Platform Dependent Verification
Engineering Verification Tools for 21st Century

Luboš Brim and Jiří Barnat

Paradise
Parallel & Distributed Systems Laboratory

Masaryk University
Czech Republic
Motivation

Complexity of developed systems is growing
- Numerical computation & text processing
- Database systems, computer games
- Simulations of physical systems
- Virtual reality
- ...

Parallelism and communication are essential

Correctness and reliability are critical measures.
Flaws of concurrent systems

Concurrent systems are everywhere
- Network applications, data communication protocols
- Client-server systems
- Multi-threaded code
- ...

Errors Related to Concurrency
- Deadlocks
- Livelocks
- Underspecification
- Overspecification
- Assumption about execution speed
Verification Methods
- Simulations, Prototyping, Testing, Formal Verification

Formal Verification Benefits
- Early integration in design process.
- More effective (higher coverage).
- Reduction in verification time.
- Reduction in development costs and time-to-market.
Press-the-button!

... a brief look at model-checking
ACM Turing Award 2007

- Edmund M. Clarke jr. (CMU, USA)
- Allen E. Emerson (Texas at Austin, USA)
- Joseph Sifakis (IMAG Grenoble, France)

Jury justification

“For their roles in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries.”
Model Checking and State Space Explosion Problem

Requirements

Formalization

Property Specification

System

Modeling

System Model

Simulation

Error

Counterexample

Model Checking

Invalid

Valid

10th PDMC, Snowbird, Utah
Fight the state explosion problem by squeezing all the power out of the contemporary hardware
What is needed ...

Contemporary computing platforms
- Distributed memory systems
- Shared memory parallelism
- GP GPU many-cores
- External memory
- ... and combinations.

Efficient utilization requires
- Parallel algorithms
- Algorithm engineering
The DiVinE Story!

... platform dependent model checking with DiVinE
What is DiVinE?

- Explicit-state automata-based LTL model checker.
- Can exploit aggregate computing power of multiple network-interconnected multi-cored workstations.

Where is it?

http://divine.fi.muni.cz
The DiVinE Story!

... parallel algorithms
Automata-based LTL Model Checking
  • Accepting cycle detection in a directed graph.

Optimal Serial Algorithms
  • Rely on difficult-to-parallelize DFS-postorder.
  • Inefficient on parallel/distributed HW architectures and non-flat memory hierarchies.

Parallel scalable algorithms needed!
# The DiVinE Story: Parallel Scalable Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
<th>Optimality</th>
<th>On-The-Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested DFS</td>
<td>$O(N+M)$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OWCTY Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>general LTL properties</td>
<td>$O(N(N+M))$</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>weak LTL properties</td>
<td>$O(N+M)$</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MAP Algorithm</td>
<td>$O(N.N.(N+M))$</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MAP-OWCTY Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>general LTL properties</td>
<td>$O(N(N+M))$</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>weak LTL properties</td>
<td>$O(N+M)$</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BLEDGE Algorithm</td>
<td>$O(N.N.(N+M))$</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NEGC Algorithm</td>
<td>$O(N.N.(N+M))$</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

$N$ – number of states  
$M$ – number of transitions
## The DiVinE Story: Parallel Scalable Algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Complexity</th>
<th>Optimality</th>
<th>On-The-Fly</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nested DFS</td>
<td>O(N+M)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>OWCTY Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>general LTL properties</td>
<td>O(N.(N+M))</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>weak LTL properties</td>
<td>O(N+M)</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>MAP Algorithm</td>
<td>O(N.N.(N+M))</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>MAP-OWCTY Algorithm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>general LTL properties</td>
<td>O(N.(N+M))</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>weak LTL properties</td>
<td>O(N+M)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BLEDGE Algorithm</td>
<td>O(N.N.(N+M))</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>NEGC Algorithm</td>
<td>O(N.N.(N+M))</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>

N – number of states  
M – number of transitions
Observations

- Distributed memory processing should not stay isolated.
- It must combine with other techniques.

Partial Order Reduction (POR)

- Key technique to fight state space explosion in explicit state model checking.
POR Principle

Orders execution of independent actions among processes of the distributed system.

Distributed system definition

Process A

```
  o  \(\rightarrow\)  o  \(\rightarrow\)  o
  ^   \(a1\)   \(a2\)
```

Process B

```
  o  \(\rightarrow\)  o  \(\rightarrow\)  o  \(\rightarrow\)  o
  ^   \(b1\)   \(b2\)   \(b3\)
```
The DiVinE Story: Partial Order Reduction Principle

POR Principle

Orders execution of independent actions among processes of the distributed system.

Unreduced State Space Graph

![Graph showing POR Principle](image-url)
Suppose actions $a_1$, $b_1$ and $b_2$ are independent.
POR Principle

By postponing execution of selected independent action (e.g. **a₁**) the state space graph is reduced.
The DiVinE Story: Partial Order Reduction Principle

POR Principle

By postponing execution of selected independent action (e.g. a1) the state space graph is reduced.

Reduced State Space Graph
POR Principle

By postponing execution of selected independent action (e.g. \textit{a1}) the state space graph is reduced.

Reduced State Space Graph
The DiVinE Story: Partial Order Reduction Principle

**POR Principle**

Postponed action must be eventually executed, otherwise the reduction is incorrect.

**Unreduced State Space Graph**

![Graph](image-url)
The DiVinE Story: Partial Order Reduction Principle

**POR Principle**

Postponed action must be eventually executed, otherwise the reduction is incorrect.

**Permanent Postponing Problem**

![Diagram of the Permanent Postponing Problem]
Por Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

Permanent Postponing Problem
POR Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

Permanent Postponing Problem
POR Principle

Postponed action must be eventually executed, otherwise the reduction is incorrect.

Permanent Postponing Problem
The DiVinE Story: Partial Order Reduction Principle

**POR Principle**

Infinite postponing is avoided if every cycle contains at least one fully expanded state \(\Rightarrow\) cycle proviso.

Cycle with Fully Expanded State
POR Principle

To detect the cycle, **depth-first search stack** is used in the standard sequential approach.

Cycle with Fully Expanded State
Suppose a directed graph.
First, indegree is computed for every vertex.
Vertices with zero indegree are “eliminated”.
Again, vertices with zero indegree are “eliminated”.

![Diagram of a graph illustrating the topological sort process. Vertices are labeled with numbers, and arrows indicate the direction of edges. The vertices are eliminated as their indegree becomes zero.]
Vertices whose indigree has decreased, are marked and removed.
Repeat until all of the graph is processed.
Repeat until all of the graph is processed.
Repeat until all of the graph is processed.
Repeat until all of the graph is processed.
Repeat until all of the graph is processed.
Repeat until all of the graph is processed.
Repeat until all of the graph is processed.
All cycles are covered.
Generation starts from an initial state.
Initial part of the state space is generated using action-postponing principle.
Topological sort proviso is applied to mark at least one state on every cycle.
Topological sort proviso is applied to mark at least one state on every cycle.
Marked states are fully re-expanded.
New states are discovered outside the previously generated part of the state space.
New part of the reduced state space is generated using action-postponing principle.
Topological sort proviso is applied to mark at least one state on every cycle.
Marked states are fully re-expanded.
New state is discovered outside the previously generated parts of the state space.
Additional part of the reduced state space is generated using action-postponing principle.
Topological sort proviso is applied to mark at least one state on every cycle.
Marked states are fully re-expanded.
No new states are generated, hence the reduced state space has been fully constructed.
The DiVinE Story!

... algorithm engineering
Algorithm engineering

“Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice.”

[Aho et al. [1997], Emerging Opportunities for Theoretical Computer Science]
Algorithm engineering

“Efforts must be made to ensure that promising algorithms discovered by the theory community are implemented, tested and refined to the point where they can be usefully applied in practice.”

[Aho et al. [1997], Emerging Opportunities for Theoretical Computer Science]

In other words ...

... implementation matters.
The DiVinE Story: Message Aggregation

**Principle**

- Individual messages

![Diagram of individual messages]

are sorted according the target and combined

![Diagram of sorted messages]

**Buffers are flushed when**

- explicitly requested by underlying algorithm, or
- maximal number of messages in buffer is reached, or
- node is otherwise idle (empty queue), or
- messages in the buffer are too old.
The DiVinE Story: Message Aggregation

Principle

- Individual messages

![Individual messages diagram]

are sorted according the target and combined

![Sorted messages diagram]

Buffers are flushed when

- explicitly requested by underlying algorithm, or
- maximal number of messages in buffer is reached, or
- node is otherwise idle (empty queue), or
- messages in the buffer are too old. ⇐ expensive and useless
The DiVinE Story: Flushing Buffers

10th PDMC, Snowbird, Utah
Processing states in local queue

Are there new messages?

Receive incoming messages

**Graph Exploration Loop**

- States to be explored locally are taken from a local queue.
- Queue contains locally generated states and states received from network.

**When to process incoming messages?**
The DiVinE Story: Polling for incoming messages

Graph Exploration Loop
- States to be explored locally are taken from a local queue.
- Queue contains locally generated states and states received from network.

When to process incoming messages?
- High rate → CPU load.
- Low rate → increased memory demands.
The DiVinE Story: Performance of OWCTY Algorithm

In cooperation with VU Amsterdam.

<table>
<thead>
<tr>
<th>Cores</th>
<th>Runtime (sec)</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>631.7</td>
<td>100%</td>
</tr>
<tr>
<td>64</td>
<td>13.3</td>
<td>74%</td>
</tr>
<tr>
<td>128</td>
<td>7.4</td>
<td>67%</td>
</tr>
<tr>
<td>256</td>
<td>5.0</td>
<td>49%</td>
</tr>
</tbody>
</table>
The DiVinE Story: Engineering on Other Platforms

**Threading in Shared-Memory**
- Avoid false sharing.
- Efficient memory allocation and deallocation.
- Lock-free communication data structures.

**Shared-Memory Memory Access**
- Memory access optimized hash tables.
- Compact data representation.

**External Memory (disks)**
- Delayed duplicate detection.
- Recompute rather than communicate.

**GP GPU Many Cores**
- Avoid hash-based graph partitioning.
- LTL model checking as matrix-vector multiplication.
What’s Next

... some conclusions
Conclusions

**General observation**
- Gap between pseudo-code and implementation is widening.
- Implementations need to be tuned for individual platforms.
- We should learn to appreciate engineering solutions.

**Explicit state verification future**
- Combination with static analysis and symbolic approaches.
- Increased importance of techniques that trade space for time.
- Platform dependent state space reductions.
The End

... thank you for your attention