DivSPIN
A SPIN-compatible Distributed Model Checker

M. Leucker  Michael Weber  V. Foreijt  J. Barnat

SENVA Meeting
November 2005
Grenoble
DiVinE & SPIN
SPIN
Reverse-Engineered SPIN
Reverse-Engineered SPIN: NIPS
Reverse-Engineered SPIN: NIPS
Never Implement Promela Semantics (again)
State space too large for exhaustive sequential verification

- Distributed Model Checking to the rescue!

Obstacles:

- Possibility to reuse already developed model?
- Available hardware? (Cluster, GRID, ...)
- Maintenance?
- Complex software dependencies: MPI, Java runtime, libraries, compiler(?), ... in non-conflicting versions!
Outline

1. Project Goals
2. DivSPIN Overview
3. NIPS: A Virtual Machine (VM) for PROMELA
4. Current State of Affairs
In a nutshell:

“Provide distributed model checking tool with extremely low access barrier which is attractive to large user base of sequential model checking tools.”

Effortless trial use

- Always up-to-date
- No installation
- No software dependency hell
- No expensive hardware to buy and maintain
DivSPIN Overview

DivSPIN

DiVinE
Library
+
Promela
VM

Cluster
GRID

User

Tool1 Tool2 Tool3

Web Interface

Model+Property

State Gen.

Algorithm

Reporter

Output – Log Files

Storage

Network

HW Monitor
Cluster

Parsecs Cluster at RWTH

26 nodes, each:
- 2x Intel Pentium 500MHz
- 512MB RAM
- 100MBit switched Ethernet

* Dramatization
Modelling Language: Promela

```plaintext
chan ch = [1] of {int};

proctype P (x) {
    do
        :: ch!x;
    od
}

proctype Q (x) {
    do
        :: ch?x;
        :: break;
    od
}
```

- Non-determinism
- Concurrency
- Synchronisation
- Priorities
- Extensions possible (clocks, heap, ...)

Only basic data types (integers, records)
NIPS VM Design

- SPIN not very accessible (to distributed settings, anyway)
- Complete formal specification from scratch [WeberSchürmans05]
  - Generally: least effort for compiler & VM implementor
  - RISC-like instruction set
  - Observable & unobservable steps in the language
  - States: binary objects, directly manipulated by VM
  - Performance: comparable to SPIN
  - Well suited for embedding (unlike SPIN . . .)
NIPS VM Design

Scheduler selects snapshots for further execution
(not part of the VM)
chan c = [1] of {int}
proctype send (int x) {
  c!x
}
init { run send(2) }

Process 1
Global Variable 0 = []
Process 1
Global Variable 0 = []
Local Variable 0 = 2
Global Variable 0 = [2]

GLOBSZ 2
LDC -31
CHNEW 1 1
STVA G 2u 0
STEP N 0
LJMP P_init

P_send: LDVA G 2u 0
TOP r0
CHFREE
NEXZ
PUSH r0
CHADD
PUSH r0
LDVA L 4 0
CHSETO 0
STEP T 0

P_init: LDC -31
LDC 2
LRUN 4 1
NEXZ
STEP T 0
Interesting Instructions

- NDET $\alpha$

Explicit notion of non-determinism
Interesting Instructions

- **NDET** $\alpha$

  Explicit notion of non-determinism

- **NEX**

  Aborts process execution
Interesting Instructions

- **NDET** $a$

Explicit notion of non-determinism

- **NEX**

Aborts process execution

- **STEP** $M'$

Finish multiple internal steps

\[ \Gamma[M] \xrightarrow[\text{step}]{} \Gamma'[M'] \]
NIPS VM Implementation

- $\approx 5k$LOC of C code (including everything)
- State size: order of tens of bytes, within few bytes of SPIN’s states
- No fancy optimizations yet
- Performance: comparable to SPIN
- Well suited for embedding (unlike SPIN . . .)

Largest example so far: LUNAR Routing Protocol
10M states generated, $\approx 6$ GB memory
Memory filled in less than two minutes
(AMD Athlon 64 3500+)
Benchmarking against SPIN

- Breadth-first state space generation (no POR)
- SPIN: generating C, compiled
- VM: compiling to bytecode, interpreted
- AMD Athlon64 3500+ (sequential)

<table>
<thead>
<tr>
<th></th>
<th>era(34)</th>
<th>leader(6,12)</th>
<th>ptrsn_N(3)</th>
<th>pftp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Total states</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIN</td>
<td>713,817</td>
<td>382,151</td>
<td>45,927</td>
<td>1,275,180</td>
</tr>
<tr>
<td>SPIN+Opt</td>
<td>342,082</td>
<td>106,449</td>
<td>25,371</td>
<td>219,167</td>
</tr>
<tr>
<td>NIPS</td>
<td>347,012</td>
<td>382,465</td>
<td>51,118</td>
<td>1,378,184</td>
</tr>
<tr>
<td>NIPS+PR</td>
<td>2,914</td>
<td>6,241</td>
<td>853</td>
<td>301,603</td>
</tr>
</tbody>
</table>

|                  |         |              |            |          |
| **State-space generation time (seconds)** |         |              |            |          |
| Spin+Opt         | 26.441  | 0.12         | 0.085      | 0.770    |
| NIPS+PR          | 0.177   | 0.176        | 0.012      | 4.996    |

M. Weber

DivSPIN

SENV A 2005/11 16
Path Reduction vs. POR

Full Transition System

\[ ab \mid cd \]

\[ a \quad c \]

\[ b \mid cd \quad ab \mid d \]

\[ b \quad c \quad a \quad d \]

\[ cd \quad b \mid d \quad ab \]

\[ c \quad b \quad d \quad a \]

\[ d \quad b \]

\[ d \quad b \]
Path Reduction vs. POR

Full Transition System

```
ab | cd
  a
  c
  bd | ab | cd
  b
  c
  a
  d
  cd | bd | ab
  cd
  b
  d
  a
  d
  b
  b

```

Partial Order Reduction

```
ab | cd
  a
  b | cd
  b
  c
  d
  cd
  d
  b
  d
```

M. Weber
Path Reduction vs. POR

Full Transition System

Partial Order Reduction

Path Reduction

- Static
- Preserves branching structure
Path Reduction vs. POR

Full Transition System

Partial Order Reduction

Path Reduction

PR+POR

- Static
- Preserves branching structure
Path Reduction

(path compression, [Yorav&Grumberg 2004]):
Collapse sequences of steps that cannot influence the value of a temporal logic specification

- Based on static program analysis.
- Identifies program locations which *may affect* the value of the specification.
- Preserves $\text{CTL}^*_{\neg x}$.
- Presented as applied to a simple high-level concurrent language (static processes, synchronous communication).

*Assumption:*
Specification is a formula over program variables.
Path Reduction for NIPS

Versus [Y&G 2004]:

- Adapted to
  - Dynamic creation of channels and processes
  - Asynchronous communication
  - Global variables \(\mapsto\) informal communication channels.
- Two phases:
  - *Transformation* of the NIPS program: preserves the model’s semantics and just alters the transition system.
  - *Transition-system construction*: delegated to the (unmodified) NIPS virtual machine.

Less involved framework (almost too simple...)  
Explicit notion of steps, program transformation  
Hard work: delegate to VM
How to present counter examples?
Currently: in terms of VM states
Need clean reflective interface to VM

How to integrate reduction techniques?
(Partial Order Reduction, \( \tau \)-confluence)
model -> New property -> New task -> tbindfs_dve Running
Extensions: Work In Progress

- More frontends (NTIF, ATMEL instruction set, ...)
- Formalizing VM in ACL2 theorem prover
- State compression
- More (semi-)static analyses (compatible with distributed settings)
  - Dead Variables ($\approx 10\%$ reduction) [Y&G 2004]
    $\implies$ Symmetry Reduction
- ...

M. Weber
DivSPIN goals

- Ready-to-use distributed model checker
- Tap large user base

- Building on DIVINE framework
- Utilizing PROMELA as modelling language
- State-space generation:
  NIPS virtual machine
  Building block!