# Mc SimGrid

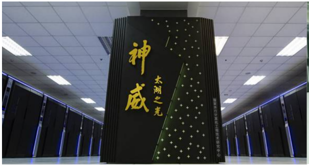
## Turning a Simulator of System Performance into a Dynamic Verification Framework

Martin Quinson
ENS Rennes / Inria, France

Northeastern University
January 23., 2018

# Modern Large Scale Distributed Systems

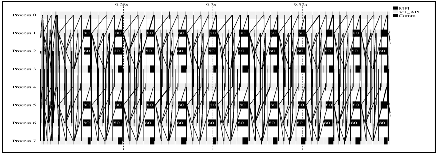
## Huge Systems



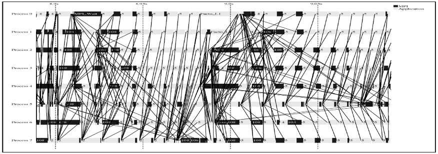


| #1 Taihu Light | #2 Tianhe 2 | #3 Piz Daint |
|---|---|---|
| 10,649,600 cores | 3,120,000 cores | 361,760 cores |
| 125 Tflops, 15MW | 56 Tflops, 18MW | 25 Tflops, 2MW |

## Complex Applications




Rigid, Regular, Hand-tuned Comm Patterns



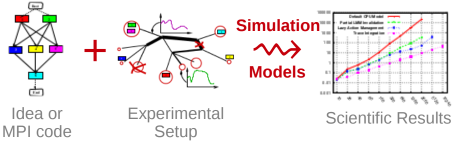

Dynamic, Irregular (task-based?)

How do we study these beasts?

# Simulating Distributed Systems

## Simulation: Fastest Path from Idea to Data

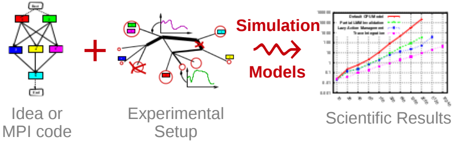
- Test your scientific idea with a fast and confortable scientific instrument



# Simulating Distributed Systems

## Simulation: Fastest Path from Idea to Data

- Test your scientific idea with a fast and confortable scientific instrument



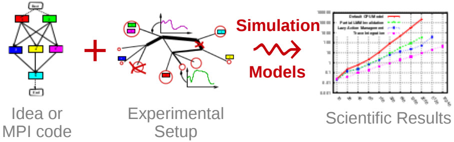

## Simulation: Easiest Way to Study Real Distributed Systems

- Centralized and reproducible setup. Don't waste resources to debug and test
- No Heisenbug, full Clairevoyance, High Reproducibility, *What if* studies

# Simulating Distributed Systems

## Simulation: Fastest Path from Idea to Data

- Test your scientific idea with a fast and confortable scientific instrument

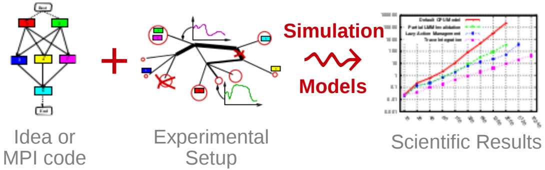


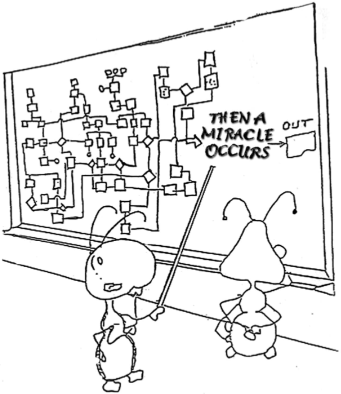

## Simulation: Easiest Way to Study Real Distributed Systems



- Centralized and reproducible setup. Don't waste resources to debug and test
- No Heisenbug, full Clairevoyance, High Reproducibility, *What if* studies
- Also software/hardware co-design, capacity planning or hardware qualification

# Methodological Challenges raised




## Challenges

- Validity: Realistic results
- Scalability: Fast enough; Big enough
- Right Focus: Aligned with users concerns

## Flourishing State of the Art

- Each group / student build its own tool
  - Short lived, Narow focus, Improvable
- Some very good domain-specific tools (HPC)

# SimGrid: Versatile Simulator of Distributed Apps

## Install a Scientific Instrument on your Laptop

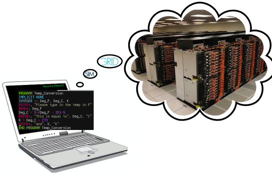
- Joint Project since 1998, mostly from french institutions
- Open Project, contributors in the USA (UHawaii, ISI), UK, Austria, Cern

## Key Strengths

- Performance Models validated with Open Science $\rightsquigarrow$ Predictive Power
- Architectured as an OS $\rightsquigarrow$ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users
- Usability: Fast, Reliable, MPI API, Visualization

## Community

- Mostly Scientists: 150 publications by 120 individuals
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, ...)
- Open Source: external Power Users (fixes & models)



# SimGrid: Versatile Simulator of Distributed Apps

## Install a Scientific Instrument on your Laptop

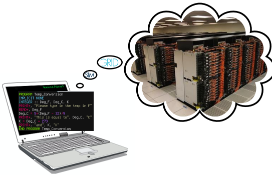
- Joint Project since 1998, mostly from french institutions
- Open Project, contributors in the USA (UHawaii, ISI), UK, Austria, Cern

## Key Strengths

- Performance Models validated with Open Science $\rightsquigarrow$ Predictive Power
- Architectured as an OS $\rightsquigarrow$ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users
- Usability: Fast, Reliable, MPI API, Visualization

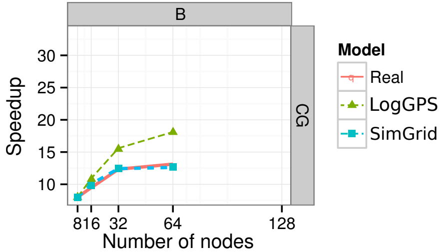
## Community

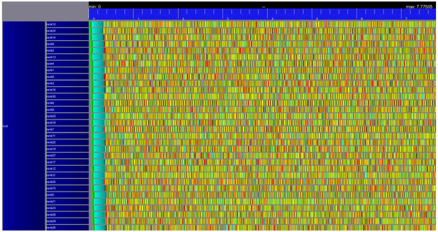
- Mostly Scientists: 150 publications by 120 individuals
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, ...)
- Open Source: external Power Users (fixes & models)



# Validity Success Stories

*unmodified* NAS CG on a TCP/Ethernet cluster (Grid'5000)



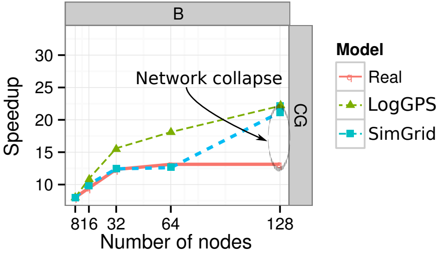



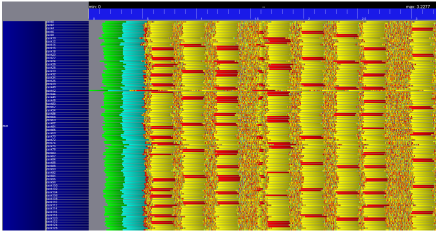
## Key aspects to obtain this result

- Network Topology: Contention (large msg) and Synchronization (small msg)
- Applicative (collective) operations (stolen from real implementations)
- Instantiate Platform models (matching effects, not docs)
- All included in SimGrid but the instantiation (remains manual for now)

# Validity Success Stories

*unmodified* NAS CG on a TCP/Ethernet cluster (Grid'5000)



## Discrepency between Simulation and Real Experiment. Why?

- Massive switch packet drops lead to 200ms timeouts in TCP!
- Tightly coupled: the whole application hangs until timeout
- Noise easy to model in the simulator, but useless for that very study
- Our prediction performance is more interesting to detect the real issue

# Have we reached the Perfect Model yet?

## What is the Perfect Model anyway?

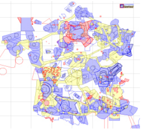
- Detailed enough to be realistic
- Efficient enough for ultra fast simulations
- Abstracted enough so that I can reason about
- In short, that's the one I could give to my students and forget about

# Perfect Model of France would be Perfect Map

Maps (and models) are abstractions

- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps

# Perfect Model of France would be Perfect Map

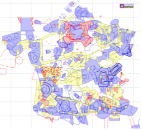



## Maps (and models) are abstractions

- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps

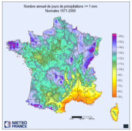
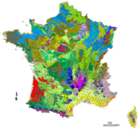
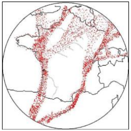
# Perfect Model of France would be Perfect Map



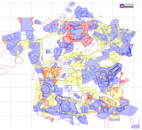
## Maps (and models) are abstractions

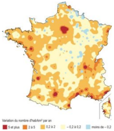
- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps

# Perfect Model of France would be Perfect Map

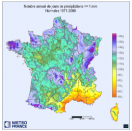
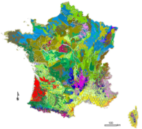
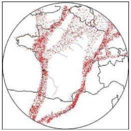


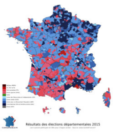
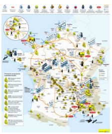



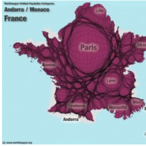
## Maps (and models) are abstractions

- Quality depends on what your usage
- More detailled $\neq$ better (not always)
- **No One True Map fitting all needs**
- Myriads of carefully adapted maps





# Perfect Model of Distributed Systems?

**the one making your Study sound**

## If you study a theoretical P2P algorithm

- You could probably go for a super-fast constant-time model

## If your study is a MPI application

- with TCP LAN, SMPI should do the trick (with correct instanciatiation)
- with InfiniBand and/or GPUs, you need our still ongoing models

## If you work on a TCP variant

- then you need a packet-level simulator such as NS3

## If your study WAN-interconnected Set Top Boxes

- SMPI model not suited! Impossible to instanciate, validated only for MPI
- Vivaldi model intended for that kind of studies

**In any case, assess the validity & soundness**

# SimGrid: Versatile Simulator of Distributed Apps

## Install a Scientific Instrument on your Laptop

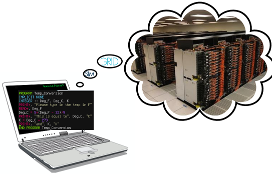
- Joint Project since 1998, mostly from french institutions
- Open Project, contributors in the USA (UHawaii, ISI), UK, Austria, Cern

## Key Strengths

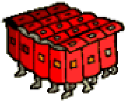
- Performance Models validated with Open Science $\rightsquigarrow$ Predictive Power
- Architectured as an OS $\rightsquigarrow$ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users
- Usability: Fast, Reliable, MPI API, Visualization

## Community

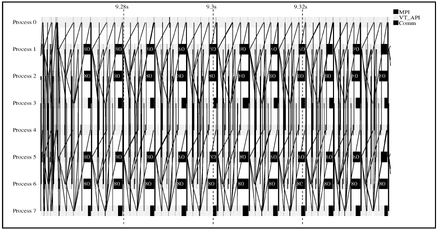
- Mostly Scientists: 150 publications by 120 individuals
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, ...)
- Open Source: external Power Users (fixes & models)



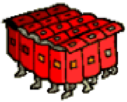
# Writting Correct Distributed Applications



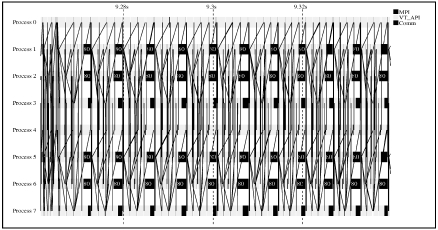
- Classical Solution: Proof of algorithms
- Pessimistic Solution: Lower performance expectations
- Optimistic Solution: Eventually Consistent
- HPC Solution: Rigid, Regular, Hand-tuned Communication Patterns

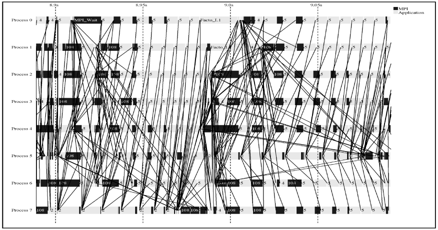


# Writting Correct Distributed Applications



- Classical Solution: Proof of algorithms
- Pessimistic Solution: Lower performance expectations
- Optimistic Solution: Eventually Consistent
- HPC Solution: Rigid, Regular, Hand-tuned Communication Patterns
- Large-Scale Hybrid Machines: Dynamic, Irregular (task-based?)



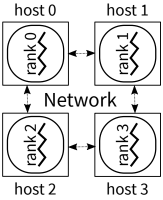

Verification: must explore **all possible execution paths**

# Virtualizing MPI Applications with SimGrid

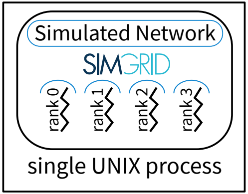
## SMPI: Reimplementation of MPI on top of MPI

- Computations emulated; Communications simulated
- Complex C/C++/F77/F90 apps run out of the box
- MPI 2.2 partially covered ($\approx$ 160 primitives supported)
  - No MPI-IO, MPI3 collectives, spawning ranks, ...
  - Monothreaded applications, no pthread nor OpenMP

## MPI Applications are *folded into* a single process




Real Settings



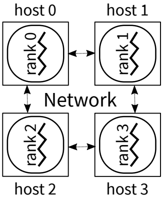

SimGrid Simulation

# Virtualizing MPI Applications with SimGrid

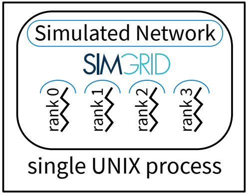
## SMPI: Reimplementation of MPI on top of MPI

- Computations emulated; Communications simulated
- Complex C/C++/F77/F90 apps run out of the box
- MPI 2.2 partially covered ($\approx$ 160 primitives supported)
  - No MPI-IO, MPI3 collectives, spawning ranks, . . .
  - Monothreaded applications, no pthread nor OpenMP

## MPI Applications are *folded into* a single process



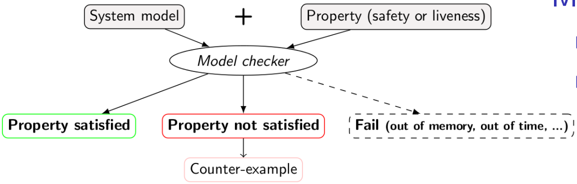

Real Settings




SimGrid Simulation

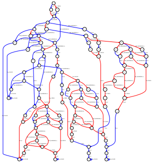
Mc SimGrid builds upon SimGrid to verify MPI applications

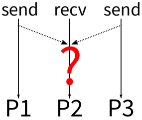
# Formal Methods in Mc SimGrid



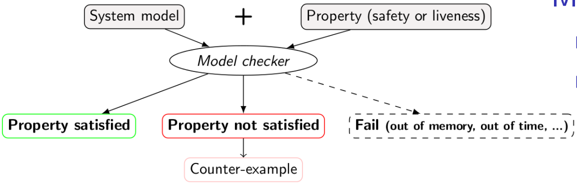

## Model Checking

- Exhaustively search for faults
- Requires an accurate model





# Formal Methods in Mc SimGrid

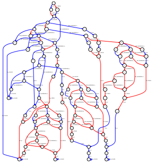


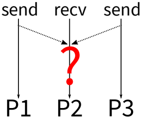

## Model Checking

- Exhaustively search for faults
- Requires an accurate model

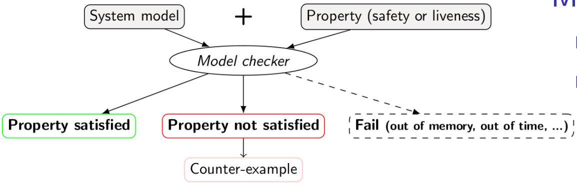
## Dynamic Verification: similar idea, applied to source code

- Mc SimGrid: Live, virtualized execution
  No static analysis (yet), no symbolic execution
- On Indecision Points: checkpoint, explore, rollback





# Formal Methods in Mc SimGrid

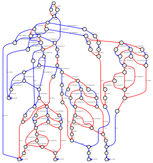



## Model Checking

- Exhaustively search for faults
- Requires an accurate model

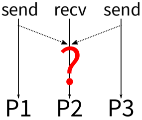
## Dynamic Verification: similar idea, applied to source code

- Mc SimGrid: Live, virtualized execution
  No static analysis (yet), no symbolic execution
- On Indecision Points: checkpoint, explore, rollback



## Execution Model in Mc SimGrid

- Mono-threaded MPI applications (CSP)
- Point-to-Point semantic: Configurable (paranoid / permissive)
- Collective semantic: Implementations of MPICH3, OpenMPI



# Use Cases: Kind of Properties

## Safety Properties: "A given bad behavior never occurs"

- e.g.: any assertion (x != 0, no deadlock)
- Verified on each state separately
- Counter example: a faulty state

## Liveness Properties: "An expected behavior will happen in all cases"

- e.g.: Any request will eventually be fulfilled; No non-progression cycle
- Verified on a full execution path
- Counter example: a cycling execution path that violates the property

## Comm Patterns: "It exists a pattern that is the same for all exec paths"

- e.g.: send-deterministic (local sending order is always the same)
- Work on all execution paths
- Counter examples: two paths exhibiting differing communication patterns

# Mitigating the State Space Explosion

## The exploration process often fails to complete

- Too many states to explore, not enough time and/or memory
- Mc SimGrid provides two reductions techniques

## Dynamic Partial Ordering Reduction (DPOR)

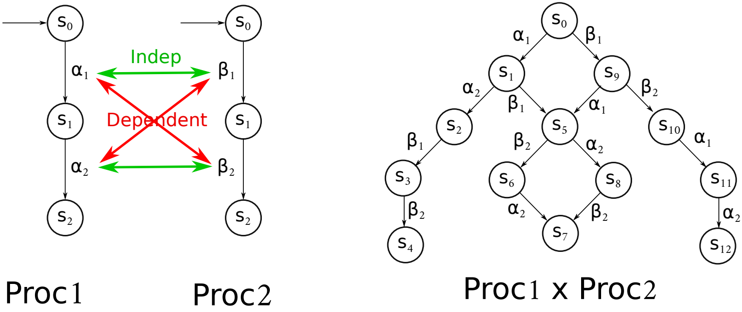
- Avoid re-exploring equivalent interleavings
- Don't explore all interleavings of local executions: they are equivalent

## System-Level State Equality

- Detect when a given state was previously explored

# Partial Ordering Reduction (DPOR)

- Avoid re-exploring Mazurkiewicz traces (don't permute independent events)




- McSimGrid: iSend and iSend are independent, etc.
- Dynamic Partial Ordering Reductions take advantage of runtime knowledge
- Many techniques (sleep sets, ample sets) are hard to understand & get right
- Ongoing work: reimplement our DPOR using Event Unfolding Structures

# But what are the transitions in Mc SimGrid?

Transition = atomic block of code between Indecision Points

- Test all interleavings of the shared state (mem+network) modifications
- Transition = (some local code +) **one** shared state's change

# But what are the transitions in Mc SimGrid?

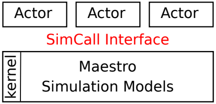
## Transition = atomic block of code between Indecision Points

- Test all interleavings of the shared state (mem+network) modifications
- Transition = (some local code +) **one** shared state's change

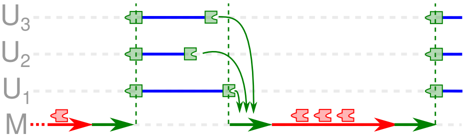
## Implementation: SimGrid is an Operating System

- Actors must use simcalls to modify the shared state
- First introduced for parallel simulation, but crucial to dynamic verification

Functional View

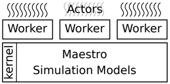



Temporal View

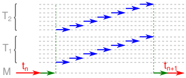



## Going parallel

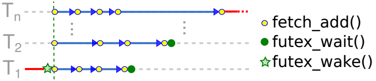
- More actors than cores ⇝ Worker Threads that execute co-routines




Functional View




Temporal View




Ideal Algorithm

# Mitigating the State Space Explosion

## The exploration process often fails to complete

- Too many states to explore, not enough time and/or memory
- Mc SimGrid provides two reductions techniques

## Dynamic Partial Ordering Reduction (DPOR)

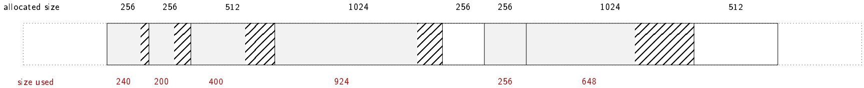
- Avoid re-exploring equivalent interleavings
- Don't explore all interleavings of local executions: they are equivalent

## System-Level State Equality

- Detect when a given state was previously explored
- Introspect the application state similarly to `gdb`
- Also with Memory Compaction

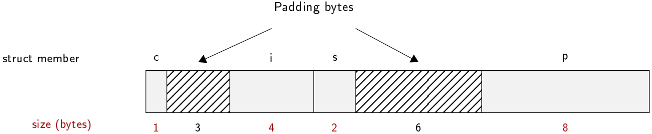
# OS-level State Equality Detection

- Memory over-provisioning

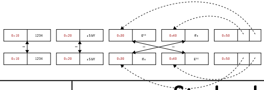



- Padding bytes: Data structure alignment

```
struct foo {
char c;
int i;
short s;
void *p;
}
```




- Irrelevant differences: system-level PID, fd, . . .
- Syntactic differences / semantic equalities:



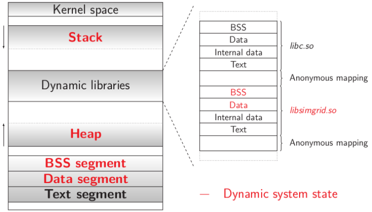
## Solutions

| Issue | Heap solution | Stack solution |
|---|---|---|
| Overprovisioning | `memset` 0 (customized `mmalloc`) | Stack pointer detection |
| Padding bytes | `memset` 0 (customized `mmalloc`) | DWARF + libunwind |
| Irrelevant differences | Ignore explicit areas | DWARF + libunwind + ignore |
| Syntactic differences | Heuristic for semantic comparison | N/A (sequential access) |

# Applicative State in Mc SimGrid

## We work at system level

- Target = legacy MPI apps
- Stack: where maestro lives
- Heap: shared between actors + actors stacks
- BSS+Data: private copy for each actor
- Network state is within libsimgrid data

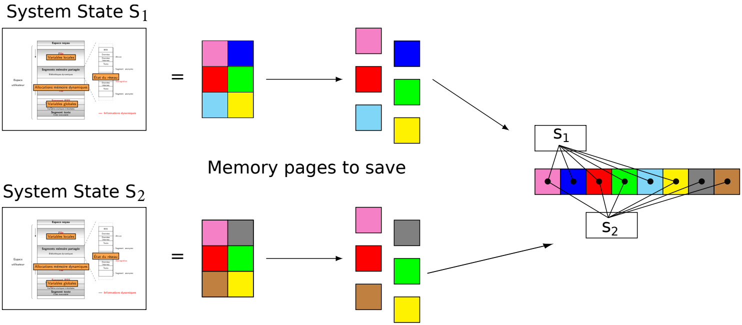



## How to privatize the BSS+data

- (this is required to fold MPI processes anyway)
- Source-to-Source: turn globals into arrays of locals
- Compiler's pass: move globals into TLS area
  changes toolchain (no `icc`) $\rightsquigarrow$ alters SEBs (as any previous solution)
- GOT injection: rewrite the ELF symbol table when switching contextes
  static variables are not part of the GOT unfortunately
- mmap of bss+data segments: preserves SEBs but forces sequential exec
- dlopen tricks: compile app with -fPIE, dlopen() it many times

# Memory Compactions

## We save literally thousands of states




- Very few modification between states in practice
- First fast hash function to distinguish new pages, then byte-wise equality
- Combines nicely with State Equality Detection (but complex implementation)

# Evaluation

## Verified small applications

- MPI2 collectives, MPICH3 test suite, Benchmarks (NAS, CORAL, NERSC)
- Safety, Liveness (no non-progressive cycle), Send-determinism

## Results

- Without reduction, only scales up to 2 to 6 processes in 24h
- Reductions (when usable) and Memory Compaction goes a bit further
- Not exactly ExaScale, but exhaustively at small size already useful

## Found bugs

- The one we intentionally added to the code
- Our own implementation of the Chord protocol (not in MPI)
- But no wild bugs in MPI yet :(

# Verification of some MPICH3 unit tests

- Looking for assertion failures, deadlocks and non-progressive cycles
- Exhaustive exploration, but no error found
- $\approx$ 1300 LOCs (per test) – State snapshot size: $\approx$ 4MB

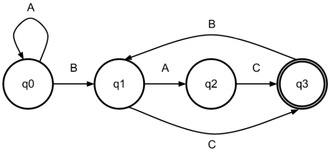
| Application | #P | Stateless exploration | | Stateful exploration | | |
|---|---|---|---|---|---|---|
| | | # States | Time | # States | Time | Memory |
| sendrecv2 | 2 | > 55 millions | > 6h | 936 | 13s | 2GB |
| | 5 | - | - | 2 284 | 43s | 5.4GB |
| | 10 | - | - | 3 882 | 2m | 11GB |
| bcastzerotype | 5 | > 12 millions | > 1h | 2 474 | 41s | 3.1GB |
| | 6 | - | - | 17 525 | 5m | 19GB |
| coll4 | 4 | > 100 millions | > 24h | 29 973 | 20m | 38GB |
| | 5 | - | - | > 150 000 | > 4h | > 200GB |
| groupcreate | 5 | > 10 millions | > 1h30 | 2 217 | 38s | 2.8GB |
| | 7 | - | - | 71 280 | 24m | 62GB |
| dup | 4 | > 57 millions | > 5h | 4 827 | 1m20 | 6.5GB |
| | 5 | - | - | 75 570 | 49m | 87GB |

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

# Checking Liveness Properties

## Enforce property $\phi$

- Search for a counter-example, ie a run of the system satisfying $\neg\phi$
- Counter examples are infinite $\rightsquigarrow$ Build the Büchi Automaton of $\neg\phi$




- Ensure that Application $\times$ Bucchi($\neg\phi$) is empty (no accepted run)
- State Equality is crucial to detect cycles

## Current state in Mc SimGrid

- Working in our tests (although fragile: equality is based on heuristics)
- We are looking for more domain-specific interesting properties

# 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: ∀ node, send are always the same, no matter the recv order
- Not liveness, not even LTL: quantifies for all execution paths within property

## Status report: we can verify such properties in Mc 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
  - NAS Parallel Benchmarks NPB 3.3 (5 kernels)
  - CORAL Benchmark codes
  - NERSC-8/Trinity Benchmarks* Conclusion

# Conclusion on Mc SimGrid

## Mc SimGrid: Dynamic Verification of MPI applications

- Unmodified C/C++/Fortran MPI applications
- Early stage, but already functional: Safety, Liveness, Send-determinism
- Reductions: DPOR and State Equality
- Scale to a few processes only, but exhaustive testing

## State of the Art

- Many testing tools (MUST): not exhaustive nor sound
- Symbolic execution (TASS, CIVL): complementary to our work
- Dynamic verification (ISP, DAMPI at U. Utah)
  - PMPI proxy at runtime to delay communications to guide execution
  - Works for safety, but not applicable to liveness (state equality)

## Ongoing Works

- Improve DPOR by using Event Unfolding structures
- Collab with NEU: Convert checkpoints taken on MPICH into SimGrid runs

# SimGrid: Versatile Simulator of Distributed Apps

## Install a Scientific Instrument on your Laptop

- Joint Project since 1998, mostly from french institutions
- Open Project, contributors in the USA (UHawaii, ISI, NEU), UK, Austria, Cern

## Key Strengths

- Performance Models validated with Open Science $\leadsto$ Predictive Power
- Architectured as an OS $\leadsto$ Efficiency; Performance & Correction co-evaluation
- Versatility: Advances in Clouds modeling reused by DataGrid users
- Usability: Fast, Reliable, MPI API, Visualization

## Community

- Mostly Scientists: 150 publications by 120 individuals
- Apps/Model co-dev : StarPU, BigDFT, TomP2P
- Some industrial users on internal projects (Intel, ...)
- Open Source: external Power Users (fixes & models)