Distributed Binary Decision Diagrams for Symbolic Reachability

Wytse Oortwijn

Formal Methods and Tools,
University of Twente

November 1, 2015

Overview

Overview

Formal Methods and Tools

Improving software reliability

- Making software *safer* in practice *(increasing quality by reducing risks)*
- Examples: static & dynamic analysis, risk analysis, *model checking*

Formal Methods and Tools

Improving software reliability

- Making software *safer* in practice *(increasing quality by reducing risks)*
- Examples: static & dynamic analysis, risk analysis, *model checking*

The model checking problem

Given a *formal model* of a software system and a *formal specification*, does the model *satisfy* the specification?

- Exhaustively analyze all model *behaviours*

Formal Methods and Tools

Improving software reliability

- Making software *safer* in practice *(increasing quality by reducing risks)*
- Examples: static & dynamic analysis, risk analysis, *model checking*

The model checking problem

Given a *formal model* of a software system and a *formal specification*, does the model *satisfy* the specification?

- Exhaustively analyze all model *behaviours*

Examples

- Finding deadlocks in software *(e.g. preventing crashes)*
- Finding solutions in games *(e.g. Chess, Sokoban)*

The reachability problem

Given a graph $G = (V, E)$, initial states $I \subseteq V$ and goal states $F \subseteq V$, check if F is *reachable* from I via edges in E

- Allows verification of temporal safety properties, that is:
- *"Something bad will never happen"*

Explicit State Reachability Analysis

The reachability problem

Given a graph $G = (V, E)$, initial states $I \subseteq V$ and goal states $F \subseteq V$, check if F is *reachable* from I via edges in E

- Allows verification of temporal safety properties, that is:
- *"Something bad will never happen"*

Limitations of model checking

- G is often *implicitly* described;
- Set of reachable states is often determined *on-the-fly*, therefore:
- *State space explosions* frequently occur

Fighting State Space Explosions

Reducing the amount of behaviours

- Partial Order Reduction *(exploit commutative transitions)*
- Bisimulation Minimization *(merge "similar" states)*

Fighting State Space Explosions

Reducing the amount of behaviours

- Partial Order Reduction *(exploit commutative transitions)*
- Bisimulation Minimization *(merge "similar" states)*

Efficiently representing state spaces

- Decision Diagrams *(e.g. BDDs, MDDs, LDDs, and ZDDs)*
- SAT-based approaches *(for example, IC3)*

Fighting State Space Explosions

Reducing the amount of behaviours

- Partial Order Reduction *(exploit commutative transitions)*
- Bisimulation Minimization *(merge "similar" states)*

Efficiently representing state spaces

- Decision Diagrams *(e.g. BDDs, MDDs, LDDs, and ZDDs)*
- SAT-based approaches *(for example, IC3)*

Adding hardware resources

- Using many-core machines or high-performance clusters:
  - More memory $\implies$ *larger* state spaces supported
  - More processors $\implies$ *faster* state space exploration

BDD-based Symbolic Reachability

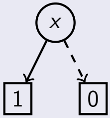
Binary Decision Diagrams (BDDs)

Efficient representation of Boolean functions ($\phi : \mathbb{B}^n \to \mathbb{B}$), e.g.:

BDD-based Symbolic Reachability

Binary Decision Diagrams (BDDs)

Efficient representation of Boolean functions ($\phi : \mathbb{B}^n \to \mathbb{B}$), e.g.:



$\phi \equiv x$

BDD-based Symbolic Reachability

Binary Decision Diagrams (BDDs)

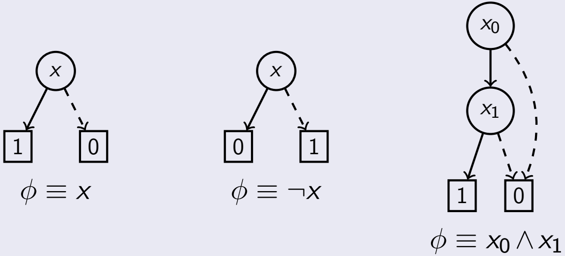
Efficient representation of Boolean functions ($\phi : \mathbb{B}^n \rightarrow \mathbb{B}$), e.g.:

$\phi \equiv x$ $\qquad$ $\phi \equiv \neg x$

BDD-based Symbolic Reachability

Binary Decision Diagrams (BDDs)

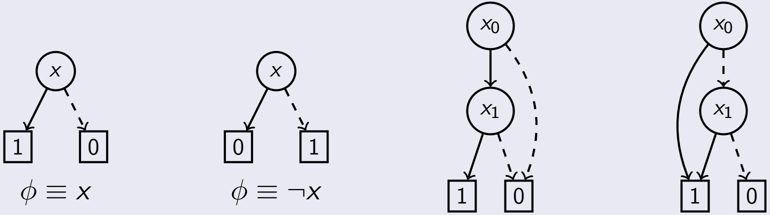
Efficient representation of Boolean functions ($\phi : \mathbb{B}^n \rightarrow \mathbb{B}$), e.g.:



BDD-based Symbolic Reachability

Binary Decision Diagrams (BDDs)

Efficient representation of Boolean functions ($\phi : \mathbb{B}^n \to \mathbb{B}$), e.g.:

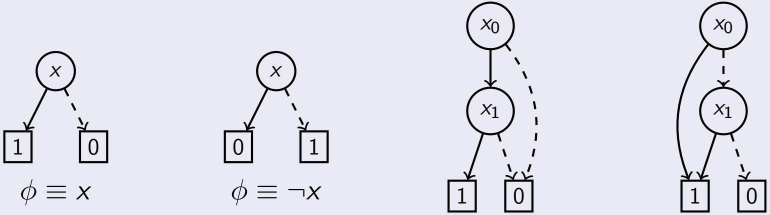


$\phi \equiv x$ | $\phi \equiv \neg x$ | $\phi \equiv x_0 \wedge x_1$ | $\phi \equiv x_0 \vee x_1$

BDD-based Symbolic Reachability

Binary Decision Diagrams (BDDs)

Efficient representation of Boolean functions ($\phi : \mathbb{B}^n \rightarrow \mathbb{B}$), e.g.:



$\phi \equiv x$ $\quad$ $\phi \equiv \neg x$ $\quad$ $\phi \equiv x_0 \wedge x_1$ $\quad$ $\phi \equiv x_0 \vee x_1$

Reachability analysis

- Represent the state space as a BDD:
- Represent initial states and the transition relation as BDDs
- Perform reachability analysis via BDD operations

Improving on Distributed Symbolic Reachability

Challenges on distributed symbolic verification

- Many memory accesses compared to computational work;
- Memory access patterns are irregular

Improving on Distributed Symbolic Reachability

Challenges on distributed symbolic verification

- Many memory accesses compared to computational work;
- Memory access patterns are irregular, therefore:
- *Good* space efficiency, but *limited* time efficiency.

Improving on Distributed Symbolic Reachability

Challenges on distributed symbolic verification

- Many memory accesses compared to computational work;
- Memory access patterns are irregular, therefore:
- *Good* space efficiency, but *limited* time efficiency.

Suggestions by Zhao et al. (2009)

Most important design considerations for improvements are:

1. Data-distribution (including exploiting data-locality);
2. Maintaining load balance;
3. Reducing communication overhead

Overview

High-Performance Infiniband Networks

Advantages of Infiniband

Specialized hardware used to construct high-performance networks:

1. Comparable in price to standard Ethernet hardware
2. Supports up to 100Gb/s
3. NICs can *directly* access main-memory via PCI-E bus
4. End-to-end latencies of $\sim 1\mu s$ have been measured

High-Performance Infiniband Networks

Advantages of Infiniband

Specialized hardware used to construct high-performance networks:

1. Comparable in price to standard Ethernet hardware
2. Supports up to 100Gb/s
3. NICs can *directly* access main-memory via PCI-E bus
4. End-to-end latencies of $\sim 1\mu s$ have been measured

Remote Direct Memory Access (RDMA)

Directly access memory of a remote machine, without invoking its CPU:

- Performance: about 20x as fast as TCP with Ethernet hardware
- Zero-copy networking
- Kernel bypassing

Partitioned Global Address Space

PGAS memory model

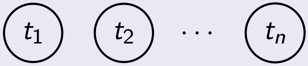
- Combines the shared & distributed memory models
- Each thread hosts a *local* block of memory
- All local memories are combined into a *shared address space*
- Accessing *remote* memory via one-sided RDMA

Partitioned Global Address Space

PGAS memory model

- Combines the shared & distributed memory models
- Each thread hosts a *local* block of memory
- All local memories are combined into a *shared address space*
- Accessing *remote* memory via one-sided RDMA

Schematically

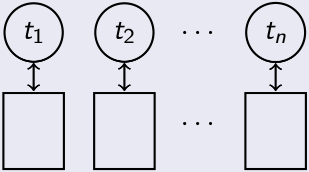


Partitioned Global Address Space

PGAS memory model

- Combines the shared & distributed memory models
- Each thread hosts a *local* block of memory
- All local memories are combined into a *shared address space*
- Accessing *remote* memory via one-sided RDMA

Schematically

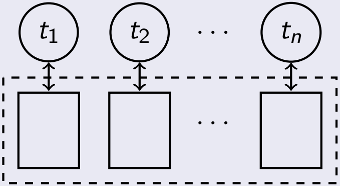


Partitioned Global Address Space

PGAS memory model

- Combines the shared & distributed memory models
- Each thread hosts a *local* block of memory
- All local memories are combined into a *shared address space*
- Accessing *remote* memory via one-sided RDMA

Schematically



Overview

Efficiently Storing BDDs

Shared distributed hash table

The hash table satisfies the following requirements:

1. Minimal number of roundtrips
2. Minimal memory overhead
3. Must be CPU efficient *(i.e. no polling)*

Efficiently Storing BDDs

Shared distributed hash table

The hash table satisfies the following requirements:

1. Minimal number of roundtrips
2. Minimal memory overhead
3. Must be CPU efficient *(i.e. no polling)*

`find-or-put`(d)

Let T be a hash table. Then, for BDD node d:

- If $d \in T$, return `found`
- If $d \notin T$, insert d and return `inserted`
- If $d \notin T$ and d cannot be inserted, return `full`

Design Considerations of `find-or-put`

Collision resolution

- Using *linear probing* for collision resolution;
- Receiving *multiple* buckets *(chunks)* per roundtrip;
- Dynamically determine chunk sizes to minimize expected number of roundtrips

Design Considerations of `find-or-put`

Collision resolution

- Using *linear probing* for collision resolution;
- Receiving *multiple* buckets *(chunks)* per roundtrip;
- Dynamically determine chunk sizes to minimize expected number of roundtrips

Calculating chunk sizes

1. Approximate the load-factor of the hash table
2. Approximate the average probe distance of linear probing
3. Apply a heuristically chosen error margin

Overview

Load Balancing via Work Stealing

Task-based parallelism

- Dividing computational problems into smaller *tasks*
- Task is a basic unit of work and only depend on intermediate *subtasks*
- Each threads maintains a *task pool*

Load Balancing via Work Stealing

Task-based parallelism

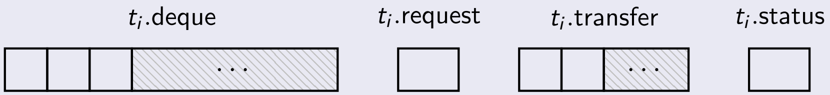
- Dividing computational problems into smaller *tasks*
- Task is a basic unit of work and only depend on intermediate *subtasks*
- Each threads maintains a *task pool*

Work stealing

- Threads are either idle or working
- When idle, threads *steal* from remote task pools
- Stealing thread is *thief*, targetted thread is *victim*
- Termination when all threads are idle

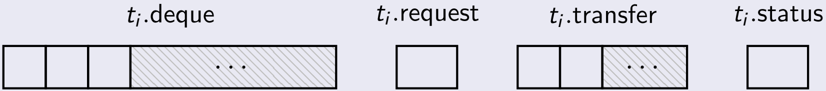
Private-Deque Work Stealing

Memory layout for thread t_i



Private-Deque Work Stealing

Memory layout for thread t_i



Handling steals

- Each thread has a shared task pool *(a private deque)*
- idle threads can *request* a steal by a victim
- Threads continuously *poll* for incoming requests
- Requests are handled by writing tasks to the *transfer* cell of the thief

Overview

Distributed Symbolic Reachability

Designing BDD operations

- Using the shared node table and work stealing operations
- Overlapping roundtrips as much as possible *(i.e. latency hiding)*
- Using a shared global memoization cache

Distributed Symbolic Reachability

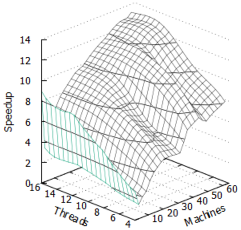
Designing BDD operations

- Using the shared node table and work stealing operations
- Overlapping roundtrips as much as possible *(i.e. latency hiding)*
- Using a shared global memoization cache

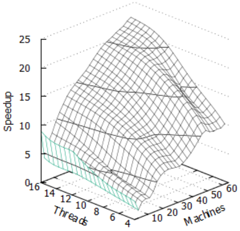
Experimental evaluation

- Performing reachability over well-known BEEM models
- Experiments performed on the DAS-5 cluster
  - We used up to 64 machines
  - Each machine has 16 CPU cores and 64GB internal memory
- Scaling along machines and threads per machine
- Measuring wall clock time and speedup

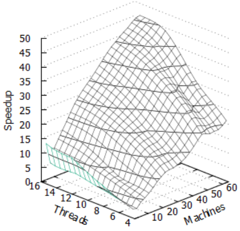
Scalability of Distributed Symbolic Reachability



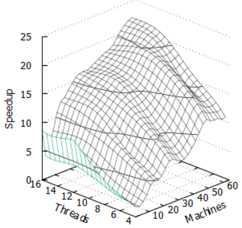
(a) anderson.8



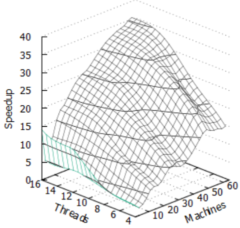
(b) at.6



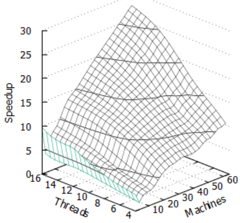
(c) at.7



(d) collision.4



(e) collision.5



(f) schedule-world.3

Overview

Conclusions and Future Work

Conclusions

- Good time-efficiency (in addition to space-efficiency)
- Highest speedups observed: 45x with 64 machines
- Combined memory of 64 machines: 4TB on DAS-5

Conclusions and Future Work

Conclusions

- Good time-efficiency (in addition to space-efficiency)
- Highest speedups observed: 45x with 64 machines
- Combined memory of 64 machines: 4TB on DAS-5

Future Work

- Performing reachability on very large models
- Experimenting with alternative decision diagrams
- Extending to full-blown CTL model checking
- Extending to GPU state space exploration