A GPGPU-based Simulator for Prism: Statistical Verification of Results of PMC (extended abstract)

Marcin Copik   Artur Rataj   Bożena Woźna-Szcześniak

RWTH Aachen University   IITiS PAN   JDU in Częstochowa

Motivation

1. integrate parallel simulator with PRISM
2. utilize capabilities of GPUs for Monte Carlo simulation
3. stochastic corectness verification of probabilistic model checking
4. ongoing research on stochastic methods for solving MDPs

PRISM

Model

Model types for simulation

- discrete-time Markov chain
- continuous-time Markov chain

Model types for simulation

- discrete-time Markov chain
- continuous-time Markov chain
- Markov decision process

Model types for simulation

- discrete-time Markov chain
- continuous-time Markov chain
- Markov decision process

Language:

- set of concurrently executing modules
- state space defined by global and module variables
- execution of modules may be synchronized
- simple arithmetical and logical functions are allowed

Model types for simulation

- discrete-time Markov chain
- continuous-time Markov chain
- Markov decision process

Command syntax

(synchronization label) (guard) & ... & (guard) -> rate : (update) & ... & (update) +

...

where:

- *guard* is a boolean expression in form: $bool = f(a, b, c, ...)$
- *rate* is a floating-point number $float = g(a, b, c, ...)$
- *update* is in form $a' = h(a, b, c, ...)$
- $a, b, c, ...$ are model variables which construct the state vector

PRISM

Model

```
module servers
 Si : [0..n_s] init n_s;
 Sl : [0..n_s] init 0;
 Sb : [0..n_s] init 0;
 [request] (Si>0) & (Cr>0) & (Sl<n_s) ->
  min(Si * r_s , Cr*r_c) : (Si'=Si-1) & (Sl'=Sl+1);
 [log] (Si<n_s) & (Sl> 0) ->
  Sl * r_l : (Si'= Si +1) & (Sl'=Sl-1);
 [break] (Si>0) & (Sb<n_s) ->
  Si * r_b : (Si'= Si -1) & (Sb'=Sb+1);
 [fix] (Sb>0) & (Si<n_s) ->
  Sb * r_f : (Si'=Si+1) & (Sb'=Sb-1);
endmodule
```

PRISM

Model

```
module clients
 Ct : [0..n_c] init n_c;
 Cr : [0..n_c] init 0;
 [request] (Si>0) & (Cr>0) & (Ct<n_c) ->
  (Cr' = Cr-1) & (Ct'=Ct + 1);
 [think] (Ct>0) & (Cr<n_c) ->
  Ct * r_t : (Cr' = Cr + 1) & (Ct' = Ct -
1);

endmodule
```

PRISM

Properties

$$\phi ::= \mathtt{P}_{\sim p}[\psi], \quad \psi ::= \mathtt{X}\phi_1 \mid \mathtt{G}^{\leq k}\phi_1 \mid \mathtt{F}^{\leq k}\phi_1 \mid \phi_1 \mathtt{U}^{\leq k}\phi_1 \mid \phi_1 \mathtt{R}^{\leq k}\phi_1,$$
$$\phi_1 ::= a \mid \phi_1 \wedge \phi_1 \mid \neg\phi_1.$$

- $a \in \mathcal{AP}$ an atomic proposition
- $\sim \in \{<, \leq, \geq, >\}$
- $p \in [0,1]$ a probability bound
- k a non-negative integer or ∞

PRISM

Properties

$$\phi ::= \mathrm{P}_{\sim p}[\psi], \quad \psi ::= \mathrm{X}\phi_1 \mid \mathrm{G}^{\leq k}\phi_1 \mid \mathrm{F}^{\leq k}\phi_1 \mid \phi_1 \mathrm{U}^{\leq k}\phi_1 \mid \phi_1 \mathrm{R}^{\leq k}\phi_1,$$
$$\phi_1 ::= a \mid \phi_1 \wedge \phi_1 \mid \neg\phi_1.$$

- $\mathrm{X}\phi$ true if ϕ satisfied in next state
- $\mathrm{G}^{\leq k}\phi$ true if ϕ holds true in all time steps less or equal to k
- $\mathrm{F}^{\leq k}\phi$ true if ϕ becomes true within k timesteps
- $\phi_1 \mathrm{U}^{\leq k}\phi_2$ true if ϕ_2 satisfied in k timesteps and ϕ_1 holds true until it happens
- $\phi_1 \mathrm{U}^{\leq k}\phi_2$ as above, but also true if ϕ_2 is true in all time steps less or equal to k

PRISM

Properties

$$\phi ::= \mathrm{P}_{\sim p}[\psi], \quad \psi ::= \mathrm{X}\phi_1 \mid \mathrm{G}^I\phi_1 \mid \mathrm{F}^I\phi_1 \mid \phi_1\mathrm{U}^I\phi_1 \mid \phi_1\mathrm{R}^I\phi_1,$$
$$\phi_1 ::= a \mid \phi_1 \wedge \phi_1 \mid \neg\phi_1.$$

- $a \in \mathcal{AP}$ an atomic proposition
- $\sim \in \{<, \leq, \geq, >\}$
- $p \in [0, 1]$ a probability bound
- I an interval of

PRISM

Properties

PRISM

Properties

Models extended with rewards enable defining:

- **transition** rewards
  value depends only on selected transitions
- **state** rewards
  value depends on state and time spent in it

Reward properties provide new operators:

- instanteous
- cumulative
- reward

PRISM

Properties

Simulator

PRISM simulator engine

1. integrated with PRISM
2. serial
3. on-the-fly interpretation of model in Java-based application

Simulator

PRISM simulator engine

1. integrated with PRISM
2. serial
3. on-the-fly interpretation of model in Java-based application

Alternatives (apmc, ymer)

1. parallel, distr
2. standalone applications
3. effectively a separate source-to-source translator

Simulator

PRISM simulator engine

1. integrated with PRISM
2. serial ➔ parallel
3. on-the-fly interpretation of model in Java-based application

Alternatives (apmc, ymer)

1. parallel, distributed
2. standalone applications
3. effectively a separate source-to-source translator

Simulator

PRISM simulator engine

1. integrated with PRISM
2. serial ➔ parallel
3. native implementation of a model

Alternatives (apmc, ymer)

1. parallel, distributed
2. standalone applications
3. effectively a separate source-to-source translator

Simulator

GPUs in simulation

Why?

1. massively parallel coprocessor
2. suitable for data parallellism and independent work items
3. more capable than a regular vector computation - **Single Instruction, Multiple Threads**
4. high bandwidth memory

Simulator

GPUs in simulation

Why?

1. massively parallel coprocessor
2. suitable for data parallellism and independent work items
3. more capable than a regular vector computation - **Single Instruction, Multiple Threads**
4. high bandwidth memory

Why not?

1. branching and diverging code paths are slow
2. small amount of private and fast memory for a single thread
3. very relaxed execution model

Simulator

GPUs in simulation

Simulator

GPUs in simulation

1. industry standard supportingmany types of accelerators and processors
2. vendor independent
3. programming languages: OpenCL SPIR, OpenCL C/C++, SYCL C/C++

Simulator

GPUs in simulation

1. industry standard supportingmany types of accelerators and processors
2. vendor independent
3. programming languages: OpenCL SPIR, **OpenCL C**/C++, SYCL C/C++

Simulator

GPUs in simulation

1. industry standard supportingmany types of accelerators and processors
2. vendor independent
3. programming languages: OpenCL SPIR, **OpenCL C**/C++, SYCL C/C++

OpenCL C

1. C99 with new keywords
2. prohibited are: recursion, function pointers, memory allocation
3. kernels implemented seperately from the main code
4. runtime compilation to device code

Simulator

GPUs in simulation

1. industry standard supportingmany types of accelerators and processors
2. vendor independent
3. programming languages: OpenCL SPIR, **OpenCL C**/C++, SYCL C/C++

OpenCL C

1. C99 with new keywords
2. prohibited are: recursion, function pointers, memory allocation
3. kernels implemented seperately from the main code
4. runtime compilation to device code → **generate kernel code on-the-fly**

Simulator

Architecture

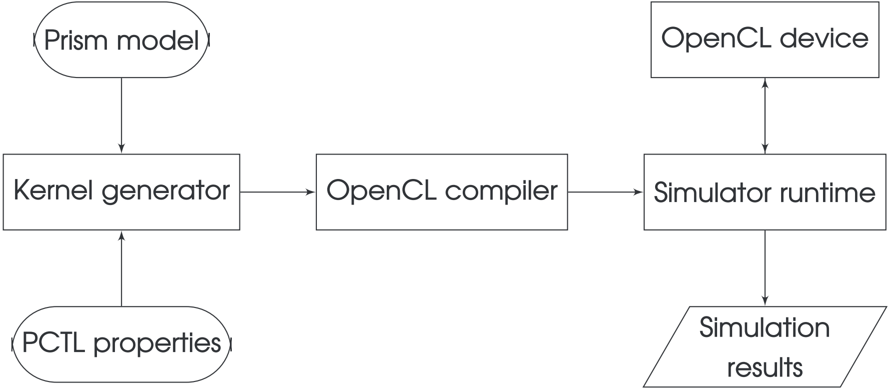

Figure: The OpenCL-based simulator engine in Prism.

Simulator

Architecture

Goals:

1.

Simulator

Architecture

Libraries used

1. *JavaCL* for OpenCL bindings in PRISM's Java code
2. *Random123* as a pseudo-random number generator in OpenCL kernels

Simulator

Stochastic verification

Example

Example

Example

Summary

Summary

Summary

Thanks for your attention

mcopik@gmail.com