SimGrid SMPI 101

Getting Started with SimGrid SMPI

Da SimGrid Team

February 6, 2018

About this Presentation

Goals and Contents

- Motivation, limits and classical use cases of SMPI
- Basic usage: running unmodified MPI applications on virtual platforms
- Advanced usage: folding memory and sampling executions for better efficiency

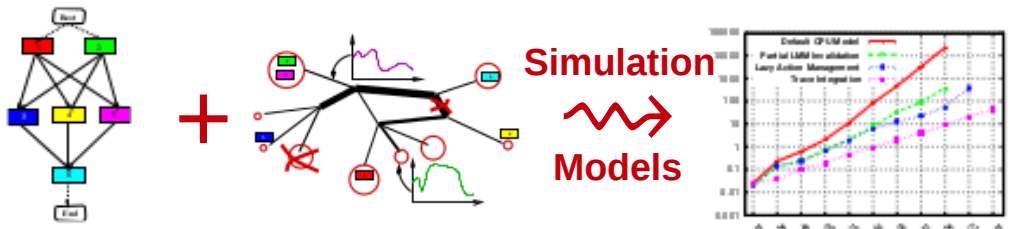
The SimGrid 101 serie

- This is part of a serie of presentations introducing various aspects of SimGrid
- SimGrid 101. Introduction to the SimGrid Scientific Project
- SimGrid User 101. Practical introduction to SimGrid and MSG
- SimGrid User::Platform 101. Defining platforms and experiments in SimGrid
- SimGrid User::SimDag 101. Practical introduction to the use of SimDag
- SimGrid User::Visualization 101. Visualization of SimGrid simulation results
- SimGrid User::SMPI 101. Simulation MPI applications in practice
- SimGrid User::Model-checking 101. Formal Verification of SimGrid programs
- SimGrid Internal::Models. The Platform Models underlying SimGrid
- SimGrid Internal::Kernel. Under the Hood of SimGrid
- SimGrid Contributer. Giving back to the community
- Get them from `http://simgrid.gforge.inria.fr/documentation.html`

Motivation Toward Simulation of HPC 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)

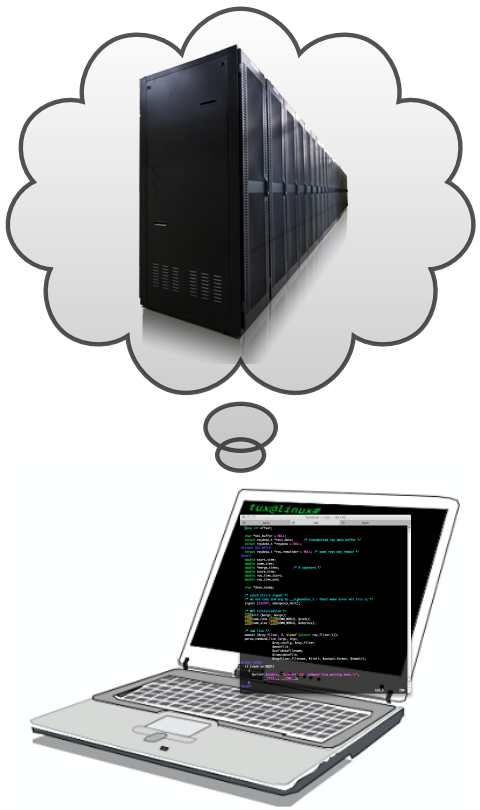


Simulation: easiest way to study distributed applications

- 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

This is **not** about HPC simulation (but about simulation of HPC)

SMPI



What is it?

- Reimplementation of MPI on top of SimGrid
- Imagine a VM running real MPI applications on platform 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
- Basic but sound coarse-grain CPU models (with multicores)
- Extensively tested on Linux, Mac and Windows

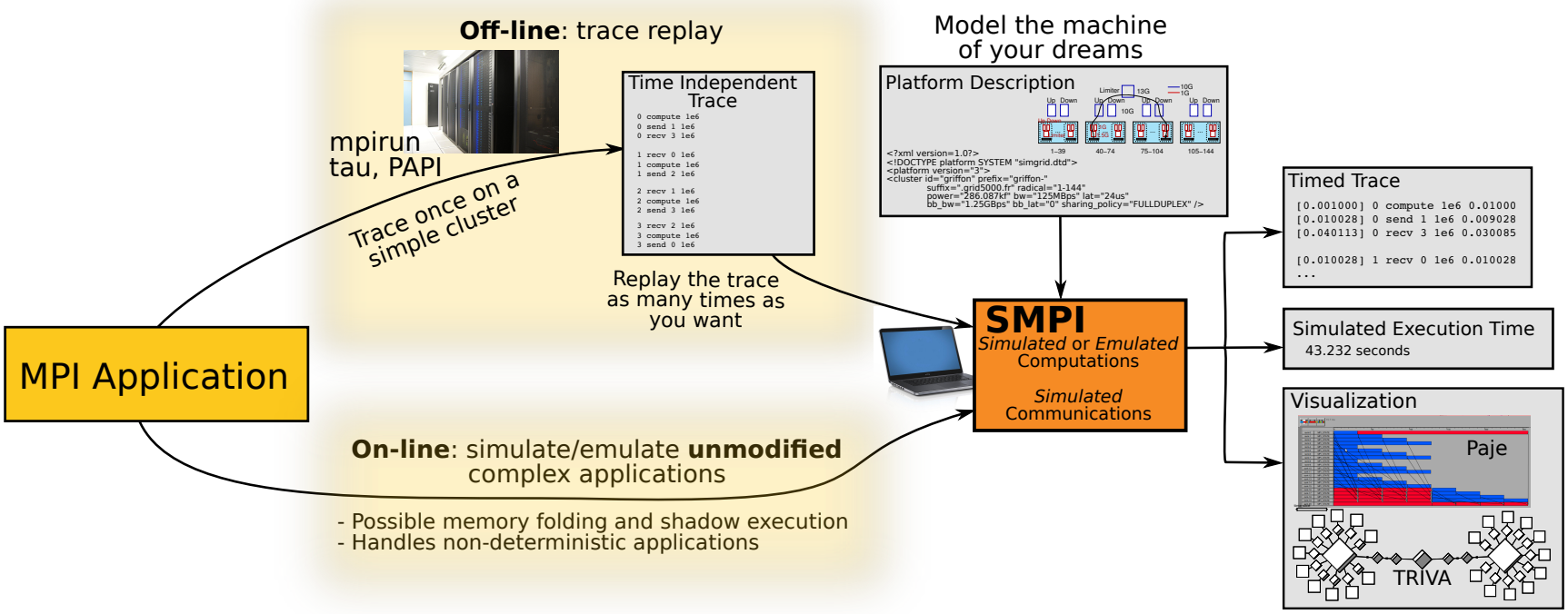
Some Success Stories

- Misprediction of BigDFT on Tibidabo turned out to be a hardware issue
- Reported to simulate 150,000+ ranks of a real application on a single node

Limitations

- MPI 2.2 partially covered: ≈160 primitives supported (more to come on need)
  - No MPI-IO, MPI3 collectives, spawning ranks, ...
  - Still passes a large amount of MPICH3 standard complience tests
- Non-multithreaded applications, neither pthread nor OpenMP

Running your code on SMPI



Offline Simulation

- Obtain a trace of your application
- Replay quickly and easily that trace
- Hard to extrapolate, adaptative apps?

Online Simulation

- Directly run your application
- Technically very challenging
- No limit (but the resources)

SMPI can do both

Which approach to choose for my app?

Simple application

- (Benchmark, teaching assignment, small application)

$\Rightarrow$ Online simulation of unmodified application: just use smpicc / smpirun $\rightsquigarrow$ p11

Larger application – network unaware, data independent

- (other data, other platform speed $\Rightarrow$ same message exchanges)

$\Rightarrow$ Offline simulation, with time independent traces

- Capture a trace at scale (right amount of ranks, on a smaller platform) $\rightsquigarrow$ p12

Larger application, network-aware – data independent

- (behavior changes with network load but not with data content)

$\Rightarrow$ Online simulation, + source annotation to fold memory and execution $\rightsquigarrow$ p18

Larger application that is both network-aware and data dependent

- If possible, "abstract away" compute kernels (use S4U::exec instead) $\rightsquigarrow$ article
- On need, you may use remote memory as a swap, eg with the nSwap project.

Installing SMPI

You just need to install SimGrid, that includes SMPI

Binary packages

- Debian, Ubuntu, etc: `sudo apt-get install simgrid`
- Windows, Mac OSX: none anymore/yet, sorry

Compiling from source

- Prefer stable archives to git, unless you have a good reason
- `tar xfz simgrid-*.tar.gz` ; `ccmake .` ; `make`
- In ccmake, you need `enable_smpi` (activated by default)
- Enable `enable_smpi_MPICH3_testsuite` to activate all the slow tests
- Do not enable `enable_model-checking` if you don't use it (perf killer)

Refer to the doc for details

http://simgrid.gforge.inria.fr/simgrid/latest/doc/install.html

Simulate your MPI application

XML Platform File

```
<?xml version='1.0'?>
<!DOCTYPE platform SYSTEM
"http://simgrid.gforge.inria.fr/simgrid.dtd">
<platform version="4">
<cluster id="acme"
 prefix="id-"  radical="0-9" suffix=".acme"
 power="1Gf"   bw="125MBps"  lat="50us"
 bb_bw="2GBps" bb_lat="500us"/>
</platform>
```

hostfile.txt

```
node-0.acme
node-1.acme
```

The application

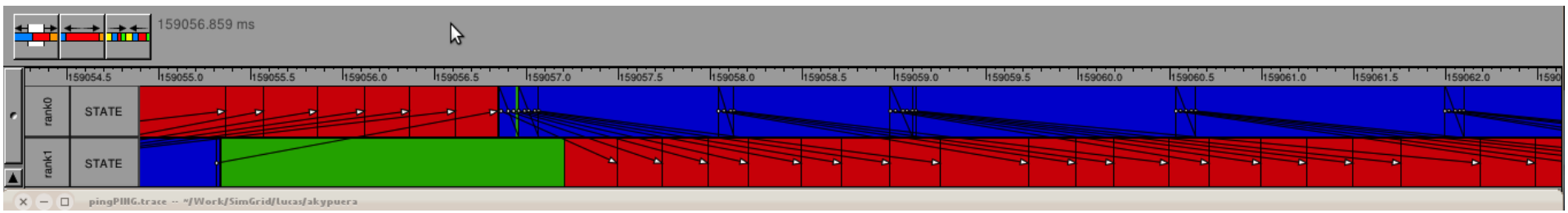
```
#include <mpi.h>
int main(int argc, char**argv) {
  int x;

  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &x);

  if (x == 0) { // rank
    x = 42;
    MPI_Send(&x, 1, MPI_INT, 1, 1,
             MPI_COMM_WORLD);
  } else {
    MPI_Recv(&x, 1, MPI_INT, 0, 1,
             MPI_COMM_WORLD, NULL);
    printf("Got %d from rank 0",x);
  }
  MPI_Finalize();
}
```

```
$ smpicc source.c -o application  # The code is now compiled
$ smpirun -platform cluster.xml -hostfile hostfile.txt  ./application # It starts
[...]  # Some debug information about your data provenance
Got 42 from rank 0
```

Vizualizing the result

- Pass option `-trace` to `smpirun`
- This generates a Pajé trace, that you can vizualize with Vite or viva

```
$ smpicc other.c -o thing
$ smpirun -platform cluster.xml -hostfile hostfile.txt -trace ./thing
$ vite smpi_simgrid.trace
```



Some bits of caution

- Our visualization framework is a currently *almost* working :-/
- The tools are getting a major lift, but it's not done yet
- See the SimGrid Visualization 101 for more info.

Simulating your MPI application online

That's about it

- It works out of the box in most cases
- Compile with `smpicxx` (C++), `smpiff` (Fortran 77), `smpif90` (Fortran 90)
- `smpirun` accepts the usual arguments (`-np` etc)
- Global variables are privatized by default (unless you pass -no-privatize)
  - Two strategies: mmap (default); dlopen (faster when it works). See online doc.
  - You need to link statically against the non-simgrid libraries

Configuring the execution

- Any SimGrid simulation accepts a few dozen command-line parameters
- 8 parameters are specific to SMPI (details in a few slides)
- `http://simgrid.gforge.inria.fr/simgrid/latest/doc/options.html`

What's going on under the hood?

- Refer to our other 101 tutorials, in particular internal::simix and internal::surf

Offline Simulation and Trace Replay

Going for SimGrid's Time Independent Traces

- Either trace your app. with `https://github.com/gmarkomanolis/mini`
- Or generate traces with `smpirun`:

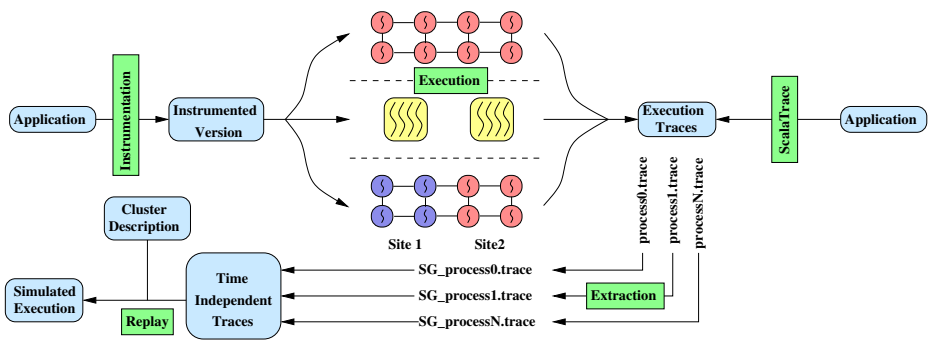
```
$ smpirun -trace-ti -hostfile machines -platform cluster.xml ./lu
```

- Replay trace files `traceProcessi` with `examples/smpi/replay` (you can test other platforms, but not extrapolating to other #processes)

```
$ smpirun -ext smpi_replay -hostfile machines -platform cluster.xml ./replay mytrace
```

Going for other formats

- ScalaTrace: was done at some point by Fred. Contact us for more info.
- Extrae: we tried but some information are missing in the trace
- OTF-2/score-P: we're working on it

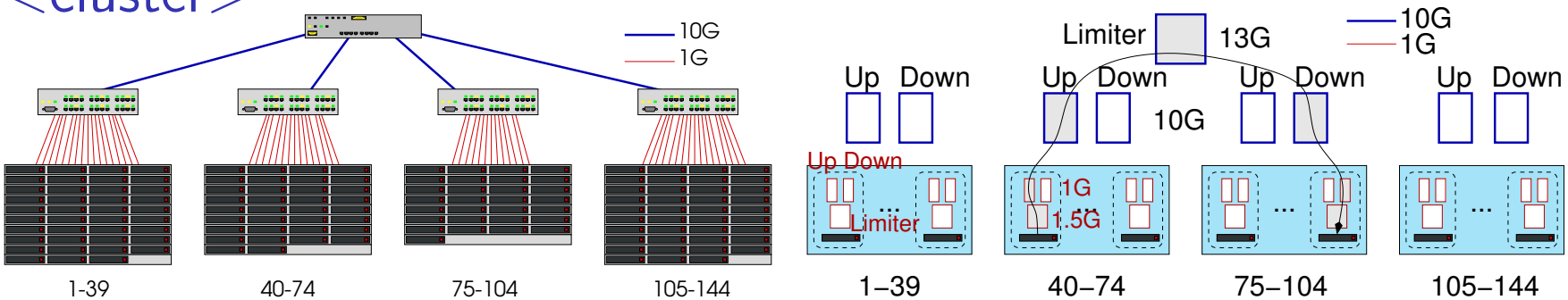


Defining Platforms

Best Part of Simulation

- Test your application on the platform of your dreams!
- SimGrid accepts XML descriptions (or C generators for non-SMPI)
- The same formalism for DataGrid, P2P, Cloud and HPC platforms
- Versatility allows combined experiments (Clouds+HPC is hype)
- Specific tags for classical constructs

<cluster>



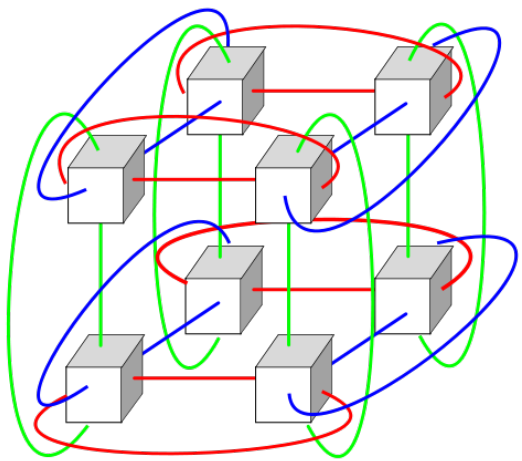

```
<cluster id="AS_sgraphene1" prefix="graphene-" suffix=".nancy.grid5000.fr"
         radical="1-39" power="16.673E9" bw="1.25E8" lat="2.4E-5"
         limiter_link="1.875E8" loopback_lat="1.5E-9" loopback_bw="6000000000"/>
```

Other HPC Typical Topologies

Torus

- <clusters> are too simplistic
- C. Heinrich prototyped a torus topology when intern at UIUC (integrated)
- Creating a n-dimension torus:

```
<cluster id="torus_cluster" radical="0-7" power="1Gf"
  bw="125MBps" lat="50us" topology="TORUS" topo_parameters="2,2,2"/>
```

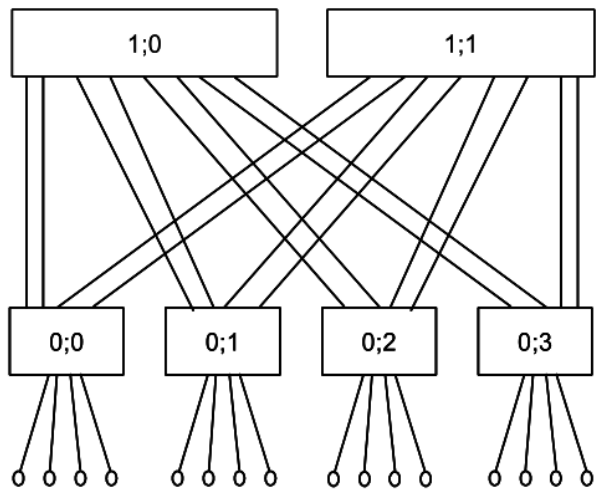


Some bits of caution

- This was not (in)validated yet. Use at own risk in production

Fat Trees

- Fat-tree network topology/routing was added to SimGrid
- Any Parallel Ports Tree Fat-tree (PGFT) cluster can be generated and instanciated in one line of XML
  - *D-Mod-K Routing Providing Non-Blocking Traffic for Shift Permutations on Real Life Fat Trees*, by Eitan Zahavi




```
<cluster id="fat_tree_cluster" radical="0-15" power="1Gf" bw="125MBps"
  lat="50us" topology="FAT_TREE" topo_parameters="2;4,4;1,2;1,2"/>
```

- TODO: (In)validation !

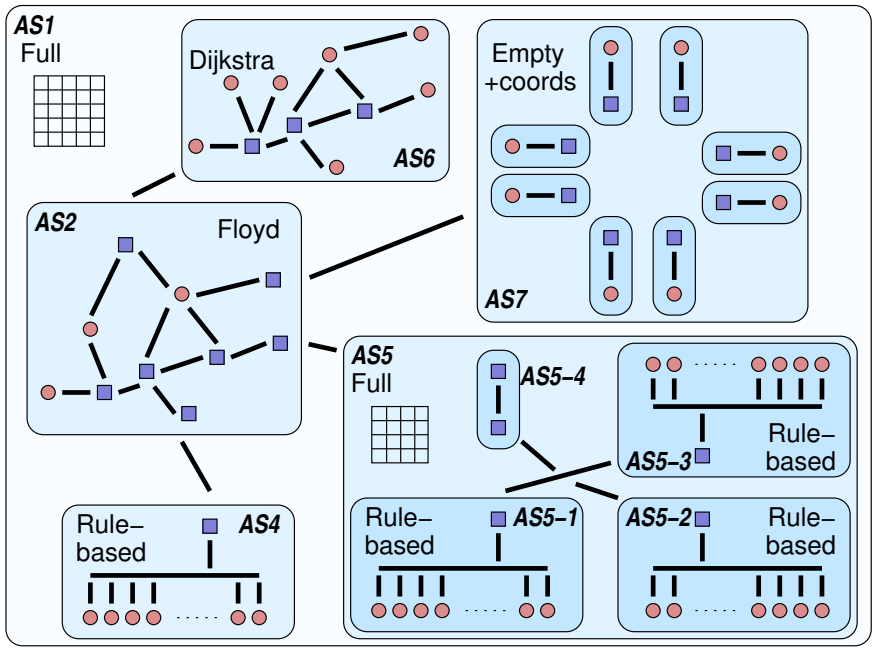
More on Platform Description in SimGrid

Versatile yet Scalable Platform Descriptions

- Hierarchical organization in ASes
  - ⇝ cuts down complexity
  - ⇝ recursive routing
- Efficient on each classical structures
  Flat, Floyd, Star, Coordinate-based
- Allow bypass at any level

⇝ Grid'5000 platform in 22KiB
(10 sites, 40 clusters, 1500 nodes)

⇝ King's dataset in 290KiB
(2500 nodes, coordinate-based)



Richer Platforms

- XML can specify some external load (power variations)
- XML can describe host and link failures (but SMPI don't like it yet)
- XML can specify the energy consumption of components with DVFS
- You can generate random platforms in C

Collectives

Real worlds (OpenMPI, MPICH)

- Dynamic selection of tuned algorithms
- Depends on the number of processes and message size
- Known to have a huge impact on application performance

Simulated world (SMPI)

- All (non MPI 3.0) algorithms of OpenMPI, MPICH, and STAR-MPI available
- `http://simgrid.gforge.inria.fr/simgrid/latest/doc/group__SMPI__API.html#SMPI_collective_algorithms`
- Configure the used algorithm with option `--cfg=smpi/coll_name:algo_name`
  Example: `--cfg=smpi/alltoall:pair`
- Configure the used automatic selector:
  - OpenMPI 1.7: `--cfg=smpi/coll_selector:ompi`
  - MPICH 3.0.4: `--cfg=smpi/coll_selector:mpich`
- Easy, isn't it?

SMPI runs on a single node

This makes things easier

- You can run it on your laptop if you want
- Don't let real life interfering with your experiments

Well it's getting too small here

- Folding a large HPC application on a laptop does not always fit
- You want to fold memory, reduce the footprint
- You want to sample execution iterations, to speed up the execution
- (the next slides explain how)

Bits of caution

- Only forcefully fold data-independent applications, silly!
- Do not try to speed up the other applications this way

Reducing the Memory Footprint

- Idea: Share arrays between processes (allocate once, use plenty)
  - Pros: Simulated times stay valid
  - Cons: Computed results become erroneous
- HowTo: Replace malloc/free in your code with these macros:

```
double* data = (double*)SMPI_SHARED_MALLOC(size);
SMPI_SHARED_FREE(data);
```

- Exact behavior controled by option `smpi/shared-malloc` :

  local: each call location returns one block, shared between processes
  global: all blocks are mmaped onto the same physical block (default)
  no: switch back to the real malloc semantic with one block per call

- For sparse data, where only parts are useful to the application logic:

```
SMPI_PARTIAL_SHARED_MALLOC(size, offsets, offsets_count)
SMPI_SHARED_FREE
```

  e.g. `SMPI_PARTIAL_SHARED_MALLOC(500, {27,42 , 100,200}, 2)`
  ⇝ mem[27...41] and mem[100...199] are shared
  ⇝ Other area remain specific to each malloc call

Please refer to the documentation for more information.

Reducing the Simulation Time

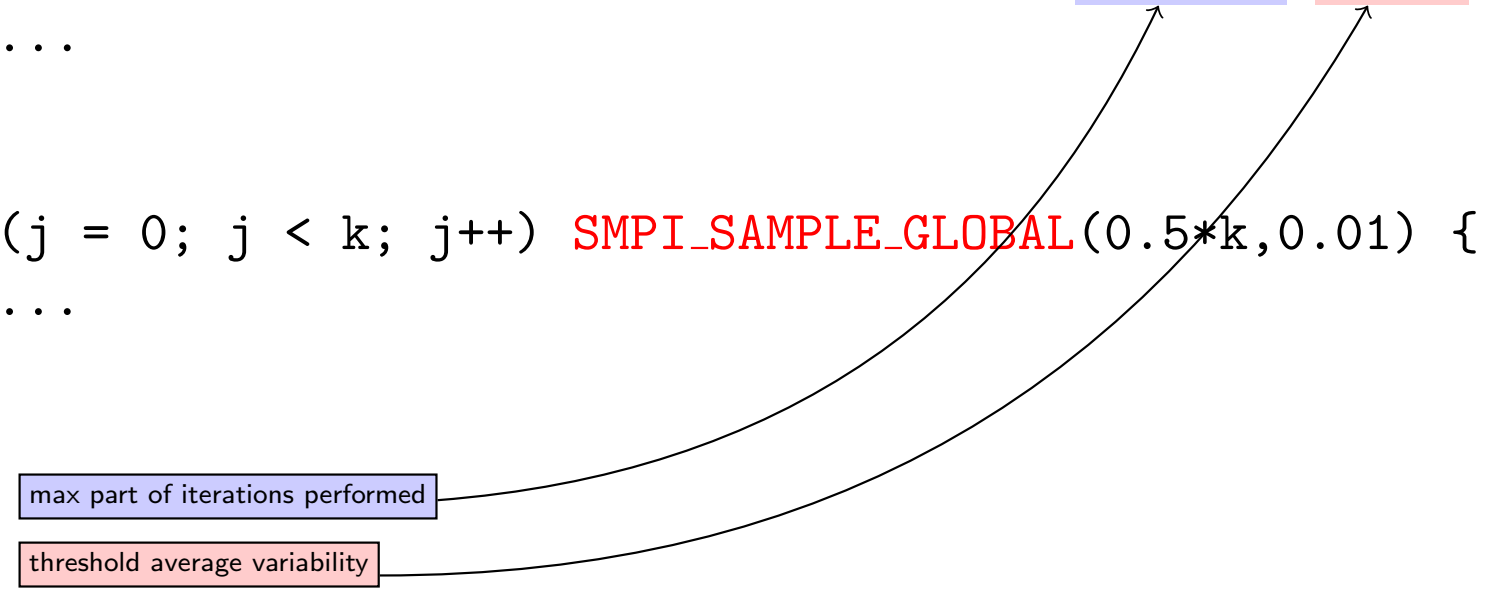
- Idea: Do not execute all the iterations
- Use sampling instead
  - LOCAL: each process executes a specified number of iterations
  - GLOBAL: a specified number of samples is produced by all processors
- Remaining iterations are replaced by average of measured values
- Implemented as (optional) macros

```
for(i = 0; i < n; i++) SMPI_SAMPLE_LOCAL( 0.75*n , 0.01 ){
    ...
}
...
for(j = 0; j < k; j++) SMPI_SAMPLE_GLOBAL(0.5*k,0.01) {
    ...
}
```

Reducing the Simulation Time

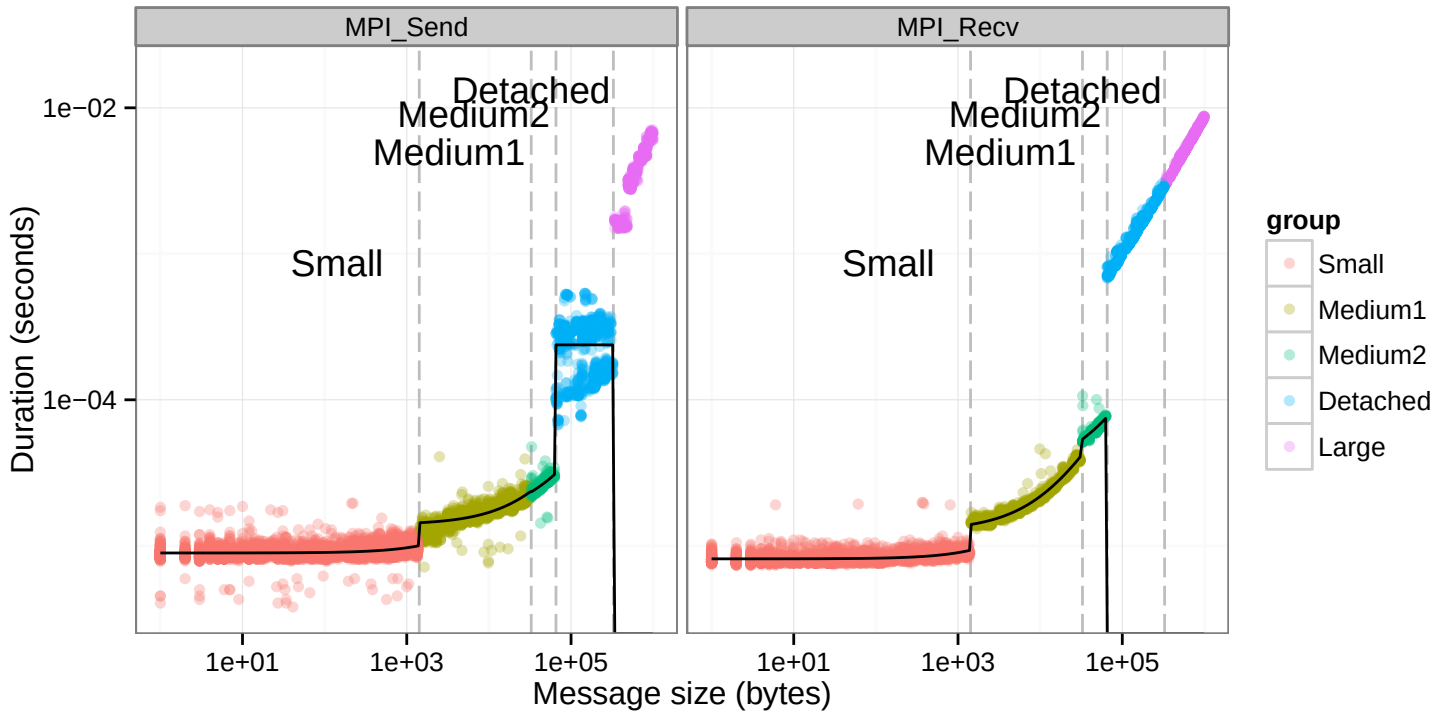
- Idea: Do not execute all the iterations
- Use sampling instead
  - LOCAL: each process executes a specified number of iterations
  - GLOBAL: a specified number of samples is produced by all processors
- Remaining iterations are replaced by average of measured values
- Implemented as (optional) macros

```
for(i = 0; i < n; i++) SMPI_SAMPLE_LOCAL( 0.75*n , 0.01 ){
    ...
}
...
for(j = 0; j < k; j++) SMPI_SAMPLE_GLOBAL(0.5*k,0.01) {
    ...
}
```



Point-to-point Communication

P2P Experimental measurements on an Ethernet cluster with OpenMPI 1.6



Calibration of simple MPI ping pong experiments

- Randomized sizes to characterize behavior : 1B-1MB
- Three modes identified in this case : eager, detached (only for sender), and rendez-vous
- Piece-wise regression for injecting times for eager sends/receives

Point-to-point Communication

SMPI parameters for the platform computed from these experiments :

- Thresholds for modes

```
<prop id="smpi/async_small_thres" value="65536"/>
<prop id="smpi/send_is_detached_thres" value="327680"/>
```

- Factors for latency, bandwidth for various message sizes, computed from regression

```
<prop id="smpi/bw_factor" value="size1:x;size2:y ..."/>
<prop id="smpi/lat_factor" value="size1:x;size2:y ..."/>
```

- Timings to inject in Send and Receive asynchronous Operations (not always the same for Send and Isend)

```
<prop id="smpi/os" value="size1:x1:x2;size2:y1:y2 ..."/>
<prop id="smpi/ois" value="size1:x1:x2;size2:y1:y2 ..."/>
<prop id="smpi/or" value="size1:x1:x2;size2:y1:y2 ..."/>
```

- Values are provided by R analysis of the experimental traces of a small benchmark. Please contact us for any help
- TODO: WTFM