Multi-core Model Checking with LTSmin

Alfons Laarman and Jaco van de Pol
9 May 2014
LTSmin for High-performance Model Checking

Goals

- Investigate high-performance model checking algorithms
- Application to complex man-made and natural systems
- Need access to/from multiple modeling languages
LTSMIN for High-performance Model Checking

Goals

▶ Investigate high-performance model checking algorithms
▶ Application to complex man-made and natural systems
▶ Need access to/from multiple modeling languages

Specification Languages

- mCRL2
- Promela
- DVE
- UPPAAL

Reachability Tools

- Distributed
- Multi-core
- Symbolic

Requirements on tool and interface (LTSMIN / PINS)

▶ Generality: support LTS with arbitrary state/edge labels
▶ On-the-fly API: next-state function to pull implicit graph
Goals

- Investigate high-performance model checking algorithms
- Application to complex man-made and natural systems
- Need access to/from multiple modeling languages

**Requirements on tool and interface (LTSmin / PINS)**

- **Generality**: support LTS with *arbitrary state/edge labels*
- **On-the-fly API**: *next-state* function to pull implicit graph
- **Efficiency**: models expose *locality* in a dependency matrix
LTSmin architecture and PINS interface

Blom, van de Pol, Weber [CAV’10], Laarman, van de Pol, Weber [NFM’11]
http://fmt.cs.utwente.nl/tools/ltsmin/

And also: LLVM, parity games, Markov Automata, C-code
Indirectly: GSPN, xUML, Signalling Networks in Biology
LTSmin architecture and PINS interface
Blom, van de Pol, Weber [CAV’10], Laarman, van de Pol, Weber [NFM’11]
http://fmt.cs.utwente.nl/tools/ltsmin/

And also: LLVM, parity games, Markov Automata, C-code
Indirectly: GSPN, xUML, Signalling Networks in Biology
# Table of Contents

1. **Multi-core Reachability**
   - Shared hash table
   - Parallel state compression

2. **Multi-core LTL model checking**
   - Büchi automata for LTL model checking
   - Parallel Nested Depth First Search

3. **Wrapup**
   - Lessons learnt
   - Perspectives
Main bottleneck for multi-core reachability: Shared Hash Table

- **State storage**: requires concurrent access (lock contention)
- **Graph traversal**: random memory access (bandwidth)
- **Computer architecture**: shared L2 caches (false sharing)
### Main bottleneck for multi-core reachability: Shared Hash Table

- **State storage**: requires concurrent access (lock contention)
- **Graph traversal**: random memory access (bandwidth)
- **Computer architecture**: shared L2 caches (false sharing)

### Design: keep it simple

- **Hash memoization**: read less data
- **On collision**: *Walking the Line*
- **In-situ locking (1 bit per bucket)**
- **Bucket operations require CAS**
Scalability Experiments from 2010 (BEEM database)

SPIN 5.2.4 (NASA/JPL)

DiVinE 2.2 (Brno,CZ)
Scalability Experiments from 2010 (BEEM database)

SPIN 5.2.4 (NASA/JPL)

DiVinE 2.2 (Brno, CZ)

LTSmin (U Twente, NL)
Table of Contents

1 Multi-core Reachability
   - Shared hash table
   - Parallel state compression

2 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Parallel Nested Depth First Search

3 Wrapup
   - Lessons learnt
   - Perspectives
State space compression

Laarman, van de Pol, Weber [spin11], Laarman, van der Vegt [memics’11]

Parallel Tree Compression

- Reuse lockless hash table: view vector as tree of pairs
  - Massive sharing of “similar” subvectors

\[ \langle 3, 5, 5, 4, 1, 3 \rangle \]
State space compression

Laarman, van de Pol, Weber [spin11], Laarman, van der Vegt [memics’11]

Parallel Tree Compression

- Reuse lockless hash table: view vector as tree of pairs
  - Massive sharing of “similar” subvectors

```
⟨3, 5, 5, 4, 1, 3⟩

⟨3, 5, 5⟩  5  ⟨4, 1, 3⟩

⟨3, 5⟩  3

⟨3, 5⟩  4  1
```

```
⟨3, 5⟩

⟨3, 5⟩  4  1
```

```
⟨3, 5⟩

⟨3, 5⟩
```
State space compression

Laarman, van de Pol, Weber [spin11], Laarman, van der Vegt [memics’11]

Parallel Tree Compression

- Reuse lockless hash table: view vector as tree of pairs
  - Massive sharing of “similar” subvectors

```plaintext
\langle 3, 5, 5, 4, 1, 3 \rangle
```

```
4 1
3 5
```

```
\langle 3, 5, 5 \rangle
```

```
\langle 3, 5 \rangle
```

```
\langle 4, 1, 3 \rangle
```

```
\langle 4, 1 \rangle
```

```
\langle 3, 5 \rangle
```

```
\langle 4, 1 \rangle
```

UNIVERSITY OF TWENTE.
State space compression
Laarman, van de Pol, Weber [spin11], Laarman, van der Vegt [memics’11]

Parallel Tree Compression

- Reuse lockless hash table: view vector as tree of pairs
  - Massive sharing of “similar” subvectors

```
4 1
6 5
1 3
3 5

⟨3, 5, 5, 4, 1, 3⟩

⟨3, 5, 5⟩  ⟨4, 1, 3⟩

6 5
1 3

⟨3, 5⟩  ⟨4, 1⟩

⟨3, 5⟩
```

State space compression

Laarman, van de Pol, Weber [spin11], Laarman, van der Vegt [memics’11]

Parallel Tree Compression

- Reuse lockless hash table: view vector as tree of pairs
  - Massive sharing of “similar” subvectors

```
<table>
<thead>
<tr>
<th>4</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
```

```
⟨3, 5, 5, 4, 1, 3⟩

⟨3, 5, 5⟩

⟨4, 1, 3⟩

6 5

2 4

1 3

3 5

3 5

⟨3, 5⟩

⟨4, 1⟩
```
Parallel Tree Compression

- Reuse lockless hash table: view vector as tree of pairs
  - Massive sharing of “similar” subvectors
- **Incremental updates**: use the Dependency Matrix
  - \((K - 1) \rightarrow \log_2(K - 1)\) lookups
Computation Experiments from 2011 [BEEM database]
Laarman, van de Pol, Weber [spin11]

- Tree compression is a recursive variant of SPIN's COLLAPSE (’97)
- Exploit combinatorial structure:
  - State vectors are highly similar
  - Impressive compression ratios
- Extreme case: firewire_tree
  - Uncompressed: 14 GB
  - Tree Compression: 96 MB
- Compression comes for free
  - Arithmetic intensity increases
  - Less memory-bus traffic
Table of Contents

1 Multi-core Reachability
   - Shared hash table
   - Parallel state compression

2 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Parallel Nested Depth First Search

3 Wrapup
   - Lessons learnt
   - Perspectives
Model Checking by Accepting Cycles

**LTL Model Checking**

- A buggy run in a system can be viewed as an infinite word
- Absence of bugs: emptiness of some Büchi automaton
Model Checking by Accepting Cycles

LTL Model Checking

- A **buggy run** in a system can be viewed as an **infinite word**
- Absence of bugs: emptiness of some Büchi automaton
- Graph problem: find a **reachable accepting state on a cycle**
- Basic algorithm: Nested Depth First Search (NDFS)

Properties of NDFS

- NDFS runs in linear time
- Inherently depends on post-order
- Post-order is P-complete [Reif’85]
- Not parallelizable (unless P=NC)
- Use BFS: OWCTY, $O(N^2)$ [Brno]
Recall: Nested Depth First Search
[CVWY’92] [Holzmann’92]

- **Blue search**: explore graph in DFS order
  - states on the blue search stack are cyan
  - on backtracking from an accepting state:
- **Red search**: find an accepting cycle
  - exit as soon as the cyan stack is reached
- Linear time, depends on post-order

**Blue search**

1: procedure dfsBlue(s)
2: add s to Cyan
3: for all successors t of s do
4:   if t \not\in Blue \cup Cyan then
5:     dfsBlue(t)
6:   if s is accepting then
7:     dfsRed(s)
8: move s from Cyan to Blue

**Red search**

1: procedure dfsRed(s)
2: add s to Red
3: for all successors t of s do
4:   if t \in Cyan then
5:     Exit: cycle detected
6:   if t \not\in Red then
7:     dfsRed(t)
Table of Contents

1 Multi-core Reachability
   - Shared hash table
   - Parallel state compression

2 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Parallel Nested Depth First Search

3 Wrapup
   - Lessons learnt
   - Perspectives
Multi-core Swarmed NDFS

- \( W \) workers perform independent random NDFS
- Visited states are stored in a shared hashtable
- All workers use their own set of colors
- Speeds up bug hunting only

Blue search

1: procedure dfsBlue\((s, i)\)
2: add \( s \) to \( \text{Cyan}[i] \)
3: for all successors \( t \) of \( s \) do
4: if \( t \not\in \text{Blue}[i] \cup \text{Cyan}[i] \) then
5: dfsBlue\((t, i)\)
6: if \( s \) is accepting then
7: dfsRed\((s, i)\)
8: move \( s \) from \( \text{Cyan}[i] \) to \( \text{Blue}[i] \)

Red search

1: procedure dfsRed\((s, i)\)
2: add \( s \) to \( \text{Red}[i] \)
3: for all successors \( t \) of \( s \) do
4: if \( t \in \text{Cyan}[i] \) then
5: Exit: cycle detected
6: if \( t \not\in \text{Red}[i] \) then
7: dfsRed\((t, i)\)
Multi-core Nested Depth First Search  
Laarman, van de Pol,...[ATVA’11][PDMC’11]; Evangelista,L,vdP [ATVA’12]

Multi-core NDFS (several variations)

▶ Collaboration between NDFS workers
  ▶ Share red and/or blue globally
  ▶ Workers backtrack on parts finished by others
  ▶ Correctness: Complicated to restore post-order
  ▶ Performance: Reasonable scalability

Blue search

1: procedure dfsBlue(s, i)  
2: add s to Cyan[i]  
3: for all successors t of s do  
4: if t \notin Blue \cup Cyan[i] then  
5: dfsBlue(t, i)  
6: if s is accepting then  
7: dfsRed(s, i)  
8: move s from Cyan[i] to Blue

Red search

1: procedure dfsRed(s, i)  
2: add s to Red  
3: for all successors t of s do  
4: if t \in Cyan[i] then  
5: Exit: cycle detected  
6: if t \notin Red then  
7: dfsRed(t, i)
OWCTY (BFS) and Swarmed NDFS versus Parallel NDFS

Experiments from [ATVA'11] on BEEM benchmarks on 16 cores

Swarmed versus Sequential NDFS

Conclusions

- Swarmed NDFS speeds up bug hunting
OWCTY (BFS) and Swarmed NDFS versus Parallel NDFS
Experiments from [ATVA'11] on BEEM benchmarks on 16 cores

Conclusions

- Swarmed NDFS speeds up bug hunting
- Parallel NDFS also speeds up verification
OWCTY (BFS) and Swarmed NDFS versus Parallel NDFS

Experiments from [ATVA’11] on BEEM benchmarks on 16 cores

Conclusions

- Swarmed NDFS speeds up bug hunting
- Parallel NDFS also speeds up verification
- Parallel NDFS finds bugs faster than OWCTY (BFS)
Experiments extended to 48 cores

From [PDMC’12]. See fmt.cs.utwente.nl/tools/ltsmin/performance/

Reachability

LTL model checking

Promela: Bakery protocol

Promela: Elevator controller
Table of Contents

1 Multi-core Reachability
   - Shared hash table
   - Parallel state compression

2 Multi-core LTL model checking
   - Büchi automata for LTL model checking
   - Parallel Nested Depth First Search

3 Wrapup
   - Lessons learnt
   - Perspectives
Evaluation: what did we learn?

Reachability: Implementation matters, keep it simple

- Leave workers alone when possible; little load balancing
- Rely on randomness to avoid most “duplicate work”
- Careful design of concurrent data structures
Evaluation: what did we learn?

Reachability: Implementation matters, keep it simple
- Leave workers alone when possible; little load balancing
- Rely on randomness to avoid most “duplicate work”
- Careful design of concurrent data structures

LTL model checking: swarmed NDFS + minimal synchronisation
- Previous parallel algorithms (OWCTY) used BFS: $O(N^2)$
- Now: linear, speedups . . . so $P = NC$??
  - #workers: $W \rightarrow \infty$ versus $W = 48$
  - Worst case $O(N \cdot W)$, no speedup
Extensions

- Multi-core LTL model checking for *Timed Automata* (with Aalborg)
- Parallel Symbolic Model Checking: *Multi-core BDD package* (Tom van Dijk)
- *Markov Automata*: extend LTSmin to probabilistic systems (Mark Timmer, Stefan Blom) (Katoen: GSPN, Stoelinga: *dynamic fault trees*)
- Solving *parity games* distributed/multi-core/symbolic (Gijs Kant)
- Link to *C-code* directly, or via LLVM (Freak vd Berg)
Perspectives (2)

Applications

- **Railway Interlockings** (xUML diagrams, INESS/ERTMS) (Jeroen Ketema, Helle Hansen, Bas Luttik)
- **Railway Maintenance** with dynamic fault trees (ProRail) (Guck, Ruijters, Stoelinga, Katoen)
- **RERS challenge in (PLC-like) software verification** (Theo Ruys)
- **CERN’s** control software of the Large Hadron Collider (Gijs Kant, Tim Willemse)
- **Energy-autonomous streaming** apps via SDF and UPPAAL (Waheed Ahmad, P. Hölzenspies)
- **System Biology** signalling networks via Timed Automata (Stefano Schivo, Langerak: ANIMO Executable Chondrocyte)
Literature on LTSmin (reachability)

**LTSmin toolset**

- Stefan Blom, Jaco van de Pol, Michael Weber, LTSmin: Distributed and Symbolic Reachability ............. (CAV 2010)
- Alfons Laarman, Jaco van de Pol, Michael Weber, Multi-Core LTSmin: Marrying Modularity and Scalability....(NFM 2011)

**Reachability**

- Alfons Laarman, Jaco van de Pol and Michael Weber, ... (FMCAD 2010) Boosting Multi-Core Reachability Performance with Shared Hash Tables
- Alfons Laarman, Jaco van de Pol, Michael Weber, Parallel Recursive State Compression for Free ............. (SPIN 2011)
- Tom van Dijk, Alfons Laarman and Jaco van de Pol, Multi-core BDD Operations for Symbolic Reachability ....(PDMC 2012)
Literature on LTSmin (advanced)

### LTL model checking
- Alfons Laarman, Jaco van de Pol, *Variations on Multi-Core Nested Depth-First Search* ...... (PDMC 2011)
- Sami Evangelista, Alfons Laarman, Laure Petrucci and Jaco van de Pol, *Improved Multi-Core Nested Depth-First Search* ............(ATVA 2012)

### Timed Automata
- A. Dalsgaard, A.W. Laarman, K.G. Larsen, M. Olesen, J. van de Pol, *Multi-Core Reachability for Timed Automata* .......... (FORMATS’12)
- Alfons Laarman, M. Olesen, A. Dalsgaard, K.G. Larsen, J. van de Pol, *Multi-core emptiness checking of timed Büchi automata using inclusion abstraction* ............................................. (CAV’13)