Efficient Distributed SAT and SAT-Based Distributed Bounded Model Checking

Malay K Ganai, Aarti Gupta, Zijiang Yang, and Pranav Ashar

NEC Laboratories America,
Princeton, NJ USA 08540
Fax: +1-609-951-2499
{malay,agupta,jyang,ashar}@nec-labs.com

Abstract. SAT-based Bounded Model Checking (BMC), though a robust and scalable verification approach, still is computationally intensive, requiring large memory and time. Interestingly, with the recent development of improved SAT solvers, it is frequently the memory limitation of a single server rather than time that becomes a bottleneck for doing deeper BMC search. Distributing computing requirements of BMC over a network of workstations can overcome the memory limitation of a single server, albeit at increased communication cost. In this paper, we present: a) a method for distributed-SAT over a network of workstations using a Master/Client model where each Client worsktation has an exclusive partition of the SAT problem and uses knowledge of partition topology to communicate with other Clients, b) a method for distributing SAT-based BMC using the distributed-SAT. For the sake of scalability, at no point in the BMC computation does a single workstation have all the information. We experimented on a network of heterogenous workstations interconnected with a standard Ethernet LAN. To illustrate, on an industrial design with ~13K FFs and ~0.5M gates, the non-disributed BMC on a single workstation (with 4 Gb memory) ran out of memroy after reaching a depth of 120; on the otherhand, our SAT-based distributed BMC over 5 similar workstations was able to go upto 323 steps with a communication overhead of only 30%.

1 Introduction

With increasing design complexity of digital hardware, functional verification has become the most expensive and time-consuming component of the product development cycle [1]. Verifying modern designs requires robust and scalable approaches in order to meet more-demanding time-to-market requirements. Formal verification techniques like symbolic model checking [2, 3], based on the use of Binary Decision Diagrams (BDDs) [4], offer the potential of exhaustive coverage and the ability to detect subtle bugs in comparison to traditional techniques like simulation. However, these techniques do not scale well in practice due to the state explosion problem. SAT solvers enjoy several properties that make them attractive as a complement to BDDs. Their performance is less sensitive to the problem sizes and they do not suffer from space explosion. As a result, various researchers have developed routines for performing Bounded Model Checking (BMC) using SAT [5-8]. Unlike symbolic model checking, BMC focuses on finding bugs of a bounded

D. Geist and E. Tronci (Eds.): CHARME 2003, LNCS 2860, pp. 334–347, 2003.

length, and successively increases this bound to search for longer traces. Given a design and a correctness property, it generates a Boolean formula, such that the formula is true if and only if there exists a witness/counterexample of length *k*. This formula is then checked by a backend SAT solver. Due to the many recent advances in SAT solvers [9-13], SAT-based BMC can handle much larger designs and analyze them faster than before.

The main limitation of current applications of BMC is that it can do search up to a maximum depth allowed by the physical memory on a single server. This limitation comes from the fact that as the search bound *k* becomes larger, the memory requirement due to unrolling of the design also increases. Especially for the memory-bound designs, a single server with a limited memory has now become the bottleneck to doing deeper search.

1.1 Motivation

Distributing computing requirements of BMC (memory and time) over a network of workstations can, however, overcome the memory limitation of a single server. In this paper, we explore this possibility, and discuss our approaches in a greater detail that made it feasible. Before we delve into that, we would like to give an intuition behind the feasibile solution.

A BMC problem (described in Section 2) originating from an unrolling of the sequential circuit in different time frames provides a natural disjoint partitioning of the problem and thereby, allows the computing resources to be configured in a linear topology. The topology using one Master and several Clients is shown in Figure 1.

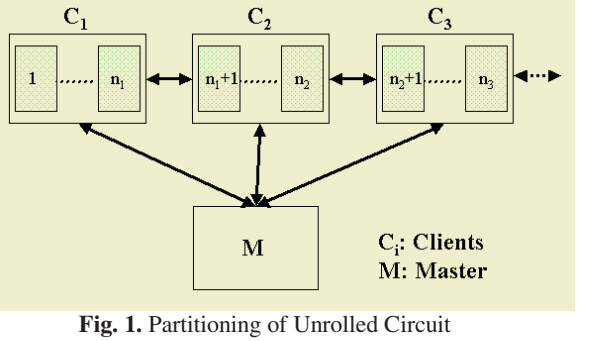

Fig. 1. Partitioning of Unrolled Circuit

Each Client C_i hosts a part of the unroll circuit i.e., from n_i+1 to n_{i+1} where n_i represents the partition depth. Each C_i (except for the terminals) is connected to C_{i+1} and C_{i-1}. The Master is connected to each of the Clients. Using the linear topology, we can distribute parts of the unroll circuit dynamically over additional Clients as and when memory resources on current Clients get close to exhaustion.

To check the satisfiability of a Boolean problem originating from BMC wherein the unrolled circuit is distributed over several servers, we must identify the part of the SAT algorithm that may be delegated to each processor without requiring any processor to have the entire problem data. Since Boolean Constraint Propagation (BCP) on clauses can be done independently on an exclusive partition, it can be delegated to each processor. Moreover, since about 80% of SAT time involves BCP,

one could achieve some level of parallelism by doing distributed-BCP. Note that any approach similar to SAT-based BMC can use similar concept to exploit parallelism.

With this motivation we now briefly describe the organization of the rest of the paper. With a brief discussion on prior related work in Section 1.2, we give a short background in Section 2, our contributions in Section 3-7, experiments in Section 8, and conclusions in Section 9.

1.2 Related Work

Parallelizing SAT solvers have been proposed by many researchers [14-19]. Most of them target performance improvement of the SAT solver. These algorithms are based on partitioning the search space on different processors using partial assignments on the variables. Each processor works on the assigned space and communicates with other processors only after it is done searching its portion of the search space. Such algorithms are not scalable memory-wise due to high data redundancy as each processor keeps the entire problem data (all clauses and variables).

In a closely related work on parallelizing SAT [16], the authors partition the problem by distributing the clauses evenly on many application specific processors. They use fine grain parallelism in the SAT algorithm to get better load balancing and reduce communication costs. Though they have targeted the scalability issue by partitioning the clauses disjointedly, the variables appearing in the clauses are not disjoint. Therefore, whenever a Client finishes BCP on its set of clauses, it must broadcast the newly implied variables to all the other processors. The authors observed that over 90% of messages are broadcast messages. Broadcasting implications can become a serious communication bottleneck when the problem contains millions of variables.

Reducing the space requirement in model checking has been suggested in several works [20-22]. These studies suggest partitioning the problem in several ways. The work in [20] shows how to parallelize the model checker based on explicit state enumeration. They achieve it by partitioning the state table for reached states into several processing nodes. The work in [21] discusses techniques to parallelize the BDD-based reachability analysis. The state space on which reachability is performed is partitioned into disjoint slices, where each slice is owned by one process. The process performs a reachability algorithm on its own slice. In [22], a single computer is used to handle one task at a time, while the other tasks are kept in external memory. In another paper [23], the author suggested a possibility of distributing SAT-based BMC but has not explored the feasibility of such an approach.

2 Background

State-of-the-Art SAT Solver

The Boolean Satisfiability (SAT) problem consists of determining a satisfying assignment for a Boolean formula on the constituent Boolean variables or proving that no such assignment exists. The problem is known to be NP-complete. Most SAT solvers [9-13] employ DPLL style [24] algorithm as shown in Figure 2 with three main engines: *decision, deduction*, and *diagnosis*. A Boolean problem can be

expressed either in CNF form or logical gate form or both. A hybrid SAT solver as in [12], where the problem is represented as both logical gates and a CNF expression, is well suited for BMC.

```
SAT_Solve(P=1) { // Check if constraint P=1 satisfiable?
  while(Decide()=SUCCESS) //Selects a new variable
    while(Deduce()=CONFLICT)//BCP till conflict/no-conflict
      if   (Diagnose()=FAILURE)   //Add   conflict   learnt
clause(s)
        return UNSAT;//Conflict found at decision level 0
  return SAT;} //No more decision to make
```

Fig. 2. DPLL style SAT Solver

Bounded Model Checking

In BMC, the specification is expressed in LTL (Linear Temporal Logic). Given a Kripke structure M, an LTL formula f, and a bound k, the translation task in BMC is to construct a propositional formula $[M, f]_k$, such that the formula is satisfiable if and only if there exists a witness of length k [25]. The satisfiability check is performed by a backend SAT solver. Verification typically proceeds by looking for witnesses or counter-examples (CE) of increasing length until *completeness threshold* [25, 26]. The overall algorithm of a SAT-based BMC for checking (or falsifying) a simple safety property is shown in the Figure 3. The SAT problems generated by the BMC translation procedure grow bigger as k increases. Therefore, the practical efficiency of the backend SAT solver becomes critical in enabling deeper searches to be performed.

```
BMC(k,P){//Falsify safety property P within bound k
  for (int i=0; i<=k ; i++) {
    P_i=Unroll(P,i);//Get property node at i^th unrolled frame
    if (SAT_Solve(P_i=0)=SAT) return CE;//Try to falsify
  }
  return NO_CE; } //No counter-example found
```

Fig. 3. SAT-based BMC for Safety Property P

3 Our Contributions

Overview of Distributed-SAT

Given an exclusive partitioning of the SAT problem, we give an overview of the fine grain parallelization of the three engines of the SAT algorithm (as described in Section 2) on a Master/Client distributed memory environment. The Master controls the execution of distributed-SAT. The decision engine is distributed in such a way that each Client selects a good local variable and the Master then chooses the globally best variable to branch on. During the deduction phase, each Client does BCP on its exclusive local partitions, and the Master does BCP on the global learned conflict clauses. Diagnosis is performed by the Master, and each Client performs a local

backtrack when request by the Master. The Master does not keep all problem clauses and variables; however, the Master maintains the global assignment stack and the global state for diagnosis. This requires much less memory than the entire problem data. To ensure proper execution of the parallel algorithm, each Client is required to be synchronized. We give details of the parallelization and different communication messages in Section 5-9.

Novelties of Our Approach

In this paper, we present a method for distributing SAT over a network of workstations using a Master/Client model where each Client worsktation has an exclusive partition of the SAT problem. Though this work is closely related to [16], there are some important differences: a) In [16], though each Client has disjoint set of clauses, variables are not disjoint. So, Clients after completing BCP, broadcast their new implications to all other Clients. After decoding the message, each receiving Client either reads the message or ignores it. In a communication network where BCP messages dominate, broadcasting implications can be an overkill when the number of variables runs into millions. In our improved distributed BCP, however, each Client has the knowledge of the SAT-problem partition topology and uses that to communicate with other Clients. This ensures that the receiving Client has to never read a message that is not meant for it. b) The algorithm in [16] is developed primarily for application specific processors, while our algorithm uses easily available existing networks of workstations. We have described several innovative optimization schemes to reduce the effect of communication overhead on performance in general-purpose networks by identifying and executing tasks in parallel while messages are in transit.

In this paper, we also extend the SAT-based BMC (as a part of our formal verification platform called DiVer) using topology-cognizant distributed-SAT to obtain a SAT-based distributed BMC over a distributed-memory environment. For the sake of scalability, our method makes sure that at no point in the BMC computation does a single workstation have all the information. We developed our distributed algorithms for a network of processors based on standard Ethernet and using the TCP/IP protocol. We can also potentially use dedicated communication infrastructures that may yield better performance, but for this work, we wanted to use an environment that is easily available, and whose performance can be considered a lower bound. We used a socket interface message passing library to provide standard bidirectional communications primitives.

4 Topology-Cognizant Distributed-BCP

BCP is an integral part of any SAT solver. We distribute BCP on multiple processes that are cognizant of topology of the SAT-problem partition running on a network of workstations. In [16], during the distributed-SAT solve each Client broadcasts its implications to all other processors. After decoding the message, each receiving process either reads the message or ignores it. We improve this approach in the following way. Each process is made cognizant of the disjoint partitioning. The process then sends out implications to only those processes that share the partitioning

interface variables with it. Each receiving process simply decodes and reads the message. This helps in two ways: a) the receiving buffer of the process is not filled with useless information; b) receiving process does not spend time in decoding useless information. This ensures that the receiving process has to never read a message that is not meant for it.

We use a distributed model with one Master and several Client processors. The Master's task is to distribute BCP on each Client that owns an exclusive partition of the problem. A bi-directional FIFO (First-in First-out) communication channel exists *only* between the process and its known neighbor, i.e., each process is cognizant of its neighbors. The process uses the partition topology knowledge for communication so as to reduce the traffic of the receiving buffer. A FIFO communication channel ensures that the channel is in-order, i.e., the messages sent from one process to another will be received in the order sent. Besides distributing BCP, the Master also records implications from the Clients as each Client completes its task.

The main challenging task for the Master is to maintain causal-effect ("happens before") ordering of implications in distributed-BCP since we cannot assume channel speeds and relative times of message arrivals during parallel BCP. *Maintaining such ordering is important because it is required for correct diagnosis during conflict analysis phase of SAT*. In the following we discuss the problem in detail and techniques to overcome it.

Consider the Master/Client model as shown in Figure 1. Client C_i can communicate with C_{i-1} and C_{i+1} besides the Master M. The Master and Clients can generate implication requests to other Clients; however, Clients can send replies to the Master *only* for the request made to it. Along with the reply message, Client also sends the message *ids* of the requests, if any, it made to the other Clients. This is an optimization step to reduce the number of redundant messages. To minimize reply wait time, the Master is allowed to send requests to the Clients even when there are implications pending from the Client provided that the global state (maintained by the Master) is not in conflict.

Let p->q denote an implication request from p to q and p<-q denote implication replies from q to p. Note that though the channel between C_i and the Master is in-order, what happens at the Event E3 cannot be guaranteed in the following.

E1: M->C1
E2: C1->C2
E3: M<-C2 or M<-C1

If M<-C2 "happens before" M<-C1, then we consider it an *out-of-order* reply since the implications due to M<-C2 depend on C1->C2, which in turn depend on M->C1. Moreover, any out-of-order reply from a Client makes subsequent replies from that Client out-of-order until the out-of-order reply gets processed.

We propose a simple solution to handle out-of-order replies to the Master. For each Client, the Master maintains a *FIFO* queue where the out-of-order replies are queued. Since the channel between a Client and Master is in-order, this model ensures that messages in the *FIFO* will not be processed until the front of the *FIFO* is processed. We illustrate this with a short event sequence. For simplicity we show the contents for *FIFO* for the Client C2.

E1: M->C1	FIFO(C2): -
E2: C1->C2	FIFO(C2): -

E3: M->C2		FIFO(C2): -
E4: M<-C2	(in response to E2)	FIFO(C2): E4
E5: M<-C2	(in response to E3)	FIFO(C2): E4,E5
E6: M<-C1	(in response to E1)	FIFO(C2): - (E4 is processed before E5)

Note that in the reply event E6, the Client C1 also notifies the Master of the event E2. Master queues E4 reply as an out-of-order reply as it is not aware of the responsible event E2 until E6 happens. E5 reply is also queued as out-of-order as earlier out-of-order reply E4 has not been processed yet. When E6 occurs, the Master processes the messages from the events E6, E4 and E5 (in the order). This maintains the ordering of the implications in the global assignment stack.

5 Distributed-SAT

We use fine grain parallelism in our distributed-SAT algorithm similar to the one proposed in [16]. However, we use the topology-cognizant distributed-BCP (as described in the previous section) to carry out distributed-SAT over network of workstations. First, we describe the task partitioning between the Master and Clients as shown in the Figure 4.

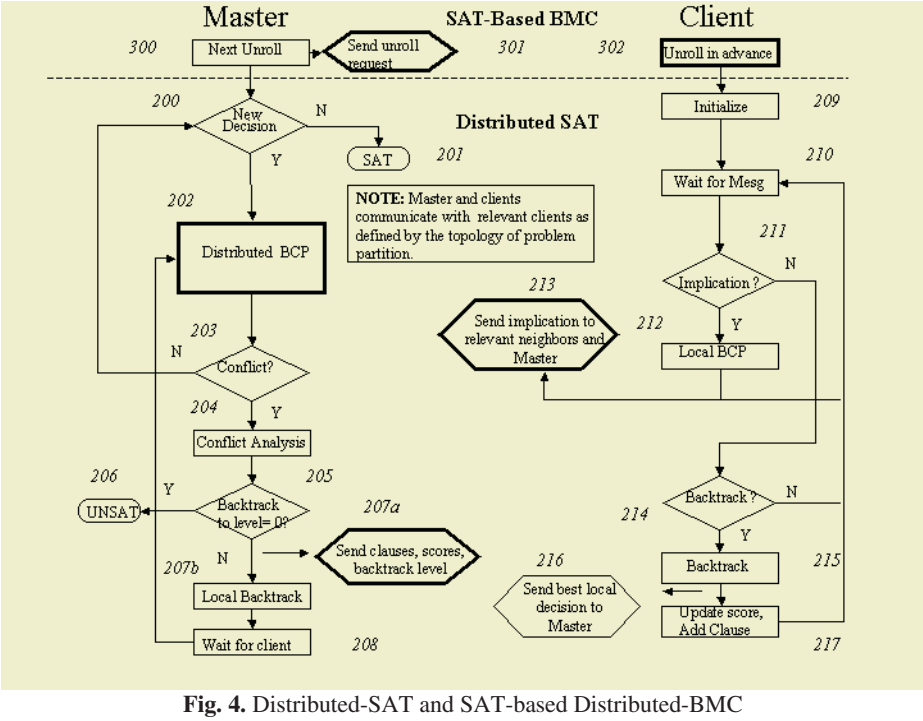

Fig. 4. Distributed-SAT and SAT-based Distributed-BMC

Tasks of the Master

- Maintains list of constraints, global assignment stack, learnt clauses, antecedents
- Selects a new decision variable from the best local decision sent by each Client
- Global conflict analysis using the assignments and antecedents
- Local BCP on clauses; manages distributed-BCP
- Receives from Ci: New implications with antecedents and best local decision
- Sends to Ci: Implication on variables local to Ci variables, backtrack request, learnt local clauses, update score request

Tasks of a Client C_i

- Maintains the ordered list of variables, scores, local assignment stack, local learnt clauses
- Keeps the exclusive partition of the problem and topological information
- Executes on request: Backtrack, decay score, update variable score, local BCP
- Receives from Master: Implications, backtrack request, update score, clause
- Receives from neighbor Cj : Implications on interface
- Sends to Master: New Implications with antecedents and best local decision, best local decision when requested, conflict node when local conflict occurs during BCP, request id when implication request comes from other Clients
- Sends to neighbor Cj: New implication requests on interface

6 SAT-Based Distributed-BMC

A SAT-based BMC problem originating from an unrolling of the sequential circuit over different time frames has a natural linear partition and thereby allows configuring the computing resources in a linear topology. The topology using one Master and several Clients is shown in Figure 1. Each Client C_i is connected to C_{i+1} and C_{i-1}. The Master *controls the execution of the SAT-based distributed BMC algorithm*. The BMC algorithm in Figure 3 remains the same except for the following changes. The *Unroll* procedure is now replaced by a *distributed* unrolling in which the procedure *Unroll* is actually invoked on the Client that hosts the partition for the depth *i.* Note that depending on the memory availability, the host Client is decided dynamically. After the unrolling, the distributed-SAT algorithm is invoked (in place of *SAT_Solve)* to check the satisfiability of the problem on the unrolled circuit that has been partitioned over several workstations. Following are the tasks distribution of the Master and Clients.

Tasks of the Master

- Allocates an exclusive problem partition to each host Client (box 300 in Figure 4)
- Requests an unrolling to the terminal Client (box 301 in Figure 4)
- Controls distributed-SAT as described in Section 5

Tasks of a Client

- Handle current unroll request and also advance by one (box 302 in Figure 4)
- Initiate a new Client as defined by the topology when new unroll size is too large
- Participate in distributed-SAT

7 Optimizations

Memory Optimizations in Distributed-SAT
The bookkeeping information kept by the Master grows with the unroll depth. The scalability of our distributed-BMC is determined by how low is the ratio of the memory utilized by the Master to the total memory used by the Clients. Following steps are taken to lower the *scalability ratio*:

- By delegating the task of choosing the local decision and maintaining the ordered list of variables to the Client, we save the memory otherwise used by the Master.
- Master does not keep the entire circuit information anytime. It relies on the Clients to send the reasons of implications that will be used during diagnosis.

In our experiments, we observed that the scalability ratio for large designs is close to 0.1, which implies that we can do a 10 times deeper search using a distributed-BMC as compared to a non-distributed (monolithic) BMC over network of similar machines (In our observation, the global learnt clauses maintained by Master is not exponentially large).

Tight Estimation of Communication Overhead
Inter-workstation communication time can be significant and adversely affects the performance. We can mitigate this overhead by hiding execution of certain tasks behind the communication latency. To have some idea of communication overhead, we first need some strategy to measure the communication overhead and actual processing time. This is non-trivial due to asynchronous clock domain of the workstations. In the following, we first discuss a novel strategy to make tight estimation of the wait time incurred by the Master due to inter-workstations communication in Parallel BMC.

Consider a *request-reply* communication . Time stamps are local to the Master and Client. At time T_s, the Master sends its request to the Client. The Client receives the message at its time t_r. The Client processes the message and sends the reply to the Master at time t_s. The Master, in the meantime, does some other tasks and then starts waiting for the message at time T_w. The Master receives the message at time T_r. *Without accounting for the Client processing time*, wait time would be simply,

$$\text{Wait\_Time} = T_r - T_w \text{ if } T_r > T_w \text{ (= 0 otherwise)}$$

This calculated wait time would be an over-estimation of the actual wait time. To account for the Client processing time, we propose the following steps:

- Master sends the request with T_s embedded in the message.
- Client replies back to the Master with the time stamp $(T_s+(t_s- t_r))$.
- The Master, depending on the time T_w , calculates the actual wait time as follows:
  - Case Tw1: $T_w < (T_s+(t_s- t_r))$ Wait _Time = $Tr-(T_s+(t_s- t_r))$
  - Case Tw2: $(T_s+(t_s- t_r)) < T_w < T_r$ Wait _Time = $T_r - T_w$
  - Case Tw3: $T_r < T_w$ Wait _Time = 0

Performance Optimizations in Distributed-SAT
Now we discuss several performance optimizations in the distributed-SAT algorithm.

- A large number of communication messages tend to degrade the overall performance. We took several means to reduce the overhead:

  - The Master waits for all Clients to stabilize before sending a new implication request. This reduces the number of implication messages sent.
  - Clients send their best local decision along with every implication and backtrack replies. At the time of decision, the Master, then, only selects from the best local decisions. It is not required to make explicit requests for a decision variable to each Client separately.
  - For all implication requests, Clients send replies to only the Master. This reduces the number of redundant messages on the network.
- Client sends active variables to the Master before doing the initialization. While the Master waits and/or processes the message, the Client does its initialization in parallel.
- When Master requests each Client to backtrack, it has to wait for the Clients to respond with a new decision variable. The following overlapping tasks are done to mitigate the wait time:
  - Local backtrack (box 207b in Figure 4) by the Master is done after the remote request is sent (box 207b in Figure 4). While the Master waits for the decision variable from the Client, the Master also sends the learnt local conflict clauses to the respective Client.
  - The function for adjusting variable score (box 217 in Figure 4) is invoked in the Client after it sends the next decision variable (during backtrack request from the Master) (box 216 in Figure 4). Since *message-send* is non-blocking, potentially the function is executed in parallel with *send.* On the downside, the decision variable that is chosen may be a stale decision variable. However, note that the local decision variable that is sent is very unlikely be chosen as decision variable. The reason is that in the next step after backtrack there will be an implication. Since the Client sends the decision variable after every implication request, the staleness of the decision variable will be eventually eliminated.

Performance Optimization in SAT-Based Distributed-BMC

- The design is read and initialization is done in all the Clients to begin with. This reduces the processing time when the unrolling is initiated onto a new Client.
- Advance unrolling is done in the Client while the Client is waiting for implication request from the Master. This includes invoking a new partition in a new Client.

8 Experiments

We conducted our evaluation of distributed -SAT and SAT-based distributed BMC on a network of workstations, each composed of dual Intel 2.8GHz Xeon Processor with 4Gb physical memory running Red Hat Linux 7.2, interconnected with a standard 10Mbps/100Mbps/1Gbps Ethernet LAN. We compare the performance and scalability of our distributed algorithm with a non-distributed (monolithic) approach. We also measure the communication overhead using the accurate strategy as described in Section 7.

We performed our first set of experiments to measure the performance penalty and communication overhead for the distributed algorithms. We employed our SAT-based distributed algorithm on 15 large industrial examples, each with a safety property. For these designs, the number of flip-flops ranges from ~1K to ~13K and number of 2-

input gates ranges from ~20K to ~0.5M. Out of 15 examples, 6 have counter examples and the rest do not have counterexample within the bound chosen. We used a Master (referred to as M) and 2 Clients (referred as C1 and C2) model where C1 and C2 can communicate with each other. We used a controlled environment for the experiment under which, at each SAT check in the distributed-BMC, the SAT algorithm executes the tasks in a distributed manner as described earlier except at the time of decision variable selection and backtracking, when it is forced to follow the sequence that is consistent with the sequential SAT. We also used 3 different settings of the Ethernet switch to show how the network bandwidth affects the communication overheads. We present the results of the controlled experiments in Table 1[a-b].

In Table 1a, the 1st Column shows the set of designs (D1-D6 have a counterexample), the 2nd Column shows the number of Flip Flops and 2-input Gates in the fanicone of the safety property in the corresponding design, the 3rd Column shows the bound depth limit for analysis, the 4th Column shows the total memory used by the non-distributed BMC, the 5th Column shows the partition depth when Client C2 took an exclusive charge of the further unrolling, Columns 6-8 show the memory distribution among the Master and the Clients. In the Column 9, we calculate the scalability ratio, *i.e.,* the ratio of memory used by the Master to that of the total memory used by Clients. *We observe that for larger designs, the scalability factor is close to 0.1* though for comparatively smaller designs, this ratio was as high as 0.8. This can be attributed to the minimum bookkeeping overhead of the Master. Note that even though some of the designs have same number of flip-flops and gates, they have different safety properties. The partition depth chosen was used to balance the memory utilization; however, the distributed-BMC algorithm chooses the partition depth dynamically to reduce the peak requirement on any one Client processor.

Table 1 [a-b]. Memory & Performance evaluation of the distributed SAT-based BMC

(a) Memory Utilization									(b) Performance Evaluation						
EX	FF (K)/ Gate (K)	D	M Mem (Mb)	Part D	P Mem (Mb)			S ratio	MT (sec)	PT (sec)	MWT (sec)			Perf Pntly	Com Ovr
					M	C1	C2				1gbs	0.1gbs	10mbs		
D1	4.2/30	16	20	5	8	5	16	0.4	8.9	12.8	11.4	34.5	991.2	1.4	0.9
D2	4.2/30	14	18	5	8	6	13	0.4	4.2	6.7	10.5	24.2	698.6	1.6	1.6
D3	4.2/30	17	21	5	9	5	17	0.4	9.7	15.6	11.2	33.2	767.9	1.6	0.7
D4	4.2/30	9	10	5	3	4	6	0.3	0.8	1.9	1.8	3.8	107.7	2.4	0.9
D5	4.2/30	15	18	5	8	5	15	0.4	5.2	8.2	10	31.4	680.5	1.6	1.2
D6	4.2/30	7	8	5	2	4	4	0.3	0.3	1.1	0.6	1.6	45.1	3.7	0.5
D7	4.2/30	21	24	5	7	4	20	0.3	9.5	14.7	9	40	855.3	1.5	0.6
D8	1.0/18	55	68	30	20	35	31	0.3	37.9	52.1	22.1	109	1895.3	1.4	0.4
D9	0.9/18	67	124	30	65	33	49	0.8	314.6	454.5	130	702.4	12922.9	1.4	0.3
D10	5.2/37	21	29	5	10	4	24	0.4	23.4	38.4	17.8	71.8	764.1	1.6	0.5
D11	12.7/448	61	1538	45	172	1071	480	0.1	919	1261.4	1135.7	2403	5893.2	1.4	0.9
D12	3.7/158	81	507	40	47	246	267	0.1	130.5	89.1	0.1	65.1	63.2	0.7	0.0
D13	3.7/158	41	254	20	24	119	141	0.1	33.7	23.2	0.4	6.3	16.1	0.7	0.0
D14	3.7/158	81	901	40	149	457	447	0.2	452.8	360.6	87.4	653.5	1288.6	0.8	0.2
D15	3.7/158	81	901	40	135	457	443	0.2	442.2	344.6	97.2	679.9	1138.5	0.8	0.3

Table 2. Comparison of monolithic and distributed BMC on Industrial designs

Ex	Mono Depth	Mono Time (sec)	Para Depth	Para Time (sec)	Para Memory (in Mb)					MWT (sec)	Comm Ovrhd	S Ratio
					M	C1	C2	C3	C4			
D11	120	1642.3	323	6778.5	634	1505	1740	1740	1730	1865.1	0.3	0.1
D12	553	4928.3	1603	13063.4	654	1846	1863	1863	1863	5947.7	0.5	0.1
D13	553	4899.5	1603	12964.5	654	1846	1864	1864	1864	5876.8	0.5	0.1
D14	567	642.8	1603	2506.2	654	1833	1851	1851	1851	1585.4	0.6	0.1
D15	567	641.9	1603	1971.5	654	1833	1851	1851	1851	879.6	0.4	0.1

In Table 1b, the 1st Column shows the cumulative time taken (over all steps) by non-distributed BMC, the 2nd Column shows the cumulative time taken (start to finish of Master over all steps) by our distributed-BMC excluding the message wait time, Columns 3-5 show the total message wait time for the Master in a 10/100/1000Mbps Ethernet Switch setting. In the Column 6, we calculate the performance penalty by taking the ratio of the time taken by distributed to that of non-distributed BMC (=Para Time/ Mono Time). In the Column 7, we calculate the communication overhead for the 1Gbps switch setting by taking the ratio of the message waiting time to distributed BMC time (=wait time for 1 Gbps/ Para Time). On average we find that the performance penalty is 50% and communication overhead is 70% with overall degradation by a factor of 2.55 (=1.5 * 1.7).

In some cases, D12-D15, however, we find an improvement in performance over non-distributed BMC. This is due to the exploitation of parallelism during the Client initialization step as described in Section 7. Note that the message wait time adversely gets affected with lowering the switch setting from 1Gbps to 10Mbps. This is attributed to the fact that Ethernet LAN is inherently a broadcast non-preemptive communication channel.

In our second set of experiments, we used the 5 largest (of 15) designs D11-D15 that did not have a witness. For distributed-BMC, we configured 5 workstations into one Master and 4 Clients C1-C4; each connected with the 1Gbps Ethernet LAN. In this setting, Clients are connected in a linear topology and the Master is connected in a star with others. In this experiment, we show the ability of the distributed-BMC to do deeper search using distributed memory. For the design D11, we used a partition of 81 unroll depths on each Client and for designs D12-15, we used partition of 401 unroll depths on each Client. The results are shown in the Table 2.

In Table 2, the 1st Column shows the set of large designs that were hard to verify, the 2nd Column shows the farthest depth to which non-distributed BMC could search before it runs out of memory, the 3rd Column shows the time taken to reach the depth in the 2nd Column, the 4th Column shows the unroll depth reached by distributed-BMC using the allocated partition, the 5th Column shows the time taken to reach the depth in the 4th Column excluding the message wait time, Columns 6-10 show the memory distribution for the Master and Clients, the 11th Column shows the total message wait time. In the Column 12, we calculate the communication overhead by taking the ratio of message wait time to the distributed-BMC time (=MWT time/ Para Time). In the Column 13, we calculate the scalability ratio by taking the ratio of memory used by the Master to that of the total memory used by the Clients.

We use the design D11 with ~13K flip-flops and ~0.5Million gates to show the performance comparison. For the design D11 we could analyze up to a depth of 323 with only 30% communication overhead, while using a non-distributed version we

could analyze only up to 120 time frames under the per-workstation memory limit. Low scalability factor, *i.e.,* 0.1 for large designs indicates that for these designs our distributed-BMC algorithm could have gone 10 times deeper compared to the non-distributed version for similar set of machines. We also observe that the communication overhead for these designs was about 45% on average, *a small penalty to pay for deeper search.*

9 Conclusions

For verifying designs with high complexity, we need a scalable and robust solution. SAT-based BMC is quite popular because of its robustness and better debugging capability. Although, SAT-based BMC is able to handle increasingly larger designs than before as a result of advancement of SAT solvers, the memory of a single server has become a serious limitation to carrying out deeper search. Existing parallel algorithms either focus on improving the SAT performance or are used in either explicit state-based model checkers or in unbounded implicit state-based model checkers. To the best of our knowledge ours is the first detailed study on providing a feasible solution for SAT-based distributed-BMC using an improved distributed SAT algorithm.

Our distributed algorithm uses the normally available large pool of workstations that are inter-connected by standard Ethernet LAN. For the sake of scalability, our distributed algorithm makes sure that no single processor has the entire data. Also, each process is cognizant of the partition topology and uses the knowledge to communicate with the other process; thereby, reducing the process's receiving buffer with unwanted information. We have also proposed several memory and performance optimization schemes to achieve scalability and decrease the communication overhead.

In the future, we would like to evaluate our distributed-SAT and SAT-based distributed-BMC on a clustered system for high performance computing that has low latency and high bandwidth communication [27].

Acknowledgements. We thank Guoqiang Pan for implementing the socket-based message-passing library.

References

[1] A. Silburt, A. Evans, G. Vrckovik, M. Diufrensne, and T. Brown, "Functional Verification of ASICs in Silicon Intensive Systems," presented at DesignCon98 On-Chip System Design Conference, 1998.
[2] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*: MIT Press, 1999.
[3] K. L. McMillan, *Symbolic Model Checking: An Approach to the State Explosion Problem*: Kluwer Academic Publishers, 1993.
[4] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. C–35(8), pp. 677–691, 1986.
[5] A. Biere, A. Cimatti, E. M. Clarke, M. Fujita, and Y. Zhu, "Symbolic model checking using SAT procedures instead of BDDs," in *Proceedings of the Design Automation Conference*, 1999, pp. 317–320.

[6] P. Bjesse and K. Claessen, "SAT-based verification without state space traversal," in *Proceedings of Conference on Formal Methods in Computer-Aided Design*, 2000.
[7] M. Ganai and A. Aziz, "Improved SAT-based Bounded Reachability Analysis," in *Proceedings of VLSI Design Conference*, 2002.
[8] P. A. Abdulla, P. Bjesse, and N. Een, "Symbolic Reachability Analysis based on {SAT}-Solvers," in *Proceedings of Workshop on Tools and Algorithms for the Analysis and Construction of Systems (TACAS)*, 2000.
[9] J. P. Marques-Silva and K. A. Sakallah, "GRASP: A Search Algorithm for Propositional Satisfiability," *IEEE Transactions on Computers*, vol. 48, pp. 506–521, 1999.
[10] H. Zhang, "SATO: An efficient propositional prover," in *Proceedings of International Conference on Automated Deduction*, vol. 1249, *LNAI*, 1997, pp. 272–275.
[11] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an Efficient SAT Solver," in *Proceedings of Design Automation Conference*, 2001.
[12] M. Ganai, L. Zhang, P. Ashar, and A. Gupta, "Combining Strengths of Circuit-based and CNF-based Algorithms for a High Performance SAT Solver," in *Proceedings of the Design Automation Conference*, 2002.
[13] A. Kuehlmann, M. Ganai, and V. Paruthi, "Circuit-based Boolean Reasoning," in *Proceedings of Design Automation Conference*, 2001.
[14] B. W. Wah, G.-J. Li, and C. F. Yu, "Multiprocessing of Combinational Search Problems," *IEEE computer*, pp. 93–108, 1985.
[15] H. Zhang, M. P. Bonacina, and J. Hsiang, "PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems," *Journal of Symbolic Computation*, 1996.
[16] Y. Zhao, "Accelerating Boolean Satisfiability through Application Specific Processing.," Ph.D. Thesis. Princeton, 2001.
[17] C. Powley, C. Fergusion, and R. Korf, "Parallel Heuristic Search: Two Approaches," in *Parallel Algorithms for Machine Intelligence and Vision*, V. Kumar, P. S. Gopalakrishnan, and L. N. Kanal, Eds. New York: Springer-Verlag, 1990.
[18] B. Jurkowiak, C. M. Li, and G. Utard, "Parallelizing Satz Using Dynamic Workload Balancing," presented at Workshop on Theory and Applications of Satisfiability Testing, 2001.
[19] M. Boehm and E. Speckenmeyer, "A Fast Parallel SAT-solver – Efficient Workload Balancing," presented at Third International Symposium on Artificial Intelligence and Mathematics, Fort Lauderdale, Florida, 1994.
[20] U. Stern and D. L. Dill, "Parallelizing the Murphi Verifier," presented at Computer Aided Verification, 1997.
[21] T. Heyman, D. Geist, O. Grumberg, and A. Schuster, "Achieving Scalability in Parallel Reachability Analysis of Very Large Circuits," presented at Computer-Aided Verification, 2000.
[22] A. Narayan, A. Isles, J. Jain, R. Brayton, and A. L. Sangiovanni-Vincentelli, "Reachability Analysis using Partitioned-ROBDDs," presented at International Conference on Computer-Aided Design, 1997.
[23] A. Yadgar, "Parallel SAT Solving for Model Checking. www.cs.technion.ac.il/~yadgar/Research/research.pdf," 2002.
[24] M. Davis, G. Longeman, and D. Loveland, "A Machine Program for Theorem Proving," *Communications of the ACM*, vol. 5, pp. 394–397, 1962.
[25] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic Model Checking without BDDs," in *Proceedings of Workshop on Tools and Algorithms for Analysis and Construction of Systems (TACAS)*, vol. 1579, *LNCS*, 1999.
[26] M. Sheeran, S. Singh, and G. Stalmarck, "Checking Safety Properties using Induction and a SAT Solver," in *Proceedings of Conference on Formal Methods in Computer-Aided Design*, 2000.
[27] A. Hasegawa, H. Matsuoka, and K. Nakanishi, "Clustering Software for Linux-Based HPC," *NEC Research & Development*, vol. vol 44, No. 1, pp. 60–63, 2003.