High Performance
Computing &
Big Data Services

hpc.uni.lu
hpc@uni.lu
@ULHPC

LUXEMBOURG
LET'S MAKE IT HAPPEN

Practical aspects of hybrid system simulation and analysis

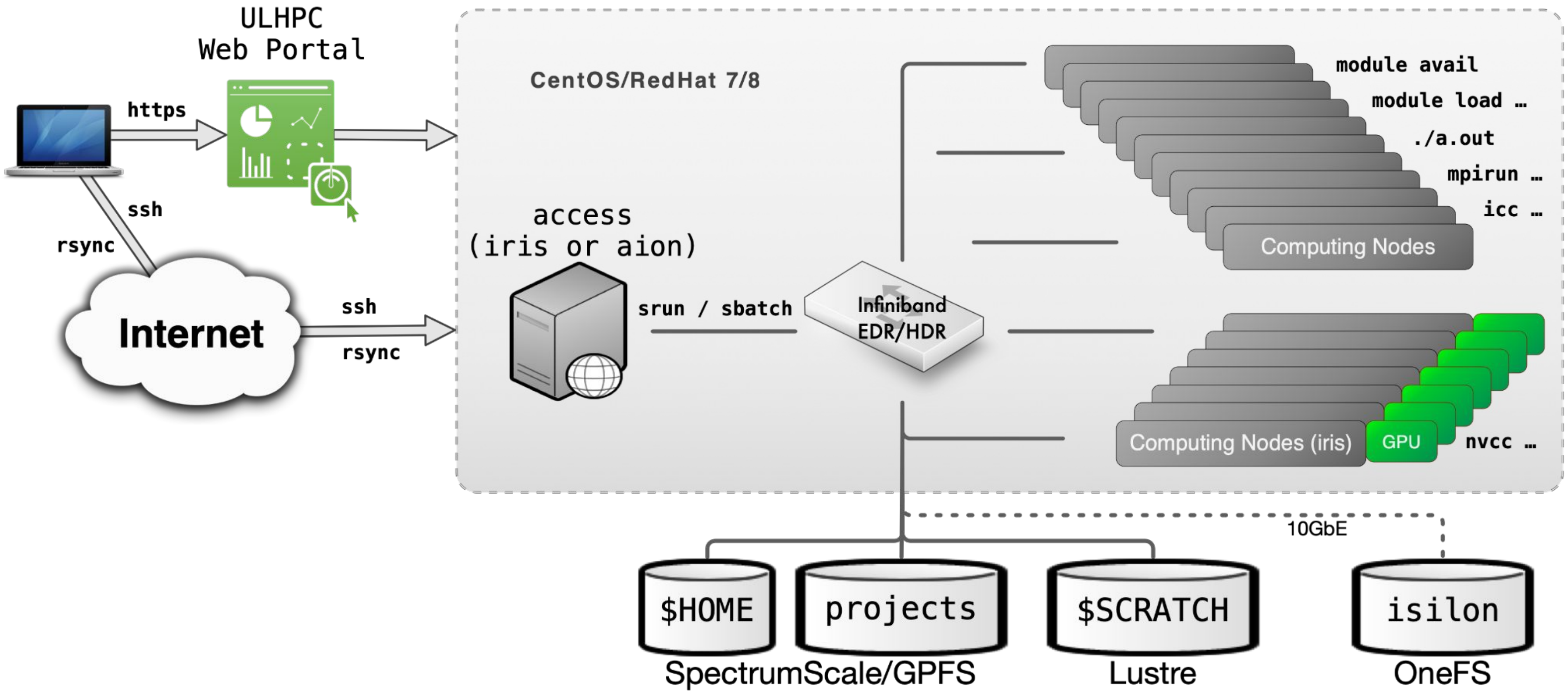
Georgios Kafanas

UNIVERSITÉ DU LUXEMBOURG

Outline

- Dynamic system simulation in HPC systems
- Safety analysis of controllers
- Overview of Modelica and OpenModelica
- Using OpenModelica in HPC systems
- Ariadne and other application specific packages
- Avenues of improvement

The University of Luxembourg HPC cluster



Example of applications deployed in UL HPC

Engineering

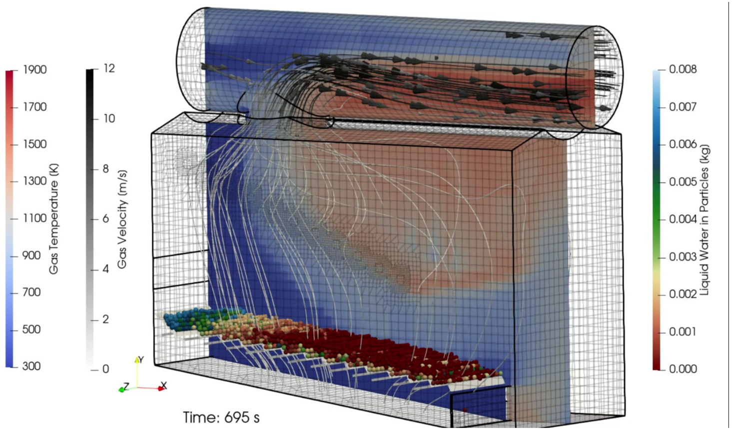
XDEM: Discrete Element Method solver with extensions for multiphysics and chemistry.

Physics

LibMBD: An implementation of the many-body dispersion (MBD) method, used in applications such crystal structure prediction.

What about dynamical systems?

Biomass furnace simulation with XDEM



Example of applications deployed in UL HPC

Engineering

XDEM: Discrete Element Method solver with extensions for multiphysics and chemistry.

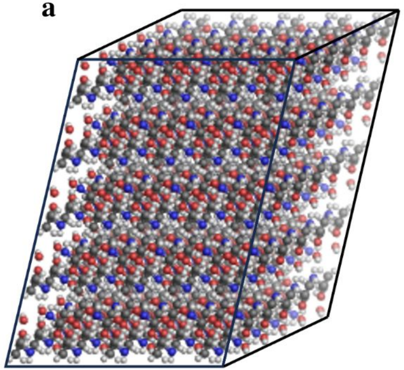
Physics

LibMBD: An implementation of the many-body dispersion (MBD) method, used in applications such crystal structure prediction.

What about dynamical systems?

Electronic structure calculation with LibMDB

a



a: J. Hermann et. al., arXiv:2308.03140 (2023)

When HPC can be useful?

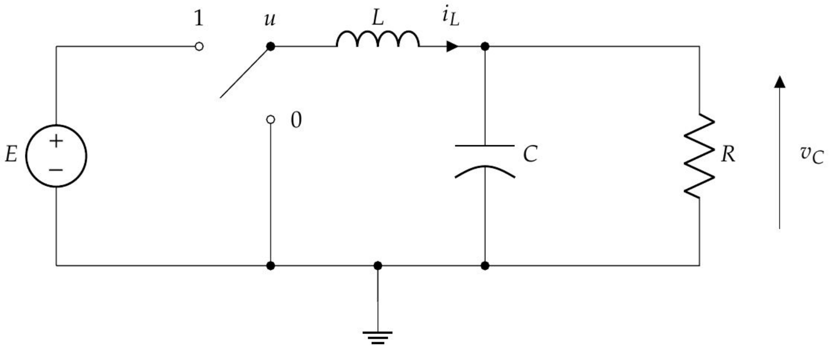
Running multiple independent simulation in parallel

Parametric investigations are easily parallelizable. When more than few 1000 simulation runs are required, the effort to parallelize the code is unusually worth the speedup.

Simulating very large systems

Laptop / desktop machines usually have 2 to 16 cores. If the system can be split in multiple independent communicating components, you can utilize 1000s of cores in a typical HPC system.

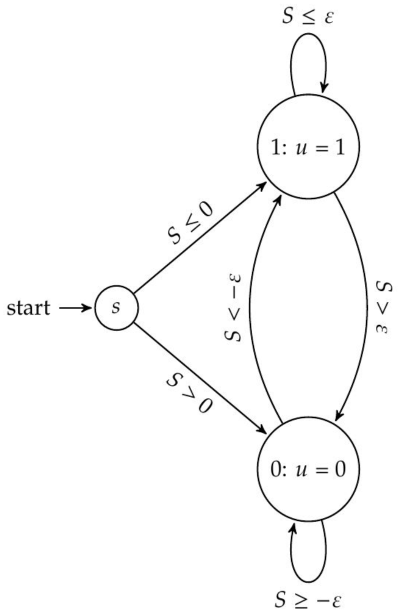
The buck converter



$$\begin{cases} D_t i_L = \frac{1}{L}(-v_C + u v_{\text{in}}) \\ D_t v_C = \frac{1}{C}\left(i_L - \frac{v_C}{R}\right). \end{cases}$$

- Objective to pin at zero: $h = v_C - v_C^*,$
- Sliding surface: $S = \alpha D_t h + h = \left(1 - \frac{\alpha}{RC}\right) v_C + \frac{\alpha}{C} i_L - v_C^*$
- Dynamics on the sliding mode: $S = 0 \Rightarrow D_t h = -\frac{1}{\alpha} h$

Switched controller



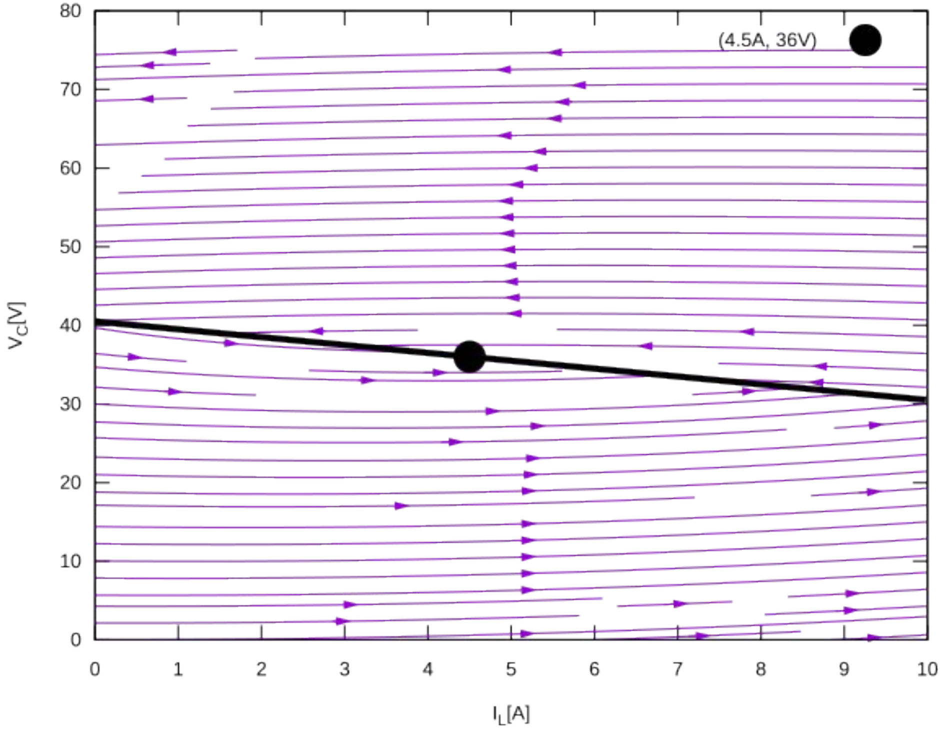
- Continuous control signal
- For simplicity, assume that the initial value of the controller is 'reasonable'
- System dynamics:

$$\begin{cases} D_t i_L = \frac{1}{L}\left(-v_C + u v_{\text{in}}\right) \\ D_t v_C = \frac{1}{C}\left(i_L - \frac{v_C}{R}\right). \end{cases}$$

- Sliding surface:

$$S = \left(1 - \frac{\alpha}{RC}\right) v_C + \frac{\alpha}{C} i_L - v_C^*$$

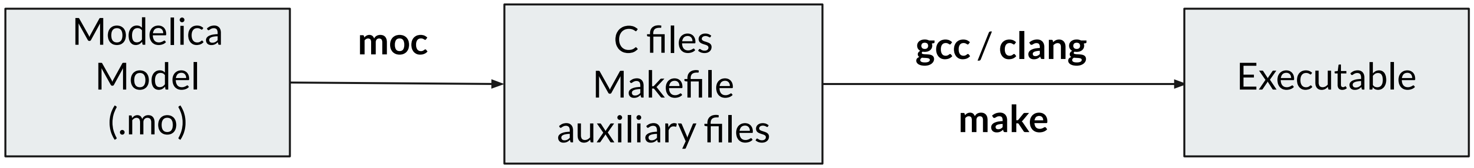
Converter dynamics



Time domain simulation

Modelica and OpenModelica

- High level domain specific language for the simulation of dynamical systems
- OpenModelica is a reference implementation that also offers good performance



OpenModelica Compiler (omc)

- Generates C code
- Various flags for code generation including OpenMP and OpenCL

A model for the buck converter in Modelica

- Modelica supports pure functions and equation models
- The switching manifold is a pure function, takes some inputs and produces an output

```
// Switching function can be loaded from a different module for flexibility
function switching_function
  input Real i_L;
  input Real v_C;
  input Real alpha;
  input Real v_C_s;
  input Real R;
  input Real C;
  output Real s;
algorithm
  s := (1 - alpha/(R*C))*v_C + (alpha/C)*i_L - v_C_s;
end switching_function;
```

A model for the buck converter in Modelica

- Dynamic components are modeled with equations
- No causal relations are specified in equation models

```
model Converter
  Real v_C;
  Real i_L;
  Real u;
  parameter Real L;
  parameter Real C;
  parameter Real R;
  parameter Real E;
  parameter Real i_L_0;
  parameter Real v_C_0;
equation
  der(i_L) = (1/L)*( -v_C + u*E );
  der(v_C) = (1/C)*( i_L - v_C/R );
initial equation
  i_L = i_L_0;
  v_C = v_C_0;
end Converter;

```

```
model Controller
  Real v_C;
  Real i_L;
  discrete Real u;
  parameter Real alpha;
  parameter Real C;
  parameter Real R;
  parameter Real v_C_s;
  parameter Real epsilon;
  parameter Real u_0;
protected
  Real h;
equation
  h = switching_function(i_L, v_C, alpha, v_C_s, R, C);
  when {h >= epsilon, h <= -epsilon} then
    u = if h >= 0 then 0 else 1;
  end when;
initial equation
  u = u_0;
end Controller;

```

A model for the buck converter in Modelica

- Equations link systems
- Large models are constructed in a modular fashion
- Currently no support for discrete signals: u is a real valued signal

```
model Automaton
  // System parameters
  parameter Real L = 1.7e-3;
  parameter Real C = 0.6e-3;
  parameter Real R = 8;
  parameter Real E = 48;
  parameter Real alpha = 0.000599944;
  parameter Real epsilon = 0.316227766;
  parameter Real v_C_s = 36;
  // Initial conditions
  parameter Real i_L_0 = 0;
  parameter Real v_C_0 = 0;

  Converter converter(L=L, C=C, R=R, E=E, i_L_0=i_L_0, v_C_0=v_C_0);
  Controller controller(alpha=alpha, epsilon=epsilon, R=R, C=C, v_C_s=v_C_s, u_0=u_0);

protected
  parameter Real s = switching_function(i_L_0, v_C_0, alpha, v_C_s, R, C);
  parameter Real u_0 = if s > 0 then 0 else 1;

equation
  converter.i_L = controller.i_L;
  converter.v_C = controller.v_C;
  converter.u = controller.u;

end Automaton;
```

Generating the model code

```
$ omc --hpcomCode=openmp --simulationCg Automaton.mo
```

`--simulationCg`	Simulation code generation, creates C files, auxiliary files, and a Makefile (`Automaton.makefile`)
`--hpcomCode=openmp`	Enables the use of OpenMp

```
$ ls
Automaton_01exo.c  Automaton_08bnd.c  Automaton_13opt.c  Automaton_18spd.c        Automaton_JacA.bin
Automaton_02nls.c  Automaton_09alg.c  Automaton_13opt.h  Automaton.c              Automaton_literals.h
Automaton_03lsy.c  Automaton_10asr.c  Automaton_14lnz.c  Automaton_functions.c    Automaton.makefile
Automaton_04set.c  Automaton_11mix.c  Automaton_15syn.c  Automaton_functions.h    Automaton_model.h
Automaton_05evt.c  Automaton_11mix.h  Automaton_16dae.c  Automaton_includes.h     Automaton_records.c
Automaton_06inz.c  Automaton_12jac.c  Automaton_16dae.h  Automaton_info.json
Automaton_07dly.c  Automaton_12jac.h  Automaton_17inl.c  Automaton_init.xml
```

Compiling and running the model code

```
$ make -j -f Automaton.makefile
```

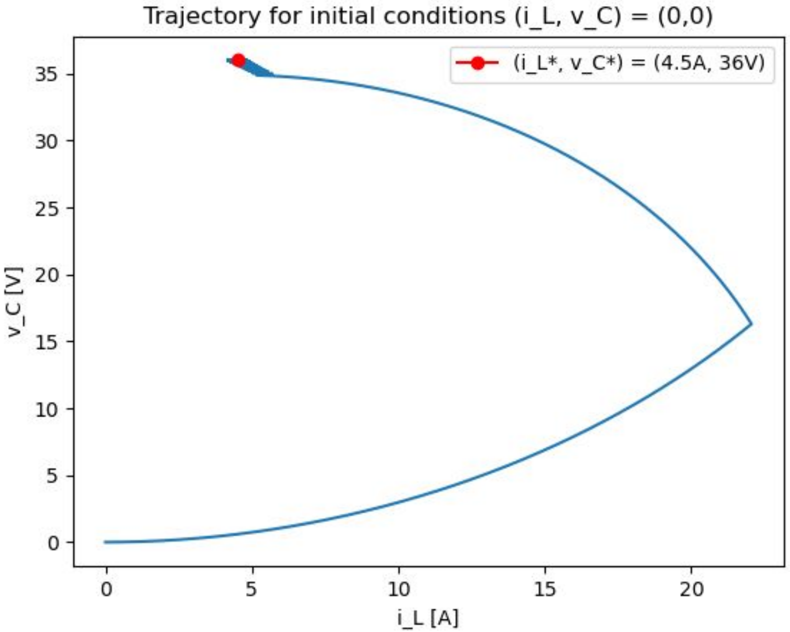
- Produces the executable: `Automaton`
- Newer versions of OpenModelica use clang as the default backend

```
$ ./Automaton -override \
  stopTime='0.02',stepSize='4e-7',i_L_0='0.0',v_C_0='0.0',outputFormat='csv' \
  -r='simulation_results.csv' \
  -parmodNumThreads='1'"
```

- The result is the file: `simulation_results.csv`
- Contains a time series with all the signals in the simulation

Safety analysis

Output of a single simulation

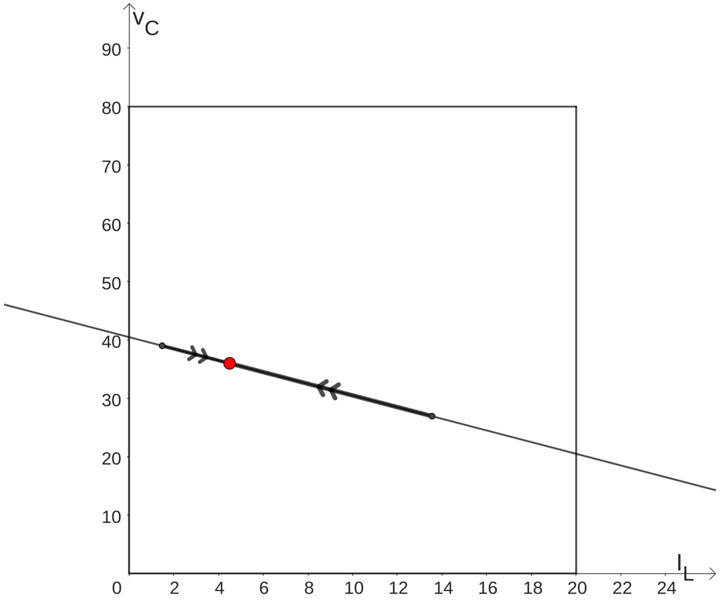


- Single simulation starting with the converter off
- Large initial excursion in the inductor current
- Repeat the simulation for a range of initial conditions.

Objective: Find range of initial conditions where the current transient does not exceed 20A.

Safety analysis of the sliding mode controller

- The maximum inductor current for each initial condition
- Inductor:
  - initial current [0, 20A]
  - step 0.2A
- Capacitor:
  - initial voltage [0, 80V]
  - step 1V
- 8181 time domain simulations



C. J. Tomlin, et. al. 2000. "A game theoretic approach to controller design for hybrid systems." *Proceedings of the IEEE* 88, no 7: 949-970

Performing an initial condition sweep

```
  cmd = [ "./Automaton",
      "-override",  override.to_string(),
      "-r", output_prefix + ".mat",
      "-parmodNumThreads", "1"
  ]
```

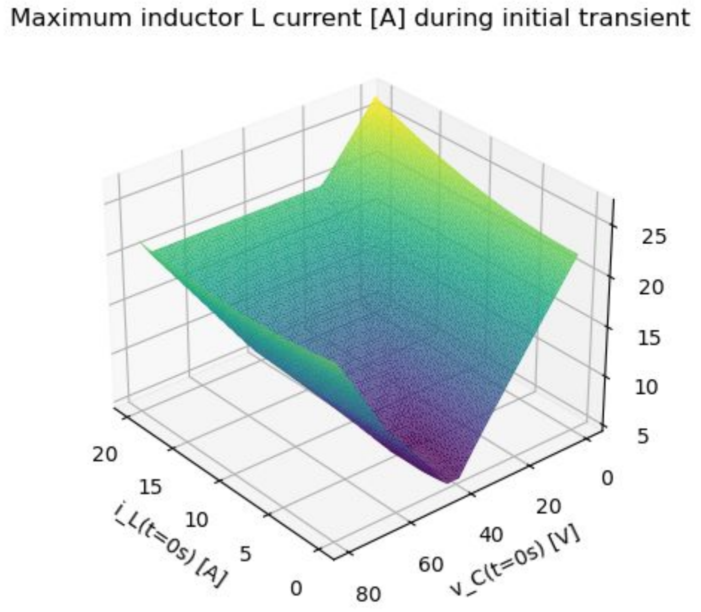
```
try:
  result = subprocess.run( cmd, capture_output=True, text=True, timeout=timeout, cwd=working_directory, env=local_env )
except subprocess.TimeoutExpired as err:
  remove_trajectory_file(trajectory_file_name)
  raise TimeoutException(
    "".join( ["Simulation exceeded the timeout=", str(timeout), "s",  error_message()] )
  )
```

Performing an initial condition sweep

Implemented the parallelization in Python, spawning multiple processes with a thread pool.

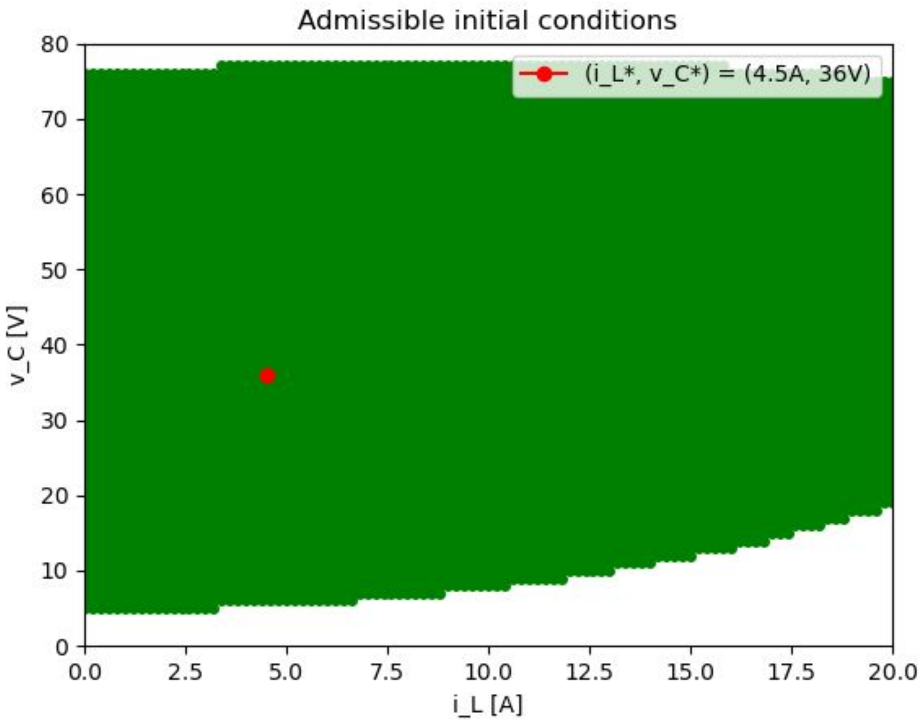
```python
output_file.write( "i_L_0,v_C_0,min_i_L,max_i_L,min_v_C,max_v_C,u_0\n" );

I_L_0 = generate_vector(I_L_0_min, I_L_0_step, I_L_0_max)
V_C_0 = generate_vector(V_C_0_min, V_C_0_step, V_C_0_max)
meshgrid = itertools.product(I_L_0, V_C_0)
with ThreadPoolExecutor( max_workers = n_procs ) as pool:
  job = {}
  job_id = 0
  for initial_conditions in meshgrid:
    i_L_0, v_C_0 = initial_conditions
    job_data = JobData( job_id, i_L_0, v_C_0, start_time, stop_time, n_samples, scratch_prefix )
    job[job_id] = pool.submit( simulate_initial_conditions_case, job_data, process_timeout )
    job_id += 1

  for job_id in list( job.keys() ):
    results = get_results( job[job_id], job_id, pool )
    if non_zero_time_simulation( results ):
      write_statistics( results.i_L_0, results.v_C_0, results.statistics, output_file )
```

Results



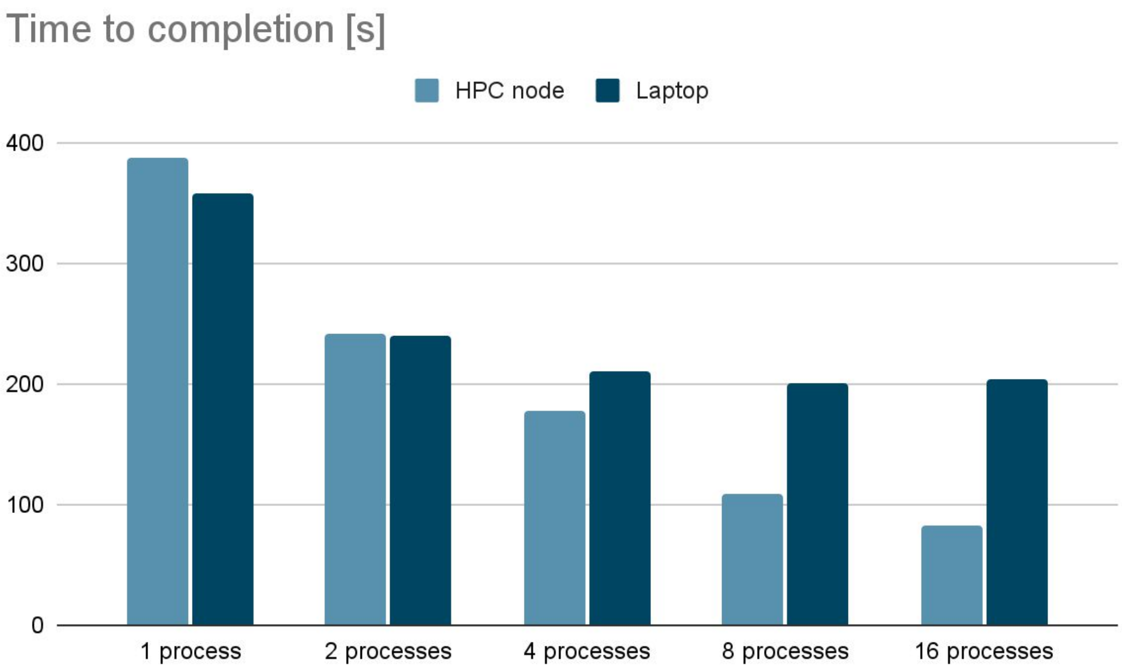
- The maximum inductor current for each initial condition
- Inductor:
  - initial current [0, 20A]
  - step 0.2A
- Capacitor:
  - initial voltage [0, 80V]
  - step 1V
- 8181 time domain simulations

Results



- Initial conditions from which the maximum inductor current is less than 20A
- Inductor:
  - initial current [0, 20A]
  - step 0.2A
- Capacitor:
  - initial voltage [0, 80V]
  - step 1V
- 8181 time domain simulations

Performance



8181 time domain simulations

- Speedup limited by file system operations
- Avoid writing files using OMPython

Running multiple simulations in parallel

Multiple tools to execute a simulation in parallel

GNU Parallel

- Simple to use
- Limited flexibility

Python

- Can parallelize code with threading or multiprocessing packages such as `concurrent.futures` or `multiprocessing`
- Can use OpenModelica’s Python interface OMPython (based on ZeroMQ)

Analysis tools

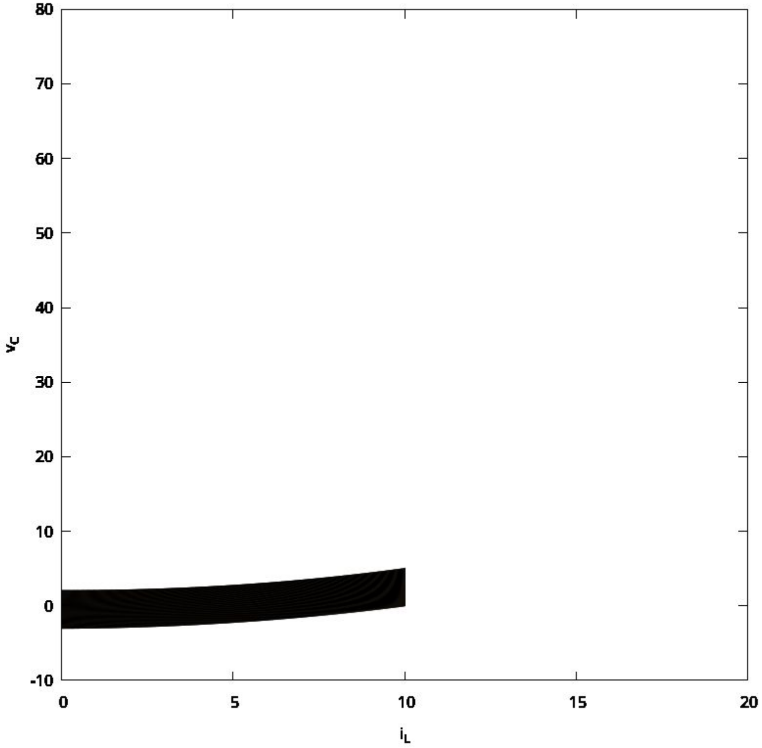
Ariadne

Library implementing:

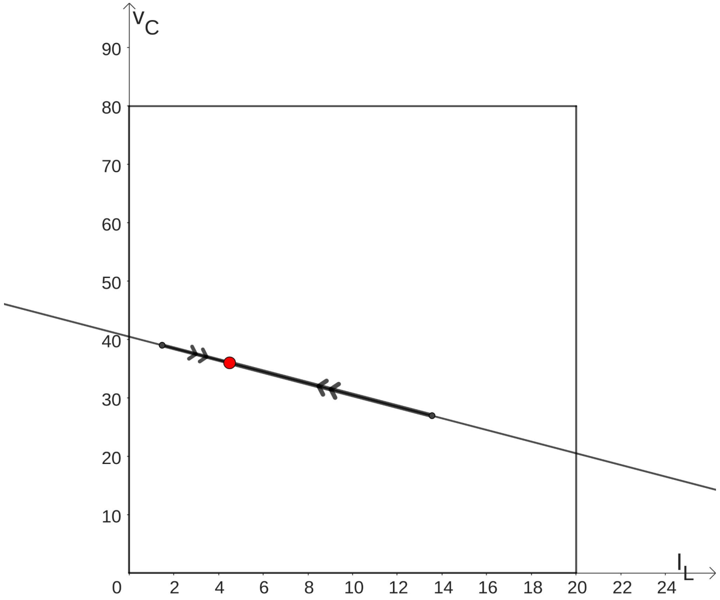
- Rigorous numerics
- Reachability analysis

Both problems are computationally expensive

Collins, P. 2020. "Computable analysis with applications to dynamic systems." *Mathematical Structures in Computer Science* 30, no. 2 (2020): 173-233



System analysis with Ariadne



- Reachability analysis can be used to verify safety properties

But not quite stable yet:

- No API to extract results (output available in `.plt` format only)
- Initial conditions can only be multidimensional intervals
- Terminating based on convergence or time limit

JuliaReach

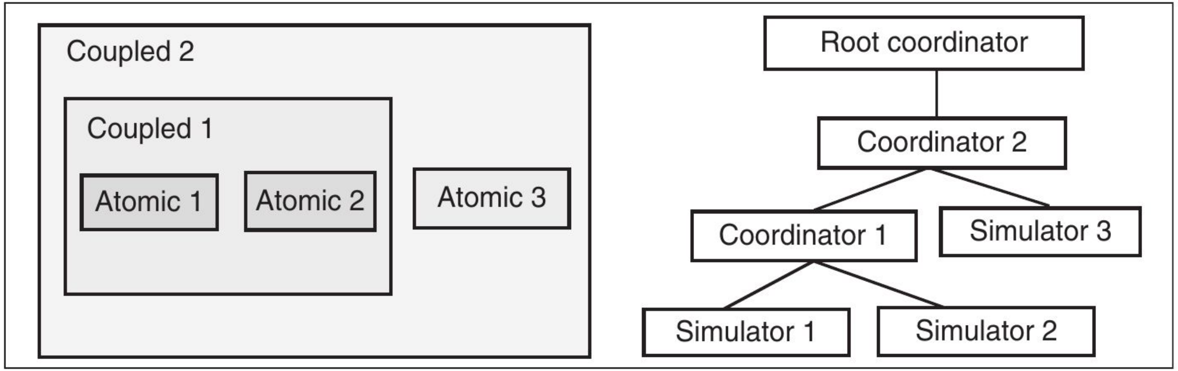
- Independent package focusing on reachability analysis
- Hybrid systems are currently supported
- Highly fragmented implementation
- Non-uniform interface

Bogomolov S., et. al. 2019. “JuliaReach: a toolbox for set-based reachability.” *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*: 39-44

Computation frameworks

PowerDEVS

- Time domain simulation
- Based on a modeling framework which automates code generation for the simulator
- The design principle has been integrated into OpenModelica

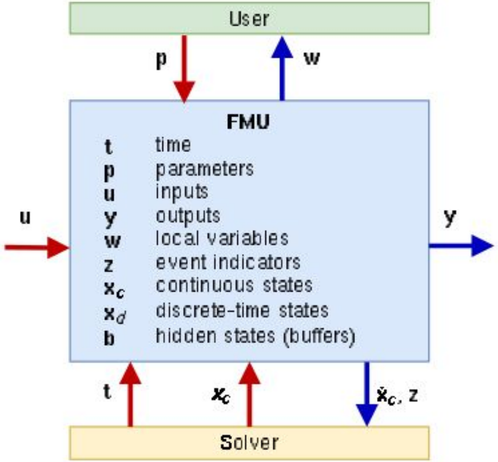


Bergero, F., et. al. 2011. "PowerDEVS: a tool for hybrid system modeling and real-time simulation." *SIMULATION* 87, no. 1-2: 113-132

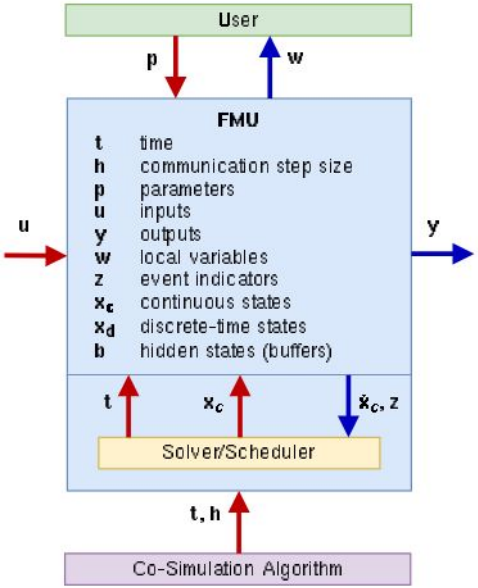
Functional Mock-up Interface

- Supported by OpenModelica and other simulation environments
- Describes 3 interface types:

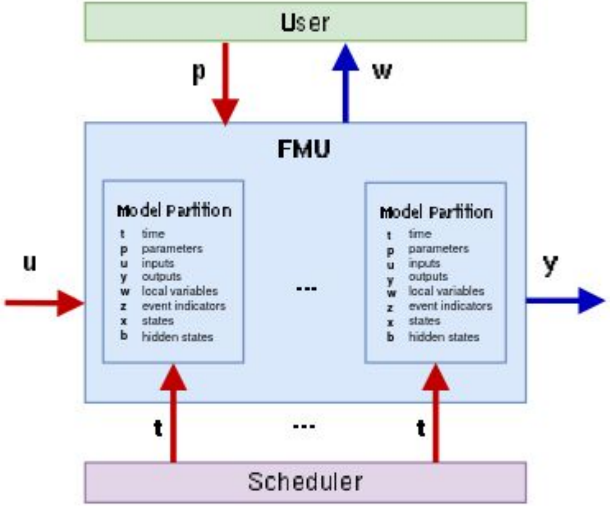
Model exchange



Co-simulation



Scheduled execution



The way forward?

Interfaces for reachability analysis?

Reachability analysis is both useful and computationally intensive, but tools that would allow its use in HPC systems are missing.

- Theoretical foundations need to be more unified
- Takes time to develop the abstractions necessary for a good interface
- More communication

Thank you!

Any questions?