Computational Science of Computer Systems

Martin Quinson

(with the SimGrid Team and others)

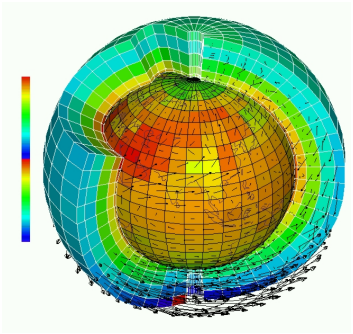
June 16th 2014

Telecom SudParis

What is Science anyway?

Doing Science = Acquiring Knowledge

$$\frac{\partial}{\partial x_j}\left(\frac{\partial \Phi}{\partial x_i}\right) = \frac{\partial}{\partial x_i}\left(\frac{\partial \Phi}{\partial x_j}\right)$$



Experimental Science

- Thousand years ago
- Observations-based
- Can describe
- Prediction tedious

Theoretical Science

- Last few centuries
- Equations-based
- Can understand
- Prediction long

Computational Science

- Nowadays
- Compute-intensive
- Can simulate
- Prediction easier

Prediction is very difficult, especially about the future. – *Niels Bohr*

Observation still base Science

Space telescope

Large Hadron Collider

Mars Explorer

NMR Spectroscope

Synchrotrons

Turntable

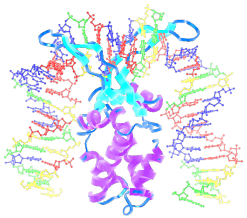
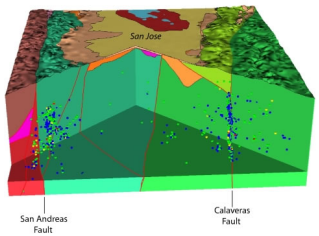
Tsunamis

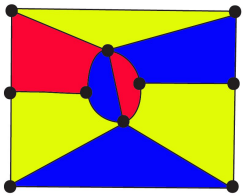
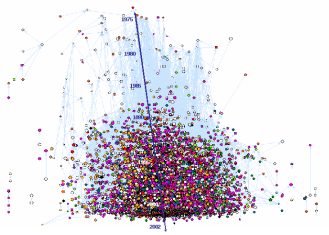
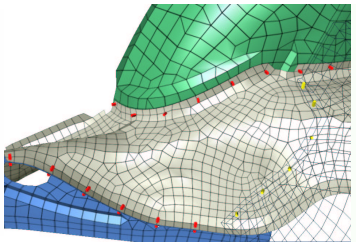
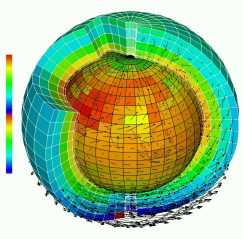
Earthquake vs. Bridge

Climate vs. Ecosystems

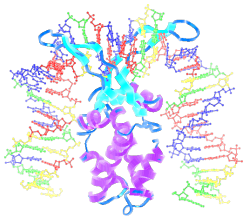
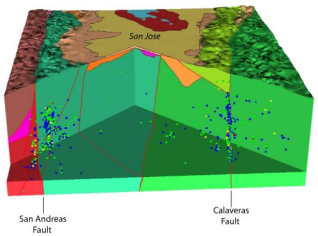
(who said that science is not fun??)

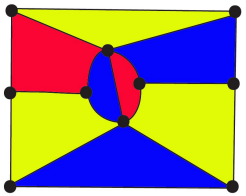
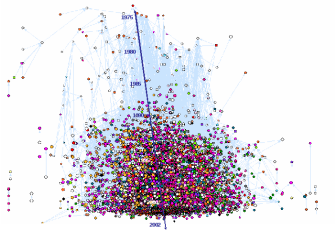
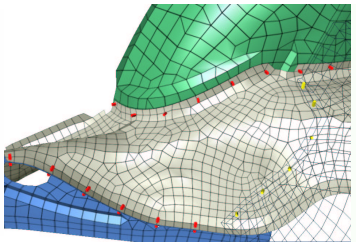
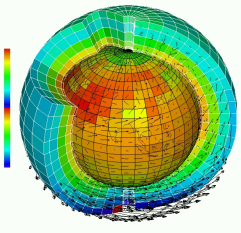
Computational Science

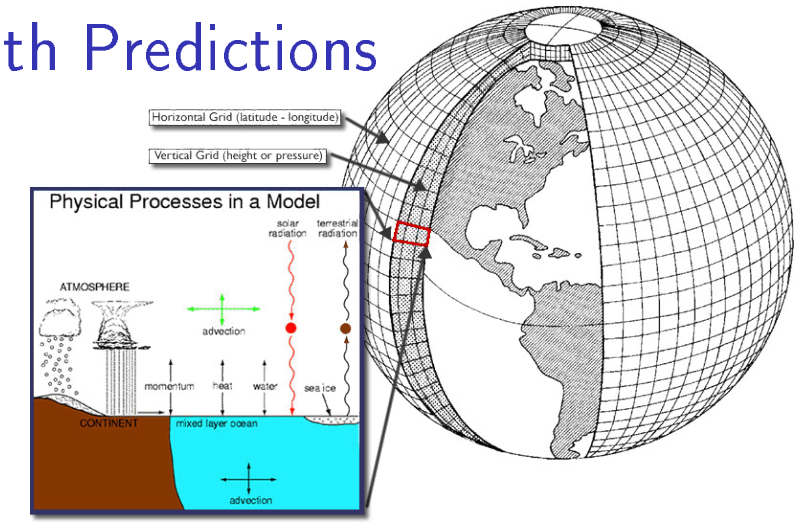
Computational Science

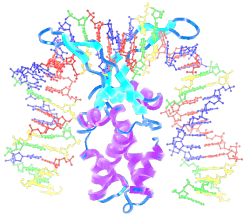
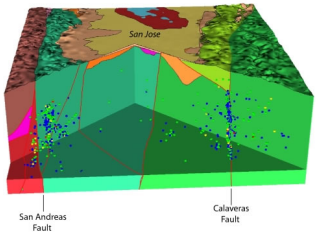





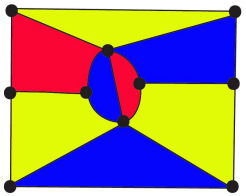
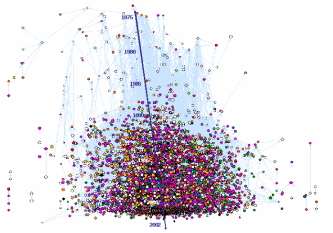
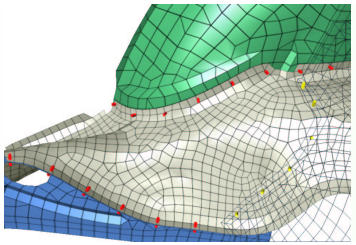
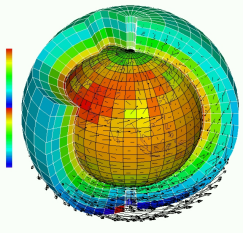
Understanding the Climate Change with Predictions





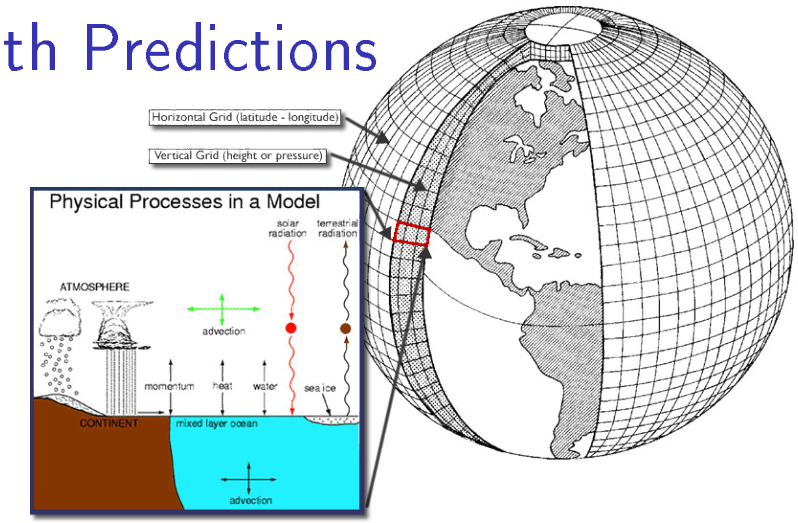
Computational Science

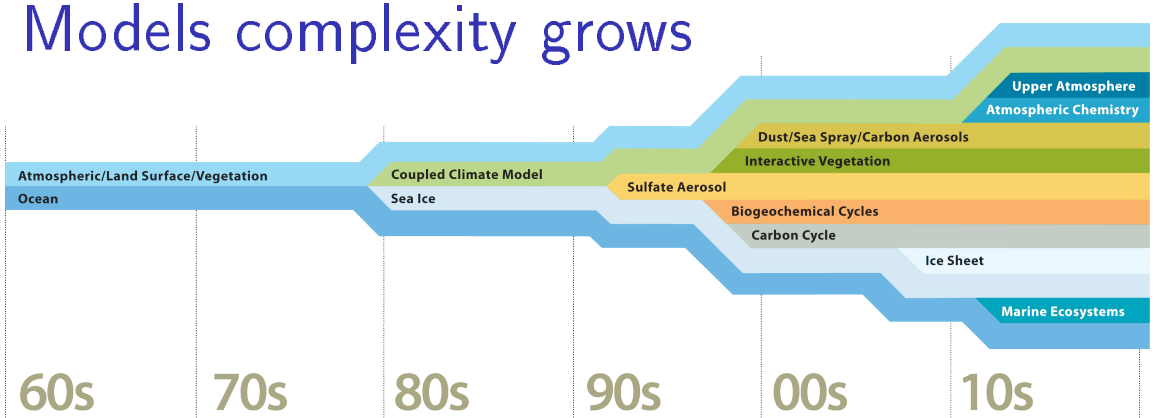





Understanding the Climate Change with Predictions





Models complexity grows



This requires **large** computers

Upscale project:
15,000 computing-years in 2012!

Modern Computers are Large and Complex

Massive Parallelism

- Cannot miniaturize further (atom limit)
- Cannot increase frequency (energy limit)
- Solution: Multiply compute cores!
- Sequoia, second fastest computer: 1,572,864 cores

ExaScale Systems, used in Computational Science

- Systems computing 1 Exaflop per second arrive (with *billions* of cores)
- 1 Exaflop = 10^{18} operations. One million million million operations...
- At humanly doable speed, that requires 10 times the age of the universe
- Each node: 20 millions lines of code (10× Encyclopedia Britannica)

Other very large computer systems in the wide

- Google computers dissipate 300MW on average (150,000 households, $\frac{1}{3}$ reactor)
- Botnets: BredoLab estimated to control 30 millions of zombie computers
- In addition, these systems are heterogeneous and dynamic

Computational Science of Computer Systems

My Research Field: **Methodologies of Experimentation**

- Assessing the performance and correctness of large-scale computer systems
- Meta-research on producing scientifically sound results
- Main contribution: SimGrid, a large-scale computer systems simulator

First title (rejected)

Simulating Applications for Research in
Simulation Applications for Research

Epistemological Stance

- Empirically consider large-scale computer systems as natural objects
- Eminently artificial artifacts, but complexity reaches "natural" levels
- Other sciences routinely use computers to understand complex systems

Computational Science of Computer Systems

My Research Field: Methodologies of Experimentation

- Assessing the performance and correctness of **large-scale computer systems**
- Meta-research on **producing scientifically sound results**
- Main contribution: **SimGrid, a large-scale computer systems simulator**

First title (rejected)

Simulating Applications for **Research** in
Simulation Applications for Research

Epistemological Stance

- Empirically consider large-scale computer systems as natural objects
- Eminently artificial artifacts, but complexity reaches "natural" levels
- Other sciences routinely use computers to understand complex systems

Assessing Distributed Applications

Correctness Study $\rightsquigarrow$ Formal Methods

- Tests: Unable to provide definitive answers

Performance Study $\rightsquigarrow$ Experimentation

- Maths: Often not sufficient to fully understand these systems

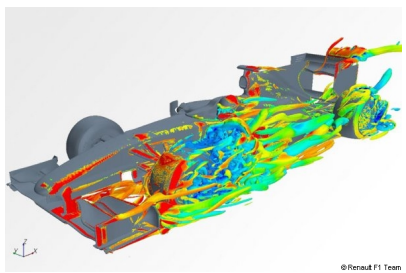
Assessing Distributed Applications

Correctness Study $\rightsquigarrow$ Formal Methods

- Tests: Unable to provide definitive answers
- Model-Checking: Exhaustive and automated exploration of state space

Performance Study $\rightsquigarrow$ Experimentation

- Maths: Often not sufficient to fully understand these systems



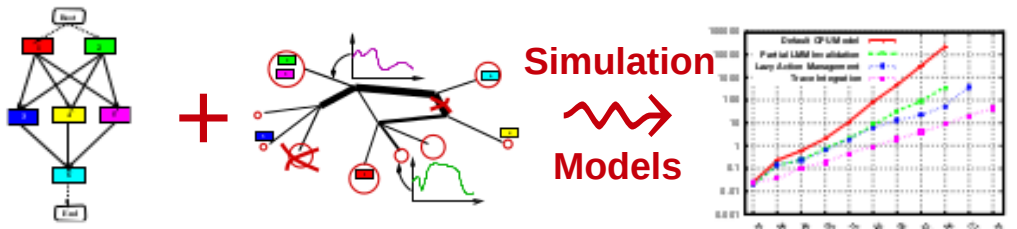
Courtesy of Lucas Nussbaum

- Experimental Facilities: Real applications on Real platform *(in vivo)*
- Emulation: Real applications on Synthetic platforms *(in vitro)*
- Simulation: Prototypes of applications on system's Models *(in silico)*

Simulating Distributed Systems

Simulation: fastest path from idea to data

- Get preliminary results from partial implementations
- Experimental campaign with thousands of runs within the week
- Test your scientific idea, don't fiddle with technical subtleties (yet)

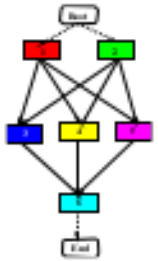


Idea or MPI code | Experimental Setup | Scientific Results

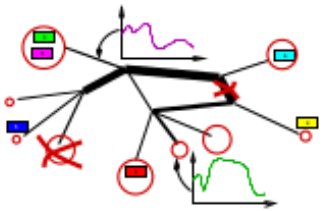
Simulation: easiest way to study distributed applications

- Everything is actually centralized: partially mock parts of your protocol
- No heisenbug: (simulated) time does not change when you capture more data
- Clairevoyance: observe every bits of your application and platform
- High Reproducibility: No variability, but in the emulated computations

Simulation Challenges



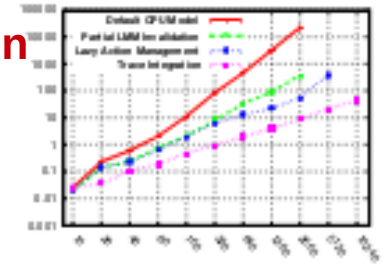
Idea or
MPI code



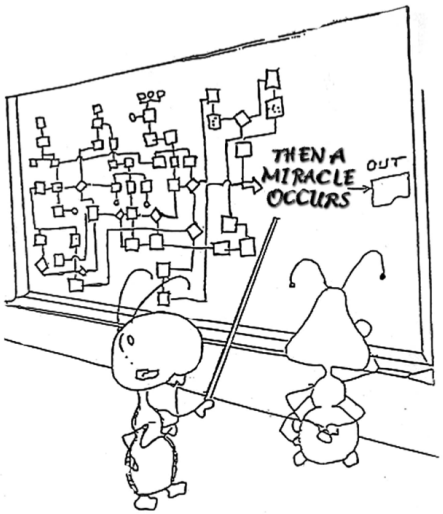
Experimental
Setup

Simulation

Models



Scientific Results



Challenges for the tool makers

- Validity: Get realistic results (controlled experimental bias). That's hard:
  - Resources: CPU and Network, maybe disks and others
  - Usage model: Predict ending time of each task in isolation
  - Contention model: Predicts how tasks interfere with each others
- Scalability: *Fast enough* and *Big enough*
- Tooling: runner, post-processing

Scientific practices sometimes unfortunate in our field

- Experimental settings not detailed enough in literature
- Many short-lived simulators; few sound and established tools

SimGrid: Versatile Simulator of Distributed Apps

Scientific Instrument

- Versatile: Grid, P2P, HPC, Volunteer Computing and others
- Sound: Validated, Scalable, Usable; Modular; Portable
- Community-driven: 30 contributors (5 not affiliated), 5 contributed tools, GPL

Scientific Object

- Allows comparison of network models on non-trivial applications
- High-Performance Simulation on realistic workload
- Full model checker of distributed applications; Emulator under way

Large Established Project

- Impact: 120 publications (110 distinct authors, 5 continents), 4 PhD
- Started in 1998 at UCSD; Now collab accross many individuals and institutions
- Funded by ANR and Inria (≈3M€), also CNRS and universities
- M. Quinson, A. Legrand, F. Suter, A. Giersch
  U. Lorraine, CNRS Grenoble, CNRS IN2P3, U. Franche Comté

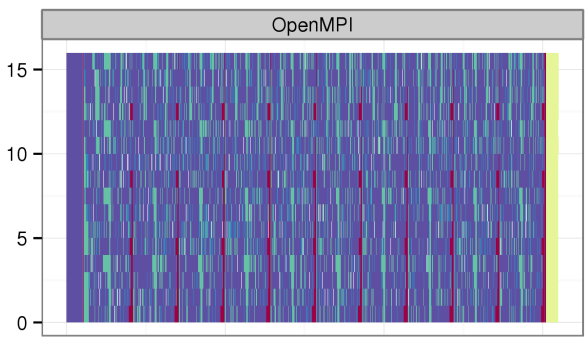
Simulation Validity

SotA: Models in most simulators are either simplistic, wrong or not assessed

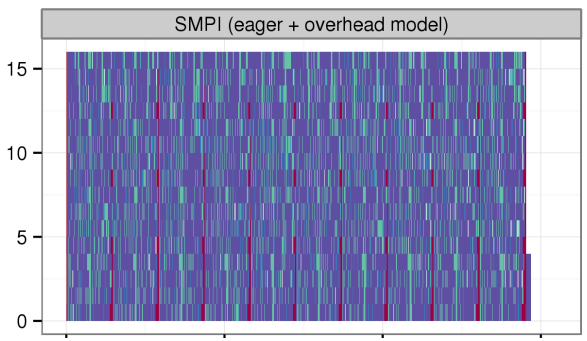
- PeerSim: discrete time, application as automaton;
- GridSim/CloudSim: naive packet level or buggy flow sharing
- OptorSim, GroudSim: documented as wrong on heterogeneous platforms
- Dimemas: aim at performance trends and bottleneck identification

SimGrid provides several Network Models

- Flow-based: Contention, Slow-start, TCP congestion, Cross-traffic effects
- Constant time: A bit faster, but no hope of realism
- Coordinate-based: Easier to instantiate in P2P scenarios
- Packet-level: NS3 bindings



Real Sweep3D



Simulated Sweep3D

The Many Faces of the SimGrid Project

SimGrid is feature-rich, we cannot present it all now

Hybrid Network Models

- Fluid model: model contention in steady state for large messages
- LogOP model: model intra-node delays and synchronization
- Also: MPI collectives, TCP (slow-start, cross-traffic), soon IB

Realistic Emulation

- SMPI: Study real MPI applications within SimGrid
- Simterpose: Study real arbitrary applications (ongoing)

High Performance Simulation

- Fast Enough: Innovative PDES; Efficient algorithms and implementations
- Big Enough: Scalable and versatile platform representation

Formal Verification of Distributed Apps

- Safety properties with DPOR, or Liveness properties with state equality

Agenda

- Introduction
- Computational Science of Computer Systems (CS^2)
- The SimGrid Project
- Emulation with SMPI
- Parallel Simulation of Discrete Event Systems
- Dynamic Verification of Distributed Applications
- Conclusion on SimGrid
- Scientific Outreach

What is SMPI?

- Reimplementation of MPI on top of SimGrid
- Imagine a VM running real MPI applications on platforms that does not exist
  - Horrible over-simplification, but you get the idea
- Computations run for real on your laptop, Communications are faked

What is it good for?

- Performance Prediction ("what-if?" scenarios)
  - Platform dimensioning; Apps' parameter tuning
- Teaching parallel programming and HPC
  - Reduced technical burden
  - No need for real hardware, or hack your hardware

Studies that you should NOT attempt with SMPI

- Predict the impact of L2 caches' size on your code
- Interactions of TCP Reno vs. TCP Vegas vs. UDP
- Claiming a simulation of 1000 billions nodes

Features and Limitations

Features

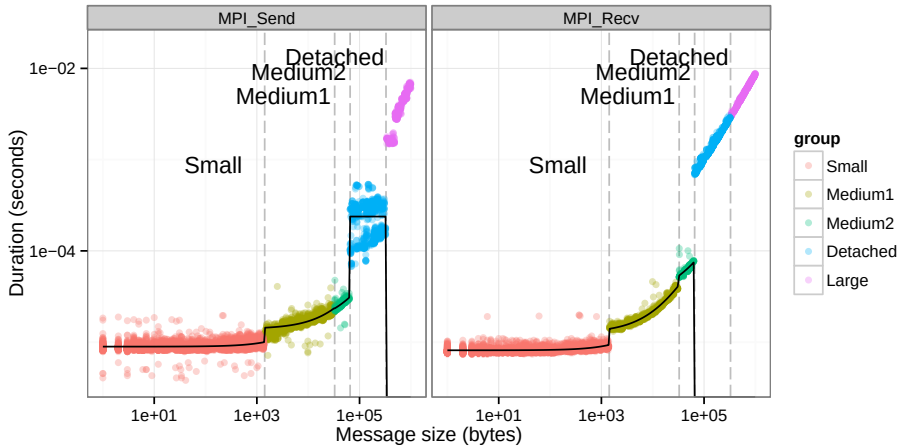
- Complex C/C++/F77/F90 applications can run unmodified out of the box
  - MPI ranks folded as threads in an unique UNIX process
  - Global variables automatically privatized
- Traces from various projects can be used offline
- Accurate Ethernet (soon IB) network models, accurate collectives
  - Misprediction of BigDFT on Tibidabo turned out to be a hardware issue
- Basic but sound coarse-grain CPU models (with multicores)
- Extensively tested on Linux, Mac and Windows

Limitations

- Partial MPI API coverage: $\approx$ 100 primitives supported (more to come on need)
  - No MPI-IO, no one-sided, MPI3 collectives, spawning ranks, ...
  - Still passes more than half of MPICH3 standard complience tests
- Non-multithreaded applications, neither pthread nor OpenMP

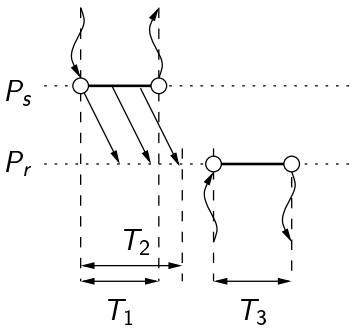
SimGrid Network Model

Measurements

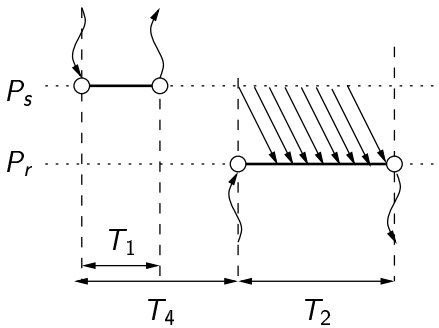


Hybrid Model

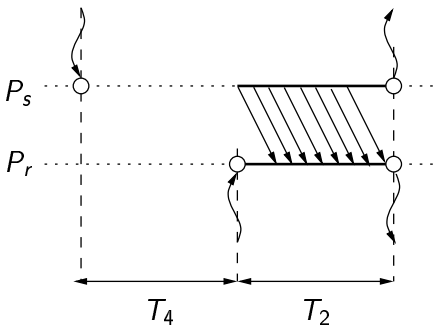
Asynchronous ($k \leqslant S_a$)



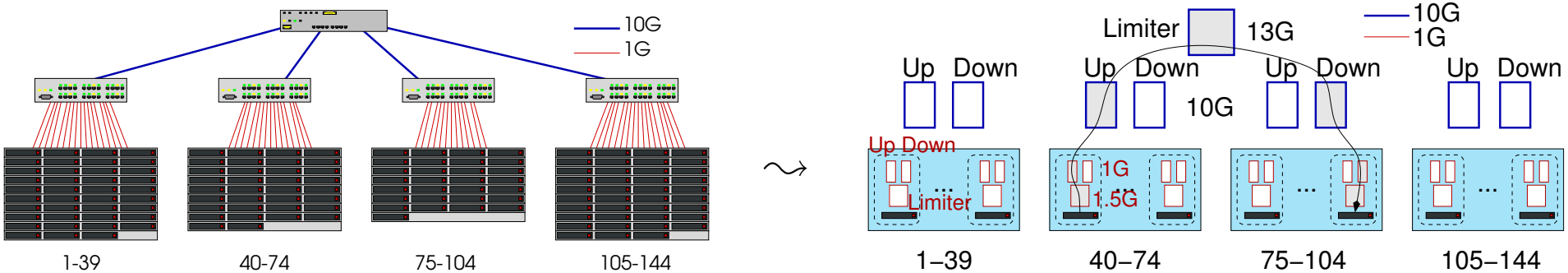
Detached ($S_a < k \leqslant S_d$)



Synchronous ($k > S_d$)



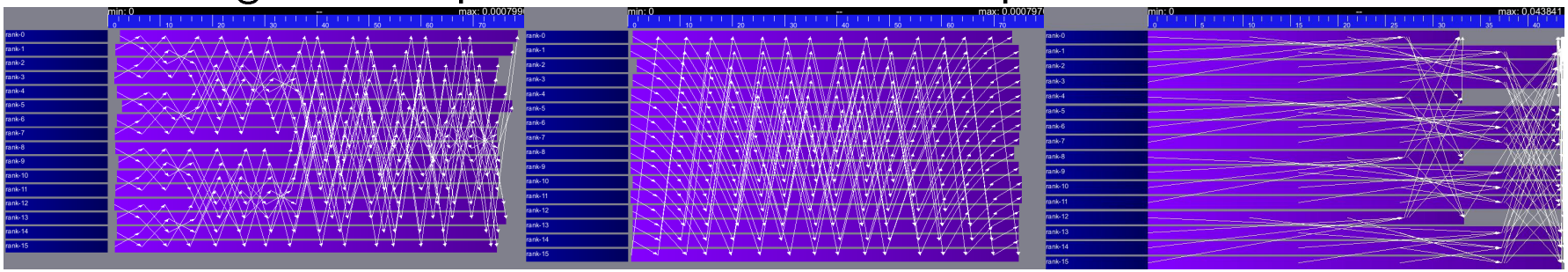
Fluid model: account for contention and network topology



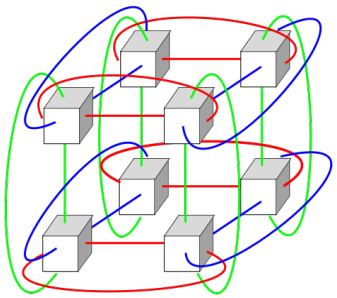
SimGrid Modeling of MPI

MPI Collectives

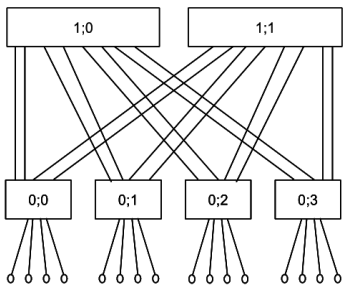
- SimGrid implements more than 120 algorithms for the 10 main MPI collectives
- Selection logic from OpenMPI, MPICH can be reproduced



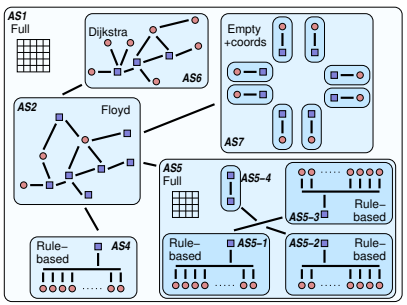
HPC Topologies



Torus



Fat-trees



Hierarchies of ASes

But also

- External load (availability changes), Host and link failures, Energy (DVFS)
- Virtual Machines, that can be migrated; Random platform generators

Agenda

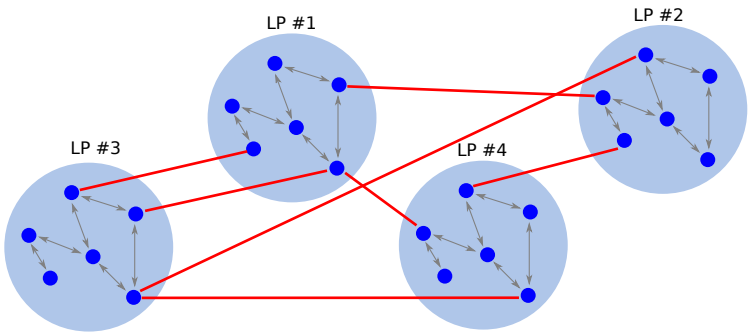
- Introduction
- Computational Science of Computer Systems (CS^2)
- The SimGrid Project
- Emulation with SMPI
- Parallel Simulation of Discrete Event Systems
- Dynamic Verification of Distributed Applications
- Conclusion on SimGrid
- Scientific Outreach

Parallel Simulation of Discrete Event Systems

- 30 years of literature on efficient Simulation Engines, FES and distribution
- Yet, all DES simulator for P2P were sequential (but dPeerSim)

The dPeerSim attempt

- Distributed implementation of PeerSim
- Classical parallelization: spreads the load over several Logical Processes (LP)



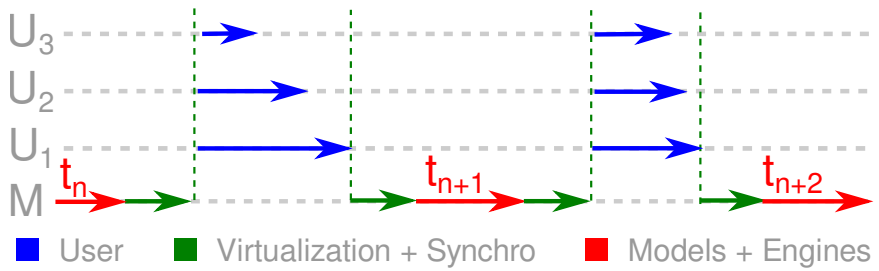
Evaluation

- Uses Chord as a standard workload: e.g. 320,000 nodes $\rightsquigarrow$ 320,000 requests
- Very good speedup results: 4h on 2 LPs $\rightsquigarrow$ 1h on 16 LPs
- But 47s in the original sequential PeerSim (and 5s in precise SimGrid)
- Yet, best known parallelization of DES simulator of P2P systems

New Parallelization Schema for DES

Split at Virtualization, not Simulation Engine

- Virtualization contains threads (user's stack)
- Engine & Models remains sequential



Simulation Workload	
	User Code
	Virtualization Layer
	Networking Models
Simulation Engine	
Execution Environment	

Understanding the trade-off

- Sequential time: $\sum_{SR}(engine + model + virtu + user)$
- Classical schema: $\sum_{SR}\left(\max_{i \in LP}(engine_i + model_i + virtu_i + user_i) + proto\right)$
- Proposed schema: $\sum_{SR}\left(engine + model + \max_{i \in WT}(virtu_i + user_i) + sync\right)$
- Synchronization protocol expensive wrt the engine's load to be distributed

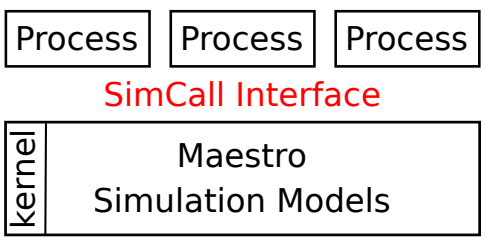
Toward Parallel P2P Simulation in SimGrid

Keep models sequential, execute processes in parallel

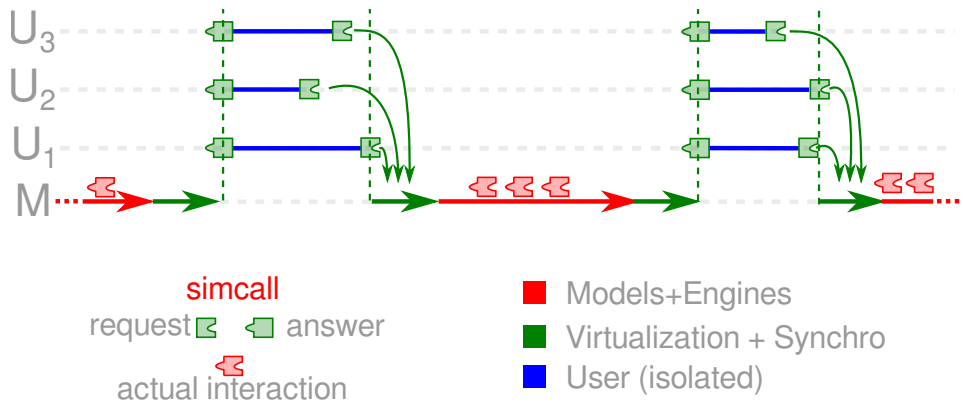
OS-inspired Approach toward Process Separation

- Fine-locking would be difficult, inefficient and would hinder reproducibility
- Mediate any process interactions through simcalls (conceptually identical to *syscalls* of real OSes)

Functional View

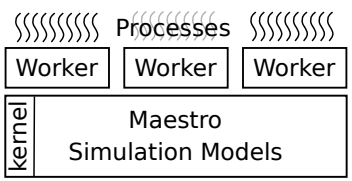


Temporal View

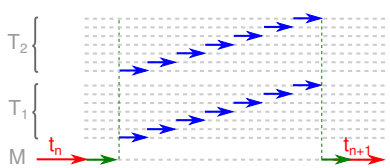


Leveraging Multicores

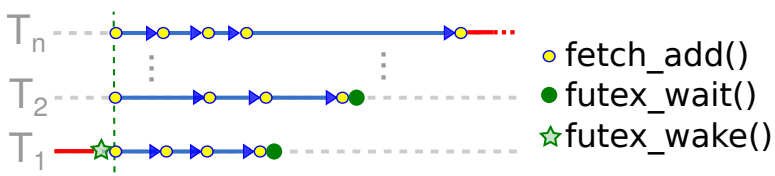
⇒ More processes than cores ⇝ Worker Threads (execute co-routines ;)



Functional View



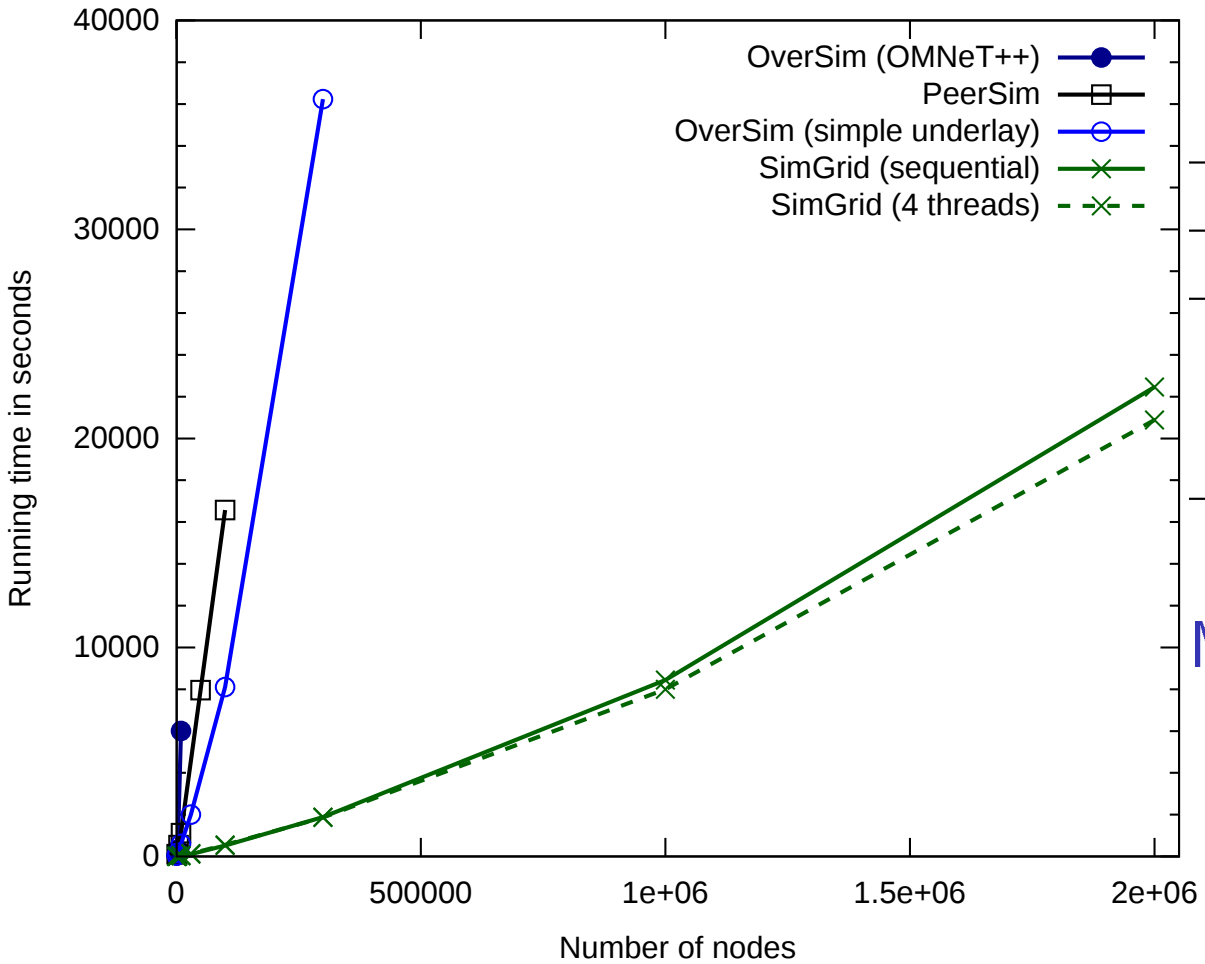
Temporal View



Ideal Algorithm

Sequential Performance in State of the Art

- Scenario: Initialize Chord, and simulate 1000 seconds of protocol
- Arbitrary Time Limit: 12 hours (kill simulation afterward)



Largest simulated scenario

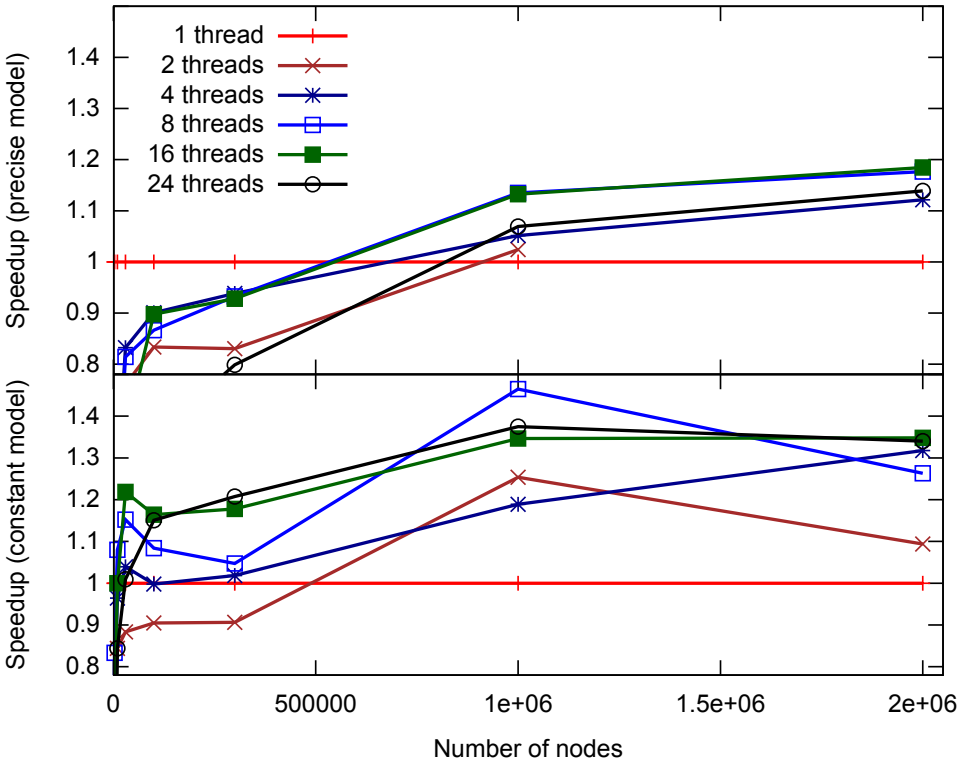
	Size	Time
Omnet++	10k	1h40
PeerSim	100k	4h36
OverSim	300k	10h
SG, precise	10k	130s
	300k	32mn
	2M	6h23
SG, simple	2M	5h30

Memory Usage

- 2M precise nodes: 32 GiB
- That is 18kiB per process (User stack: 12kiB)

Extra complexity of parallel execution doesn't impact sequential performance

Benefits of the Parallel Execution



- Speedup ($\frac{t_{seq}}{t_{par}}$): up to 45%
- More efficient with simple model:
  - Less work in engine + Amhdal law
- Speedup depends on thread amount
  - 8 threads (of 24 cores) often better
  - Synch costs remain hard to amortize
  - They depend on thread amount

Parallel Efficiency ($\frac{speedup}{\#cores}$) for 2M nodes

Model	4 threads	8 th.	16 th.	24 th.
Precise	0.28	0.15	0.07	0.05
Constant	0.33	0.16	0.08	0.06

- Baaaaad efficiency results
- Remember, P2P and Chord: Worst case scenarios

Yet, first time that Chord's parallel simulation is faster than best known sequential

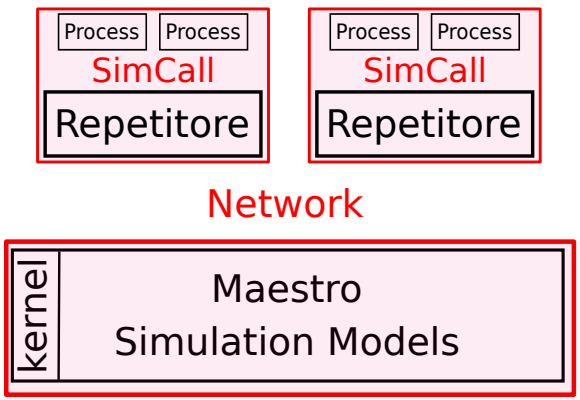
Future Work on HPS

Distributed Simulation toward size

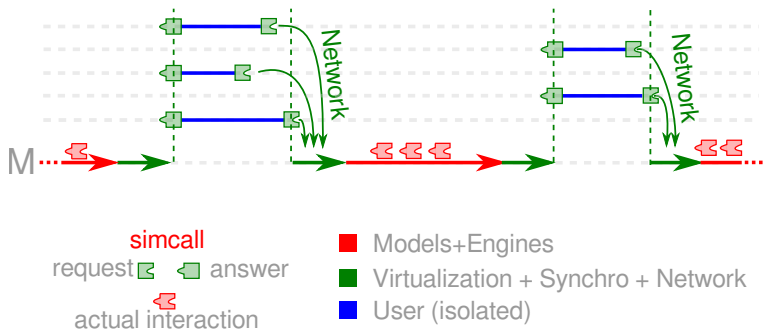
- Leverage the memory of more nodes; Useless in P2P, more adapted to SMPI

Design: split our design under the simcall layer

Functional View



Temporal View



Increase level of parallelism

- Pessimistic execution (as now): efficient for 500,000 processes and more...
- Optimistic execution unfeasible because of our complex state
- Vision: realistic execution run optimistically only if it is safe to do so
  Determining independent actions is easy using formal methods

Agenda

- Introduction
- Computational Science of Computer Systems (CS^2)
- The SimGrid Project
- Emulation with SMPI
- Parallel Simulation of Discrete Event Systems
- Dynamic Verification of Distributed Applications
- Conclusion on SimGrid
- Scientific Outreach

Assessing the Correctness of HPC codes?

The Good Old Days

- Writing distributed applications is notoriously difficult
- MPI codes circumvented the difficulty with rigid communication patterns
- Correctness established through testing
- Only performance matters anyway:
  - Most prefer a fast code that rarely fail-stop to a slow code that always work
  - (at least, that's my feeling for most of the numerical applications)

These Days are Now Over

- But rigid patterns do not scale! We now have to release the grip
- But this is dangerous! We now have to explicitly seek for correctness

Slowly, old ignored problems resurface...

Model Checking and Dynamic Verification

These are Automated Formal Methods

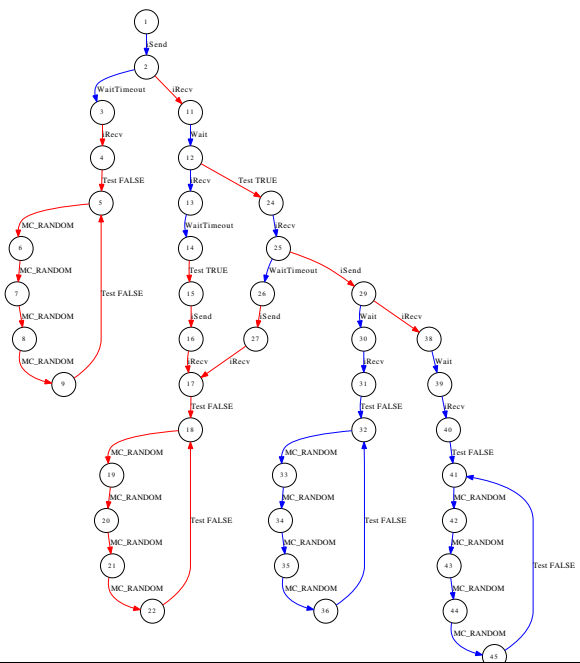
- Try to assess the correctness of a system by actively searching for faults
- If you find a fault, then you have something to work on
- If don't find any after an exhaustive search, correctness is experimentally proved
- Dynamic Verification: Model Checking applied to real applications

Model Checking and Dynamic Verification

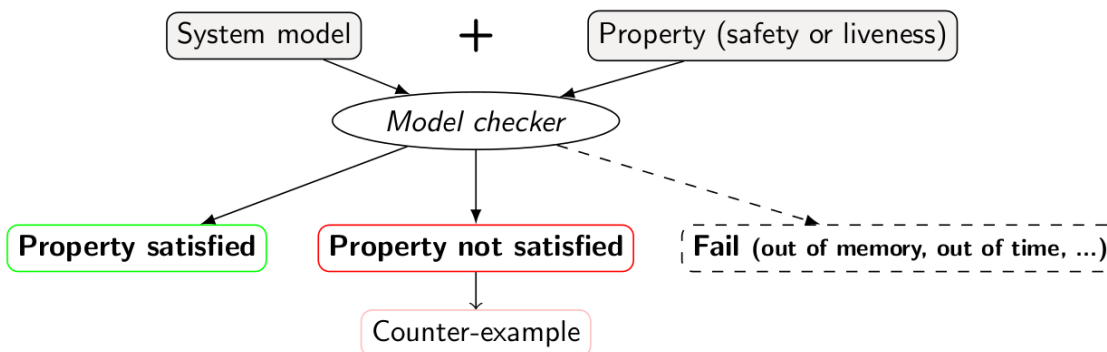
These are Automated Formal Methods

- Try to assess the correctness of a system by actively searching for faults
- If you find a fault, then you have something to work on
- If don't find any after an exhaustive search, correctness is experimentally proved
- Dynamic Verification: Model Checking applied to real applications

Exhaustive Exploration



Model Checking: the Big Idea



- My preferred outcome: a counter-example
  If not, I fear my property to be wrongly expressed
- We tend to bug finding, not certification

Formal Properties

Safety Properties

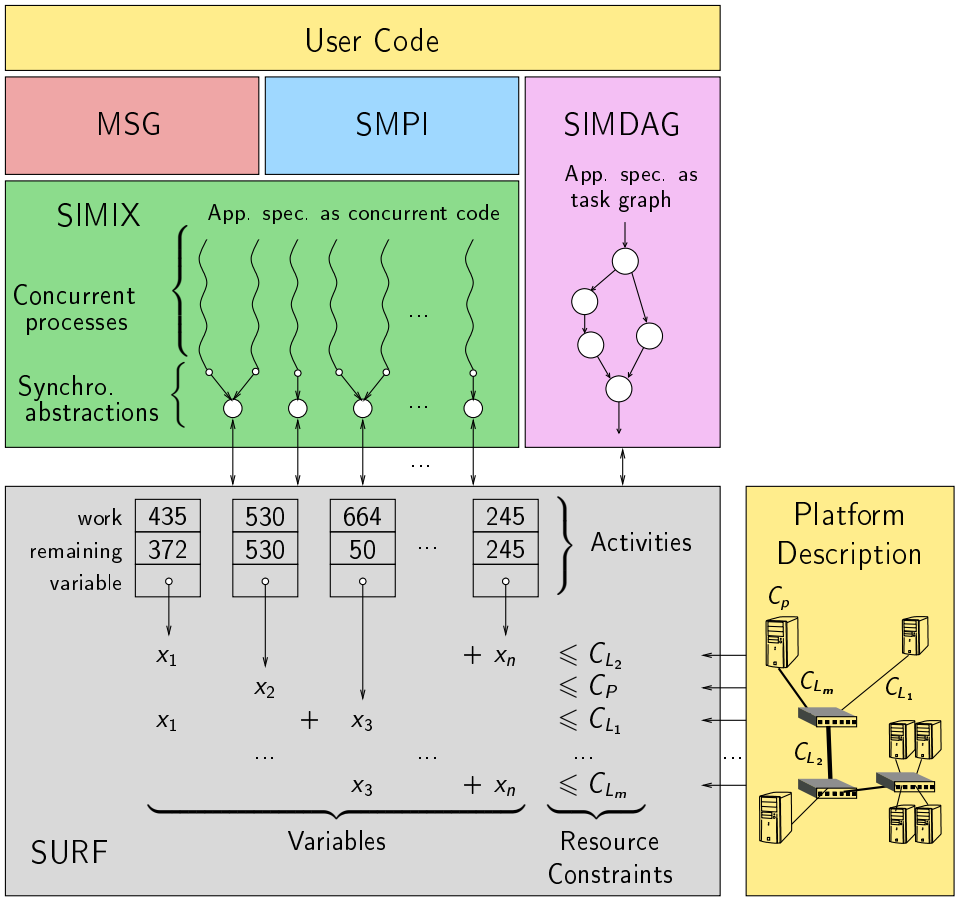
- "A given bad behavior never occurs"
- Can be expressed as boolean (assertion): no deadlock, $x \neq 0$, ...
- Work on all states separately
- Counter example: a faulty state

Liveness Properties

- "An expected behavior will happen in all cases"
- Example: Any process that asks a resource will obtain it eventually
- Must be expressed in a temporal logic such as CTL (safety ones *could* too)
- Work on execution path
- Counter example: an infinite path (ie, a cycle) that violates the property

Liveness properties are much more challenging to verify in practice

SimGrid and SMPI

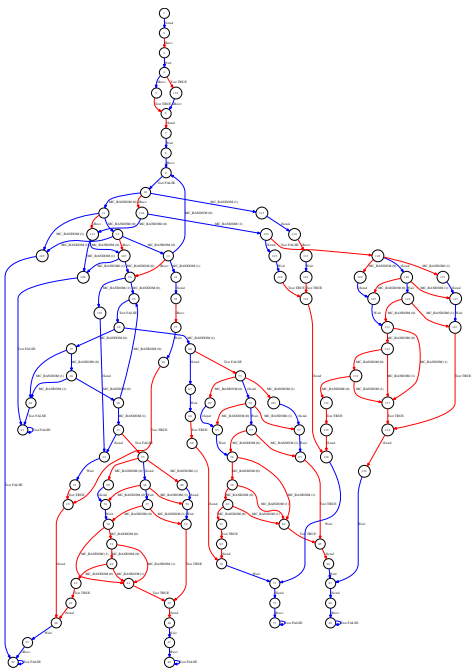


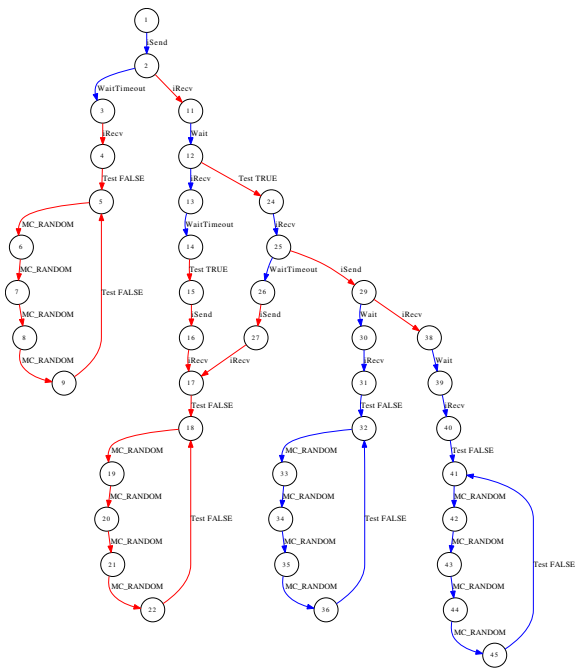
- SMPI an run complex C/C++/Fortran applications on top of SimGrid
- Let's leverage this unconventional virtualization layer for verification!

SimGridMC: Formal Methods in SimGrid

Verify any application that would run in SimGrid

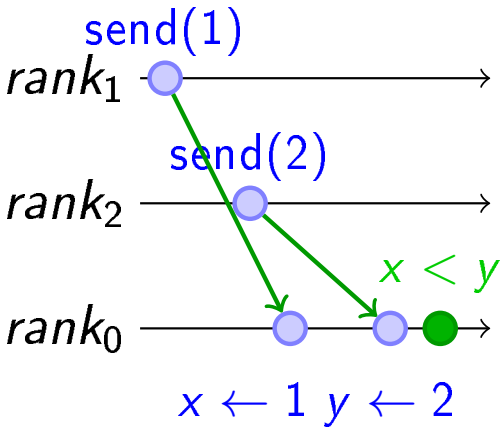
- Replace the simulation kernel underneath with a model checker
- Tests all causally possible orders of events to dynamically verify the app
- Reuse the mediation mechanism that base the simulator
- System-level checkpoints the app to then rewind and explore another path
- Works with SMPI, and MSG (our simple API for the study of CSP algorithms)



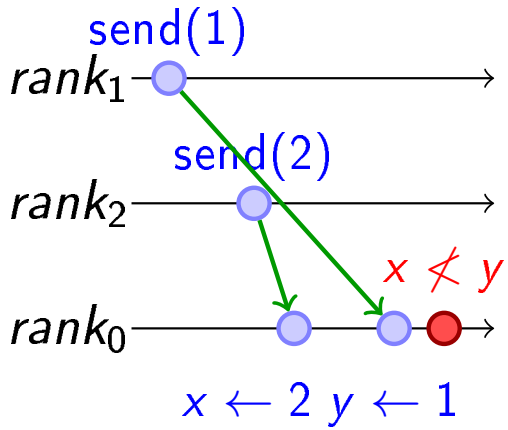


Example: Out of order receive

- Two processes send a message to a third one
- The receiver expects the message to be in order
- This may happen. . . or not




```
if (MPI_rank() == 0) {
  MPI_Recv(&x , MPI_ANY_SOURCE);
  MPI_Recv(&y , MPI_ANY_SOURCE);
  MC_assert(x < y);
} else {
  MPI_Send (&rank , 0);
}
```




```
**************************
*** PROPERTY NOT VALID ***
**************************
Counter-example execution trace:
[(1)recver] iRecv (dst=recver, buff=(verbose only), size=(verbose only))
[(3)sender] iSend (src=sender, buff=(verbose only), size=(verbose only))
[(1)recver] Wait (comm=(verbose only) [(3)sender -> (1)recver])
[(1)recver] iRecv (dst=recver, buff=(verbose only), size=(verbose only))
[(2)sender] iSend (src=sender, buff=(verbose only), size=(verbose only))
[(1)recver] Wait (comm=(verbose only) [(2)sender -> (1)recver])
```

Mitigating the State Space Explosion

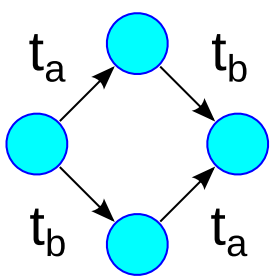
Many execution paths are redundant $\rightsquigarrow$ cut exploration when possible

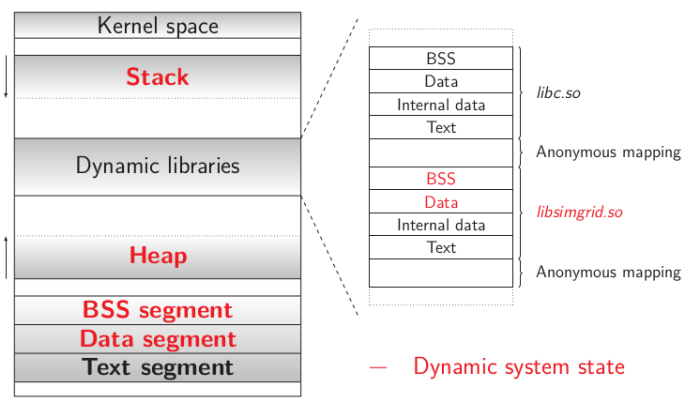
Dynamic Partial Ordering Reduction (DPOR)

- Works on histories: test only one transitions' interleaving if independent
- *Independence theorems:* Local events are independent; iSend+iRecv also; ...
- Must be conservative (exploration soundness at risk!)
- It works well (for safety properties)

System-Level State Equality

- Works on states: detect when a given space was previously explored
- Complementary to DPOR (but not compatible yet)
- Introspect the C/C++/Fortran app just like gdb (+some black magic)





Some Results

Wild safety bug in our Chord implementation (≈ 500 lines of C)

- Simulation: bug on large instances only; MC finds small trace (1s with DPOR)

Mocked liveness bug

- Buggy centralized mutual exclusion: last client never obtains the CS
- About 100 lines – state snapshot size: 5Mib
- Verified with up to 9 processes (12,000 states, 9 minutes, 45Gb).

Verifying MPICH3 complience tests

- Looking for assertion failures, deadlocks and non-progressive cycles
- 6 tests; ≈ 1300 LOCs (per test) – State snapshot size: ≈ 4MB

- With no reduction: no test concluded in a few hours
- With state equality: Exhaustive exploration up to 10 procs, but no error found

- We verified several MPI2 collectives too: all good so far ☹

Verification of Protocol-wide Properties

Motivation

- Clever checkpoint algorithms exist, provided that the application is nice enough
- *On communication determinism in parallel HPC applications*, F. Cappello, A. Guermouche and M. Snir (2010)
  - Manual inspection of 27 HPC applications, seeking for such properties

Protocol-wide properties

- deterministic: On each node, send and receive events are always in same order
- send deterministic: $\forall$ node, send are always the same, no matter the recv order
- Not liveness, not even CTL: quantifies for all execution paths within property

Status report: we can verify such properties in SimGrid

- Explore one path to learn the communication order, deduce the property
- Enforce that this order holds on all other execution path
- We reproduced the conclusions of previous paper on several benchmarks
  - All good ☹

Take Away Messages

SimGrid will prove helpful to your research

- Versatile: Used in several communities (scheduling, GridRPC, HPC, P2P, Clouds)
- Accurate: Model limits known thanks to validation studies
- Sound: Easy to use, extensible, fast to execute, scalable to death, well tested
- Open: User-community much larger than contributors group; AGPL
- Around since over 10 years, and ready for at least 10 more years

Welcome to the Age of (Sound) Computational Science

- Discover: `http://simgrid.gforge.inria.fr/`
- Learn: 101 tutorials, user manuals and examples
- Join: user mailing list, #simgrid on irc.debian.org
  We even have some open positions ;)

Agenda

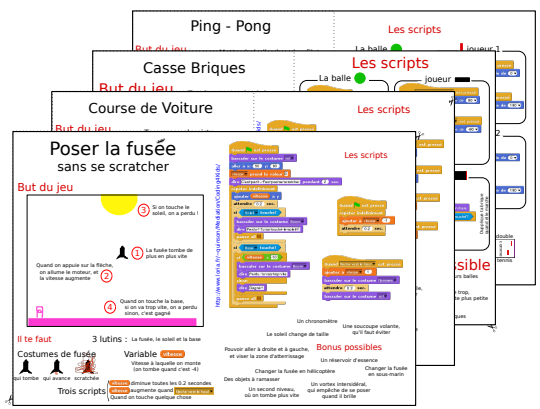
- Introduction
- Computational Science of Computer Systems (CS^2)
- The SimGrid Project
- Emulation with SMPI
- Parallel Simulation of Discrete Event Systems
- Dynamic Verification of Distributed Applications
- Conclusion on SimGrid
- Scientific Outreach

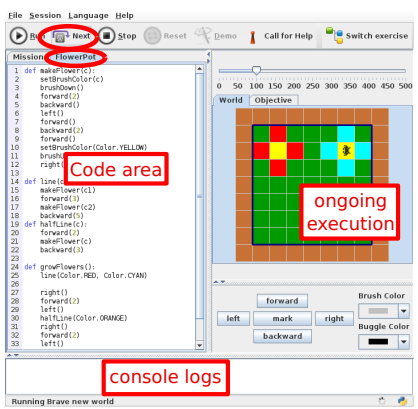
(Diffusion de la culture scientifique)

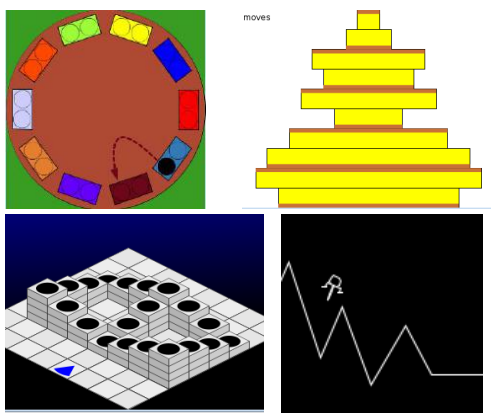
(diapo pour «Décodez le code» de Cap Digital/Inria au CNAM le 2/4/2014)

Apprendre l'informatique par la programmation

Quoi?	Bon sens, idées fortes
36 minutes	Activités débranchées
36 heures	Coding goûters avec *Snap!*
36 jours	Exerciseur interactif
36 semaines	Tous mes cours  sur 
36 ans	Communautés de logiciels libres
Avec qui?	Mise en réseau avec je.code();





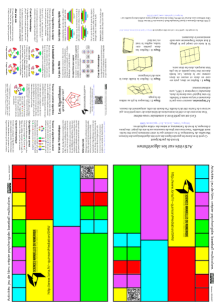


Sciences Manuelles du Numérique

Quoi: activités débranchées pour présenter l'informatique

- Parler d'informatique sans ordinateur, et même sans électricité
- Ne pas étudier d'algorithmes fournis; promouvoir une démarche expérimentale






Motivation

- Inventé pour la fête de la science: capter le chaland et passer le message
- `http://csunplugged.org/` super, mais licence CC-BY-NC-ND

État des lieux

- Séquence *algorithmes* marche très bien; d'autres en cours
- Matériel facile à dupliquer, tant pour le participant que pour l'asso
- Projet ouvert, contribution attendue: `http://www.loria.fr/~quinson/SMN/`

coding goûter

- «des kids, du code et du cake» — `http://codinggouter.org/`
- Activités ludiques pour apprendre la programmation créative
- En famille, ou non. Régulier ou une seule fois. Dans divers endroits

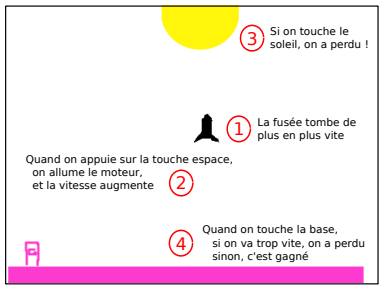


Pourquoi organiser des Coding Goûters?

- La programmation comme porte d'entrée sur l'informatique
  - L'informatique est bien plus vaste, c'est sûr
  - La programmation pour aborder les notions par la pratique
  - C'est mon pot de miel vers notre science
- Lire, écrire, compter . . . et programmer
  - *Programmez, ou soyez programmés*
  - Programmer: faire même ce que le vendeur n'avait pas prévu

http://www.loria.fr/~quinson/C4K/

Poser la fusée
sans se scratcher



Premier jeu à faire en Snap! (ou Scratch)

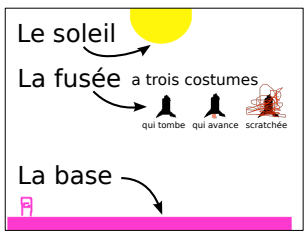
Principe du jeu

On doit poser sa fusée sans la scratcher.

Si on appuie sur *espace*, le moteur la fait accélérer vers le haut. Si on ne fait rien, elle tombe (de plus en plus vite).

On a gagné si on arrive tout doucement sur la base. On a perdu si on se pose trop vite, ou si on touche le soleil.

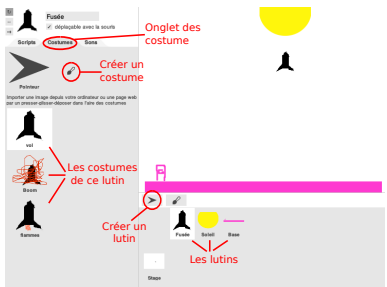
Il te faut trois lutins



Comment créer ces lutins ?

Qu'est ce qu'un lutin ?

Les lutins sont des dessins à l'écran que l'on peut commander en écrivant des scripts.

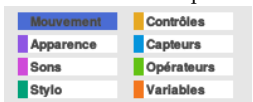


Crée tes lutins, puis dessine leurs costumes.

Comment faire bouger mes lutins ?

Qu'est ce qu'un script ?

Les briques sont des ordres que le lutin comprend. Elles sont triées par couleur :



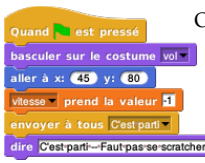
Il y a beaucoup de briques possibles, mais heureusement, on en utilise peu à la fois ! Essaye de combiner les briques suivantes :

Faire descendre la fusée

Il est temps de faire notre premier vrai script, puis de cliquer dessus.

Mince, j'ai perdu la fusée (en bas) !

Ne plus perdre la fusée



On revient au départ. On note que la vitesse est -1. On déclenche les autres scripts.

Quand ça démarre, on descend comme avant.

La fusée a maintenant deux scripts.

Tomber toujours plus vite

Ce troisième script de la fusée change régulièrement la vitesse.

Si la fusée glisse mollement ou se précipite au sol, corrige les délais et la vitesse.

Le moteur fait remonter

Un quatrième script ajoute de la vitesse vers le haut quand on active le moteur.

Ne touche pas le soleil !

Pour détecter si la fusée se brûle dans le soleil, il faut ajouter un cinquième script qui bloque jusqu'au moment du contact.

Toucher la base

Posée ou scratchée ?

gagné ou perdu ? Ça dépend de la vitesse !

(nouvelle version du dernier script)

Ton jeu est prêt ! !

Amuse toi bien !

C'est ton jeu

Ce livret t'apprend à fabriquer ton premier jeu en Snap!, étape après étape.

Quand il est terminé, joue avec tes copains et fais-le évoluer comme tu le désires.

Bonus possibles

Un chronomètre *Un réservoir d'essence*

Le soleil change de taille

Un second niveau, où on tombe plus vite

Pouvoir aller à droite et à gauche

Viser la base *Ramasser des choses*

Éviter des méchants

Changer la fusée en sous-marin *Changer la fusée en hélicoptère*

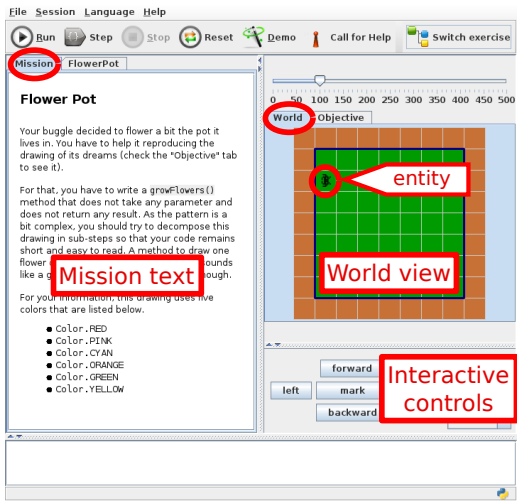
Savoir programmer permet

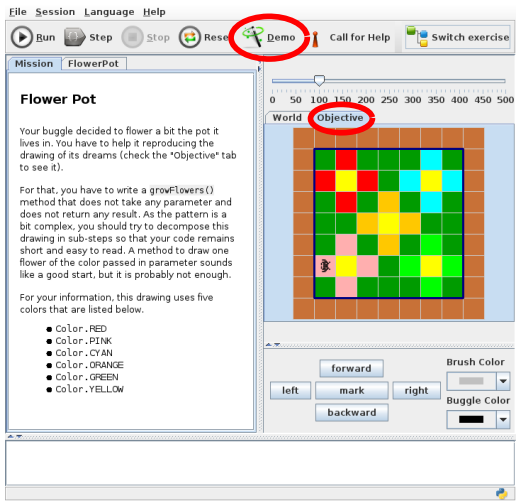
de créer des choses auxquelles personne n'avait pensé. Profites-en pour inventer !

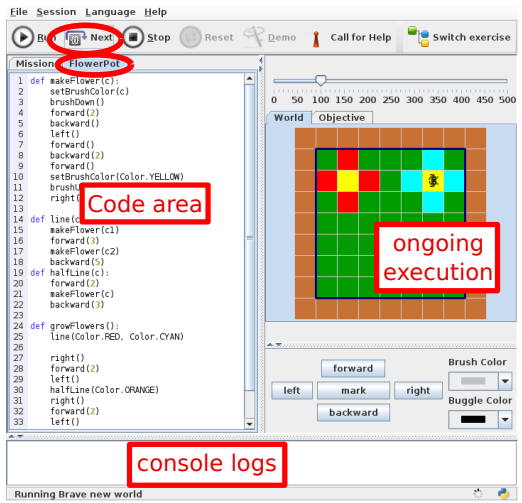
La Programmer's Learning Maching

Exerciseur interactif dédié à la programmation

- Outil interactif et graphique pour apprendre à coder
- C'est en forgeant qu'on devient forgeron (et qu'on apprend à aimer ça)







Usage classique

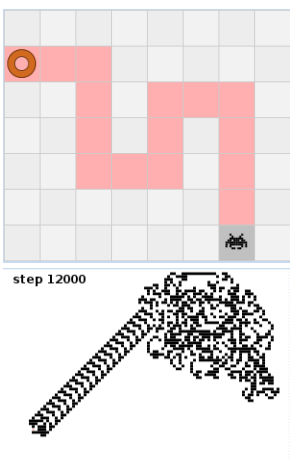
- On lit la mission à gauche, compare à droite l'état initial et l'état désiré
- On tape le code, on clique sur un bouton, et *ça s'anime* à droite
- Boucle de *feedback* très courte (et motivante pour les élèves)

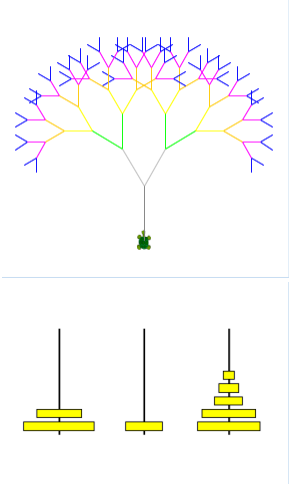
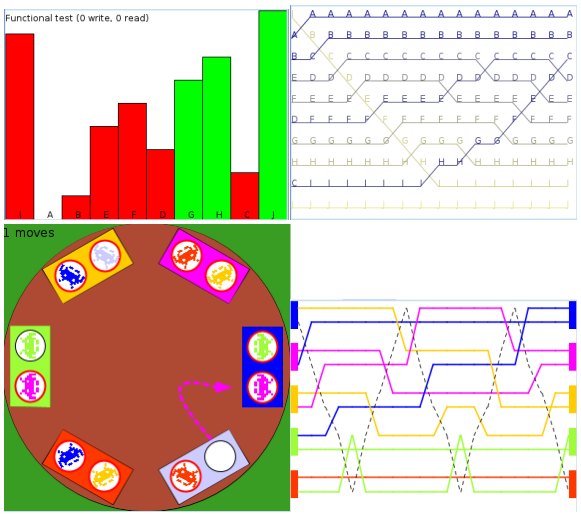
Bonnes propriétés de la PLM

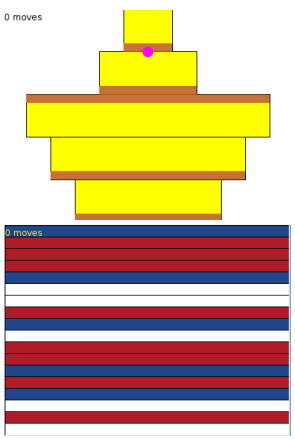
- L'interface est trilingue anglais/français/italien (+ brésilien soon?)
- Les exercices sont trilingues Java/Scala/Python (+C soon; javascript/ruby?)
- Mode démo, exécution pas à pas, vitesse d'animation, sessions
- Plusieurs mondes parallèles pour mieux tester le code élève
- Documentation embarquée dans l'outil

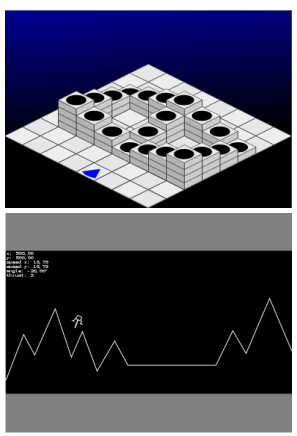
Différents types d'univers pour différentes situations-problèmes

- Micro-mondes génériques; Problèmes classiques; Tris; Jeux; (jUnit)
- Des vues d'état et des vues temporelles; Des interactions à la souris







- Ajouter un univers: 100 à 300 lignes de code (Java ou Scala)

Projet de recherche autour de PLM

Objectif: PLM comme terrain d'expérimentations

- Pédagogie: quelle est la bonne façon d'enseigner la programmation?
- Modélisation: gamification, tuteur intelligent
- Data-mining: erreurs corrélées, corrélation au *background* des élèves, erreurs typiques (identification d'obstacles didactiques)
- Collaboratif: susciter la collaboration sans triche
- Systèmes distribués: former suffisament d'ingénieurs MPI pour l'avenir, diffuser SimGrid (projet de MOOC sur le sujet)

Participants

- M. Granbastien: Pédagogie et numérique (émérite)
- A. Brun et A. Boyer: User modeling and personalization (Kiwi)
- Y. Toussaint: Knowledge Extraction from Texts (Orpailleur)
- G. Oster: Computer-Supported Cooperative Work (Score)
- M. Quinson: Modélisation sémantique de systèmes distribués (Algorille)

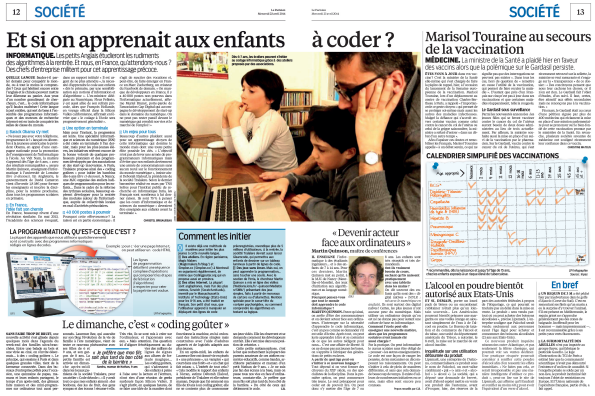
De l'importance d'apprendre à programmer

Décoder le code fait salle comble

De l'importance d'apprendre à programmer

Décoder le code fait salle comble

Les JT de CANAL+ et TF1



La Croix (26 mai)

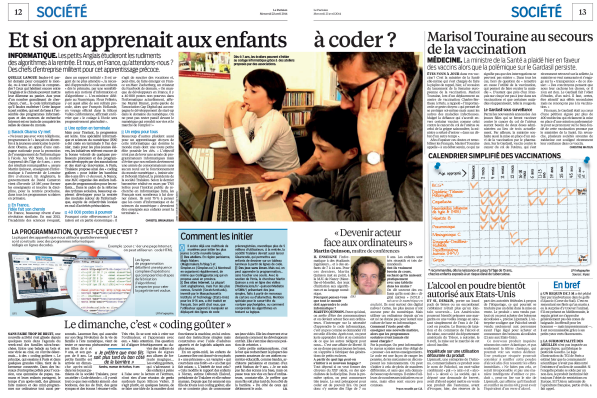
✚ L'initiation informatique des élèves français se fait attendre

▶ École, le gouvernement ouvre le chantier de la notation

De l'importance d'apprendre à programmer

Décoder le code fait salle comble

Les JT de CANAL+ et TF1



La Croix (26 mai)

✚ L'initiation informatique des élèves français se fait attendre

▶ École, le gouvernement ouvre le chantier de la notation

La question n'est plus «Faut-il» mais plutôt «Comment faire»

Allons-y maintenant! je.code();

Agenda

- Introduction
- Computational Science of Computer Systems (CS^2)
- The SimGrid Project
- Emulation with SMPI
- Parallel Simulation of Discrete Event Systems
- Dynamic Verification of Distributed Applications
- Conclusion on SimGrid
- Scientific Outreach