Design Approach for a Generic and Scalable Framework for Parallel FMU Simulations

Martin Flehmig, Marc Hartung, Marcus Walther

Linköping, 02. February 2015

E-Mail: martin.flehmig@tu-dresden.de
Outline

1. Introduction and Motivation
2. Coupled Simulations Using FMI
3. Design Approach
4. Summary and Outlook
Outline

1. Introduction and Motivation
2. Coupled Simulations Using FMI
3. Design Approach
4. Summary and Outlook
The three main tasks/goals within the HPC-OM project are:

1. Domain independent parallelization of Modelica simulations.
   - Equation based parallelization using task graph.
   - Algorithms for task merging and clustering.
   - Implementation of various schedulers.
   - Code generation for OpenMP, PThreads and Intel TBB.
   - Exploiting repeated structures and vectorization in Modelica.

2. Parallel time integration.

3. Coupling of interactive simulations and an HPC system.
Tasks within HPC-OM

The three main tasks/goals within the HPC-OM project are:

1. Domain independent parallelization of Modelica simulations.
   - Equation based parallelization using task graph.
   - Algorithms for task merging and clustering.
   - Implementation of various schedulers.
   - Code generation for OpenMP, PThreads and Intel TBB.
   - Exploiting repeated structures and vectorization in Modelica.

2. Parallel time integration.

3. Coupling of interactive simulations and an HPC system.

4. *High speedups.*
Task Graph Parallelization

...is promising, but current restrictions are:

- Model dependent benefit, because
  - tasks are too lightweight,
  - one large equation system thwarts whole parallel computation.
- OpenModelica has some issues regarding large models.
**How to Achieve High Speedups?**

**Idea:** Build simulation from several FMUs to obtain multiple levels of parallelism:

- Task graph parallelization with FMUs as nodes.
- Use task graph parallelization generated by OMC in each FMU.
Outline

1. Introduction and Motivation
2. Coupled Simulations Using FMI
3. Design Approach
4. Summary and Outlook
Can bring together sub-models from distinctive authoring tools (language, libraries, solvers, etc.).

Models can be used in already existing production codes and frameworks.

Protects intellectual property.
Can bring together sub-models from distinctive authoring tools (language, libraries, solvers, etc.).

Models can be used in already existing production codes and frameworks.

Protects intellectual property.

Single-threaded simulations using coupled FMUs have limitations:

- Suitable for real time applications?
- Large and complex models have long simulation execution times and demand for high memory consumption.
Can bring together sub-models from distinctive authoring tools (language, libraries, solvers, etc.).
Models can be used in already existing production codes and frameworks.
Protects intellectual property.

Single-threaded simulations using coupled FMUs have limitations:
- Suitable for real time applications?
- Large and complex models have long simulation execution times and demand for high memory consumption.

Therefore, FMU simulations exploiting today’s multi-core hardware are needed.
Synchronisation Step Approach

- Synchronisation of FMUs happens after predefined time intervals.
- At every synchronisation point, numerical error is calculated.
- In between inputs are extrapolated for single solver steps.

![Diagram showing error control and extrapolation in synchronous steps]
Synchronisation Step Approach

- Synchronisation of FMUs happens after predefined time intervals.
- At every synchronisation point, numerical error is calculated.
- In between inputs are extrapolated for single solver steps.

+ No communication between synchronisation steps.
+ Clear and straightforward implementation possible.
- Revert FMUs to last synchronisation point or even rerun simulation.
- Communication leads to delays during synchronisation points.
Generic Approach - Idea

- Values of every valid solver step are communicated.
- Dependent FMUs take most recent values as inputs.
- Solvers can base the error estimation on profound data.
- Less interfering in solver step size.

[Diagram showing the interaction between FMU 1 and FMU 2 with error control and extra-/interpolation highlighted.]
Generic Approach - Aspects and Challenges

Aspects

+ Just single solver steps need to be rerun.
+ Replaces unsafe extrapolation with interpolation.
+ Direct error handling, i.e.,
  - change of numerical behaviour is treated on occurrence,
  - solver step size depends only on input values, not on synch. points.
- Increasing communication effort.
- Sophisticated implementation with complex data structures.

Challenges

Asynchronous communication is needed.

→ Field of parallel computing provides several solutions.

FMUs need to be smartly distributed on system for high efficiency.

→ Knowledge transfer from task scheduling.

Adoptable to different simulation setups.

! Requires interchangeable modular structure of the system and components.
Generic Approach - Aspects and Challenges

Aspects
+ Just single solver steps need to be rerun.
+ Replaces unsafe extrapolation with interpolation.
+ Direct error handling, i.e.,
  • change of numerical behaviour is treated on occurrence,
  • solver step size depends only on input values, not on synch. points.
- Increasing communication effort.
- Sophisticated implementation with complex data structures.

Challenges
• Asynchronous communication is needed.

• FMUs need to be smartly distributed on system for high efficiency.

• Adoptable to different simulation setups.
Generic Approach - Aspects and Challenges

Aspects

+ Just single solver steps need to be rerun.
+ Replaces unsafe extrapolation with interpolation.
+ Direct error handling, i.e.,
  - change of numerical behaviour is treated on occurrence,
  - solver step size depends only on input values, not on synch. points.
- Increasing communication effort.
- Sophisticated implementation with complex data structures.

Challenges

- Asynchronous communication is needed.
  → Field of parallel computing provides several solutions.
- FMUs need to be smartly distributed on system for high efficiency.
- Adoptable to different simulation setups.
Generic Approach - Aspects and Challenges

Aspects
+ Just single solver steps need to be rerun.
+ Replaces unsafe extrapolation with interpolation.
+ Direct error handling, i.e.,
  - change of numerical behaviour is treated on occurrence,
  - solver step size depends only on input values, not on synch. points.
- Increasing communication effort.
- Sophisticated implementation with complex data structures.

Challenges
- Asynchronous communication is needed.
  → Field of parallel computing provides several solutions.
- FMUs need to be smartly distributed on system for high efficiency.
  → Knowledge transfer from task scheduling.
- Adoptable to different simulation setups.
Generic Approach - Aspects and Challenges

Aspects

+ Just single solver steps need to be rerun.
+ Replaces unsafe extrapolation with interpolation.
+ Direct error handling, i.e.,
  - change of numerical behaviour is treated on occurrence,
  - solver step size depends only on input values, not on synch. points.
- Increasing communication effort.
- Sophisticated implementation with complex data structures.

Challenges

- Asynchronous communication is needed.
  → Field of parallel computing provides several solutions.
- FMUs need to be smartly distributed on system for high efficiency.
  → Knowledge transfer from task scheduling.
- Adoptable to different simulation setups.
  ! Requires interchangeable modular structure of the system and components.
FMU Simulation Framework

Final goal is a generic and scalable framework for coupled FMU simulations.

Framework for Parallel FMU Simulation

- OpenMP
- MPI
- C++
- Error control/handling
- Simulation
- FMI
- Communication
- Data Management
- I/O
- Solver
- Initialization
- User friendliness
- Adaptive step sizes
- Generic
- Scalable
Outline

1. Introduction and Motivation
2. Coupled Simulations Using FMI
3. Design Approach
4. Summary and Outlook
Challenge: How to provide input/output data for asynchronous simulation?
**Challenge:** How to provide input/output data for asynchronous simulation?

**Answer:** Use buffers!

- Saves relevant state values of FMUs.
- Input values are written remotely.
- Interface for accessing input values on local storage.
Challenge: How to achieve efficient communication and handling of parallel FMU simulation?

Answer:
- Use a manager!
- Initiates shared and distributed memory writes for input values.
- Writes result data to file.
- Controls DataHistory, e.g., flushes unnecessary data.
- Interpolation/extrapolation of input data.
**DataManager**

**Challenge:** How to achieve efficient communication and handling of parallel FMU simulation?

**Answer:** Use a manager!

- Initiates shared and distributed memory writes for input values.
- Writes result data to file.
- Controls DataHistory, e.g., flushes unnecessary data.
- Interpolation/extrapolation of input data.

→ **Hides communication and data flow from solvers.**

![Diagram showing communication between FMUs through a buffer](image-url)
Outline

1. Introduction and Motivation
2. Coupled Simulations Using FMI
3. Design Approach
4. Summary and Outlook
Presented approach for efficient asynchronous parallel simulation of coupled FMUs.

Key features and main challenges have been identified:

- Use task graph parallelized FMUs generated from OpenModelica.
- Use model exchange FMUs in order to obtain asynchronous simulation.
  - Need scalable data structures and communication.
  - Need localized buffers to provide input values for FMUs.
Outlook

What has to be done?

- Finish implementation.
- Show scalability by performing large simulation with numerous FMUs.
- Find a fancy name for this piece of software - suggestions are welcome.
- Make a release available.
Thank you for your attention.
error control of input values only at synchronization points
in between extrapolated inputs only based on previous interval
in several cases FMUs need to be set back to last synchronization point
FMU Input Extrapolation

- error control of input values based on most recent values
- solver can directly check error after every step
- leading to higher numerical stability
- less interfering in solver step size
- more dynamical error handling possible