 500 lines of C code) is currently being integrated into the generic BES resolution library CÆSAR_SOLVE (9 300 lines of C code) [11], available under SOLARIS, LINUX, and WINDOWS operating systems. The CÆSAR_SOLVE library is built using the generic OPEN/CÆSAR environment [7] for on-the-fly exploration of transition systems, which is itself part of the CADP verification toolbox [8]. The DSOLVE implementation uses a generic communication library (4 000 lines of C code) based on TCP/IP sockets. This communication layer allows a fine control of memory consumption, by tuning the size of communication buffers appropriately.

Platform architecture. To make an efficient use of the memory (since the workload is distributed equally among nodes by the static hash function), we assume that all nodes are homogeneous in terms of operating systems, processor and memory. Our experiments have mostly been carried out on a cluster of 17 XEON 2.4 GHz PCs, with 1.5 GB of RAM, running LINUX, and interconnected by a 1 Gigabit network. We also made a few experiments on a cluster of 216 PENTIUM III 733 MHz PCs, with 256 MB of RAM, running LINUX, essentially for scalability measures, as well as on a local network of SUN workstations for development purpose.

Random generation of BESs. In order to test our implementation of DSOLVE, we developed a random BES solver tool (400 lines of C code) that provides the successor func-

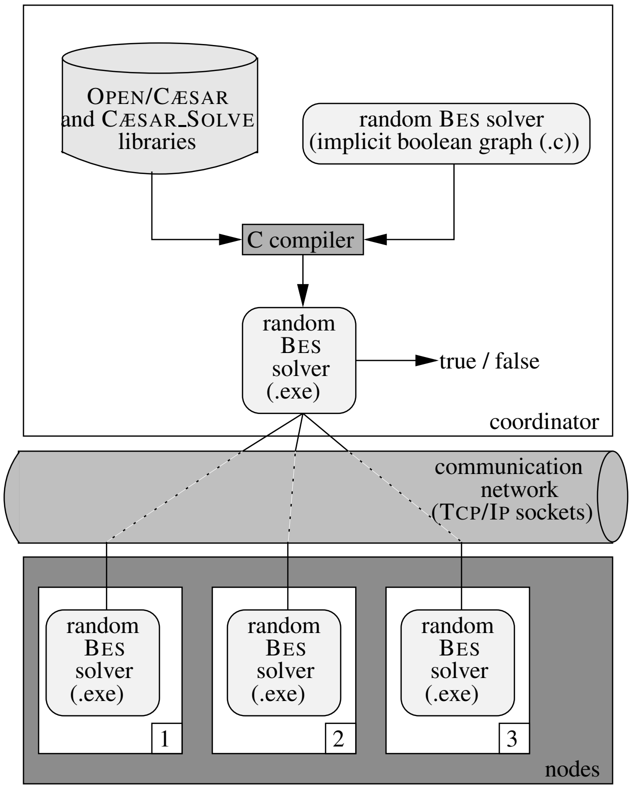


Figure 4. The random BES solver tool

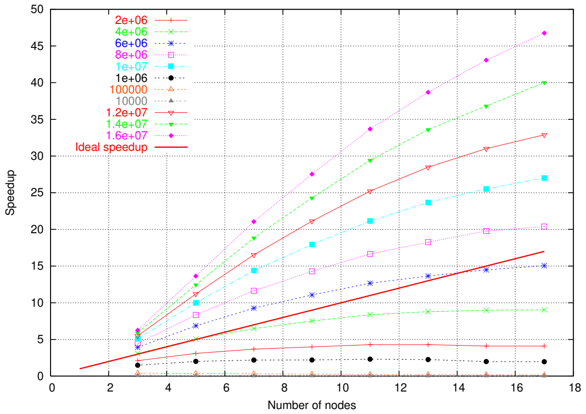
tion of a BES (edges going out of a variable in the boolean graph) characterized by a set of parameters. Figure 4 shows the programming interface and the runtime environment allowing to distribute the local resolution of a BES generated randomly. It is the role of the coordinator to duplicate and launch the executable program over all nodes. The random BES solver provides a very accurate way of measuring DSOLVE performances, since the cost of computing the successor function is negligible w.r.t. distributed BES resolution. Moreover, an appropriate tuning of the parameters allows to build a wide variety of BESs, including those encountered in specific application areas, such as verification (equivalence checking, model checking, partial order reduction) and Horn clause resolution. The parameters used for defining a class of BESs are: percentage of variable kind alternation, i.e., proportion of $\wedge$ (resp. $\vee$) variables going out of a $\vee$ (resp. $\wedge$) variable; percentage of boolean constants in the BES; minimum number of variables; and average boolean equation length (branching factor of the boolean graph). According to the above parameters, boolean variables (represented as integers) and their kinds ($\wedge$, $\vee$) are generated randomly by means of a seed also given as a parameter to the BES generation.

Speedup. Given a BES local resolution problem, if T_s is the time taken by a sequential algorithm to solve the problem, and T_P is the time taken by DSOLVE for the same problem on a parallel machine with P nodes, the speedup S_P is T_s/T_P. The sequential BES resolution algorithm we use is one of the algorithms proposed in [11], which is based on a BFS exploration strategy similar to the one used by DSOLVE.

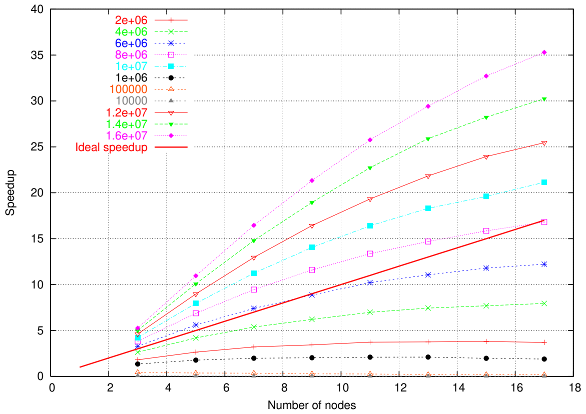
Figure 5 shows the speedups obtained for three different classes of BESs. Speedups are computed on the basis of real distributed execution time, i.e., including the initialization phase (file copies, set up of communication channels, creation of data structures). Moreover, one of the nodes is used as coordinator process in addition to its resolution task. The results shown in Figure 5 have been obtained with boolean equations whose right-hand sides contain 10 variables on average, and a BES size varying from 10^4 to $1.6 \cdot 10^7$ variables.

Figure 5(a) represents the BES class on which DSOLVE achieves the best speedup. It is characterized by 0% of variable kind alternation and 0% of boolean constants. Since no constants are present in the BES, no stabilization occurs, and the BES resolution reduces to a forward exploration of the entire boolean graph. Each curve on Figure 5(a) gives the speedup for a given problem using an increasing number of XEON nodes (from 3 to 17). It can be seen that speedup increases with BES size and becomes superlinear for $6 \cdot 10^6$ variables. This can be explained by the fact that both DSOLVE and the sequential algorithm in [11] use hash tables to store sets of boolean variables, with a search complexity $O(N/H)$, when H is the number of entries (collision lists) in the hash table, and N the number of variables present in a table. Hence, updating hash tables during DSOLVE execution takes $(|E|/P) \cdot ((|V|/P)/H) = (|E| \cdot (|V|/H))/P^2$ operations, since the set of dependencies E was divided equally in P subsets, whereas it takes $|E| \cdot (|V|/H)$ operations in the sequential algorithm. Therefore, for large problems (i.e., $|V| >> H$, e.g., $6 \cdot 10^6$ variables) and for a high number of nodes (i.e., division by a high factor P^2), the speedup becomes superlinear. Updating the hash tables of variables during sequential resolution becomes more expensive than communication overhead of distributed resolution. We believe that a similar behaviour will occur with other data structures (e.g., balanced search trees).

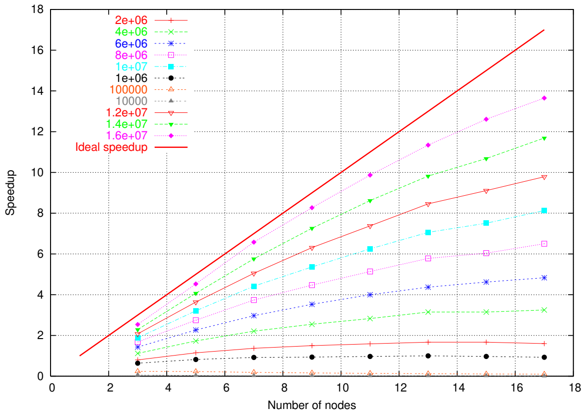
Figure 5(b) represents a class of BES with 100% of variable kind alternation (i.e., each $\vee$-variable has only $\wedge$-successors, and vice-versa), and 10% of boolean constants. This class corresponds to the verification of nondeterministic systems (equivalence checking [11] and partial order reduction [13]), and also to Horn clause resolution [9]. Speedup is slightly lower than for $5(a)$, because boolean constants induce stabilization (hence addi-



(a) 0% const., 0% alt. BES class



(b) 10% const., 100% alt. BES class



(c) 1% const., 2% alt. BES class

Figure 5. Speedup for 3 BES classes

tional messages) during BES resolution. Thus, the overall communication cost in the distributed resolution is doubled, since each expansion message will most likely induce a stabilization message. However, back propagation of stable variables is limited due to the 100% alternation parameter, which allows to propagate stabilized variables only on immediate predecessors (e.g., a $\vee$-variable stabilized to $\top$ will not necessarily stabilize its $\wedge$-predecessors). Hence, both sequential and distributed resolution have to explore most of the BES and the sequential resolution is again penalized when BESs become very large (speedup becomes superlinear for BESs exceeding $8 \cdot 10^6$ variables).

Figure 5(c) shows the BES class with lowest speedups. The graph corresponds to BESs with 2% of variable kind alternation and 1% of boolean constants. Such BESs, containing long paths of $\vee$-variables ended by $\top$ constants ($\wedge$-sinks), are often encountered in verification (equivalence checking of deterministic systems and model checking [11]). Hence, backward propagation of stable variables can be very effective and often reaches the variable of interest. The sequential algorithm implements an efficient propagation mechanism (since all information about predecessor dependencies is stored locally), whereas the cost of stabilization messages in DSOLVE (which are as numerous as expansion messages) cannot be overlapped by computation activities. This explains the low speedup obtained with DSOLVE until a BES size of $1.6 \cdot 10^7$ variables, where it becomes close to linear, due to the overhead of searching and putting variables in hash tables. We observe the same behaviour for BESs characterized by any percentage of variable kind alternation in $]0, 100[$ and any percentage of boolean constants in $]0, 100[$.

Scalability. Scalability can be expressed in terms of variation of processing speed (increasing the BES size on a fixed set of nodes), or in terms of execution time (increasing the number of nodes on a fixed BES size).

Figure 6 presents both scalabilities. Each curve represents the processing speed of boolean variables with a fixed set of XEON nodes varying from 3 to 17. Each point on a curve is an instance of resolution for a BES size between 10^4 to $2.4 \cdot 10^8$ variables (y-axis), and whose execution time varies from 0.1 to $1\,800$ seconds (x-axis). The BESs considered have 0% of boolean constants, 0% of variable kind alternation, and an average equation length of 10. Points on a same horizontal line represent a same problem instance.

The shape of the curves for large BESs is close to linear (it does not converge immediately to a horizontal asymptote) showing that the processing speed scales well with the number of nodes. Increasing the number of nodes on a fixed BES size yields superlinear gains (see the discussion on Figure 5(a)). Hence, solving a BES with 100 million variables will be more than two times faster on 13 nodes than on 7

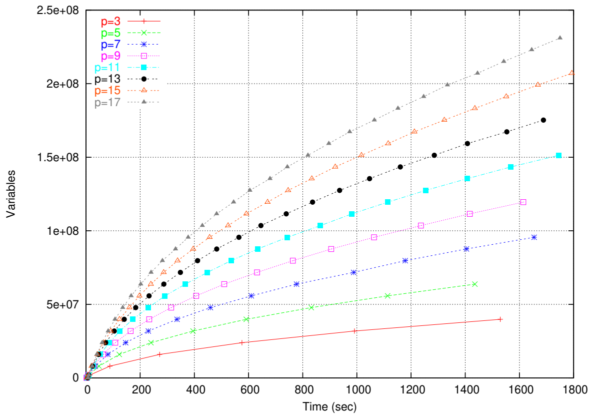


Figure 6. Scalability w.r.t. BES size and node number

nodes. For the class of BES considered, DSOLVE handles a BES with 240 million variables, 1.2 billion transitions on 17 XEON nodes in less than 28 minutes, whereas the sequential algorithm starts running out of memory around 16 million variables on a single XEON node. The same BES resolution has been made with 81 PENTIUM nodes, and it achieved similar results both in execution time and number of messages exchanged. The number of messages exchanged is about 1.1 billion, among which 100 thousand messages are for termination detection. This communication cost is due to a lack of data locality induced by the static hash function, and explains the shape of the scalability curve, which is lower than linear.

It would be interesting to study the effect of the exploration strategy on the scalability of DSOLVE, for instance by considering a combination of sequential depth-first search (performed locally by each node) and distributed breadth-first search (performed globally by all nodes). We leave this topic for future work.

Memory and communication cost. All our experiments show a perfect load balancing achieved by the hash function (sets of boolean variables are equally divided among the P nodes). The memory overhead due to distributed resolution is not significant compared to the memory allocated for data: if M_s is the sequential memory cost, then M_P, the memory cost on each of the P nodes, is approximately M_s/P. The communication cost can also be evaluated, assuming that messages are sent for each cross-dependency (edge $(x, y) \in E$ such that $h(x) \neq h(y)$). Since the hash function h shares variables equally among nodes, it also shares dependencies equally. Thus, the number of cross-dependencies can be evaluated to $((P\text{-}1)/P) \cdot |E|$, since statistically only $|E|/P$ transitions will be local. Hence, at most $((P\text{-}1)/P) \cdot |E|$ expansion messages are exchanged, and at most $((P\text{-}1)/P) \cdot |E|$ corresponding stabilization messages.

5. Conclusion and Future Work

We have presented DSOLVE, a distributed algorithm for local resolution of BESs, which runs on widely-used loosely-coupled parallel machines such as PC clusters and NOWs. An extensive set of experiments performed on large BESs with various forms (including typical ones encountered in applications) revealed linear speedups, which even become superlinear for large BESs of particular forms. Moreover, speedups are stable w.r.t. the BES size and the number of processors, therefore showing a good scalability of DSOLVE. As far as we know, this is the first attempt to devise a generic implementation of distributed local BES resolution. We are currently applying DSOLVE for verification of concurrent programs (equivalence checking, model checking, and partial order reduction), which already allowed to scale up verification capabilities to larger programs than with currently available sequential tools.

We plan to continue our work along several directions. Firstly, we will generalize the DSOLVE algorithm for the case (encountered in model checking) of BESs having several blocks of equations with acyclic inter-block dependencies; this requires to handle the interleaving of several simultaneous resolutions. Secondly, we will develop and experiment other applications of DSOLVE, e.g., resolution of Horn clauses, for which succinct translations to BESs (linear BES size w.r.t. the number of literals and operators in the Horn clause) are available [9]. Finally, DSOLVE could be extended for solving general systems of (monotonic) equations defined over complete lattices, with direct applications in the field of abstract interpretation and data flow analysis [6].

Acknowledgements

We are grateful to Nicolas Descoubes for implementing the generic communication library used by the DSOLVE algorithm.

References

[1] H. R. Andersen. Model checking and boolean graphs. *Theoretical Computer Science*, 126(1):3–30, April 1994.

[2] H. R. Andersen and B. Vergauwen. Efficient Checking of Behavioural Relations and Modal Assertions using Fixed-Point Inversion. In P. Wolper, editor, *Proceedings of the 7th International Conference on Computer Aided Verification CAV '95 (Liege, Belgium)*, volume 939 of *Lecture Notes in Computer Science*, pages 142–154. Springer Verlag, July 1995.

[3] A. Arnold and P. Crubillé. A linear algorithm to solve fixed-point equations on transition systems. *Information Processing Letters*, 29:57–66, 1988.

[4] Benedict Bollig, Martin Leucker, and Michael Weber. Local Parallel Model Checking for the Alternation Free Mu-Calculus. In D. Bonaki and S Leue, editors, *Proceedings of the 9th International SPIN Workshop on Model checking of Software (SPIN '02)*, volume 2318 of *LNCS*, pages 128–147. Springer Verlag, 2002.

[5] R. Cleaveland and B. Steffen. A Linear-Time Model-Checking Algorithm for the Alternation-Free Modal Mu-Calculus. *Formal Methods in System Design*, 2:121–147, 1993.

[6] Christian Fecht and Helmut Seidl. An Even Faster Solver for General Systems of Equations. In Radhia Cousot and David A. Schmidt, editors, *Proceedings of the 3rd International Symposium on Static Analysis SAS'96 (Aachen, Germany)*, volume 1145 of *Lecture Notes in Computer Science*, pages 189–204. Springer Verlag, September 1996.

[7] Hubert Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In Bernhard Steffen, editor, *Proceedings of the First International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'98 (Lisbon, Portugal)*, volume 1384 of *Lecture Notes in Computer Science*, pages 68–84, Berlin, March 1998. Springer Verlag. Full version available as INRIA Research Report RR-3352.

[8] Hubert Garavel, Frédéric Lang, and Radu Mateescu. An Overview of CADP 2001. *European Association for Software Science and Technology (EASST) Newsletter*, 4:13–24, August 2002. Also available as INRIA Technical Report RT-0254 (December 2001).

[9] X. Liu and S. A. Smolka. Simple Linear-Time Algorithms for Minimal Fixed Points. In Kim G. Larsen, Sven Skyum, and Glynn Winskel, editors, *Proceedings of the 25th International Colloquium on Automata, Languages, and Programming ICALP'98 (Aalborg, Denmark)*, volume 1443 of *Lecture Notes in Computer Science*, pages 53–66. Springer Verlag, July 1998.

[10] Angelika Mader. *Verification of Modal Properties Using Boolean Equation Systems*. VERSAL 8, Bertz Verlag, Berlin, 1997.

[11] Radu Mateescu. A Generic On-the-Fly Solver for Alternation-Free Boolean Equation Systems. In Hubert Garavel and John Hatcliff, editors, *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems TACAS'2003 (Warsaw, Poland)*, volume 2619 of *Lecture Notes in Computer Science*, pages 81–96. Springer Verlag, April 2003. Full version available as INRIA Research Report RR-4711.

[12] J. Matocha and T. Camp. A taxonomy of distributed termination detection algorithms. *Journal of Systems and Software*, 43:207–221, 1998.

[13] Gordon Pace, Frédéric Lang, and Radu Mateescu. Calculating τ-Confluence Compositionally. In Jr Warren A. Hunt and Fabio Somenzi, editors, *Proceedings of the 15th International Conference on Computer Aided Verification CAV'2003 (Boulder, Colorado, USA)*, volume 2725 of *Lecture Notes in Computer Science*, pages 446–459. Springer Verlag, July 2003. Full version available as INRIA Research Report RR-4918.

[14] B. Vergauwen and J. Lewi. Efficient Local Correctness Checking for Single and Alternating Boolean Equation Systems. In S. Abiteboul and E. Shamir, editors, *Proceedings of the 21st ICALP (Vienna)*, volume 820 of *Lecture Notes in Computer Science*, pages 304–315, Berlin, July 1994. Springer Verlag.