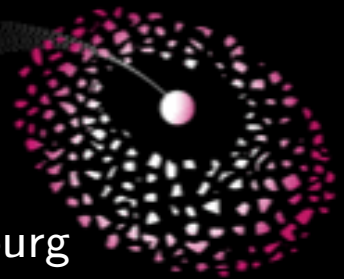
UNIVERSITY OF TWENTE.
Formal Methods & Tools.

Scalable Multi-core Model Checking: Technology & Applications of Brute Force
Day I: Reachability

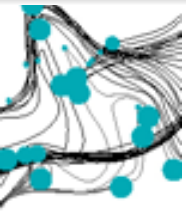


Jaco van de Pol

30, 31 October 2014

VTSA 2014, Luxembourg

Table of Contents



The Reachability Problem

Reachability Problem – Instances:

- Find assertion violations in multi-core software
- Find safety risks in Railway Interlockings
- Find solutions to games/puzzles, e.g. Sokoban

The Reachability Problem in general graphs

- Given a graph $G = (V, R)$ (nodes, edges)
- Initial states $I \subseteq V$ and goal/error states $F \subseteq V$
- Check: is there a path in G from I to F? i.e. is F reachable?

- Typically, the graph is given implicitly,
  as the state space of a program or a specification.

Reasons for State Space Explosion

Concurrency: exponential growth

- System of n components, each can be in m states
- The total state space may consist of m^n states.
- Example: Railway safety systems (signals, points, tracks)

Data variables: exponential growth

- Given n different variables, each may take m values
- Potential number of different state vectors: m^n
- Example: model checking software, rather than models

How to handle $> 10^{100}$ states??

- Partial Order Reduction: Avoid certain states systematically
- Symbolic model checking: Treat sets of states simultaneously
- Focus of my lectures: Brute force parallel computation

Motivation for High-Performance Model Checking

Solution to State Space Explosion?

- Model checking suffers from the state space explosion, Therefore it is very time and memory intensive
- Reaching the memory bound is an immediate show stopper, But also excessive waiting times put a bound on applicability
- Why not simply throw more computer power at the problem?

Will this help in practice? Is this scientifically interesting?

- Is the problem embarrassingly parallel?
- No: Graph algorithms are not easy to parallelize efficiently, so clever algorithm engineering is necessary.
- But: only linear improvement for an exponential problem...
- Yes, orthogonal to clever reduction techniques: start simple

Various possibilities regarding underlying hardware

Distributed computing:

- network of workstations, clusters, Grid - cheap
- this allows accumulation of available memory
- But: limited bandwidth, high latency

Parallel computing (shared memory):

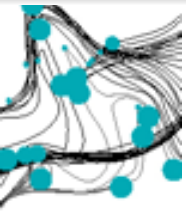
- Multi-core, supercomputers - expensive, but price dropping
- 64-bit machines, $>$ 120GB RAM, 8-64 cores: quite popular
- But: Scalability is imperfect, heterogeneous (so distributed?)

Several alternatives are under investigation:

- Use hard disk as substitute for RAM
- CUDA (GPU), Cell processors, FPGA, cloud, map/reduce

In all cases: algorithms must be fundamentally revised!

Table of Contents



Model Checking made Practical and Widespread?

Main obstacles

- Scalability
  - parallel components
  - data, buffers, ...
- Modeling effort
  - many languages
  - avoid modeling?
- Complex tools
  - algorithms, heuristics
  - low-level details

Algorithmic solutions (combinatorics: locality)

- on-the-fly model checking
- symbolic model checking
- bounded model checking
- partial-order reduction
- symmetry reduction
- parallel model checking

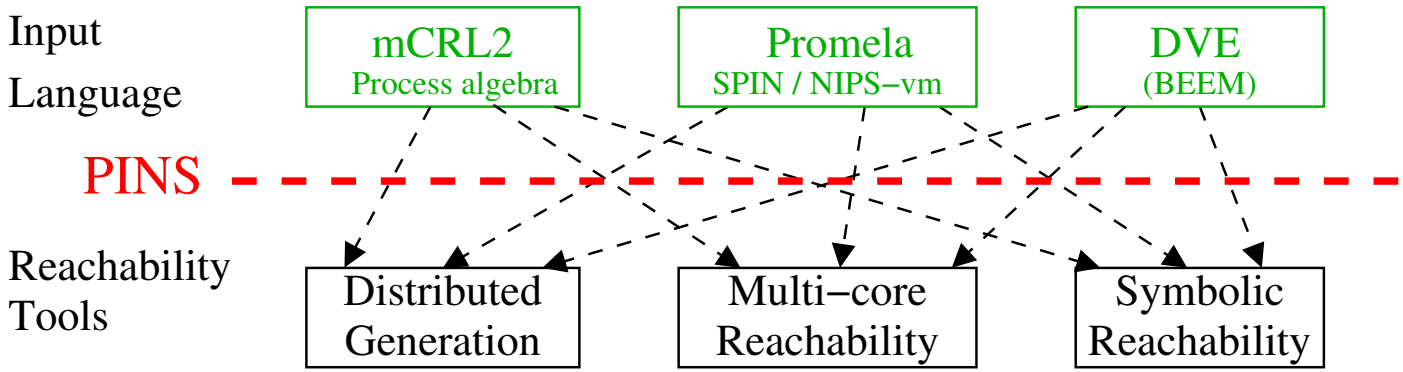
Problem: algorithms are often tied to specification languages

- No particular technique suits all applications / models
- A user needs to rewrite his model into different languages

Solution Direction

Where to draw the line?

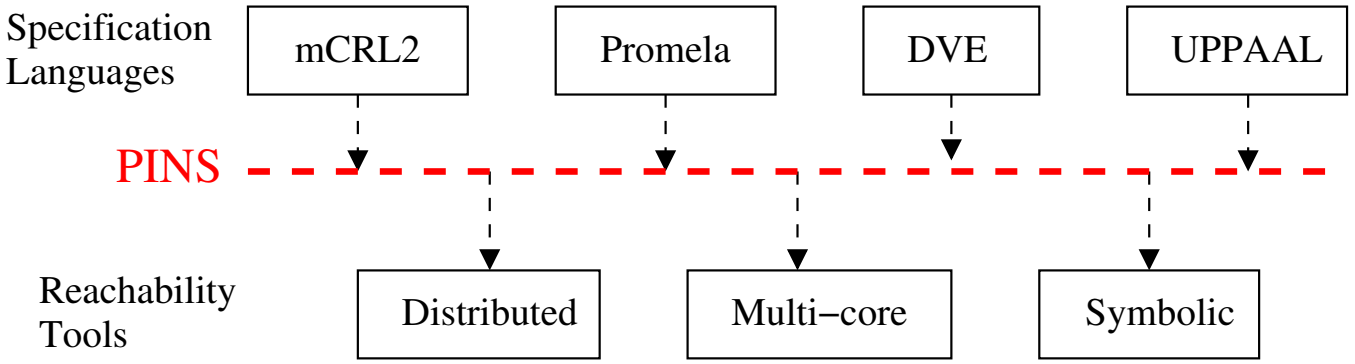
- Separate languages and algorithms via a clean interface (API)
- API should be simple: allow many different languages
- API should be rich: expose locality structure to algorithms



PINS interface of LTSmin toolset:

- Frontends provide on-the-fly access to a state space
- Backend algorithms determine the verification strategy

High-performance Model Checking for the Masses



	x	y	z
t_1	r	w	–
t_2	–	r	w
t_3	w	–	rw

Advantages of tool and interface (LTSmin / PINS)

- General and flexible: support for arbitrary state/edge labels
  - Also: LLVM, parity games, Markov Automata, C-code, B||CSP
  - Indirectly: GSPN, xUML, Signalling Networks in Biology
- On-the-fly API: next-state function to pull the 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/

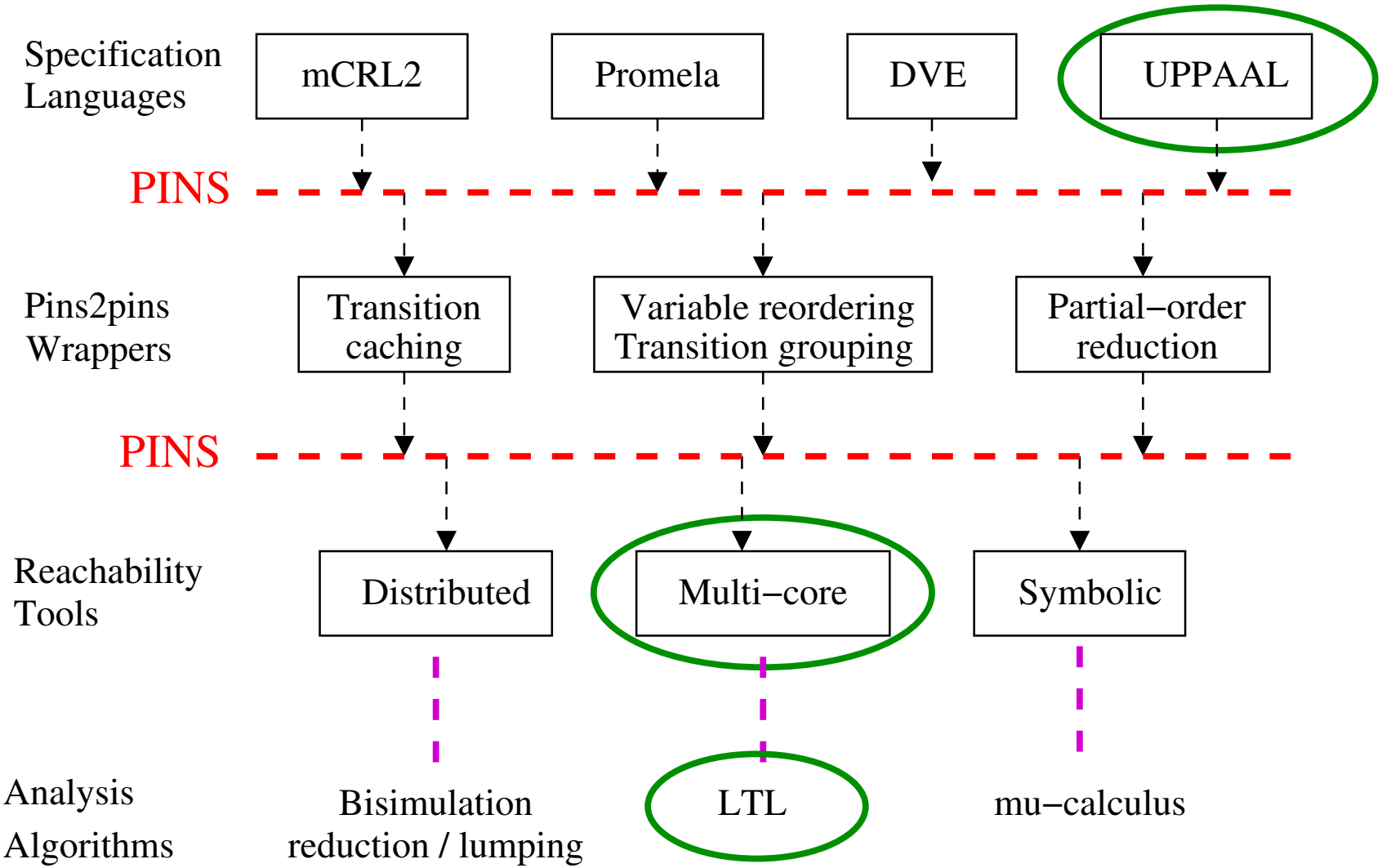
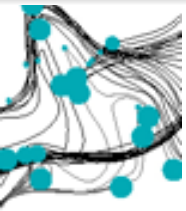


Table of Contents



Lecture on High-performance Model Checking

High-level Goals

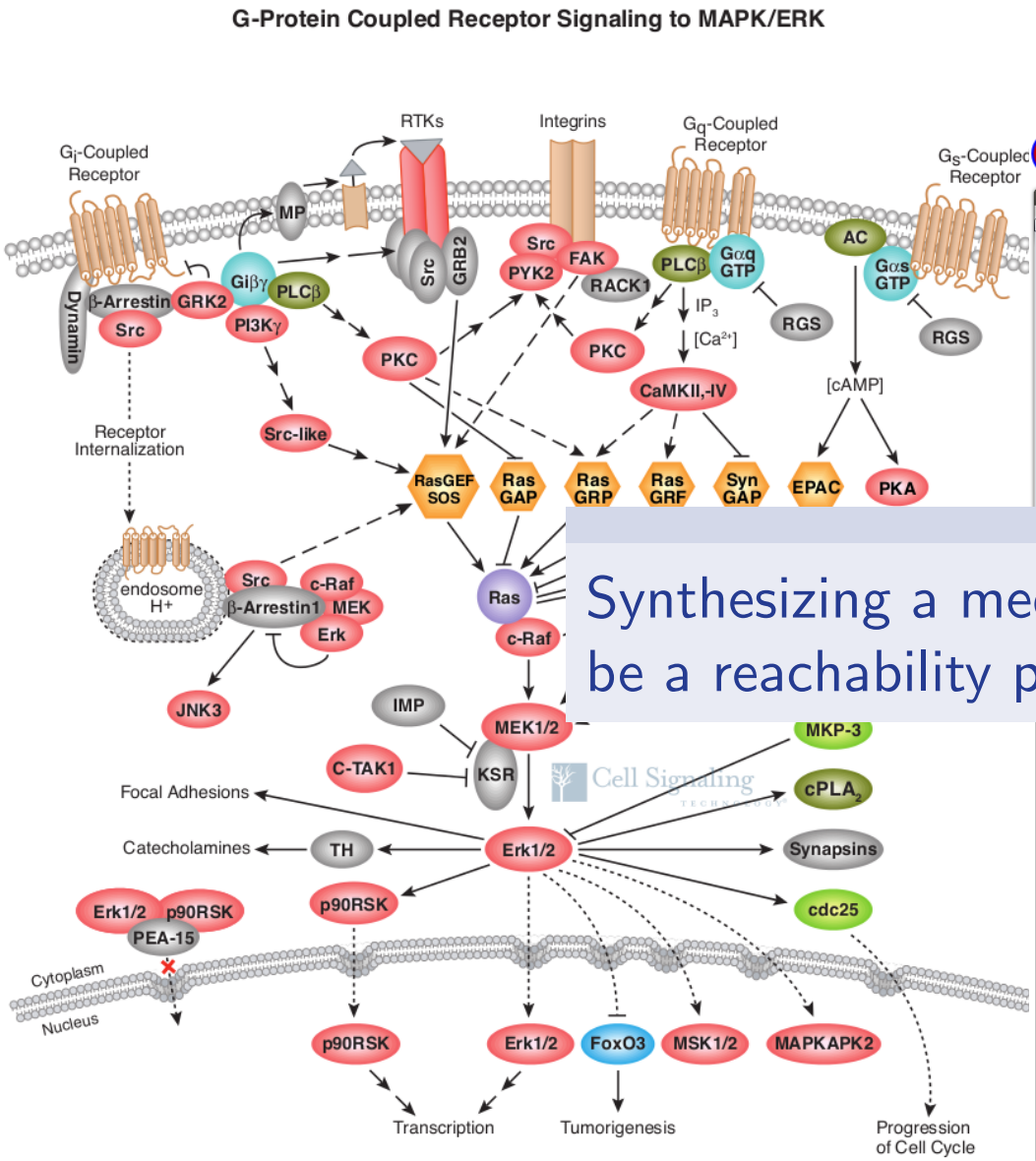
- Investigate high-performance model checking algorithms
- Applications to complex man-made and natural systems

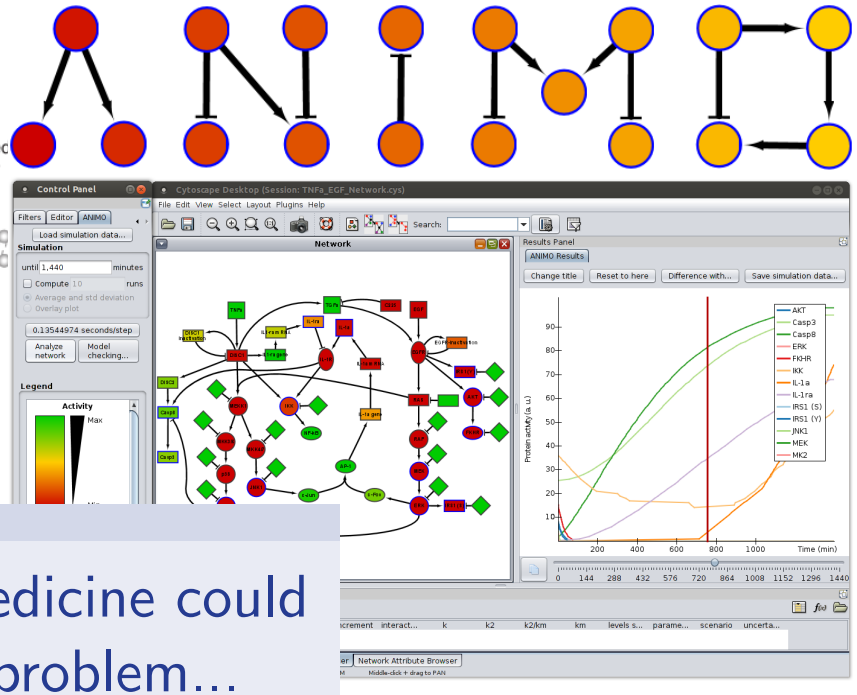
Ingredients

- Basic multi-core datastructures for Reachability
- Checking liveness properties – LTL, multi-core Nested DFS
- Symbolic representation: LTL for Timed Automata
- Symbolic representation: Multi-core Decision Diagrams
- Application to Biological Signaling Pathways
- Application to xUML diagrams for Railway Safety

Signaling Pathways with Timed Automata

Stefano Schivo, Langerak, van de Pol etal. [BIBE'12] [GENE'13] [J-BHI'14]





Synthesizing a medicine could be a reachability problem...

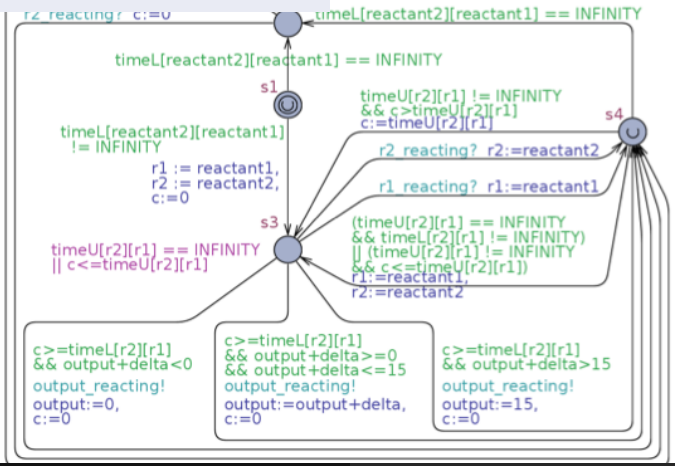
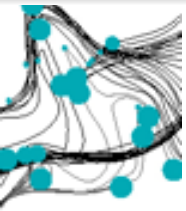
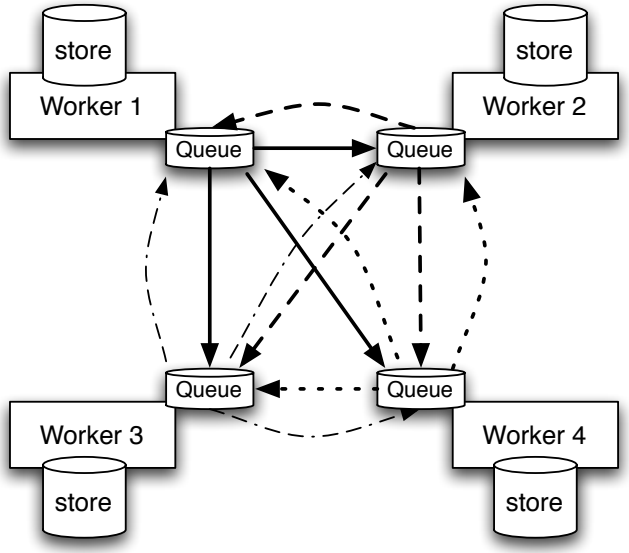
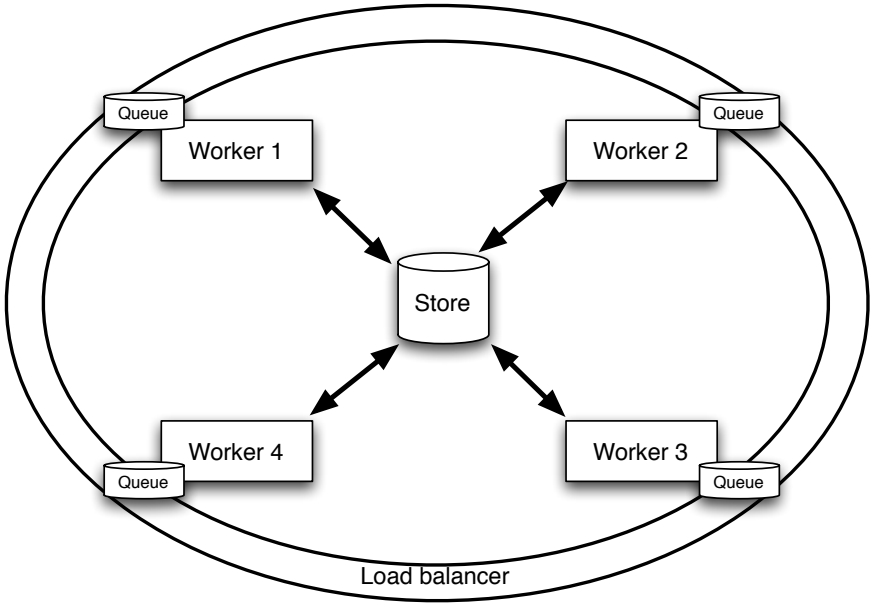


Table of Contents



Which architecture suits Multi-core Model Checking?





Static partitioning

- Distributed memory solution
- Communication: W^2 queues
- (Relaxed) BFS only

Shared hash table

- (Pseudo) DFS & BFS
- Communication: shared hash table
- Load balancing

Algorithm: parallel reachability

Data: Global set $V = \emptyset$, Local sets $S_0 = I$, $S_1 = \cdots = S_{N-1} = \emptyset$
for $0 \leq id < N$ **do in parallel**
  while LoadBalance(S_{id}) **do**
    while *some work to do and no timeout* **do**
1     $state \leftarrow S_{id}$.Get()
      $count \leftarrow 0$
      check invariants on *state*
      for $s \in$ NextState(*state*) **do**
        increment *count*
2   **if not** V.FindOrPut(s) **then**
          S_{id}.Put(s)
      if $count = 0$ **then** report deadlock

(1) "Open" set S influences search order (e.g.: BFS, DFS)

(2) Shared-Memory synchronization point

- Locking the hashtable is not an option

Lockless Hash Table: Design

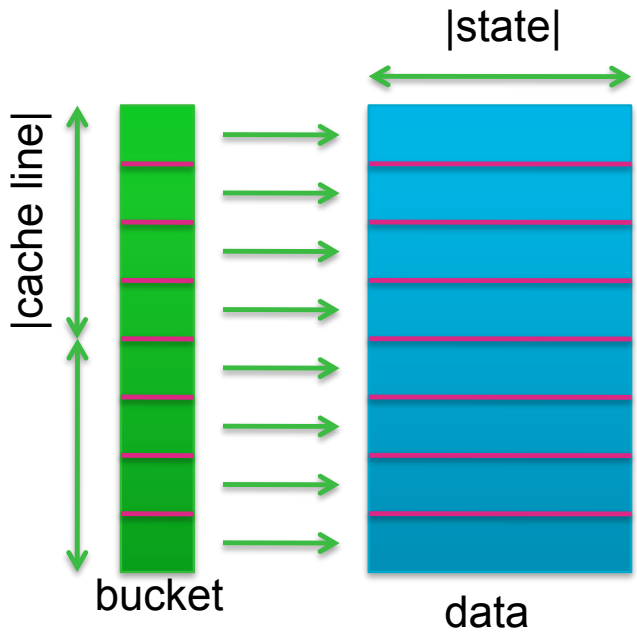
Alfons Laarman, van de Pol, Weber [fmcad10]

Main bottlenecks for scalable implementation

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

Design: keep it simple

- Open addressing
- Hash memoization: read less data
- Separate hash and data
- On collision: Walking the Line
- In-situ locking (1 bit per bucket)
- Bucket operations require CAS
- Not strictly wait-free

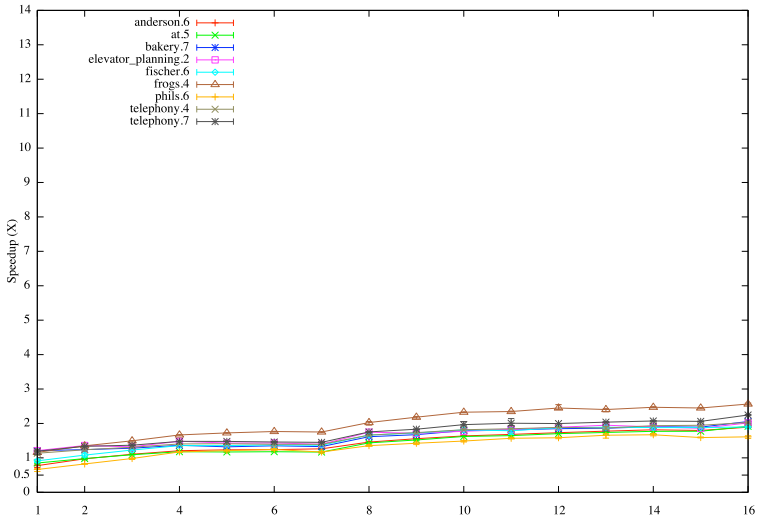


Algorithm: multi-core FindOrPut

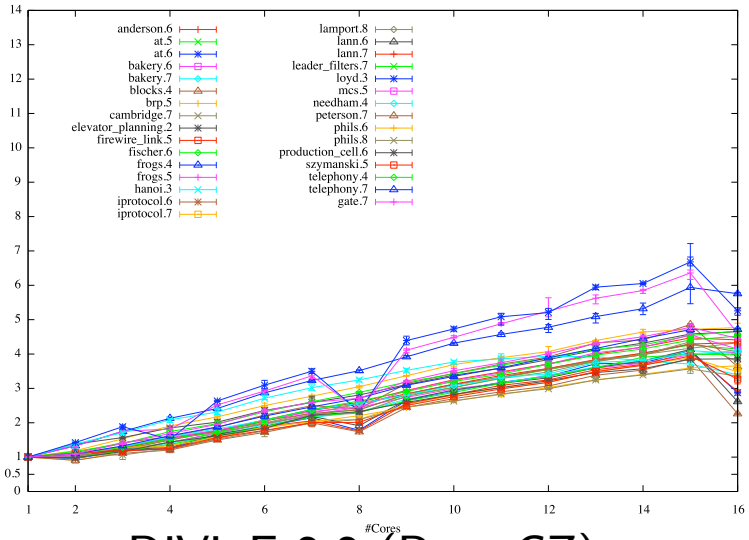
Input : state
Output: true if seen, false otherwise
Data: size, Bucket[size], Data[size]

```
1  h ← Hash(state);   index ← h  mod size
2  for i in WalkTheLineFrom(index) do
3      if empty = Bucket[i] then
4          if CompareAndSwap(Bucket[i], empty, ⟨h, write⟩) then
5              Data[i] ← state
6              Bucket[i] ← ⟨h, done⟩
7              return false
8      if ⟨h, ?⟩ = Bucket[i] then
9          while ⟨?, write⟩ = Bucket[i] do ... wait ...
10         if Data[i] = state then return true
```

Scalability Experiments from 2010 (BEEM database)



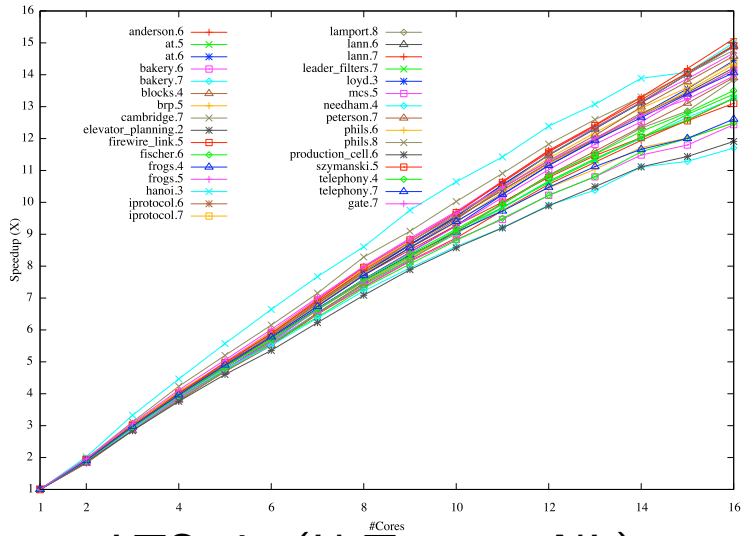
SPIN 5.2.4 (NASA/JPL)



DiVinE 2.2 (Brno,CZ)

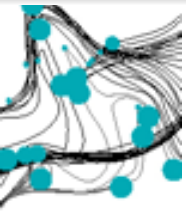
Barnat (2007)

- "our shared hash tables do not scale beyond 8 cores"
- "could not investigate lockless hash table solution"
- "haven't found the cause of the scalability issues"



LTSmin (U Twente, NL)

Table of Contents



State space compression

Where is the bottleneck for parallel reachability?

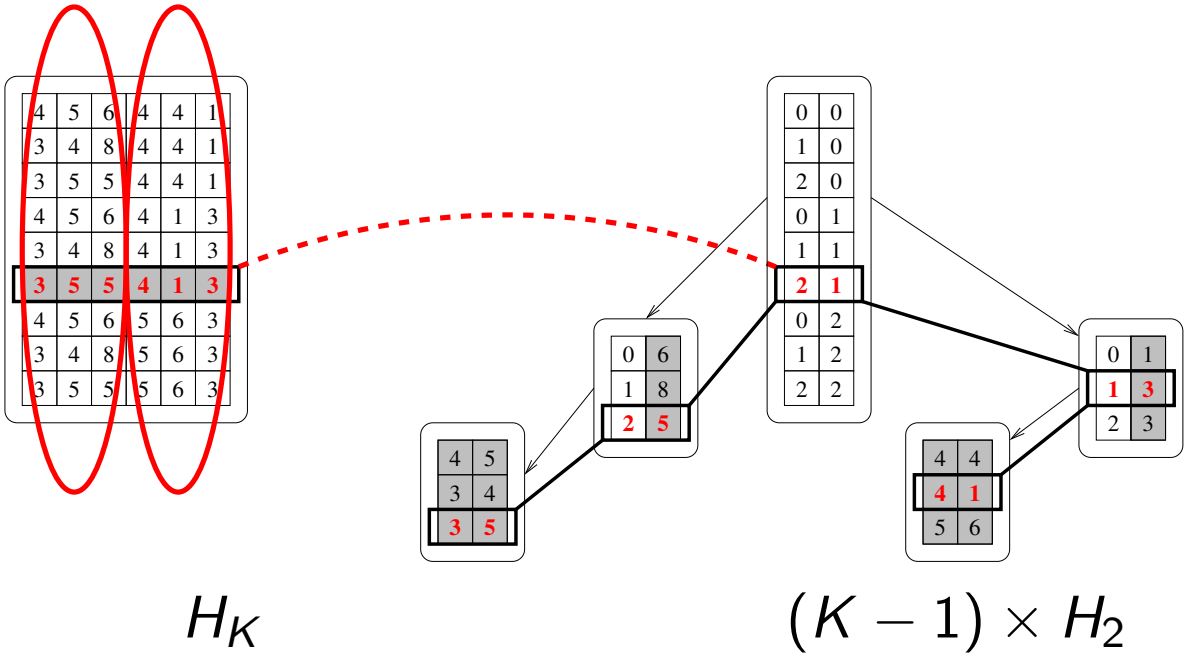
- In every step: read and write long state vectors
- Memory: puts an upper limit to the state space
- Time: memory bus becomes the bottleneck for speedup

Exploit locality

- Due to locality: subsequent state vectors have a lot of overlap
- The set of state vectors can be greatly compressed
- Requirement: quick check if a state has been visited
- (otherwise the specification is a very good compression)

Recursive indexing (Tree Compression)

Blom, Lisser, van de Pol, Weber [PDMC'07, JLC'09]



Analysis

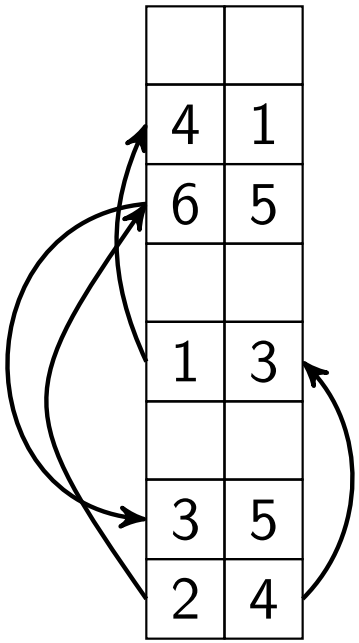
- Locality $\Longrightarrow$ balanced tree $(N + 2\sqrt{N} + 4\sqrt[4]{(N)} \cdots \approx N)$
  Compresses states of length K to almost 2 (!)
- Hard to parallelize:
  - Sequential operation on tree of tables
  - Many small (variable size) hash tables

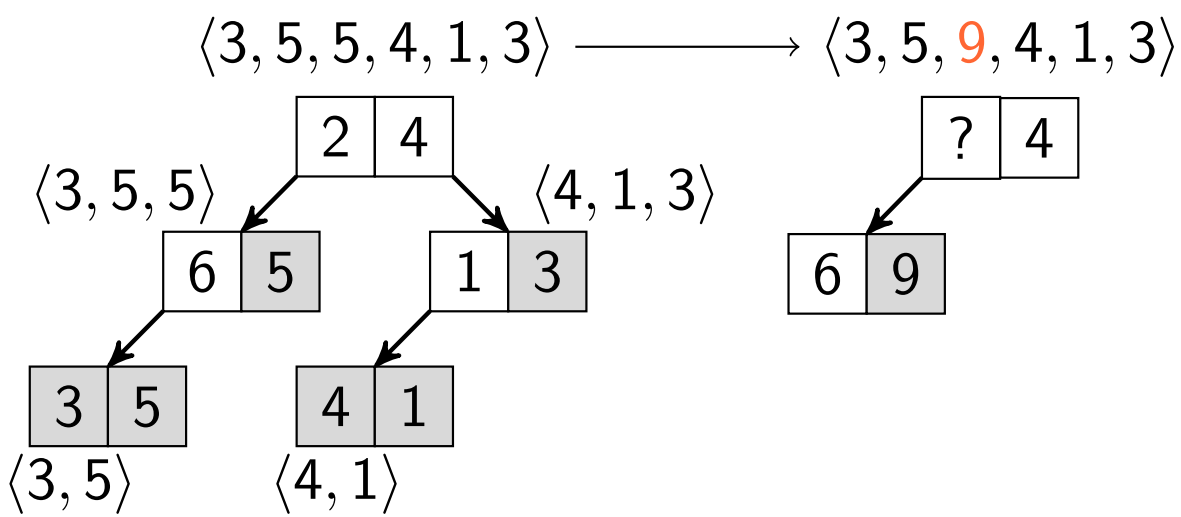
Parallel Tree Compression

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

Solution

- Reuse lockless hash table: merge tree of tables into one
- Incremental updates: use the Dependency Matrix
  - $(K-1) \rightarrow \log_2(K-1)$ lookups

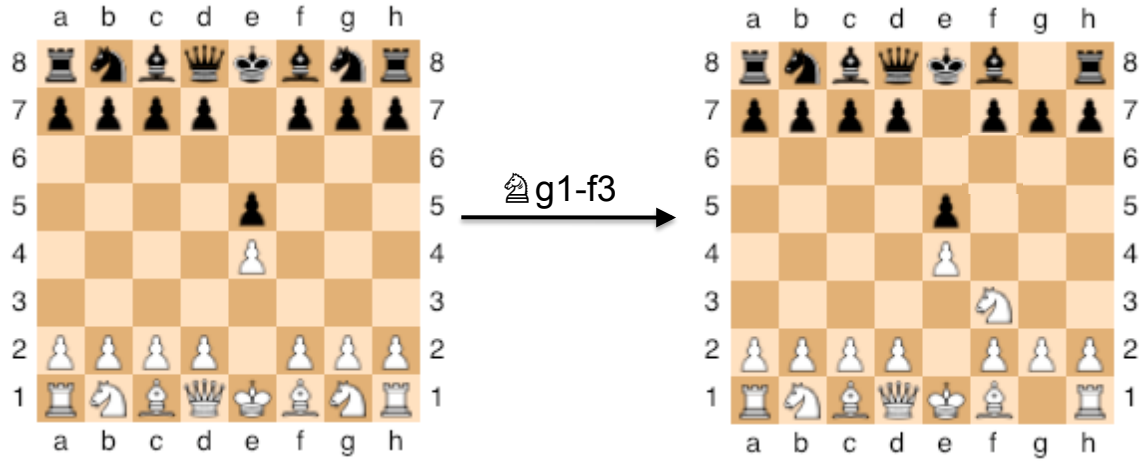




Exploiting locality once more

Dependency Matrix $D_{M \times N}$ predicts changing state parts:

- Incremental tree insertions:
  - Traverse only the changing paths in the Tree of Tables
- Incremental hashing, based on Albert L. Zobrist (1969):



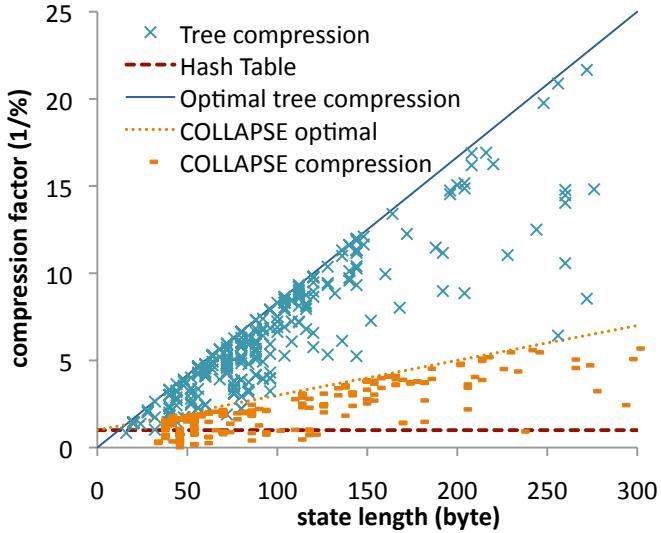
- Even further compression:
  - J.G. Cleary (1984): infer part of hash value from its address
  - Vegt/Laarman (2012): Parallel Compact Hash Table
- Can now compress $2^{35} = 3.4 \cdot 10^{10}$ states into 160GB

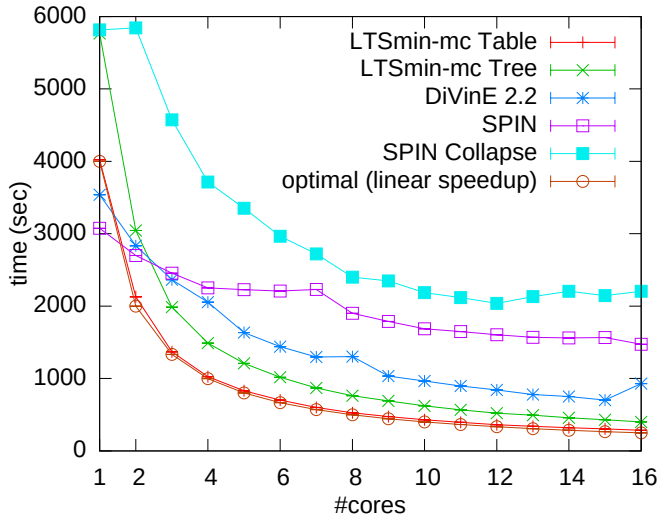
Compression Experiments from 2011 [BEEM database]

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





Literature on LTSmin (reachability)

LTSmin toolset

- http://fmt.cs.utwente.nl/tools/ltsmin/
- 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 and State Compression

- 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)
- Steven van der Vegt, Alfons Laarman,
  A Parallel Compact Hash Table ..... (MEMICS 2011)