GPU-accelerated stochastic simulator engine for PRISM model checker.

Marcin Cópik

January 19, 2014

Plan of presentation

1. Model checking
   - Computer system failures
   - Automatic verification
   - Model checking
2. Methods of model checking
   - Model
   - Properties
3. GPGPU
   - OpenCL
4. Technical aspects
   - Kernel generation
   - Technical details
5. Bibliography

Ariane V

Ariane V

- Ariane 5 missile, made by European Space Agency
- self-destruction 37 seconds after launch on June 4,1996
- uncaught exception: conversion of 64-bit floating point into 16-bit integer
- loss of more than 370 milion USD

Ariane V

- Ariane 5 missile, made by European Space Agency
- self-destruction 37 seconds after launch on June 4,1996
- uncaught exception: conversion of 64-bit floating point into 16-bit integer
- loss of more than 370 milion USD

Ariane V

- Ariane 5 missile, made by European Space Agency
- self-destruction 37 seconds after launch on June 4,1996
- uncaught exception: conversion of 64-bit floating point into 16-bit integer
- loss of more than 370 milion USD

Ariane V

- Ariane 5 missile, made by European Space Agency
- self-destruction 37 seconds after launch on June 4,1996
- uncaught exception: conversion of 64-bit floating point into 16-bit integer
- loss of more than 370 milion USD

Intel Pentium II

- bug in floating-point division unit
- estimation that 1 in 9 billion divides would produce wrong results
- loss of about 475 milion USD
- much more expensive loss: **Intel's reputation**
- could it get even worse...?

Intel Pentium II

- bug in floating-point division unit
- estimation that 1 in 9 billion divides would produce wrong results
- loss of about 475 milion USD
- much more expensive loss: **Intel's reputation**
- could it get even worse...?

Intel Pentium II

- bug in floating-point division unit
- estimation that 1 in 9 billion divides would produce wrong results
- loss of about 475 milion USD
- much more expensive loss: **Intel's reputation**
- could it get even worse...?

Intel Pentium II

- bug in floating-point division unit
- estimation that 1 in 9 billion divides would produce wrong results
- loss of about 475 milion USD
- much more expensive loss: **Intel's reputation**
- could it get even worse...?

Intel Pentium II

- bug in floating-point division unit
- estimation that 1 in 9 billion divides would produce wrong results
- loss of about 475 milion USD
- much more expensive loss: **Intel's reputation**
- could it get even worse...?

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

The London Ambulance Service

- 1500sq kilometeres
- 6.8 milion people
- 5000 patients per day
- 2000-2500 calls per day
- 1000-1200 999 calls per day

- computer aided despatch system, introduced in 1992
- severe failures: incorrectly recorderd vehicle positions, multiple vehicles sent to the same location
- result: **20-30 deaths**

Modern computer systems

The tar pit of software engineering will continue to be sticky for a long time to come. One can expect the human race to continue attempting systems just within or just beyond our reach; and software systems are perhaps the most intricate of man's handiworks. This complex craft will demand our continual development of the discipline, our learning to compose in larger units, our best use of new tools, our best adaptation of proven engineering management methods, liberal application of com- mon sense, and a God-given humility to recognize our fallibility and limitations.

Frederick Brooks, "The Mythical Man-Month"

Modern computer systems

What we need to guarantee?

- corectness, safety, reliability
- performance, resource usage(battery life!)
- security, privacy, anonymity

Modern computer systems

What we need to guarantee?

- corectness, safety, reliability
- performance, resource usage(battery life!)
- security, privacy, anonymity

Modern computer systems

What we need to guarantee?

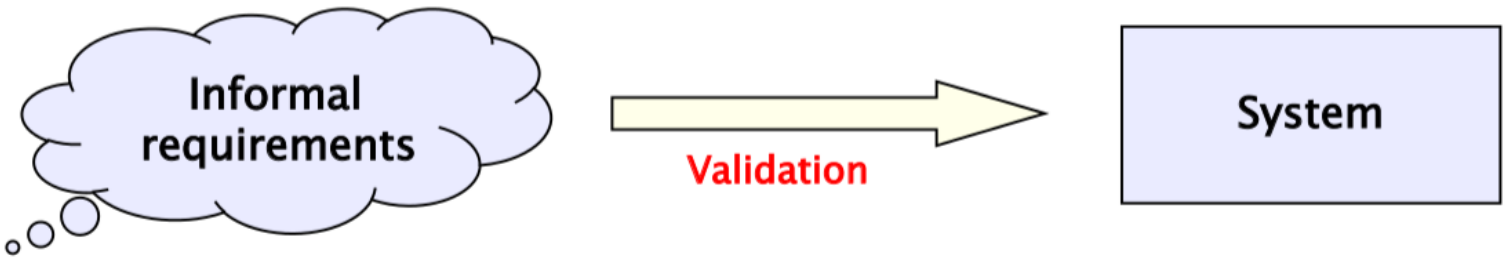
- corectness, safety, reliability
- performance, resource usage(battery life!)
- security, privacy, anonymity

Modern computer systems

What we need to guarantee?

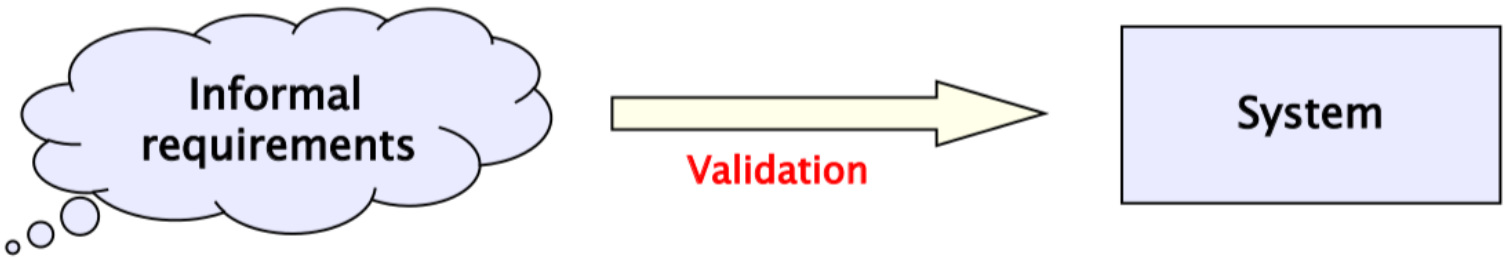
- corectness, safety, reliability
- performance, resource usage(battery life!)
- security, privacy, anonymity

Testing



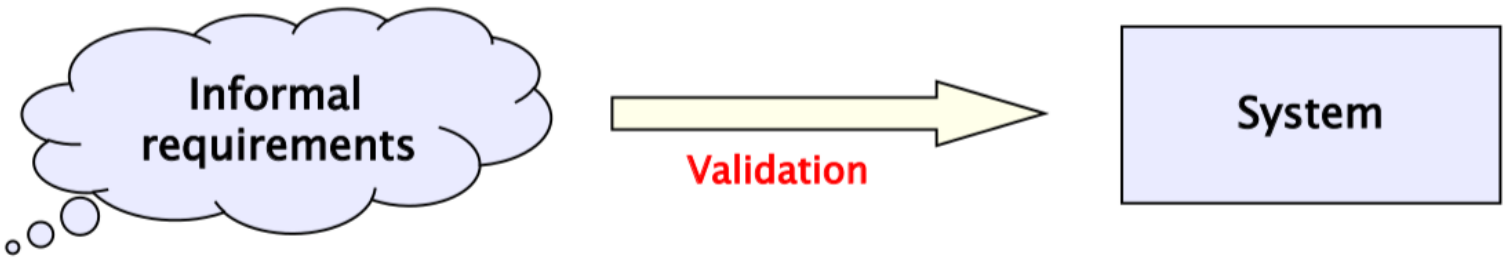
- design methodologies
- validation: testing, code walkthroughs etc.
- "Testing can only show the presence of errors, not their absence."

Testing



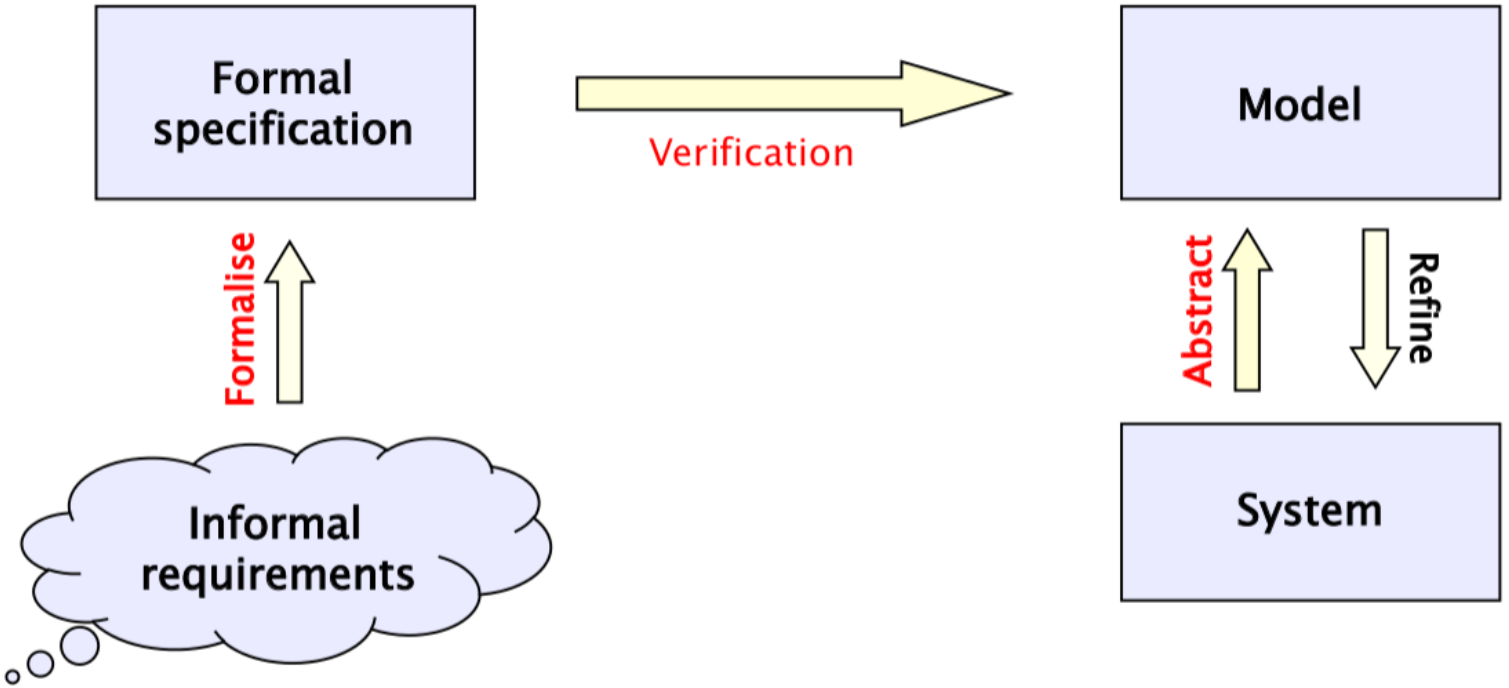
- design methodologies
- validation: testing, code walkthroughs etc.
- "Testing can only show the presence of errors, not their absence."

Testing



- design methodologies
- validation: testing, code walkthroughs etc.
- "Testing can only show the presence of errors, not their absence."

Formal verification



- **formally** verify requirements

Automatic verification

- **Formal verification**: the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems
- **Automatic verification**: formal verification without human intervention

Automatic verification

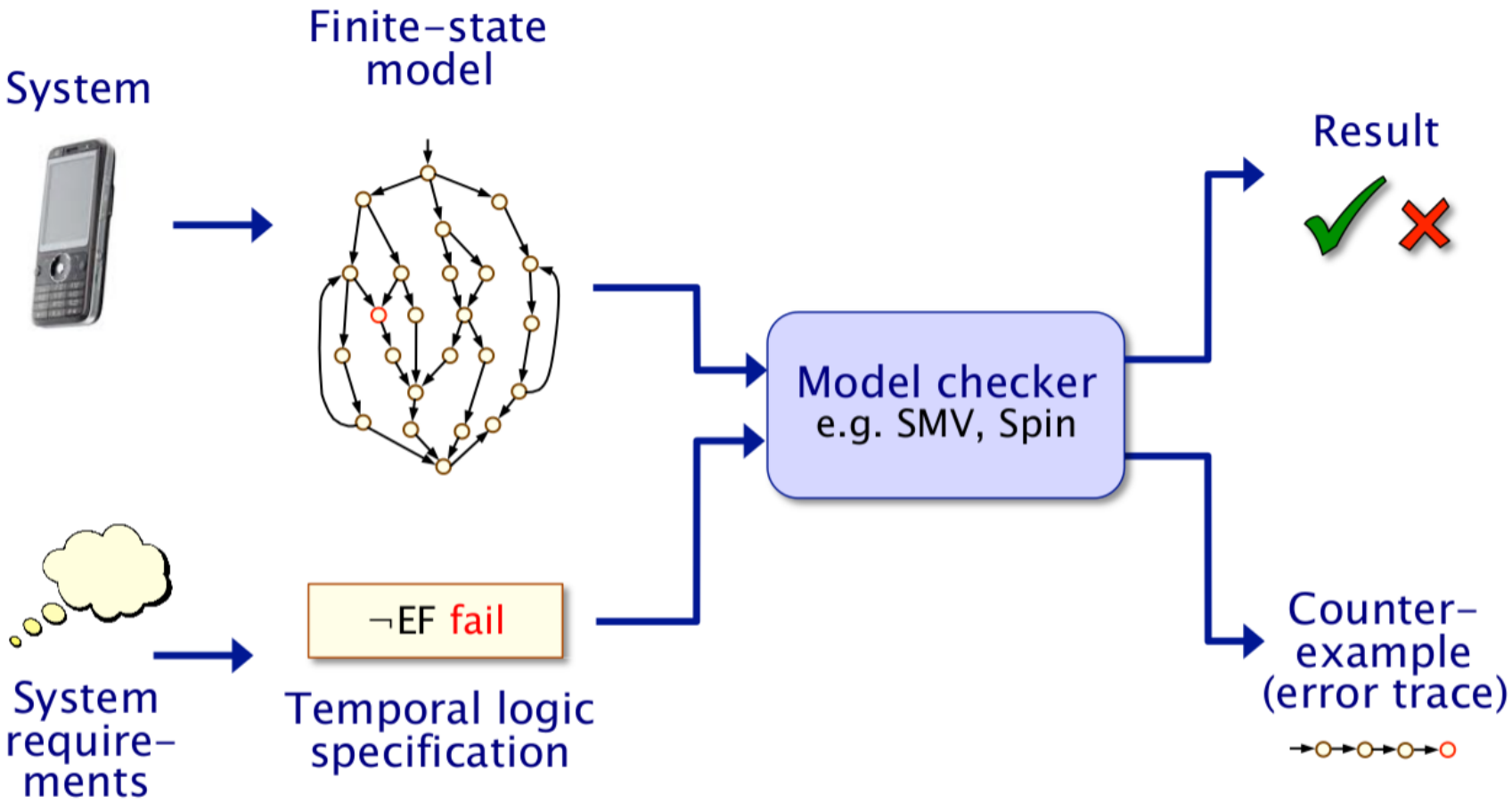
- **Formal verification**: the application of rigorous, mathematics-based techniques to establish the correctness of computerised systems
- **Automatic verification**: formal verification without human intervention

Automatic verification

> *Formal methods should be part of the education of every computer scientist and software engineer, just as the appropriate branch of applied maths is a necessary part of the education of all other engineers.*

Report of an investigation by the FAA and NASA about the use of formal methods

Model checking



Qualitive vs quantitive

Qualitive properties:

- is it possible for system to stay infinitely often in error state?
- every request will eventually be granted - no matter when!
- if a message is sent, will it be delivered within 10 steps?

Quantitive properties:

- how reliable is the system?
- how efficient is my phone's power management policy?

Qualitive vs quantitive

Qualitive properties:

- is it possible for system to stay infinitely often in error state?
- every request will eventually be granted - no matter when!
- if a message is sent, will it be delivered within 10 steps?

Quantitive properties:

- how reliable is the system?
- how efficient is my phone's power management policy?

Qualitive vs quantitive

Qualitive properties:

- is it possible for system to stay infinitely often in error state?
- every request will eventually be granted - no matter when!
- if a message is sent, will it be delivered within 10 steps?

Quantitive properties:

- how reliable is the system?
- how efficient is my phone's power management policy?

Qualitive vs quantitive

Qualitive properties:

- is it possible for system to stay infinitely often in error state?
- every request will eventually be granted - no matter when!
- if a message is sent, will it be delivered within 10 steps?

Quantitive properties:

- how reliable is the system?
- how efficient is my phone's power management policy?

Qualitive vs quantitive

Qualitive properties:

- is it possible for system to stay infinitely often in error state?
- every request will eventually be granted - no matter when!
- if a message is sent, will it be delivered within 10 steps?

Quantitive properties:

- how reliable is the system?
- how efficient is my phone's power management policy?

Qualitive vs quantitive

Qualitive properties:

- is it possible for system to stay infinitely often in error state?
- every request will eventually be granted - no matter when!
- if a message is sent, will it be delivered within 10 steps?

Quantitive properties:

- how reliable is the system?
- how efficient is my phone's power management policy?

Qualitive vs quantitive

Qualitive properties:

- is it possible for system to stay infinitely often in error state?
- every request will eventually be granted - no matter when!
- if a message is sent, will it be delivered within 10 steps?

Quantitive properties:

- how reliable is the system?
- how efficient is my phone's power management policy?

Probabilistic model checking

Why probability?

- modelling uncertainty and performance - eg. rate of failures
- probability equal to 0/1 -¿ express qualitive properties

Some systems are inherently probabilistic

- randomised back-off schemes: 802.3 CSMA/CD, 802.11
- random choice of waiting time: Firewire(root contention)
- random choice over a set of possible addresses: IPv4 Zeroconf dynamic addressing

Probabilistic model checking

Why probability?

- modelling uncertainty and performance - eg. rate of failures
- probability equal to 0/1 -¿ express qualitive properties

Some systems are inherently probabilistic

- randomised back-off schemes: 802.3 CSMA/CD, 802.11
- random choice of waiting time: Firewire(root contention)
- random choice over a set of possible addresses: IPv4 Zeroconf dynamic addressing

Probabilistic model checking

Why probability?

- modelling uncertainty and performance - eg. rate of failures
- probability equal to 0/1 -¿ express qualitive properties

Some systems are inherently probabilistic

- randomised back-off schemes: 802.3 CSMA/CD, 802.11
- random choice of waiting time: Firewire(root contention)
- random choice over a set of possible addresses: IPv4 Zeroconf dynamic addressing

Probabilistic model checking

Why probability?

- modelling uncertainty and performance - eg. rate of failures
- probability equal to 0/1 -¿ express qualitive properties

Some systems are inherently probabilistic

- randomised back-off schemes: 802.3 CSMA/CD, 802.11
- random choice of waiting time: Firewire(root contention)
- random choice over a set of possible addresses: IPv4 Zeroconf dynamic addressing

Probabilistic model checking

Why probability?

- modelling uncertainty and performance - eg. rate of failures
- probability equal to 0/1 -¿ express qualitive properties

Some systems are inherently probabilistic

- randomised back-off schemes: 802.3 CSMA/CD, 802.11
- random choice of waiting time: Firewire(root contention)
- random choice over a set of possible addresses: IPv4 Zeroconf dynamic addressing

Probabilistic model checking

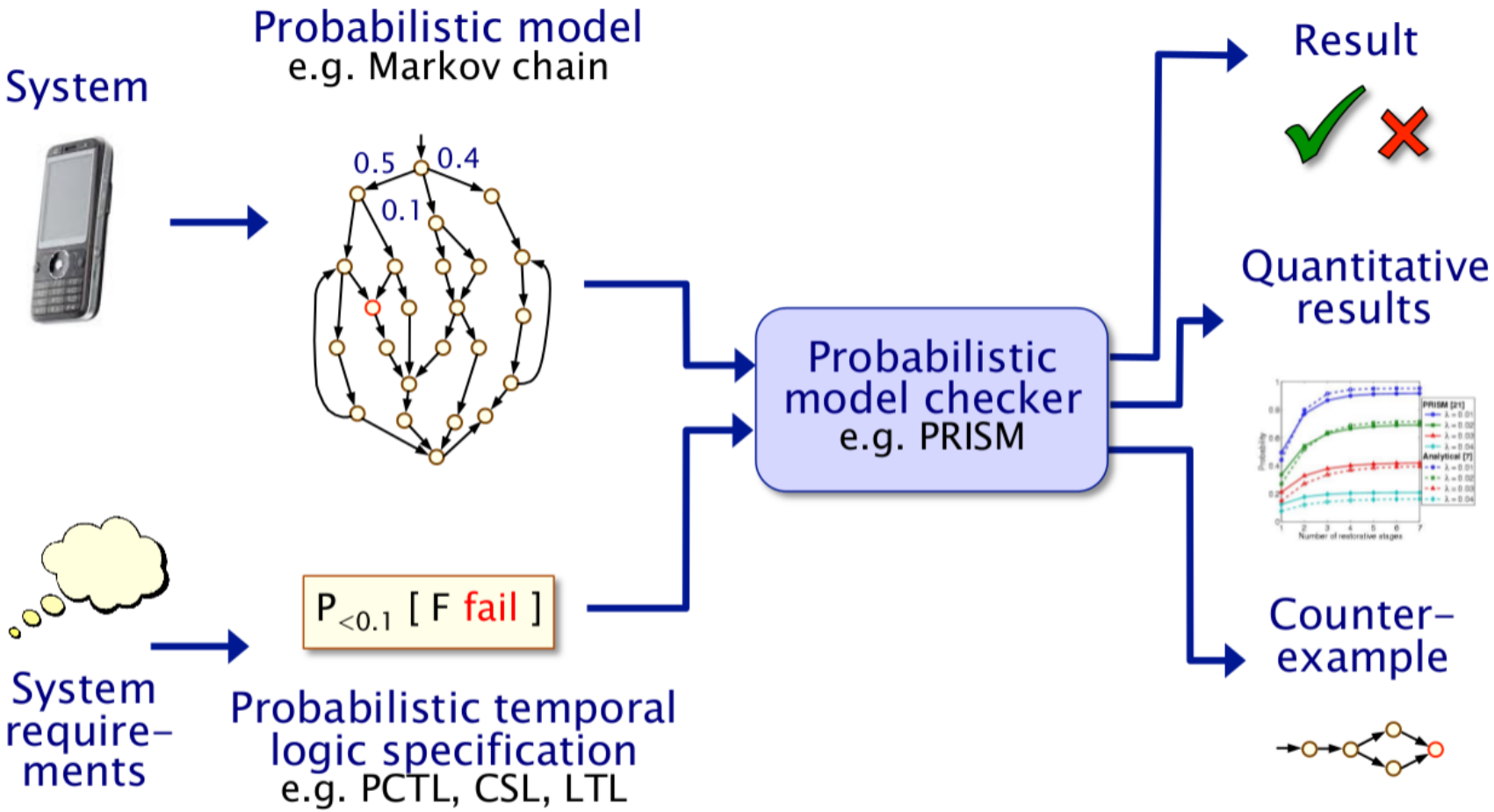
Why probability?

- modelling uncertainty and performance - eg. rate of failures
- probability equal to 0/1 -¿ express qualitive properties

Some systems are inherently probabilistic

- randomised back-off schemes: 802.3 CSMA/CD, 802.11
- random choice of waiting time: Firewire(root contention)
- random choice over a set of possible addresses: IPv4 Zeroconf dynamic addressing

Probabilistic model checking



PRISM model checker

- free, open-source probabilistic model checker
- developed mainly at Oxford University
- supports many different probabilistic models
- **CPU-based sequential simulator**

PRISM model checker

- free, open-source probabilistic model checker
- developed mainly at Oxford University
- supports many different probabilistic models
- **CPU-based sequential simulator**

PRISM model checker

- free, open-source probabilistic model checker
- developed mainly at Oxford University
- supports many different probabilistic models
- CPU-based sequential simulator

PRISM model checker

- free, open-source probabilistic model checker
- developed mainly at Oxford University
- supports many different probabilistic models
- **CPU-based sequential simulator**

Model types

	Fully probabilistic	Nondeterministic
Discrete time	Discrete-time Markov chains (DTMCs)	Markov decision processes (MDPs) (probabilistic automata)
Continuous time	Continuous-time Markov chains (CTMCs)	CTMDPs/IMCs
		Probabilistic timed automata (PTAs)

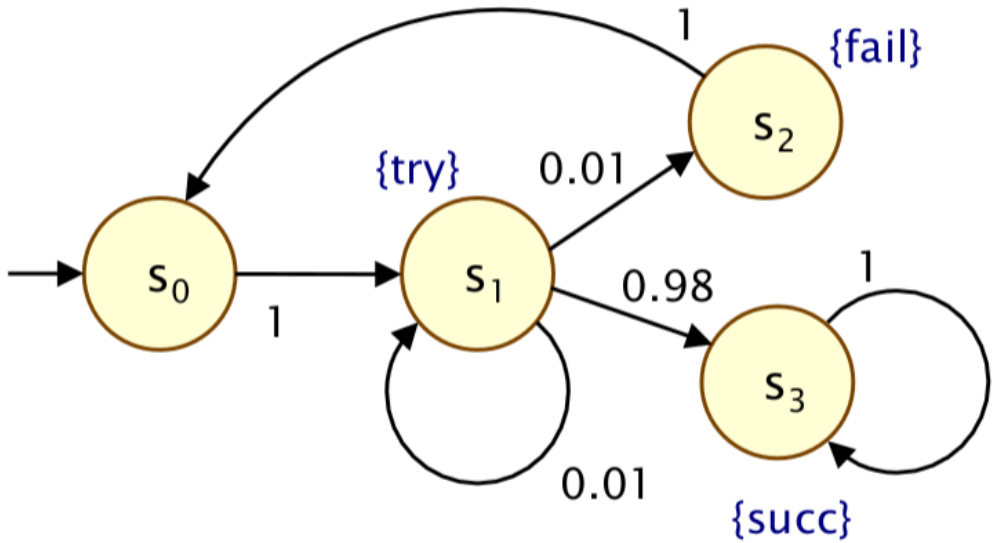
DTMC

$D = (S, s_{init}, \mathbf{P}, L)$

$S = \{s_0, s_1, s_2, s_3\}$
$s_{init} = s_0$

$AP = \{try, fail, succ\}$
$L(s_0) = \varnothing$,
$L(s_1) = \{try\}$,
$L(s_2) = \{fail\}$,
$L(s_3) = \{succ\}$

$$P = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0.01 & 0.01 & 0.98 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

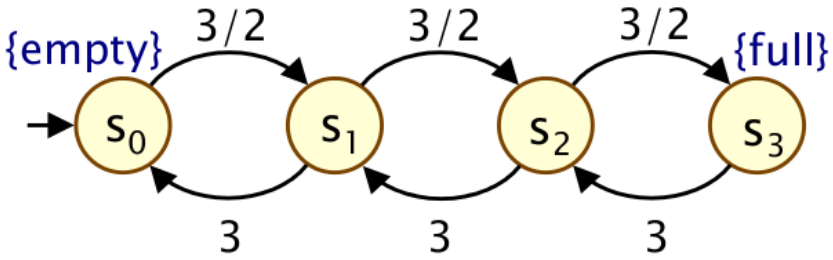


CTMC

$C = (S, s_{init}, \mathbf{R}, L)$

$S = \{s_0, s_1, s_2, s_3\}$

$s_{init} = s_0$



$AP = \{empty, full\}$

$L(s_0)=\{empty\}$, $L(s_1)=L(s_2)=\varnothing$ and $L(s_3)=\{full\}$

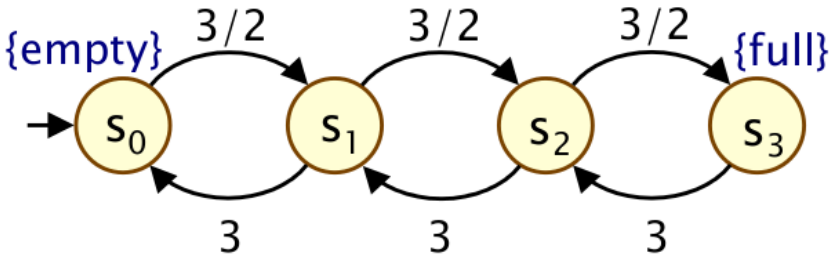
- transition rates instead of probabilities
- continous time delays, exponentially distributed
- supports many different probabilistic models
- race condition

CTMC

$C = (S, s_{init}, \mathbf{R}, L)$

$S = \{s_0, s_1, s_2, s_3\}$

$s_{init} = s_0$



$AP = \{empty, full\}$

$L(s_0)=\{empty\}$, $L(s_1)=L(s_2)=\varnothing$ and $L(s_3)=\{full\}$

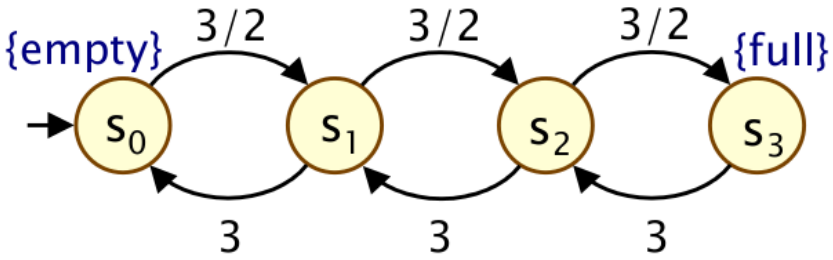
- transition rates instead of probabilities
- continous time delays, exponentially distributed
- supports many different probabilistic models
- race condition

CTMC

$C = (S, s_{init}, \mathbf{R}, L)$

$S = \{s_0, s_1, s_2, s_3\}$

$s_{init} = s_0$



$AP = \{empty, full\}$

$L(s_0)=\{empty\}$, $L(s_1)=L(s_2)=\varnothing$ and $L(s_3)=\{full\}$

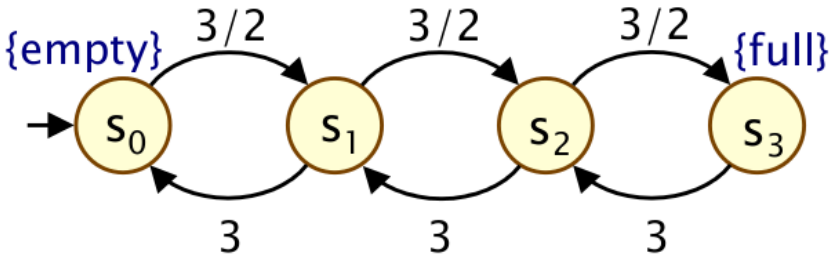
- transition rates instead of probabilities
- continous time delays, exponentially distributed
- supports many different probabilistic models
- race condition

CTMC

$C = (S, s_{init}, \mathbf{R}, L)$

$S = \{s_0, s_1, s_2, s_3\}$

$s_{init} = s_0$



$AP = \{empty, full\}$

$L(s_0)=\{empty\}$, $L(s_1)=L(s_2)=\varnothing$ and $L(s_3)=\{full\}$

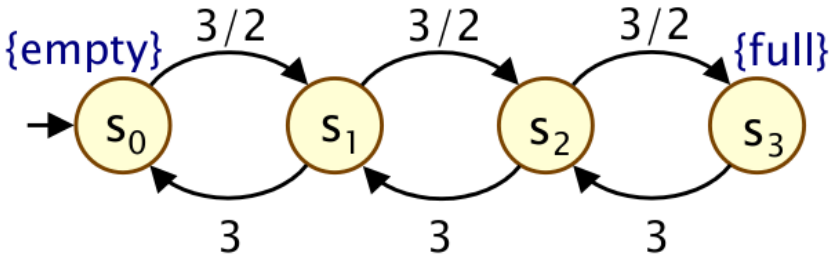
- transition rates instead of probabilities
- continous time delays, exponentially distributed
- supports many different probabilistic models
- race condition

CTMC

$C = (S, s_{init}, \mathbf{R}, L)$

$S = \{s_0, s_1, s_2, s_3\}$

$s_{init} = s_0$



$AP = \{empty, full\}$

$L(s_0)=\{empty\}$, $L(s_1)=L(s_2)=\varnothing$ and $L(s_3)=\{full\}$

- transition rates instead of probabilities
- continous time delays, exponentially distributed
- supports many different probabilistic models
- race condition

Property languages

CTL LTL	Φ ψ	non–probabilistic (e.g. LTSs)
PCTL LTL + prob. PCTL*	Φ Prob(s, ψ) Φ	probabilistic (e.g. DTMCs)

Temporal logic: formal language for specifying and reasoning about how the behaviour of a system changes over time

PCTL

- $P < 0.05\,[F\ \frac{err}{total} > 0.1]$
  "with probability at most 0.05, more than 10 percent of the NAND gate outputs are erroneous?"
- $P >= 0.8\,[F <= k\ replyCount = n]$
  "the probability that the sender has received n acknowledgements within k clock-ticks is at least 0.8"
- $P < 0.4\,[!fail_A\ U\ fail_B]$
  "the probability that component B fails before component A is less than 0.4"
- $P >= 1\,[F\ (P > 0.99\,[G <= 100\ oper])]$
  "the system almost surely reaches a state from which it has a greater than 0.99 chance of staying operational for 100 time units"

PCTL

- $P < 0.05\,[F\ \frac{err}{total} > 0.1]$
  "with probability at most 0.05, more than 10 percent of the NAND gate outputs are erroneous?"
- $P >= 0.8\,[F <= k\ replyCount = n]$
  "the probability that the sender has received n acknowledgements within k clock-ticks is at least 0.8"
- $P < 0.4\,[!fail_A\ U\ fail_B]$
  "the probability that component B fails before component A is less than 0.4"
- $P >= 1\,[F\ (P > 0.99\,[G <= 100\ oper])]$
  "the system almost surely reaches a state from which it has a greater than 0.99 chance of staying operational for 100 time units"

PCTL

- $P < 0.05\,[F\ \frac{err}{total} > 0.1]$
  "with probability at most 0.05, more than 10 percent of the NAND gate outputs are erroneous?"
- $P >= 0.8\,[F <= k\ replyCount = n]$
  "the probability that the sender has received n acknowledgements within k clock-ticks is at least 0.8"
- $P < 0.4\,[!fail_A\ U\ fail_B]$
  "the probability that component B fails before component A is less than 0.4"
- $P >= 1\,[F\,(P > 0.99\,[G <= 100\ oper])]$
  "the system almost surely reaches a state from which it has a greater than 0.99 chance of staying operational for 100 time units"

PCTL

- $P < 0.05\,[F\ \frac{err}{total} > 0.1]$
  "with probability at most 0.05, more than 10 percent of the NAND gate outputs are erroneous?"
- $P >= 0.8\,[F <= k\ replyCount = n]$
  "the probability that the sender has received n acknowledgements within k clock-ticks is at least 0.8"
- $P < 0.4\,[!fail_A\ U\ fail_B]$
  "the probability that component B fails before component A is less than 0.4"
- $P >= 1\,[F\,(P > 0.99\,[G <= 100\ oper])]$
  "the system almost surely reaches a state from which it has a greater than 0.99 chance of staying operational for 100 time units"

Stochastic model checking

Monte-Carlo experiment

- compute number of paths through model
- generate random simulation paths
- approximate PCTL results

Stochastic model checking

Monte-Carlo experiment

- compute number of paths through model
- generate random simulation paths
- approximate PCTL results

Stochastic model checking

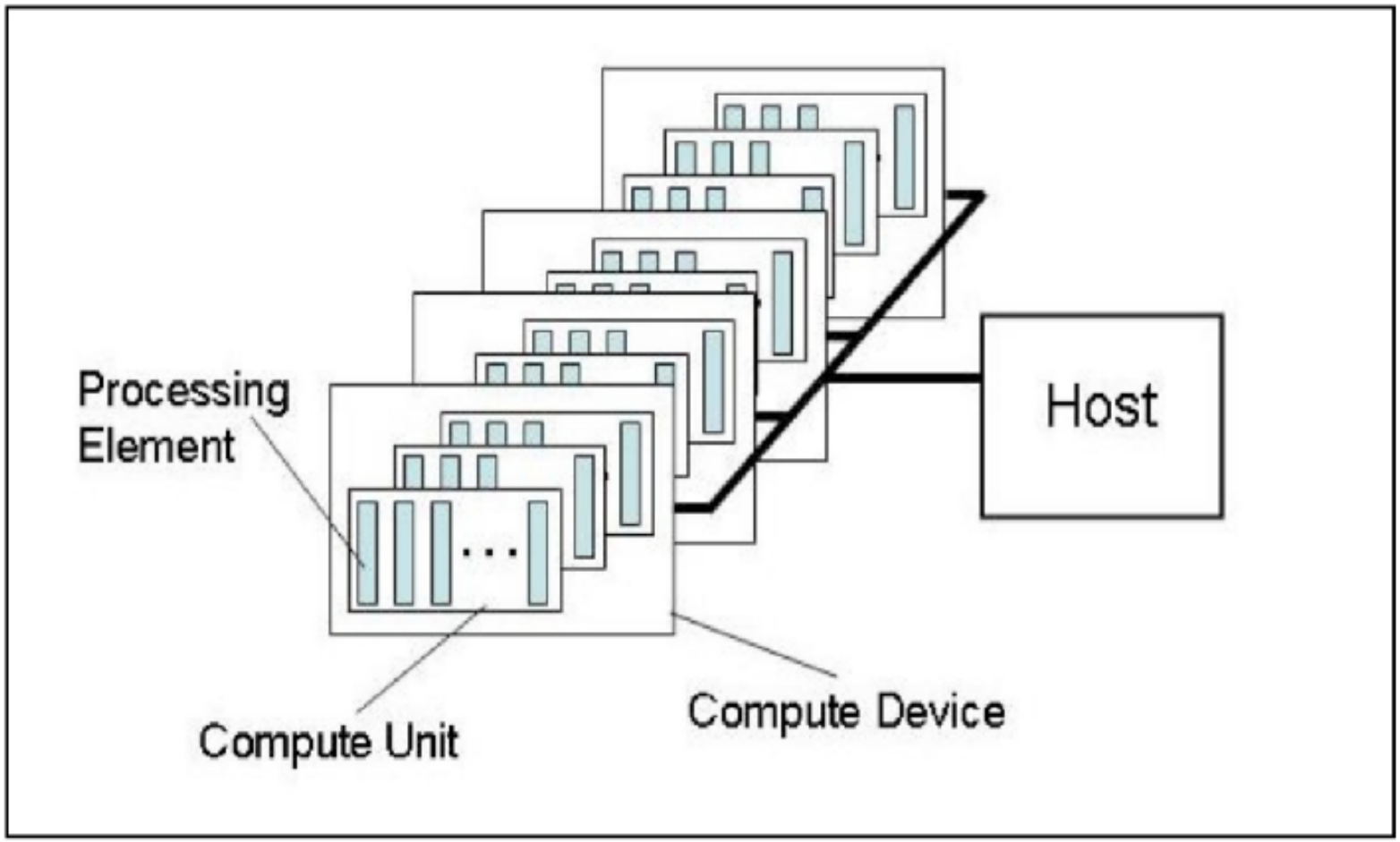
Monte-Carlo experiment

- compute number of paths through model
- generate random simulation paths
- approximate PCTL results

OpenCL


KHRONOS GROUP
OpenCL
3DLABS
ACTIVISION
BLIZZARD
AMD
ARM
BROADCOM
EA
codeplay
ERICSSON
freescale
IBM
intel
Imagination
MOTOROLA
movidia
NOKIA
NVIDIA
Petapath
QNX
RAPID
S3 GRAPHICS
SAMSUNG
Seaweed SYSTEMS
TAKUMI
TEXAS INSTRUMENTS

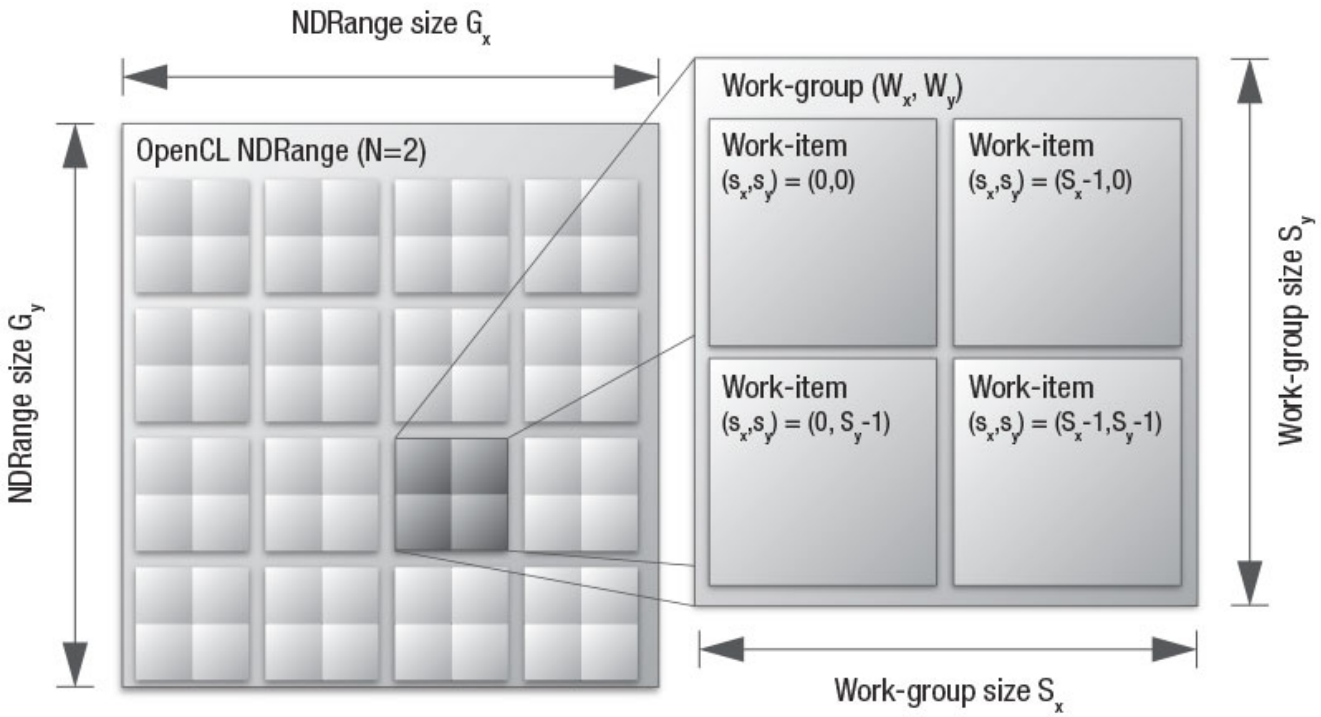
Device model



Differences with CUDA

CUDA term	OpenCL term
GPU	Device
Multiprocessor	Compute Unit
Scalar core	Processing element
Global memory	Global memory
Shared (per-block) memory	Local memory
Local memory (automatic, or local)	Private memory
kernel	program
block	work-group
thread	work item

Work-group and work-items



PRISM language

```
dtmc

module die

// local state
s : [0..7] init 0;
// value of the die
d : [0..6] init 0;

[] s=0 -> 0.5 : (s'=1) + 0.5 : (s'=2);
[] s=1 -> 0.5 : (s'=3) + 0.5 : (s'=4);
[] s=2 -> 0.5 : (s'=5) + 0.5 : (s'=6);
[] s=3 -> 0.5 : (s'=1) + 0.5 : (s'=7) & (d'=1);
[] s=4 -> 0.5 : (s'=7) & (d'=2) + 0.5 : (s'=7) & (d'=3);
[] s=5 -> 0.5 : (s'=7) & (d'=4) + 0.5 : (s'=7) & (d'=5);
[] s=6 -> 0.5 : (s'=2) + 0.5 : (s'=7) & (d'=6);
```

PRISM language

```
ctmc
const int q_max = 20;
const double rate_arrive = 1/0.72;

module SQ

q : [0..q_max] init 0;
// A request arrives
[request] true -> rate_arrive : (q'=min(q+1,q_max));
// A request is served
[serve] q>1 -> (q'=q-1);
// Last request is served
[serve_last] q=1 -> (q'=q-1);

endmodule
```

PRISM language

```
// Rate of service (average service time = 0.008s)
const double rate_serve = 1/0.008;
module SP

// Power state of SP: 0=sleep, 1=idle, 2=busy
sp : [0..2] init 1;
// Synchronise with service queue (SQ):
[request] sp=1  -> (sp'=2);
[request] sp!=1 -> (sp'=sp);

// Serve a request from the queue
[serve]      sp=2 -> rate_serve : (sp'=2);
[serve_last] sp=2 -> rate_serve : (sp'=1);
endmodule
```

PRISM language

Potential problems

- synchronization between modules
- choosing between modules(probability,race condition)
- multiple initial states
- formulas
- costs and rewards

PRISM language

Potential problems

- synchronization between modules
- choosing between modules(probability,race condition)
- multiple initial states
- formulas
- costs and rewards

PRISM language

Potential problems

- synchronization between modules
- choosing between modules(probability,race condition)
- multiple initial states
- formulas
- costs and rewards

PRISM language

Potential problems

- synchronization between modules
- choosing between modules(probability,race condition)
- multiple initial states
- formulas
- costs and rewards

PRISM language

Potential problems

- synchronization between modules
- choosing between modules(probability,race condition)
- multiple initial states
- formulas
- costs and rewards

Technology

- Java, JUnit, Swing

Technology

- Java, JUnit, Swing
- OpenCL, JavaCL

Technology

- Java, JUnit, Swing
- OpenCL, JavaCL
- Eclipse IDE

Development

- refactor and clean existing simulator

Development

- refactor and clean existing simulator
- implement necessary data structures

Development

- refactor and clean existing simulator
- implement necessary data structures
- kernel generation - DTMC

Development

- refactor and clean existing simulator
- implement necessary data structures
- kernel generation - DTMC
- kernel generation - CTMC

Development

- refactor and clean existing simulator
- implement necessary data structures
- kernel generation - DTMC
- kernel generation - CTMC
- PCTL properties

Development

- refactor and clean existing simulator
- implement necessary data structures
- kernel generation - DTMC
- kernel generation - CTMC
- PCTL properties
- efficiency improvements

Development

- refactor and clean existing simulator
- implement necessary data structures
- kernel generation - DTMC
- kernel generation - CTMC
- PCTL properties
- efficiency improvements
- costs and rewards

Development

- refactor and clean existing simulator
- implement necessary data structures
- kernel generation - DTMC
- kernel generation - CTMC
- PCTL properties
- efficiency improvements
- costs and rewards

PRISM supervisor: Dave Parker, University of Birmingham/Oxford

Books

- 'Statistical Approaches for Probabilistic Model Checking' - Vincent Nimal MSc Thesis, University of Oxford
- 'Principles of model checking' - Christel Baier, Joost-Pieter Katoen
- 'Stochastic Model Checking' -Marta wiatkowska, Gethin Norman, Dave Parker