Distributed Binary Decision Diagrams for Symbolic Reachability

Wytse Oortwijn

Formal Methods and Tools, University of Twente

July 12, 2016
Overview

1. Introduction
2. High-performance networking
3. Distributed unique table
4. Fine-grained task-parallelism
5. Experimental evaluation
6. Conclusion
Overview

1. Introduction

2. High-performance networking

3. Distributed unique table

4. Fine-grained task-parallelism

5. Experimental evaluation

6. Conclusion
Model checking: exhaustive analysis

image source: http://https://d.ibtimes.co.uk
Model checking: exhaustive analysis

Well-known limitation of model checking: state space explosions

image source: http://https://d.ibtimes.co.uk
Fighting state space explosions: adding hardware

more memory: larger state spaces supported
more processors: faster state space generation

image source: http://www.extremetech.com
Fighting state space explosions: problem representation

Partial order reduction

Bisimulation minimisation

SAT solving, IC3

Decision diagrams
Fighting state space explosions: problem representation

Partial order reduction

Bisimulation minimisation

SAT solving, IC3

Binary Decision Diagrams
BDDs: efficient representation of Boolean functions
Distributed symbolic reachability: challenges

Memory accesses dominate computational work

image sources: www.sqlskills.com (left) and www.qnap.com (right)
Distributed symbolic reachability: challenges

Memory accesses dominate computational work

Memory access patterns are often irregular

image sources: www.sqlskills.com (left) and www.qnap.com (right)
Distributed symbolic reachability: challenges

Memory accesses dominate computational work

Previous work achieves:
Good space complexity, but limited time complexity

image sources: www.sqlskills.com (left) and www.qnap.com (right)
Improving on distributed symbolic reachability

Most important design considerations for improvements
(Ciardo, 2009)

1. Data-distribution and exploiting data-locality
2. Maintaining load balance
3. Reducing communication overhead
Overview

1. Introduction
2. High-performance networking
3. Distributed unique table
4. Fine-grained task-parallelism
5. Experimental evaluation
6. Conclusion
Infiniband networking

1. Relatively cheap
2. Bandwidth: up to 100Gb/s
3. End-to-end latency: \( \sim 1\mu s \)
4. Direct access to main-memory

image source: http://www.storagereview.com
Infiniband networking

1. Relatively cheap
2. Bandwidth: up to 100Gb/s
3. End-to-end latency: $\sim 1\mu s$
4. Direct access to main-memory

image source: http://www.storagereview.com
RDMA: Remote Direct Memory Access

1. CPU efficient
2. 20x faster than TCP over Ethernet
3. Zero-copy networking
4. Kernel by-passing

image source: https://www.youtube.com/watch?v=dLw5bA5ziwU (modified)
PGAS: Partitioned Global Address Space

\[ t_1 \quad t_2 \quad \cdots \quad t_n \quad \text{threads} \]
PGAS: Partitioned Global Address Space

\[ t_1 \quad t_2 \quad \ldots \quad t_n \quad \text{threads (fixed } n) \]
PGAS: Partitioned Global Address Space

\begin{center}
\begin{tikzpicture}[node distance=1.5cm, every node/.style={circle,draw,fill=blue!20,minimum size=0.7cm}, >=stealth,font=
ormalsize]
    \node (t1) at (0,0) {$t_1$};
    \node (t2) at (1,0) {$t_2$};
    \node (tn) at (2,0) {$t_n$};
    \node (m1) at (0,-1) {\ldots};
    \node (m2) at (1,-1) {\ldots};
    \node (m3) at (2,-1) {\ldots};
    \node (mem) at (1,-2) {private memory};

    \draw[->] (t1) -- (m1);
    \draw[->] (t2) -- (m2);
    \draw[->] (tn) -- (m3);
    \draw[->] (mem) -- (m1);
    \draw[->] (mem) -- (m2);
    \draw[->] (mem) -- (m3);
\end{tikzpicture}
\end{center}

\textbf{threads} (fixed $n$)
PGAS: Partitioned Global Address Space

threads (fixed $n$)

private memory

global address space
PGAS: Partitioned Global Address Space

threads \((\text{fixed } n)\)

private memory

partitioned global address space
**Shared array**

Global view of a simple array of length $n$
PGAS memory abstraction

**Partitioned shared array**
Split up array into equal parts and distribute among threads
Overview

1. Introduction

2. High-performance networking

3. Distributed unique table

4. Fine-grained task-parallelism

5. Experimental evaluation

6. Conclusion
Unique table operations: find-or-put

Given a hash table $T$ and a BDD node $B$

- $\text{find-or-put}(B)$ returns \textit{found} if $B \in T$
- $\text{find-or-put}(B)$ inserts $B$ and returns \textit{inserted} if $B \not\in T$
- $\text{find-or-put}(B)$ returns \textit{full} if $B \not\in T$ and $B$ cannot be inserted
Unique table implementation

- **Indexing array**
- **BDD array**

\[
\begin{array}{c}
t_1 \quad t_2 \quad \ldots \\
\text{indexing array} \quad \ldots \\
\text{BDD array} \quad \ldots
\end{array}
\]
Unique table implementation

- indexing array
- BDD array

\[ t_1 \quad t_2 \quad t_n \]

Linear probing
Obtain chunks of buckets
Dynamically determine chunk size
Obtain chunks of buckets

Wytse Oortwijn, FMT, University of Twente
Unique table implementation

- Linear probing
- Obtain chunks of buckets
- Dynamically determine chunk size
- Obtain chunks of buckets

Wytse Oortwijn, FMT, University of Twente
Overview

1. Introduction
2. High-performance networking
3. Distributed unique table
4. Fine-grained task-parallelism
5. Experimental evaluation
6. Conclusion
Private-deque work stealing

- 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*
Private-deque work stealing

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

$t_1$  
$\ldots$  
$t_2$  
$\ldots$  
$t_n$  
$\ldots$  

private deque of $t_1$  
private deque of $t_2$  
private deque of $t_n$
Overview

1. Introduction
2. High-performance networking
3. Distributed unique table
4. Fine-grained task-parallelism
5. Experimental evaluation
6. Conclusion
Experimental setup

- 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 over BEEM models

(a) anderson.8
(b) at.6
(c) at.7
(d) collision.4
(e) collision.5
(f) schedule-world.3
Overview

1 Introduction
2 High-performance networking
3 Distributed unique table
4 Fine-grained task-parallelism
5 Experimental evaluation
6 Conclusion
Conclusion

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

Conclusion

- 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