EFFICIENT IMPLEMENTATIONS AND CO-SIMULATION TECHNIQUES IN MULTIBODY SYSTEM DYNAMICS

Francisco Javier González Varela

Doctoral thesis

University of A Coruña

Ferrol, May 3rd, 2010

Outline

Introduction

Software Architecture for MBS Simulation

Linear Algebra Implementations

Parallelization

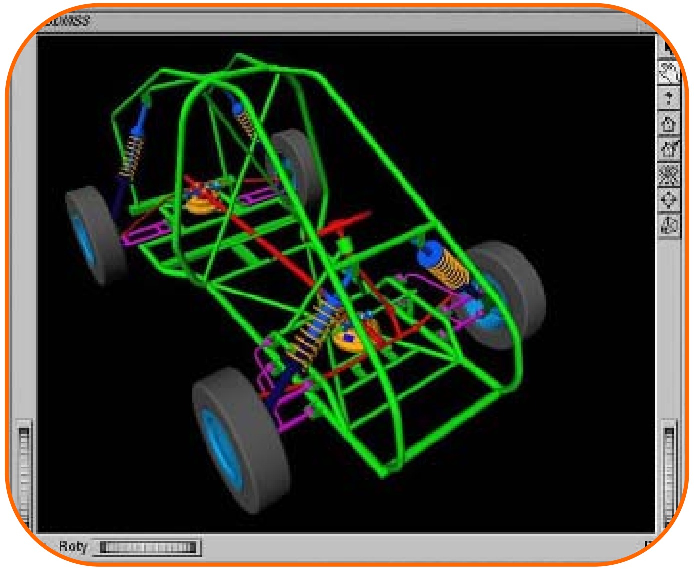
Integration with MATLAB/Simulink

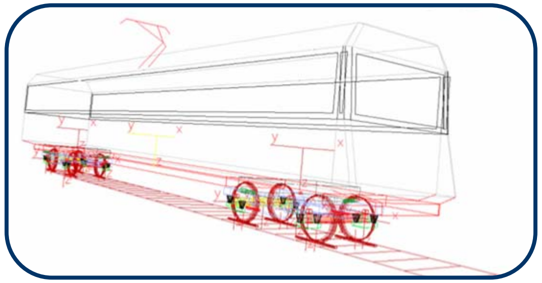
Multirate Co-simulation Methods

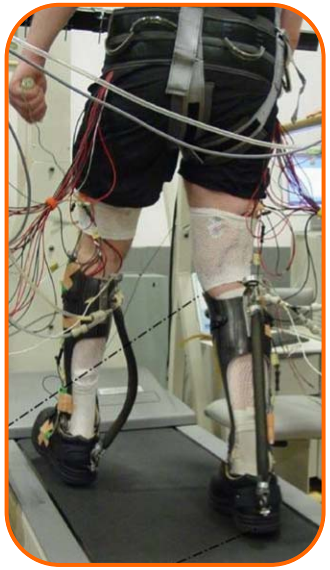
Conclusions and Future Research

Motivation

- **Multibody systems (MBS) dynamic simulation is present in a wide range of applications today**







- **MBS simulation is heavily dependent on available software features**
  - Simulated systems are very complex and often multi-disciplinary
  - High efficiency required in real-time applications and what-if analyses

Objectives of this thesis

- **Two main goals in current research in MBS dynamic simulation**
  - Efficiency
  - Addition of new functionality (multiphysics, contact, impacts, etc.)
- **1- Efficient implementations in MBS software**
  - Linear Algebra routines
  - Parallelization
- **2 - Communication with external packages**
  - Comparison of available communication techniques
  - Multirate co-simulation
- **Intermediate goal: MBS software architecture**

Outline

Introduction

Software Architecture for MBS Simulation

Linear Algebra Implementations

Parallelization

Integration with MATLAB/Simulink

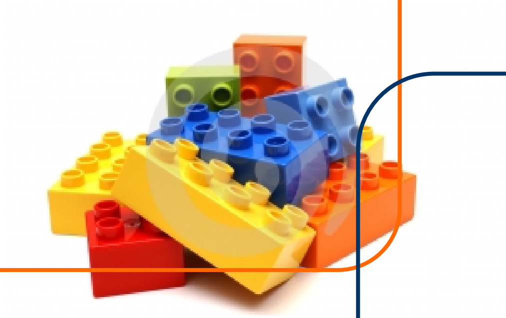
Multirate Co-simulation Methods

Conclusions and Future Research

Software requirements

- Research software for MBS simulation

Modular



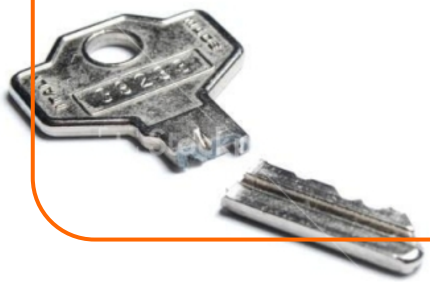
Collaborative

Object-oriented paradigm

C++

Flexibility - Efficiency

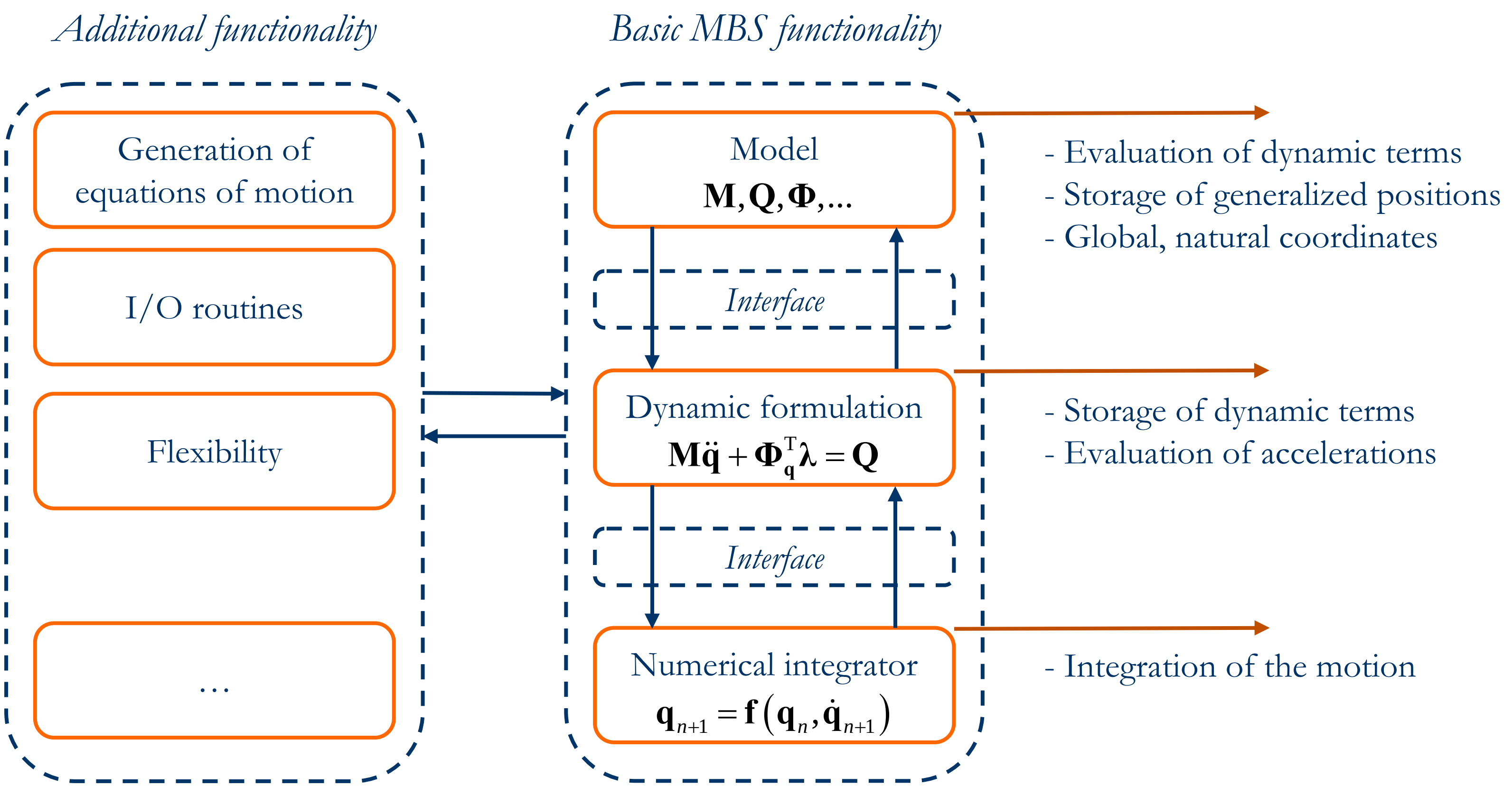
Open



Platform-independent

Structure of the simulation software

- **Modular structure**



Outline

Linear algebra operations in MBS simulation

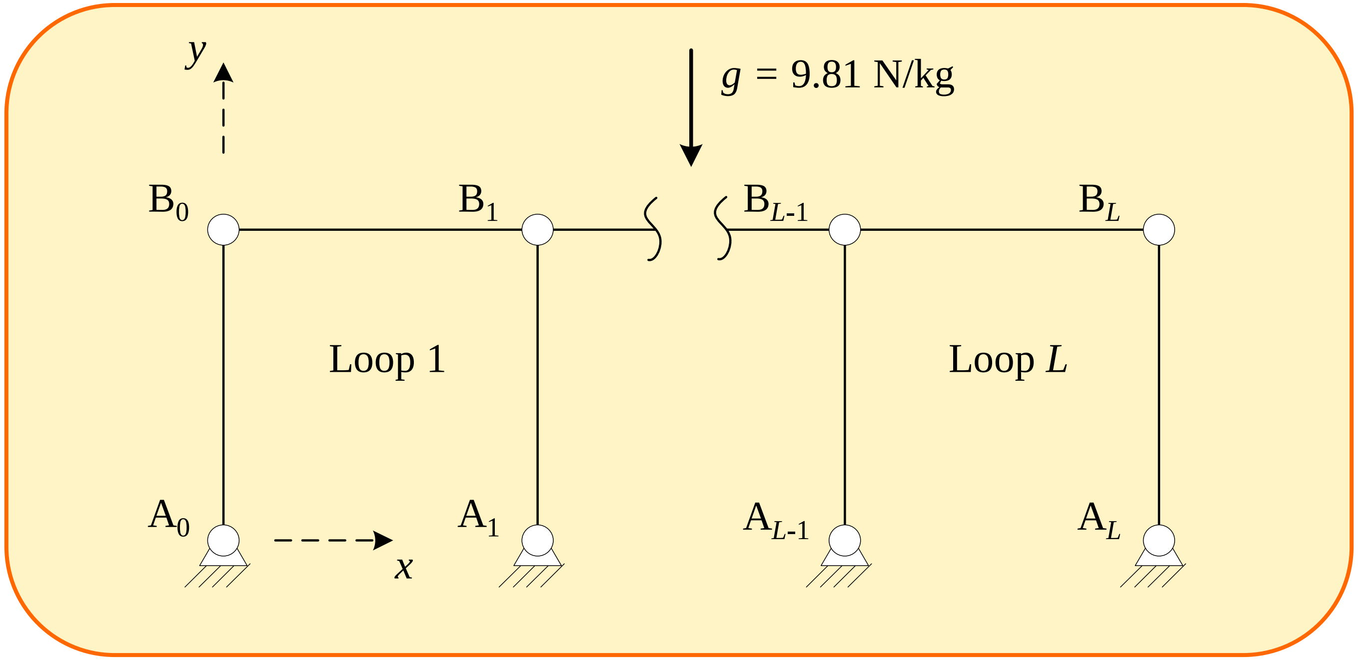
- **MBS codes make intensive use of linear algebra operations**
  - Low-level scalar-matrix-vector operations $\mathbf{B} = \mathbf{A}^{\mathrm{T}}\mathbf{A}$
  - Solution of linear systems of equations $\mathbf{A}\mathbf{x} = \mathbf{b}$

- **Reference Fortran implementation**
  - Dense: IMSL solver
  - Sparse: MA27 solver

Fraction of elapsed time in computations	**Dense**	**Sparse**
Residual and tangent matrix	48%	15%
Factorization and back-substitutions	44%	51%
Velocity and acceleration projections	4%	13%
Other	4%	21%

Benchmark setup: test problem

- 2D assembly of four-bar linkages
- 1 degree of freedom; variable number of loops and size
- Simulation time: 20 s



Benchmark setup: dynamic formulation

- **Index-3 augmented Lagrangian (natural coordinates)**

$$\mathbf{M}\ddot{\mathbf{q}} + \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\alpha\boldsymbol{\Phi} + \boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\boldsymbol{\lambda}^{*} = \mathbf{Q}$$

$$\boldsymbol{\lambda}_{i+1}^{*} = \boldsymbol{\lambda}_{i}^{*} + \alpha\boldsymbol{\Phi}_{i+1}; \quad i = 0,1,2,\ldots$$

- **Trapezoidal rule as integrator (implicit)**
- **Newton-Raphson iteration with approximate tangent matrix (SPD)**

$$\mathbf{f}(\mathbf{q}) = \mathbf{M}\mathbf{q}_{n+1} + \frac{\Delta t^2}{4}\boldsymbol{\Phi}_{\mathbf{q}(n+1)}^{\mathrm{T}}\left(\alpha\boldsymbol{\Phi}_{n+1} + \boldsymbol{\lambda}_{n+1}\right) - \frac{\Delta t^2}{4}\mathbf{Q}_{n+1} + \frac{\Delta t^2}{4}\mathbf{M}\hat{\ddot{\mathbf{q}}}_{n}$$

$$\left[\frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}}\right] \cong \mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\left(\boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\alpha\boldsymbol{\Phi}_{\mathbf{q}} + \mathbf{K}\right)$$

- **Mass-orthogonal projection of velocities and accelerations**

$$\left[\frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}}\right]\dot{\mathbf{q}} = \left[\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\mathbf{K}\right]\dot{\mathbf{q}}^{*} - \frac{\Delta t^2}{4}\boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\alpha\boldsymbol{\Phi}_{t}$$

$$\left[\frac{\partial \mathbf{f}(\mathbf{q})}{\partial \mathbf{q}}\right]\ddot{\mathbf{q}} = \left[\mathbf{M} + \frac{\Delta t}{2}\mathbf{C} + \frac{\Delta t^2}{4}\mathbf{K}\right]\ddot{\mathbf{q}}^{*} - \frac{\Delta t^2}{4}\boldsymbol{\Phi}_{\mathbf{q}}^{\mathrm{T}}\alpha\left(\dot{\boldsymbol{\Phi}}_{\mathbf{q}}\dot{\mathbf{q}} + \dot{\boldsymbol{\Phi}}_{t}\right)$$

Efficient dense implementations

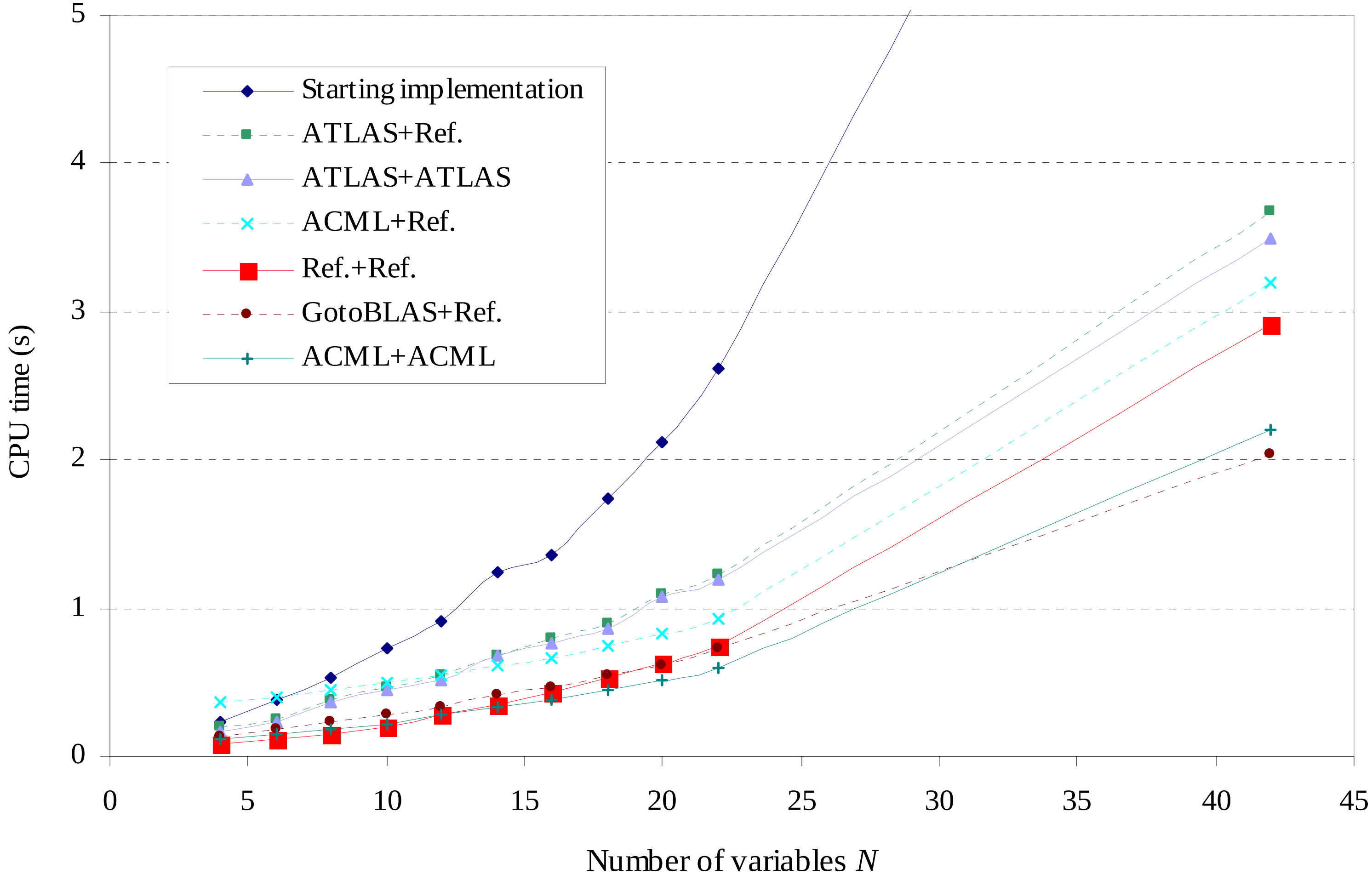
- **Dense storage is *supposed* to be faster for small problems**

- **Standard libraries for linear algebra operations:**
  - Low-level scalar-matrix-vector operations: **BLAS**
  - Linear equation solvers: **LAPACK**

- **BLAS/LAPACK are available in several implementations:**
  - Reference — Original Fortran77 implementation (not tuned)
  - ATLAS — Tuned for different hardware architectures
  - GotoBLAS — Tuned for different CPUs
  - ACML — Tuned for AMD CPUs
  - Other

Performance of BLAS/LAPACK implementations

- **As a function of problem size**



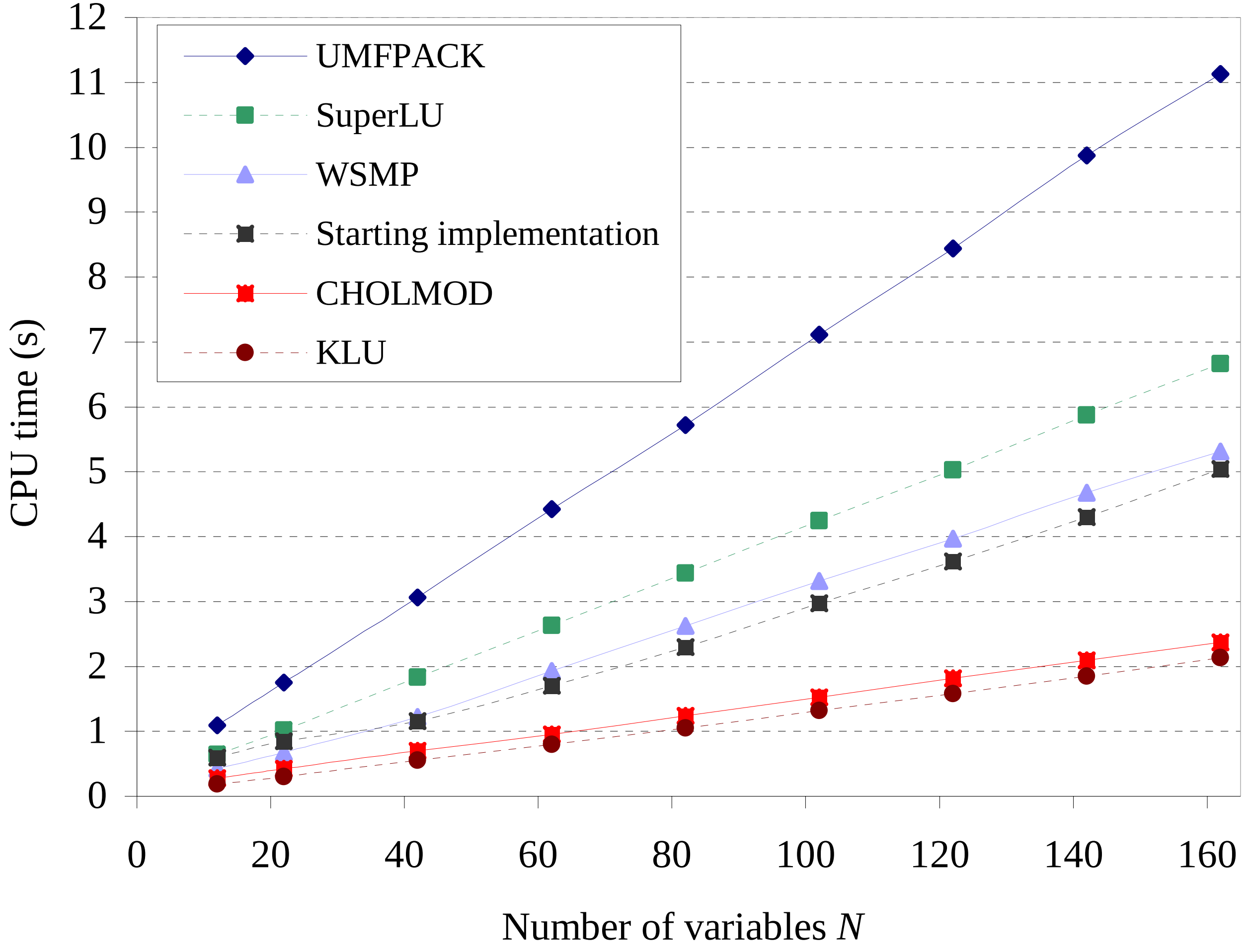
Efficient sparse implementations

- **Sparse linear equation solver is critical (50% of CPU)**
- **Use of optimized matrix handling routines**
  - **A + B**
  - **$A^T A$**
  - **Access to Jacobian matrix**
- **Different solvers have been tested (all CCS)**

Sparse linear solver	Matrix type
Cholmod	Symmetric positive definite
KLU	General
SuperLU	General
Umfpack	General
WSMP	Symmetric indefinite

Performance of sparse linear solvers

- **As a function of problem size**



Effect of matrix filling on sparse implementations

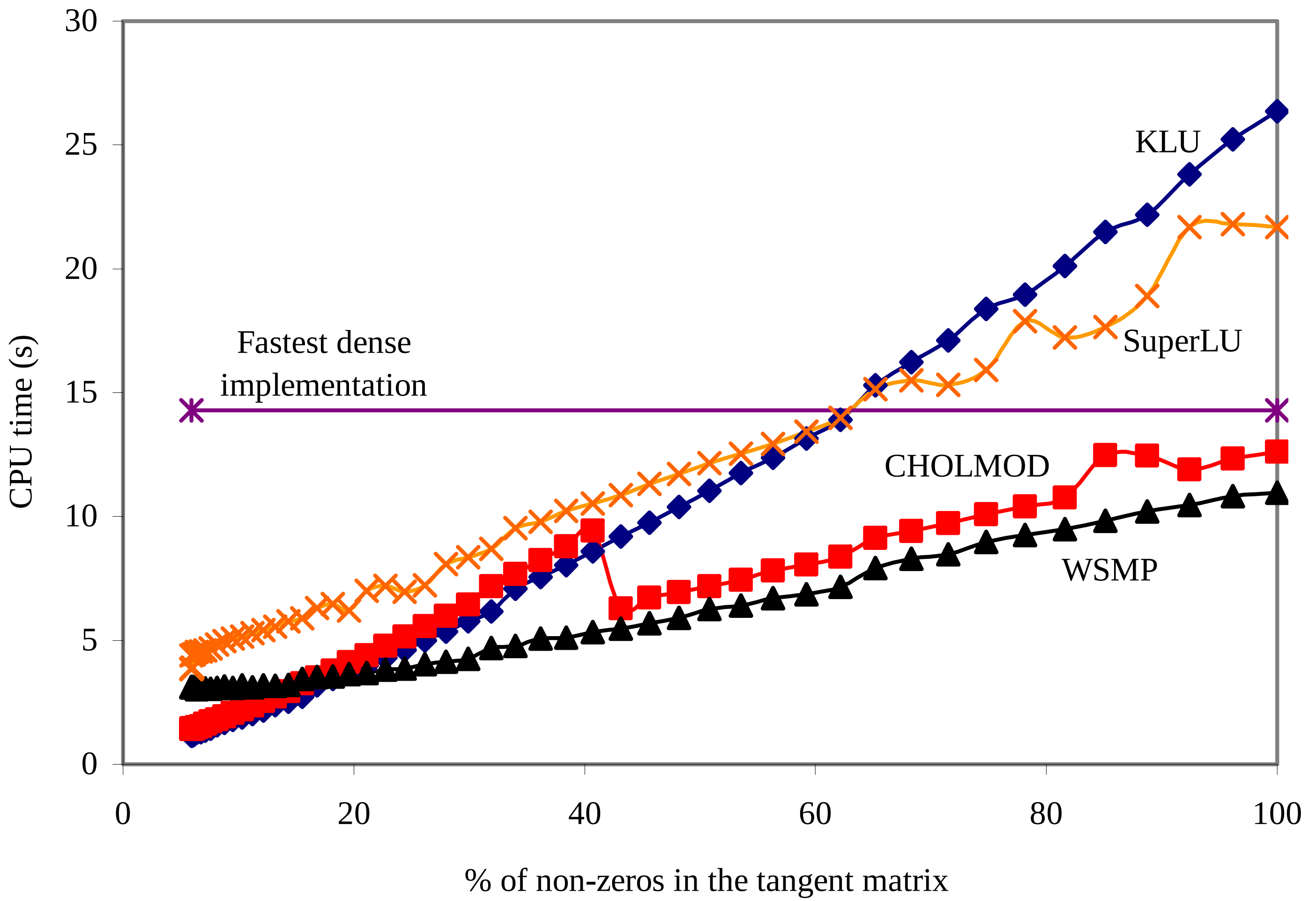
- **The percentage of non-zeros in the test problem is small**
  - Global formulation
  - 6% of non-zeros for $N = 100$ variables

- **The % of non-zeros can increase**
  - Recursive and semi-recursive formulations
  - Some methods for flexible bodies

- **Modification of the test problem**
  - Addition of artificial non-zeros to the leading matrix
  - Evaluation of solver performance vs. % of non-zeros

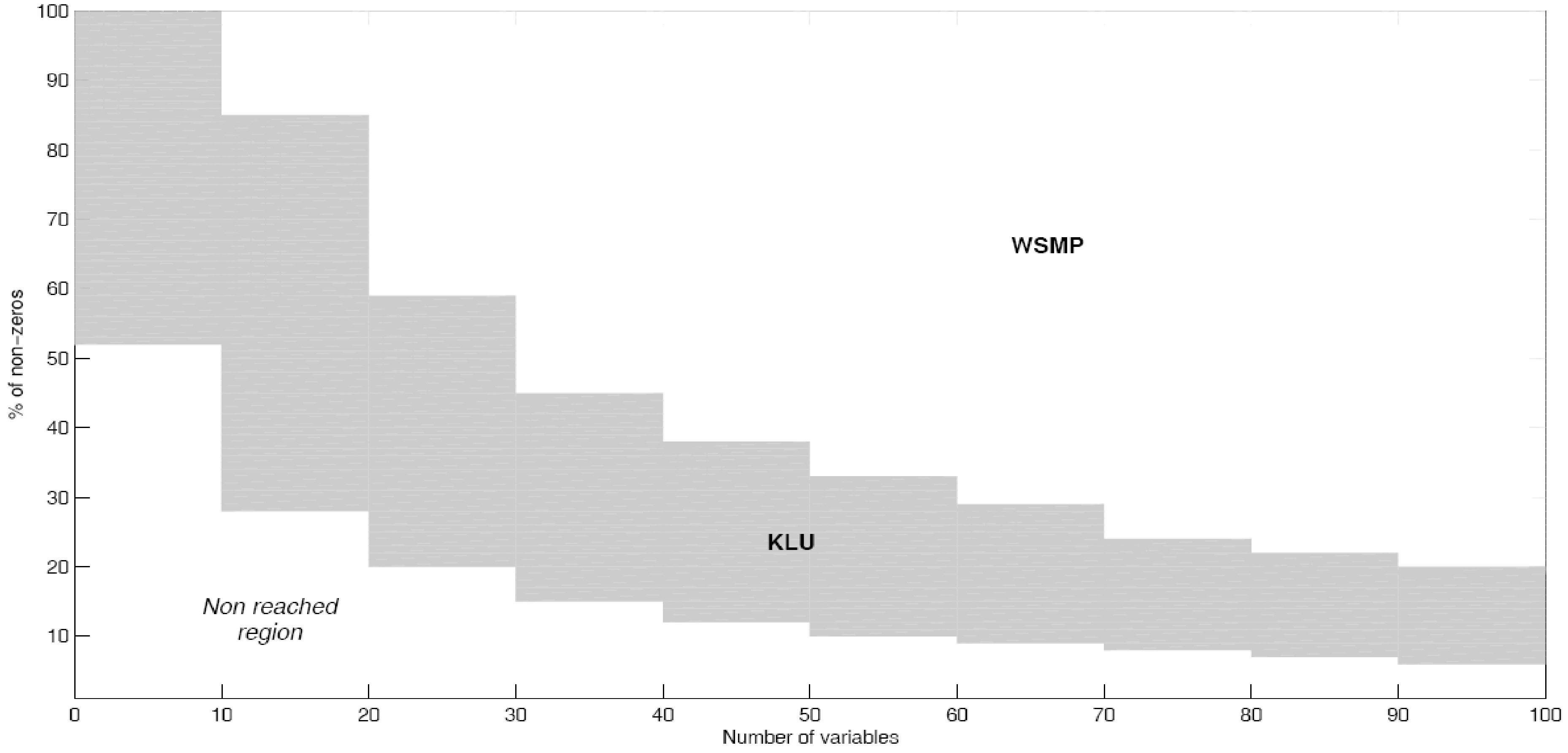
Effect of matrix filling on sparse implementations

- As a function of percentage of non-zeros (N = 100)



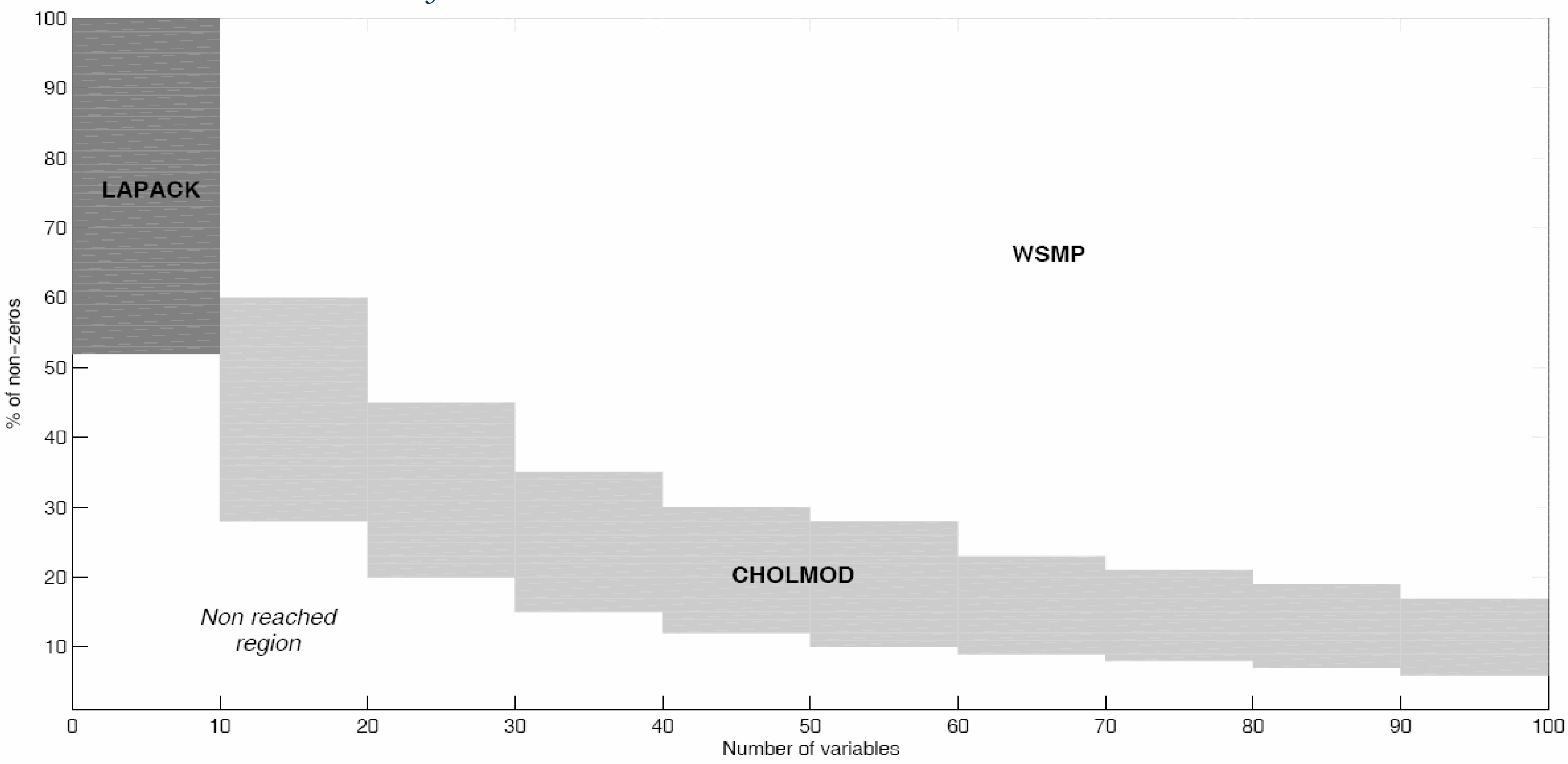
Best linear equation solver

- As a function of problem size and % of non-zeros in leading matrix



Best linear equation solver

- **As a function of problem size and % of non-zeros in leading matrix**
  - Without KLU *refactor* routine



Conclusions

- **Efficient linear algebra implementations can speedup simulations**
  - With respect to our starting implementation, in a factor of 2 – 3
- **Sparse solvers have performed better: KLU, Cholmod, WSMP**
  - Selection rule based on matrix type, size (*N*) and non-zeros (*NNZ*)

Type of leading matrix	$N \cdot (NNZ - 10)$	
	< 900	**> 900**
Symmetric positive definite	KLU (smooth problems) Cholmod (rough problems)	WSMP
Symmetric	KLU	WSMP
Unsymmetric	KLU	KLU

- **Future work:**
  - Test the optimization with recursive and/or flexible formulations

Outline

Introduction

Software Architecture for MBS Simulation

Linear Algebra Implementations

Parallelization

Integration with MATLAB/Simulink

Multirate Co-simulation Methods

Conclusions and Future Research

Non-intrusive parallelization in MBS simulations

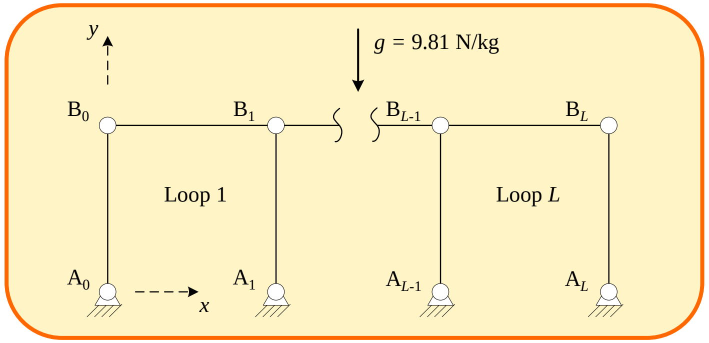
- **In MBS, parallel computing is usually achieved through**
  - Parallel algorithms: recursive formulations, sub-structuring…
  - Implemented with Message Passing Interface (MPI)

- **These are intrusive methods**
  - They require particular code designs and implementations
  - Difficult to apply to existing sequential codes

- **Objective: parallelization of existing sequential codes with minimum effort**
  - Use of non-intrusive techniques (with minor changes in the code)
    - Not as efficient as intrusive methods
    - Easy to apply to existing sequential MBS simulation tools

Benchmark setup

- **Same problem and dynamic formulation used in previous chapter**
  - L-loop four-bar linkage
  - Index-3 augmented Lagrangian formulation with projection of velocities and accelerations

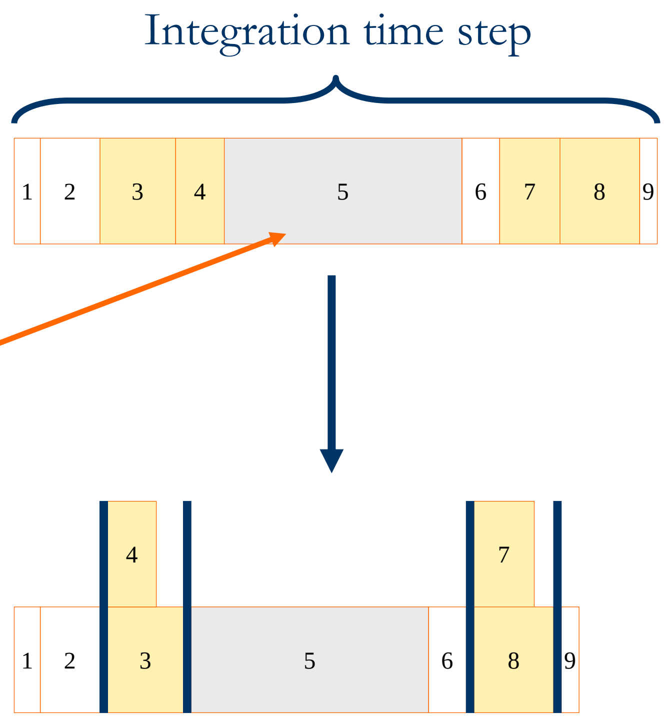


- **Heavily optimized for sequential execution**
  - Difficult to gain advantage from parallelization
- **Tests in 2-core computer**
  - Intel Core Duo E6300

Starting sequential implementation

- **Profiling of the initial implementation, for N variables**

Task	Description	% of elapsed time N = 1000	N = 8000
1	Predictor (Trapezoidal rule)	4.1	4.0
2	Evaluate dynamic terms	9.3	9.8
3	Evaluate tangent matrix	11.8	11.8
4	Evaluate residual vector	7.6	7.6
5	Factorize leading matrix	36.8	36.7
6	Back-substitution	5.9	5.8
7	Project velocities	9.4	9.3
8	Project accelerations	12.3	12.2
9	Other	2.8	2.8
Total elapsed time (s)		10.0	102.4



Parallel linear equation solvers

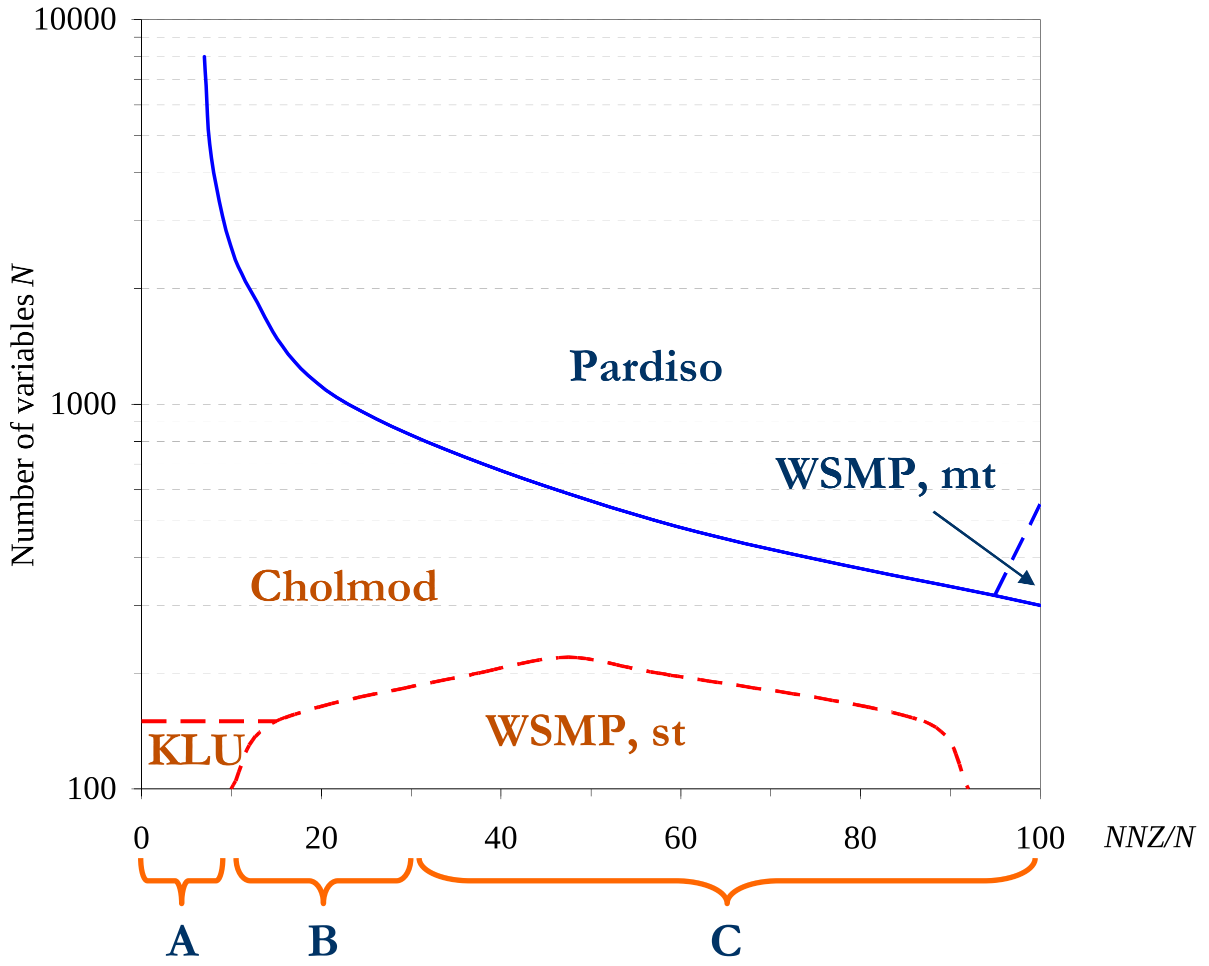
- **Different parallel solvers tested as a function of:**
  - Matrix size (number of variables N from 100 to 8,000)
  - Matrix filling
    - Filling ratio NNZ/N (NNZ = number of non-zeros in the leading matrix)

Type of problem and dynamic formulation		***NNZ / N***
A)	Rigid bodies – Global formulations	< 10
B)	Rigid bodies – Recursive formulations Flexible bodies – Component mode synthesis	10 – 30
C)	Flexible bodies – Finite element mesh	30 – 100

- **Effect of sparse pattern diminished**
  - Use of reordering strategies: METIS, AMD…

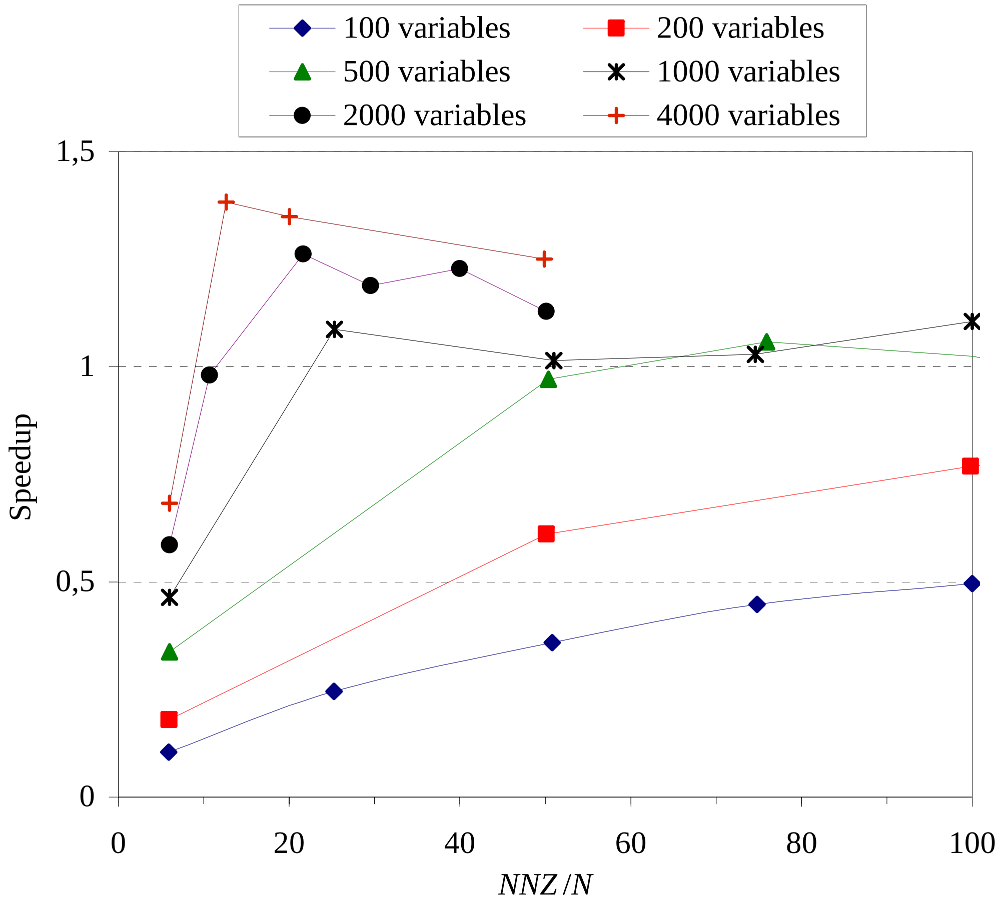
Parallel linear equation solvers: Results

- **Best solver as a function of size *N* and matrix filling *NNZ/N***



Parallel linear equation solvers: Results

- **Speedup of Pardiso (best parallel solver) vs. best sequential solver**



$$S = \frac{elapsed\ \ time_{sequential}}{elapsed\ \ time_{parallel}}$$

- **Theoretical maximum, for 2 CPUs, is 1.53**
- **Speedups close to 70% of the theoretical maximum, for N > 2,000**
- **Easy replacement of solvers in code**

OpenMP: Description

- **Set of compiler directives**
  - Guide the compiler to parallelize the code

OpenMP: Description

- **Set of compiler directives**
  - Guide the compiler to parallelize the code

- **Example**

  Calls 2 functions in parallel

```
void example1()
{
        #pragma omp parallel sections
        #pragma omp section
        function1();
        #pragma omp section
        function2();
}
```

OpenMP: Description

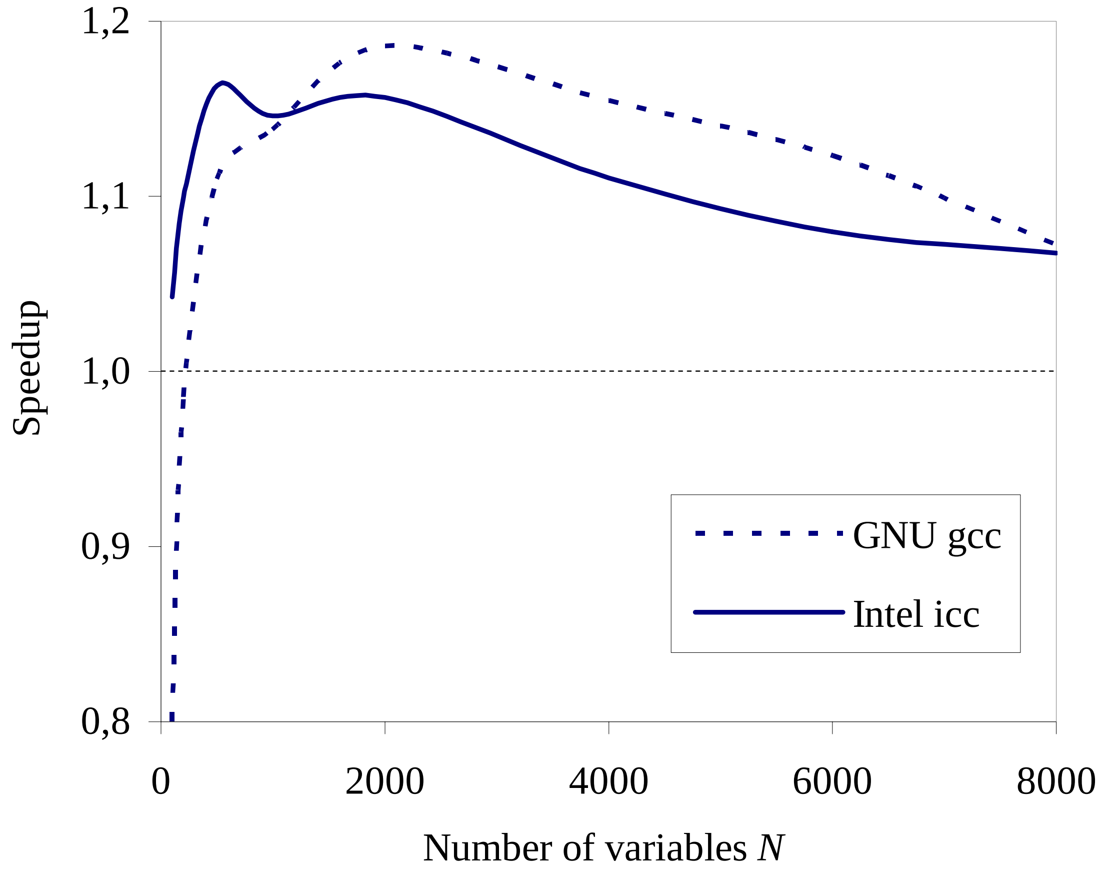
- **Set of compiler directives**
  - Guide the compiler to parallelize the code

- **Advantages (over MPI)**
  - Does not change the design of the code
  - Compiler does the hard work of parallelization in a transparent way
  - Can be applied incrementally

- **Disadvantages (over MPI)**
  - Only supports shared-memory hardware architectures
  - Cannot achieve the same performance as MPI in some cases

OpenMP: Results

- **Speedup of the OpenMP parallel implementation**



$$S = \frac{elapsed\ \ time_{sequential}}{elapsed\ \ time_{parallel}}$$

- **Theoretical maximum, for 2 CPUs, is 1.20**
- **Speedups close to 90% of the theoretical maximum**
- **Effect of compiler toolchain**

Conclusions

- **OpenMP and parallel linear equation solvers can be used in MBS simulation**
  - Actually non-intrusive and straightforward to implement
  - Can be applied to parallelize existing sequential codes
  - Speedups above 70% of maximum theoretical values

- **Parallel linear equation solvers**
  - Suitable for N > 2000 and NNZ/N > 10

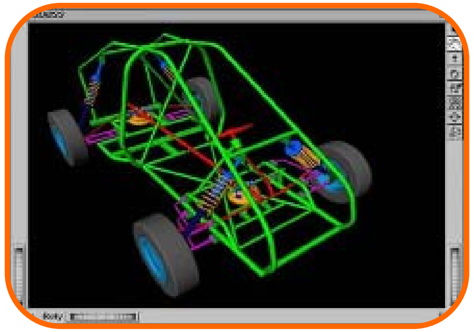
- **OpenMP**
  - Suitable for N > 100

Outline

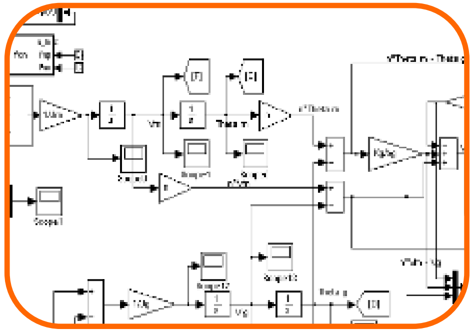
Introduction

- Multibody dynamics often needs multiphysics modelling

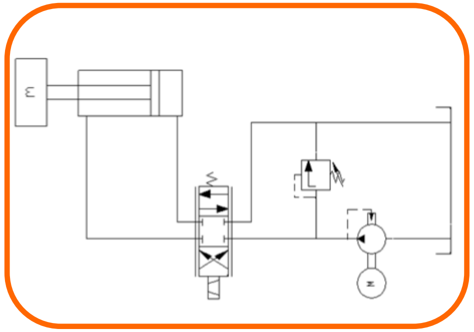
Mechanics



Control



Hydraulics



Other

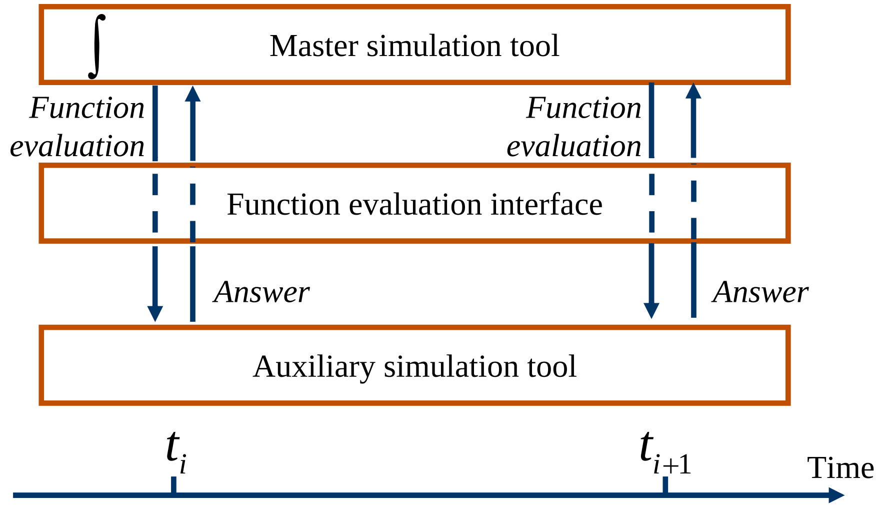
...

Multibody software

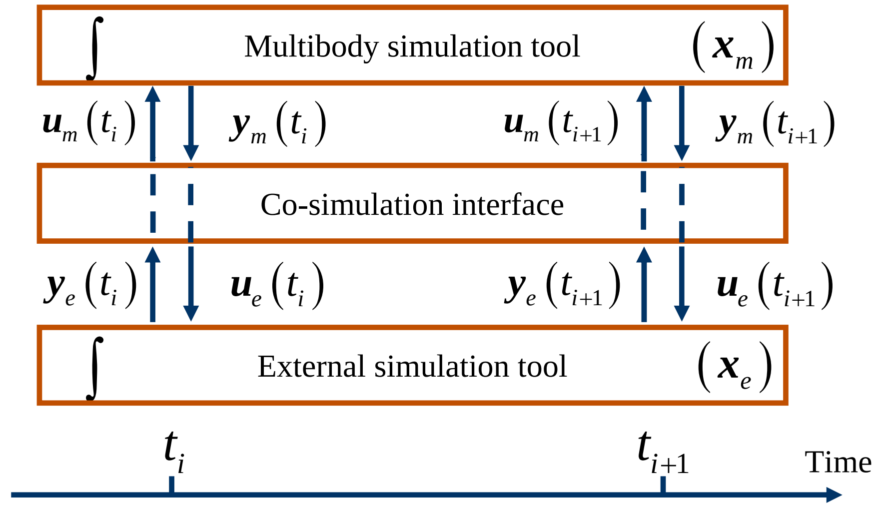
Math tools, block diagram simulators, other software

Communication cases

- **Function evaluation**
  - MBS software as master
  - No numerical integration in auxiliary tool

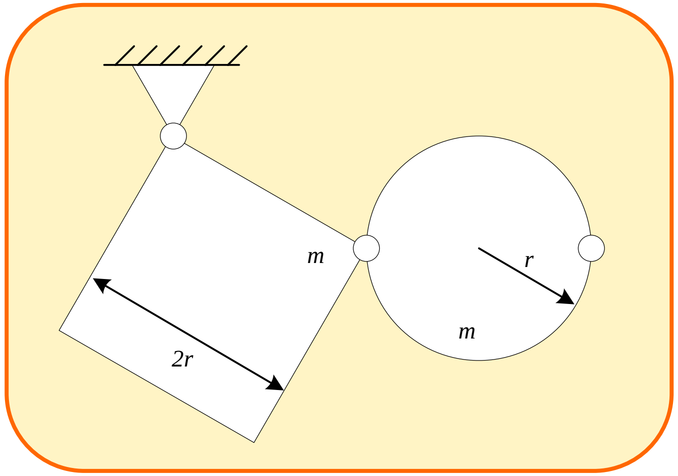


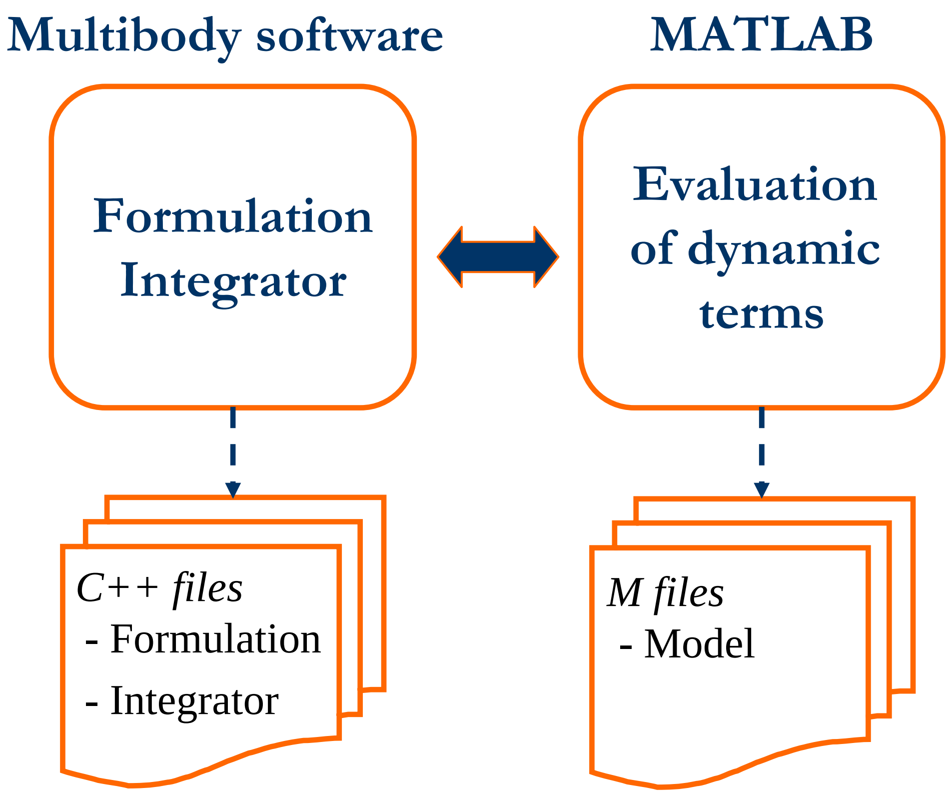
- **Co-simulation**
  - Each software package carries out its numerical integration
  - Data sharing at defined synchronization points



Function evaluation

- **Test problem: dynamic simulation of a double-pendulum (10 s)**



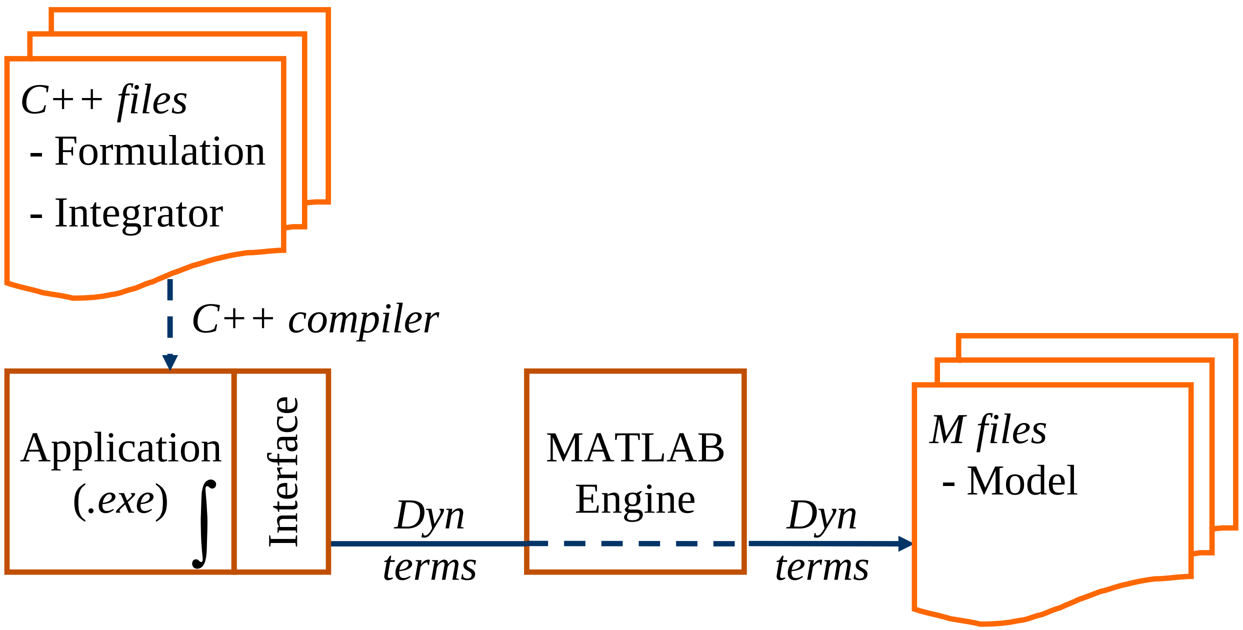


- **3 communication techniques:**
  - MATLAB Engine
  - MATLAB Compiler
  - MEX functions

- **Measure CPU times and compare to standalone C++ code**

Function evaluation: MATLAB Engine

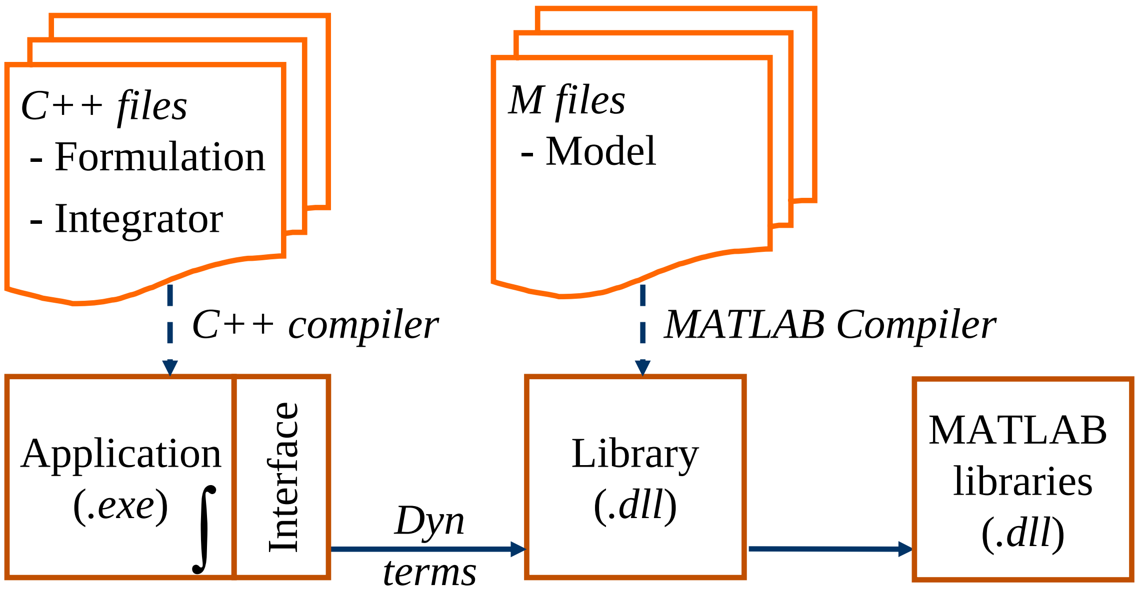
- **MATLAB Engine**
  - Inter-process communication



  - Easy to implement (direct call to MATLAB)
  - Slow: parsing of instructions (overhead about 0.25 ms per call)

Function evaluation: MATLAB Compiler

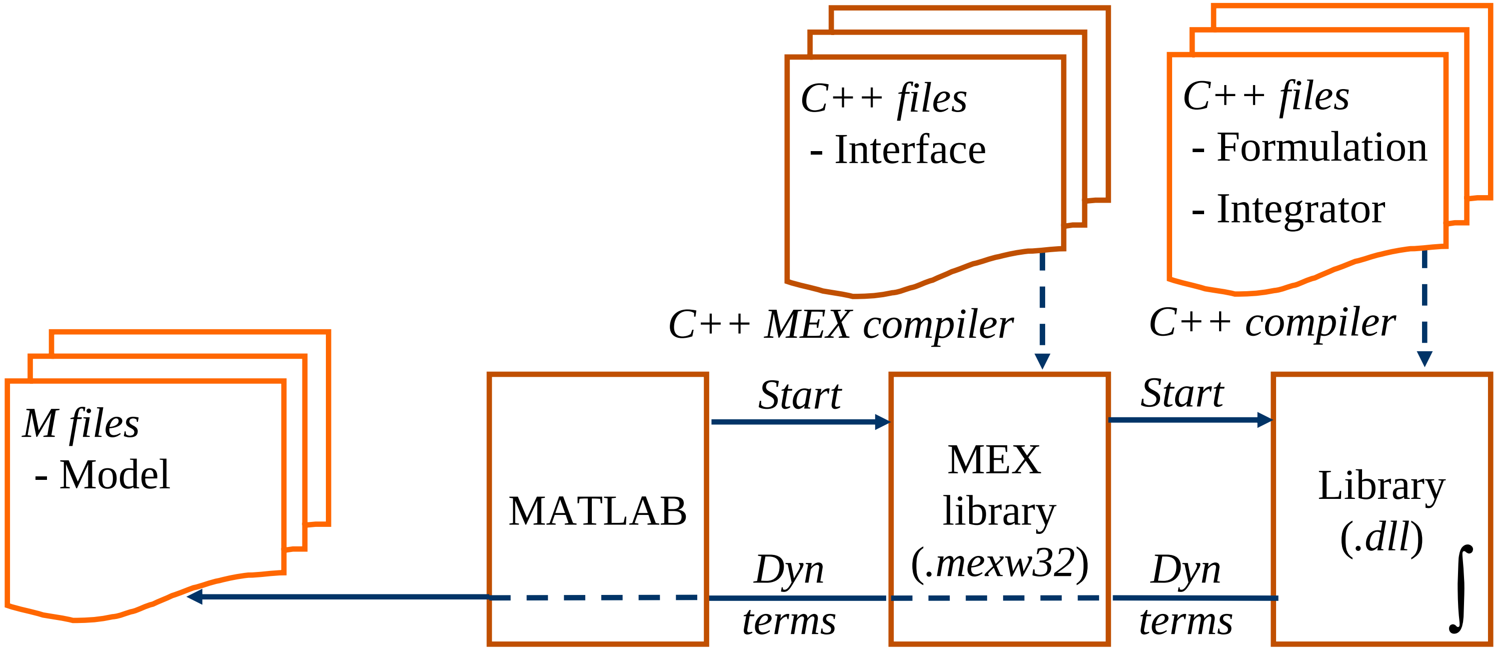
- **MATLAB Compiler**
  - Invocation of MATLAB code translated to C and compiled into a library



  - Claimed to be the fastest method
  - Changes in MATLAB code force re-compilation

Function evaluation: MEX functions

- **MEX functions**
  - Originally designed to call C code from MATLAB



  - More complex than previous techniques: MEX interface required
  - MATLAB code is not compiled

Function evaluation: computational efficiency

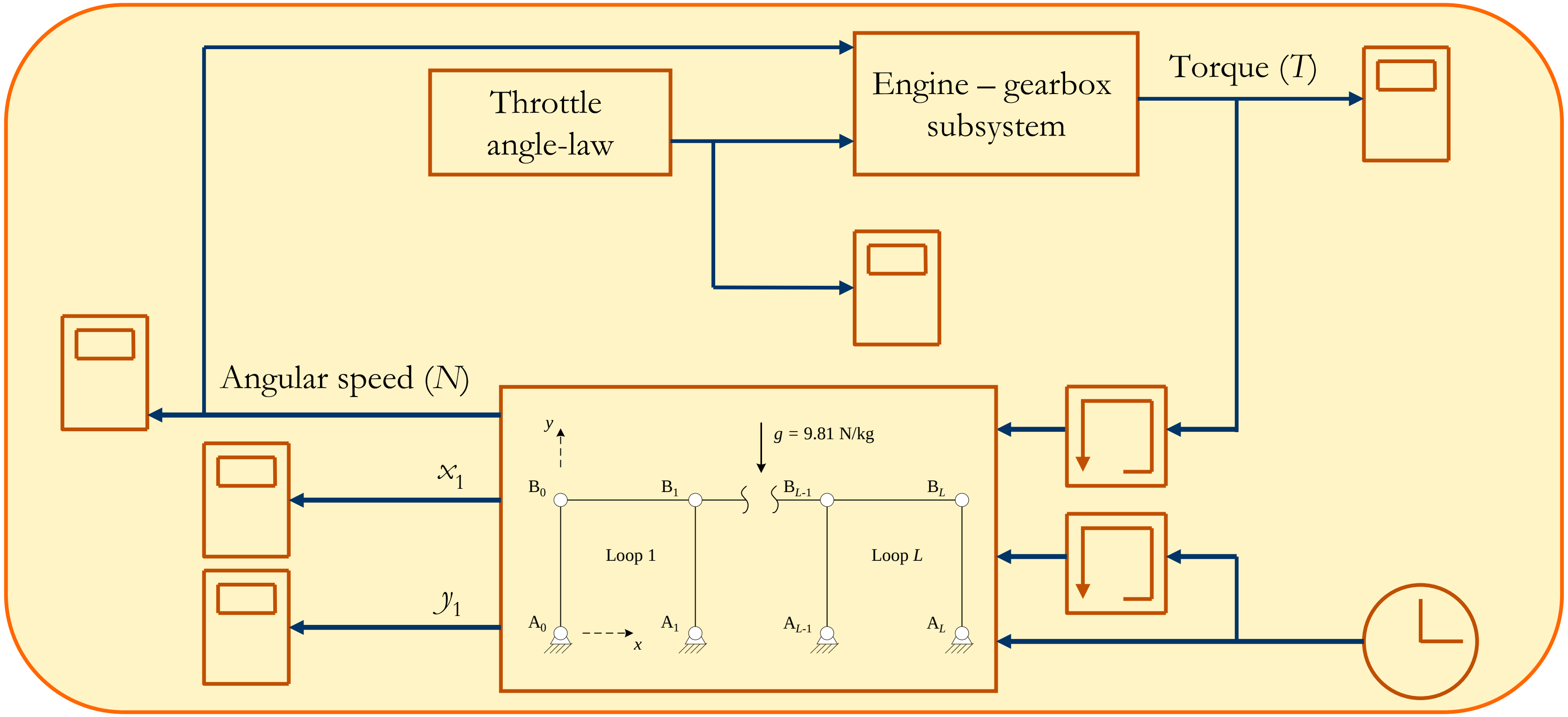
- **Comparison of CPU-times**
  - For two different time-steps ($\Delta t = 10^{-3}$ s and $\Delta t = 10^{-2}$ s)

Method	CPU-time ($\Delta t = 10^{-3}$ s)	Ratio	CPU-time ($\Delta t = 10^{-2}$ s)	Ratio
Standalone MBS code (reference)	$5.02 \cdot 10^{-2}$ s	1	$8.40 \cdot 10^{-3}$ s	1
MATLAB Engine	18.12 s	361.0	3.32 s	395.2
MATLAB Compiler	5.56 s	110.8	1.07 s	127.4
MEX functions	0.64 s	12.7	0.12 s	14.3

  - MEX functions are 7 times faster than MATLAB Compiler and 25 than MATLAB Engine

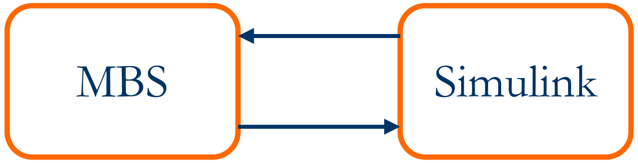
Co-simulation

- **Test problem: dynamic simulation**
  - *L*-loop four-bar linkage (MBS software)
  - Powered by an internal combustion engine (Simulink)



Co-simulation: implementation techniques

- **Network connection**
  - Inter-process communication
  - Use of TCP/IP sockets



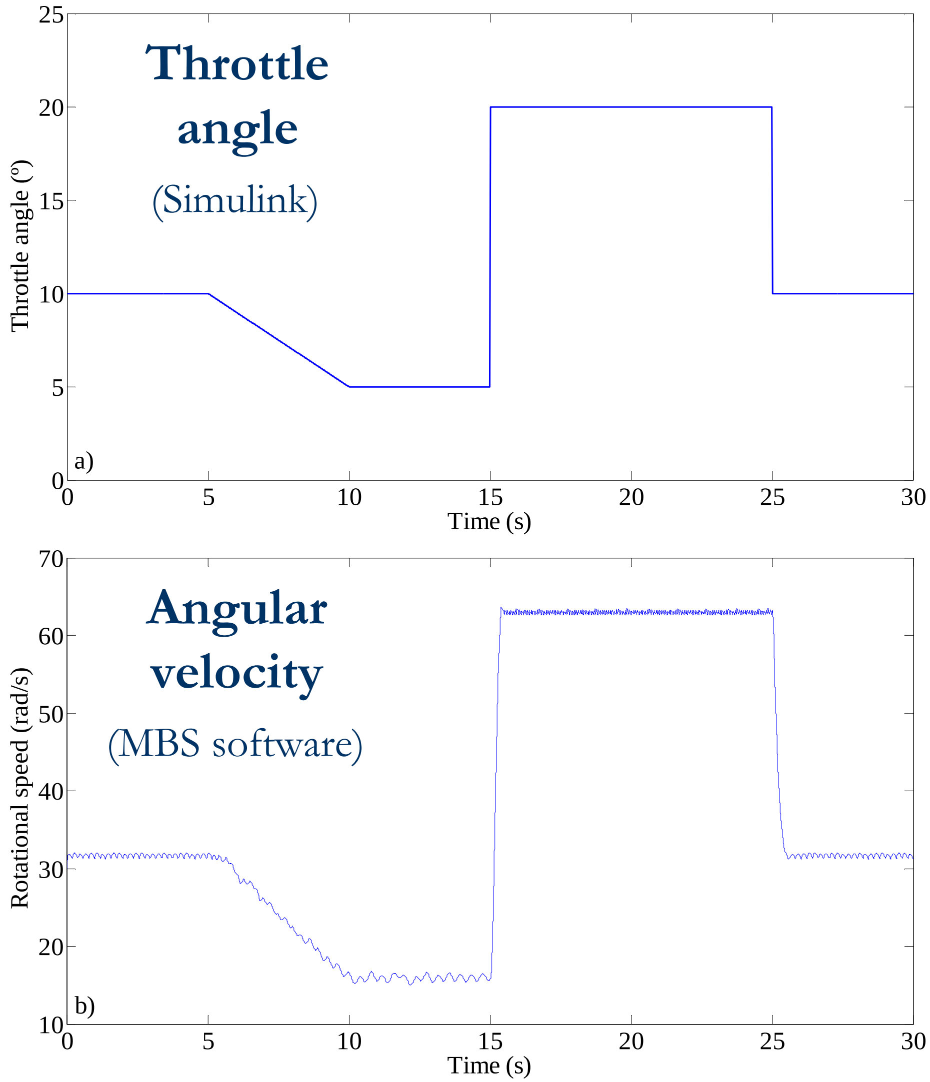
- **Simulink as master**
  - MBS code compiled as a .dll
  - Embedded in an S-function block

- **MBS as master**
  - Simulink model compiled as a .dll
  - Use of Real-Time Workshop

- **Compared to monolithic counterparts**
  - Simulink model with SimMechanics elements
  - C equivalent compiled with Real-Time Workshop

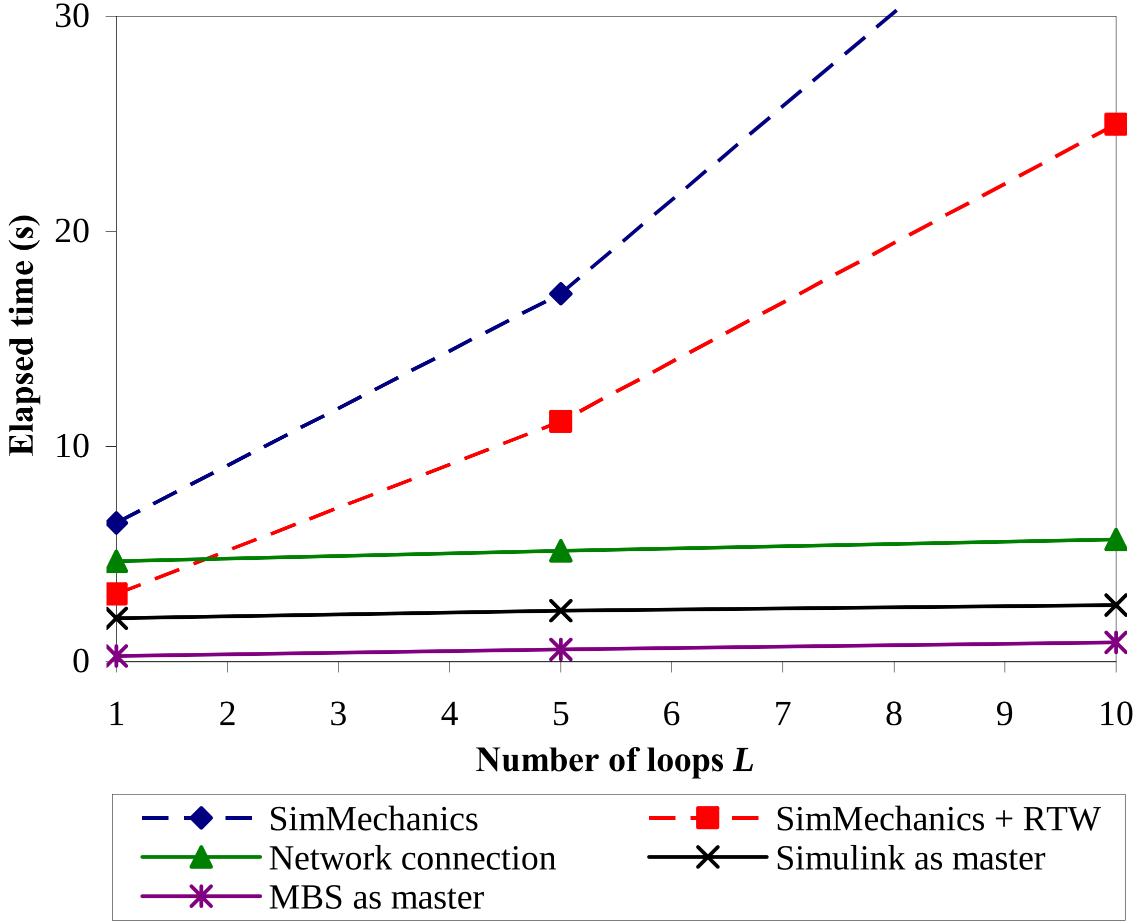
Co-simulation

- **Dynamic response for a 1-loop mechanism**



Co-simulation: computational efficiency

- **Comparison of CPU-times**
  - For Δt = 1 ms
  - 30 s simulation



Conclusions

- **Different coupling techniques with MATLAB/Simulink have been explored**

- **Function evaluation**
  - Recommended use of MEX functions

- **Co-simulation**
  - Able to efficiently simulate models up to 300 global variables
  - *Simulink as master* recommended in development stages (easy to modify)
  - *MBS as master* recommended for real-time applications (efficient)

Outline

Introduction

Software Architecture for MBS Simulation

Linear Algebra Implementations

Parallelization

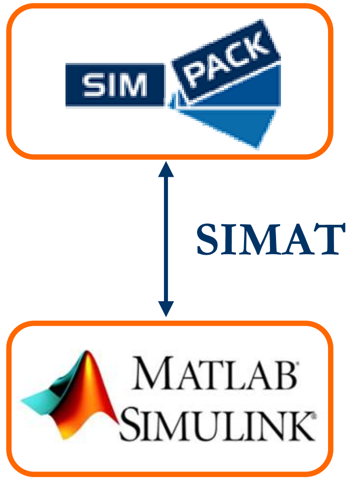
Integration with MATLAB/Simulink

Multirate Co-simulation Methods

Conclusions and Future Research

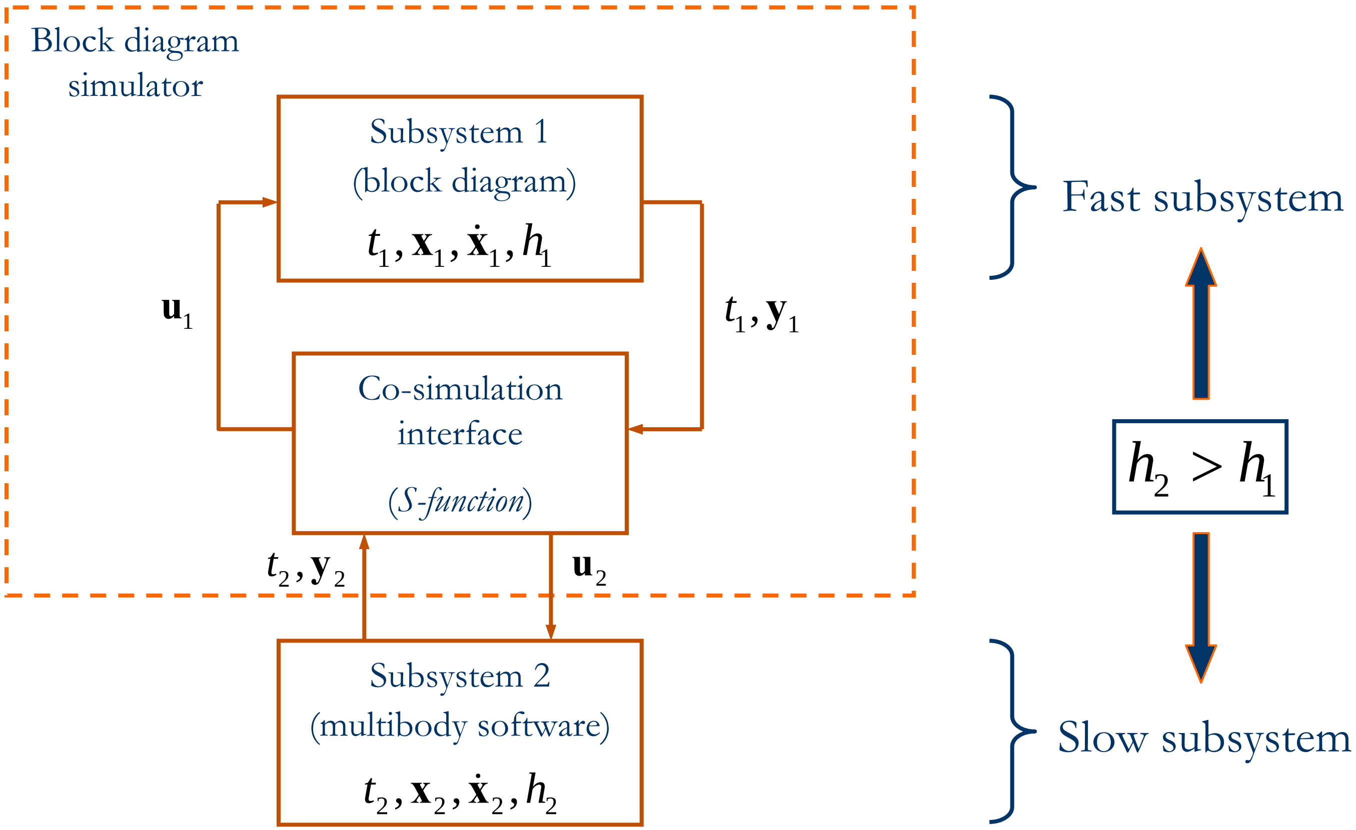
Introduction

- **Weakly coupled co-simulation**
  - Each solver integrates a subsystem
  - Commercial packages only allow exchange of data at fixed rates
- **Multirate integration**
  - Different time-steps in each subsystem
  - Reduces the elapsed time in simulations
  - Difficult to implement with commercial block diagram simulators
    - Non-modifiable integration schemes
    - Iterative coupling schemes cannot be used
    - Variable-step integrators not supported by interfaces
- **Development and test of a multirate co-simulation interface between MBS software and block diagram simulators**



Multirate co-simulation interface

- **Weakly coupled co-simulation scheme**



Coupling strategy

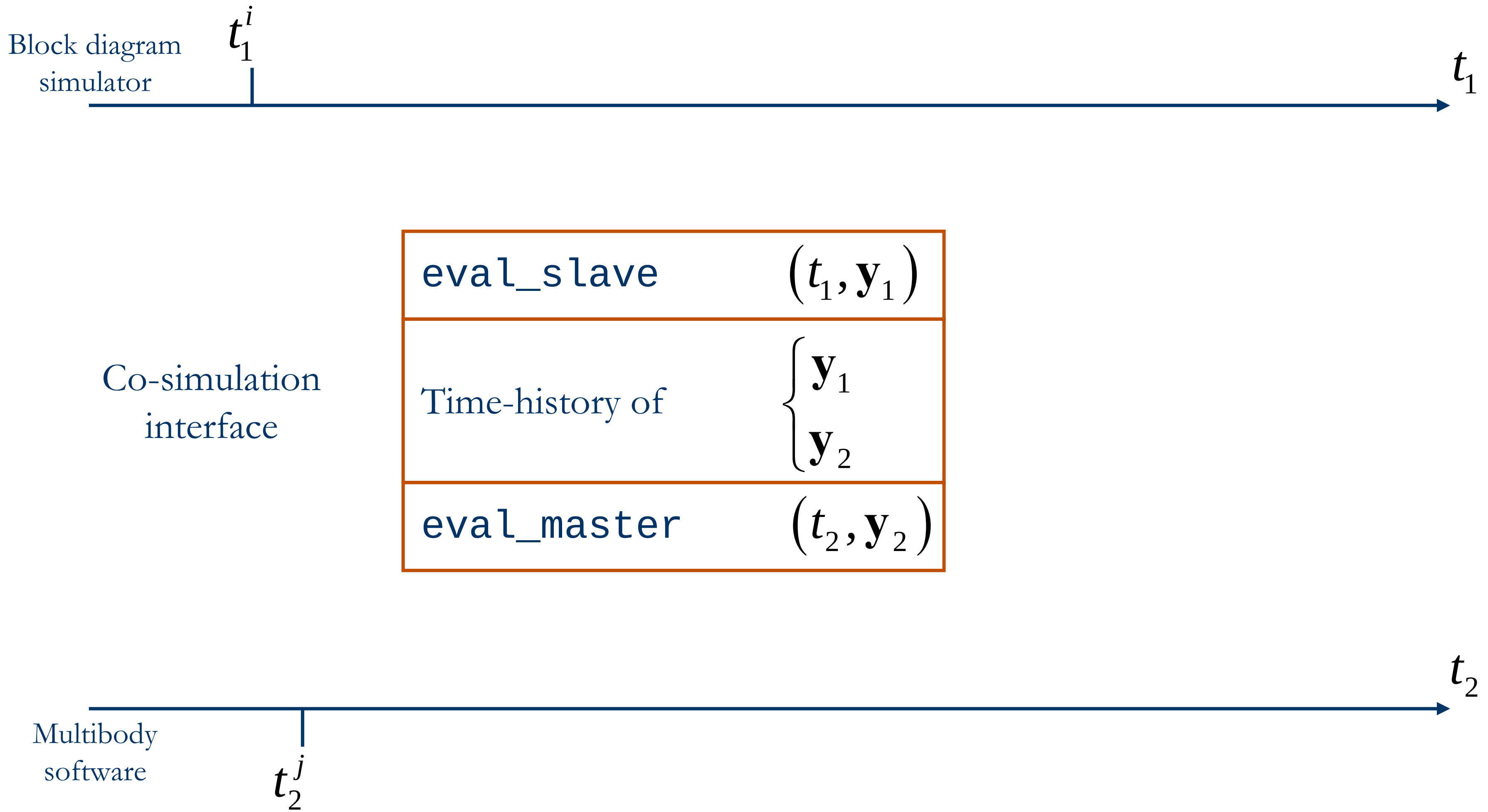
- **Two methods:**
  - Slowest-first (*SF*): slow subsystem is ahead in the integration
  - Fastest-first (*FF*): fast subsystem is ahead in the integration

- **Interpolation / Extrapolation polynomials used for approximating the values of the inputs between time-steps**

- **Smoothing:**
  - Averaging of the outputs of the fast subsystem during a time-step of the slow one

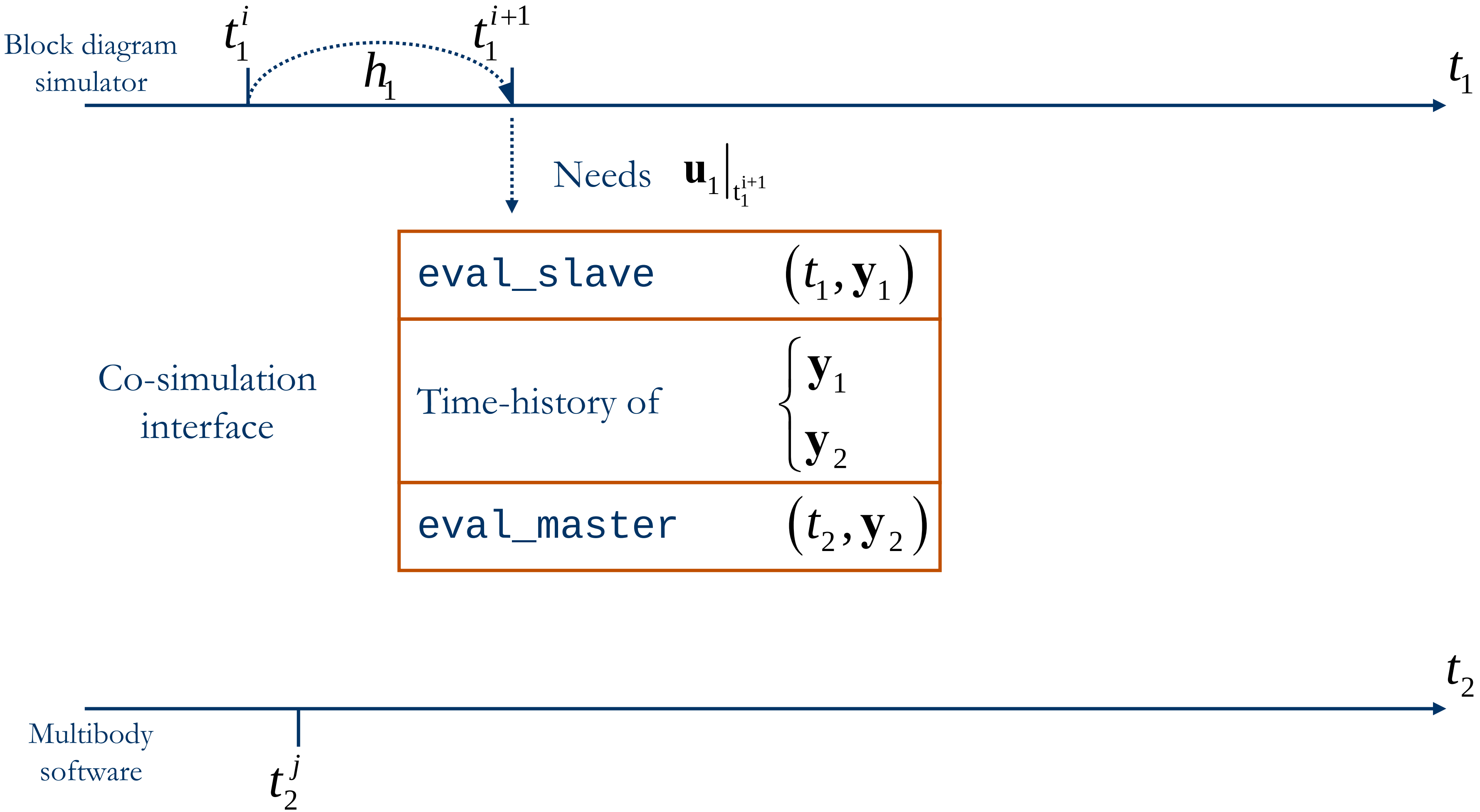
Coupling strategy

- **Initial situation (*slowest-first* configuration)**



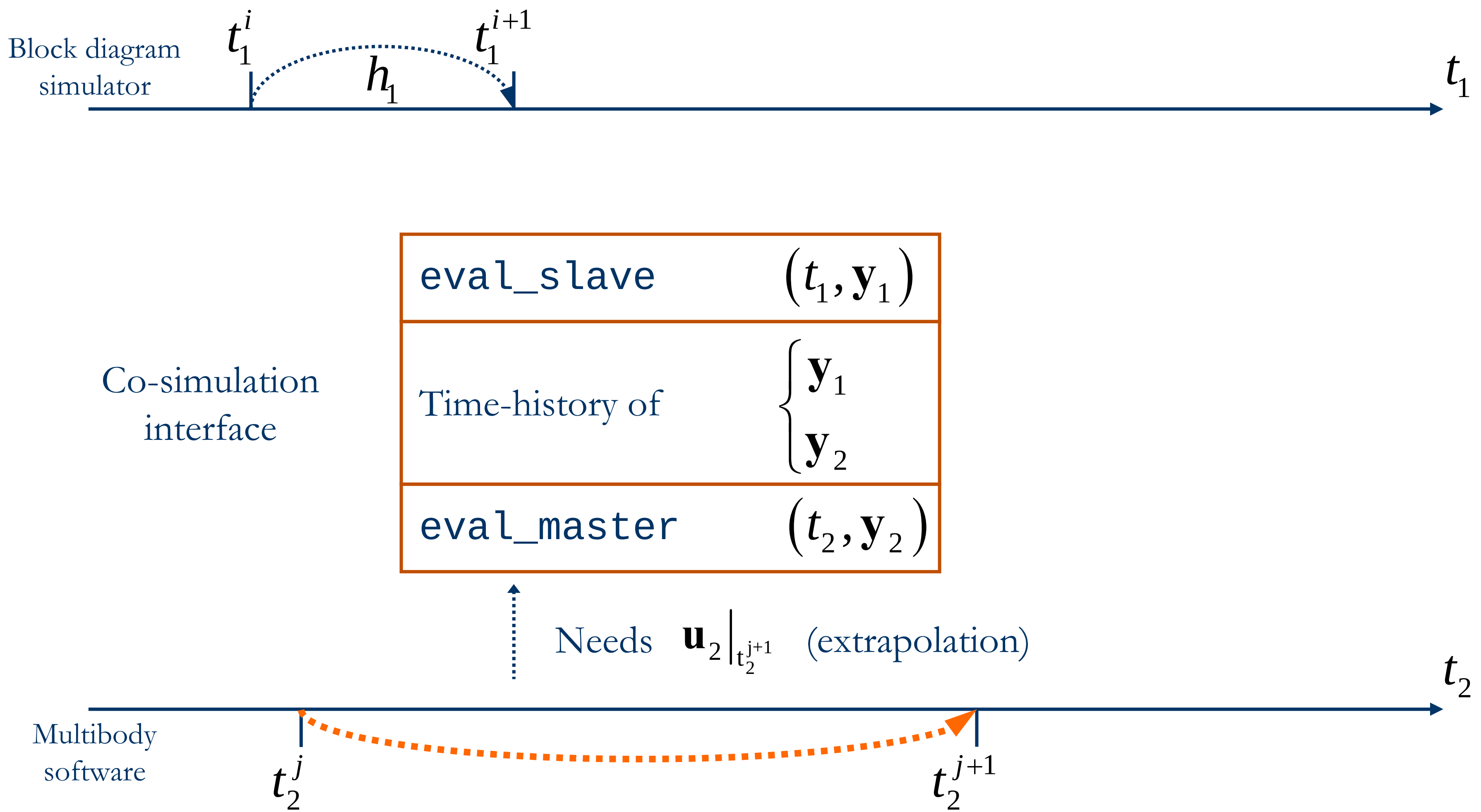
Coupling strategy

- **Block diagram simulator starts a time-step (h_1)**



Coupling strategy

- **MBS software advances a time-step (h_2)**

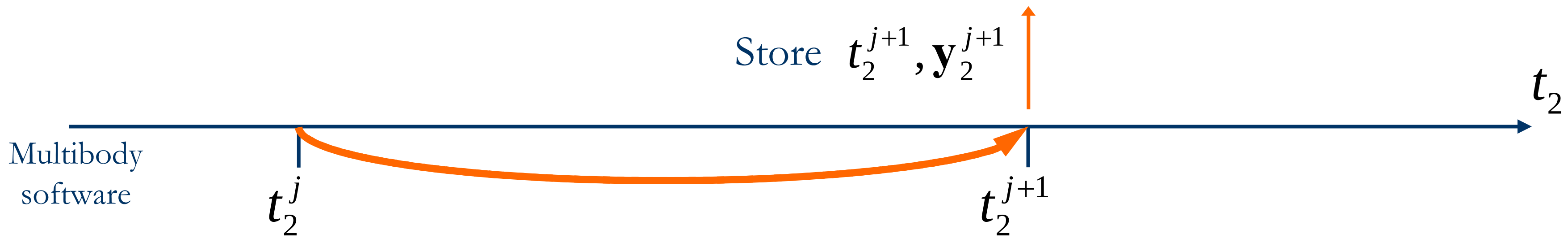


Coupling strategy

- **MBS software advances a time-step (h_2)**

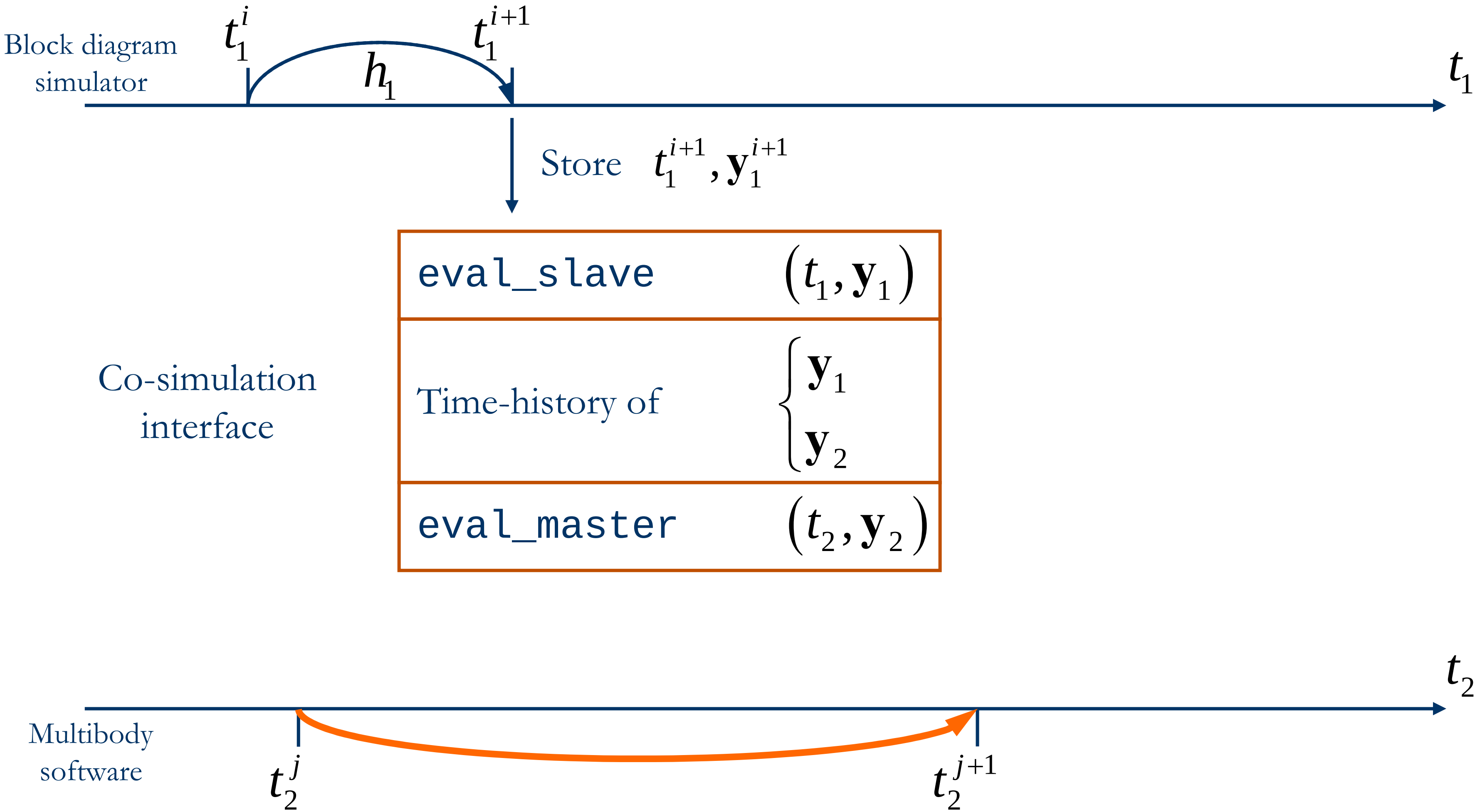
Co-simulation interface

`eval_slave`	$(t_1, \mathbf{y}_1)$
Time-history of	$\begin{cases} \mathbf{y}_1 \\ \mathbf{y}_2 \end{cases}$
`eval_master`	$(t_2, \mathbf{y}_2)$



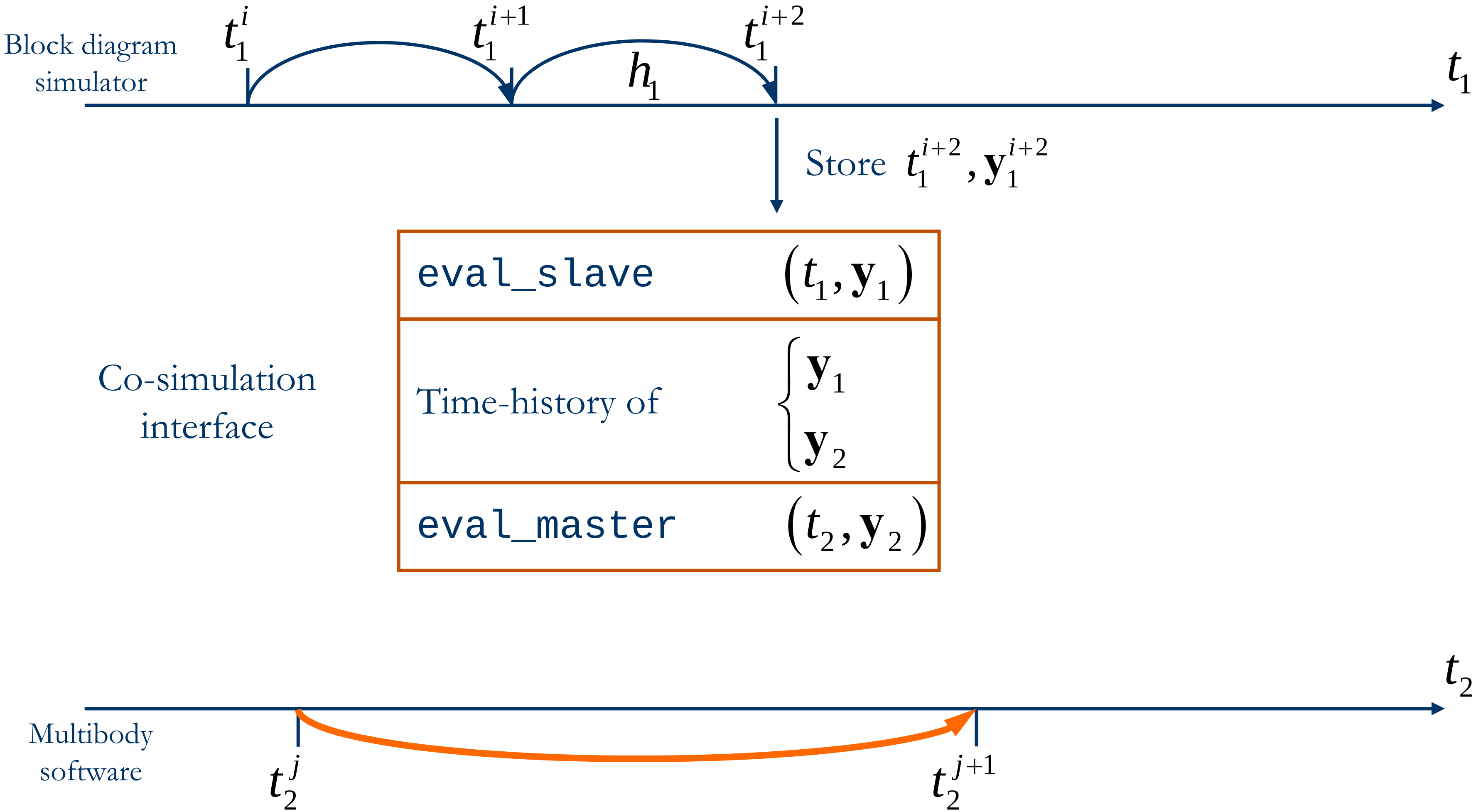
Coupling strategy

- **The block diagram simulator can resume its time-step**



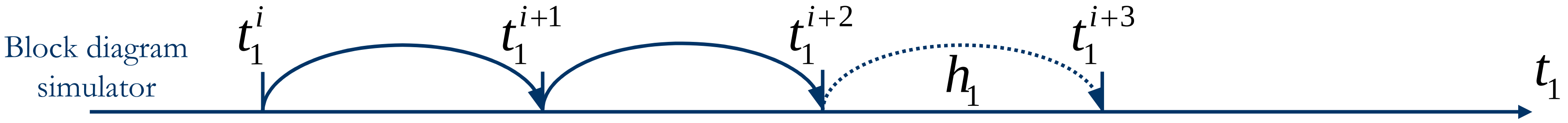
Coupling strategy

- **And more time-steps can be taken**



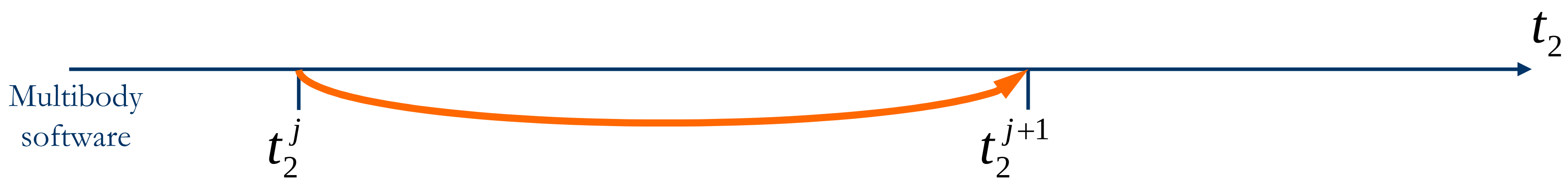
Coupling strategy

- **And the process starts again…**



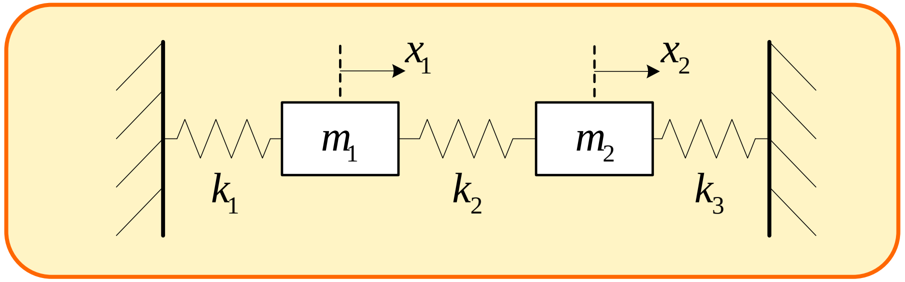
Co-simulation interface

`eval_slave`	$\left(t_1, \mathbf{y}_1\right)$
Time-history of	$\begin{cases} \mathbf{y}_1 \\ \mathbf{y}_2 \end{cases}$
`eval_master`	$\left(t_2, \mathbf{y}_2\right)$



Test problem: modelling approach

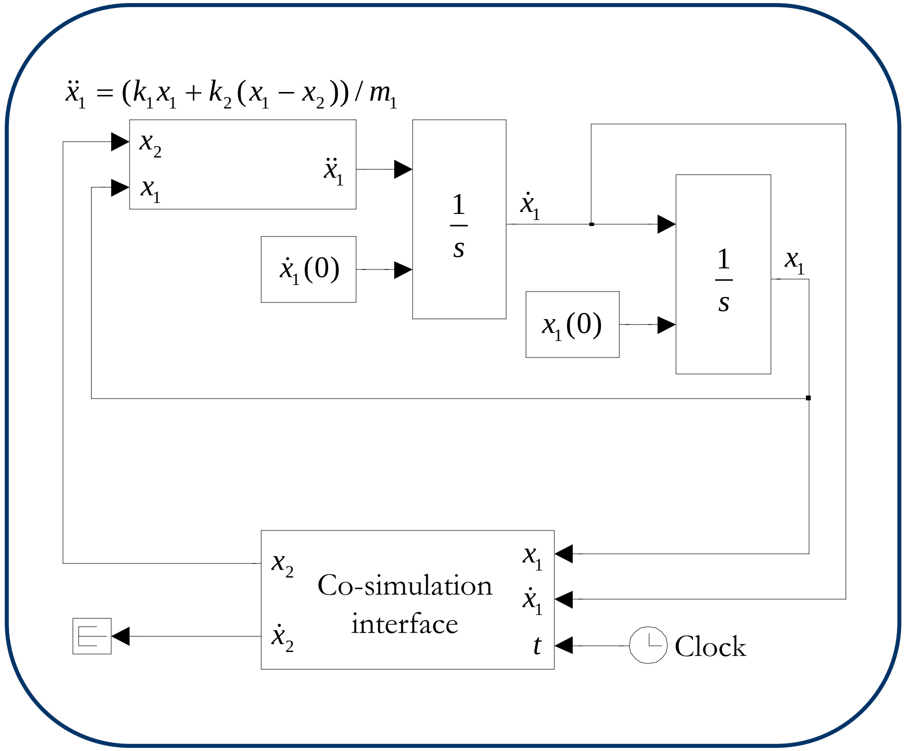
- **Double-mass, triple-spring assembly (linear system)**



- **Purely mechanical**

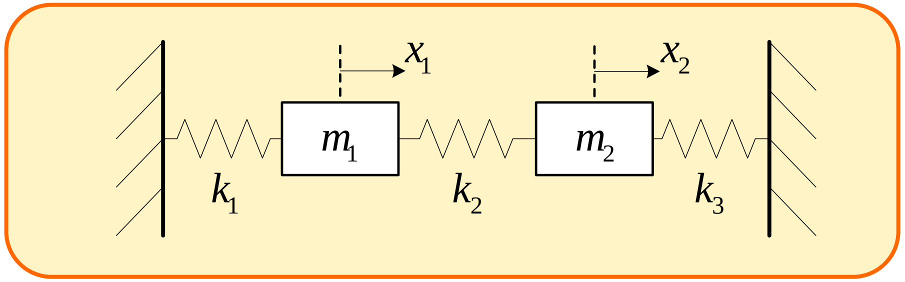
m$_1$: Simulink

m$_2$: MBS software embedded in *S-function* block



Test problem

- **Known analytical solution (reference to measure error in position and energy)**



- **Simulation: 100 cycles of the fast subsystem**

$$\begin{cases} x_1(t) = C_{11}\cdot\cos(\omega_1 t) + C_{13}\cdot\cos(\omega_2 t) \\ x_2(t) = C_{21}\cdot\cos(\omega_1 t) + C_{23}\cdot\cos(\omega_2 t) \end{cases}$$

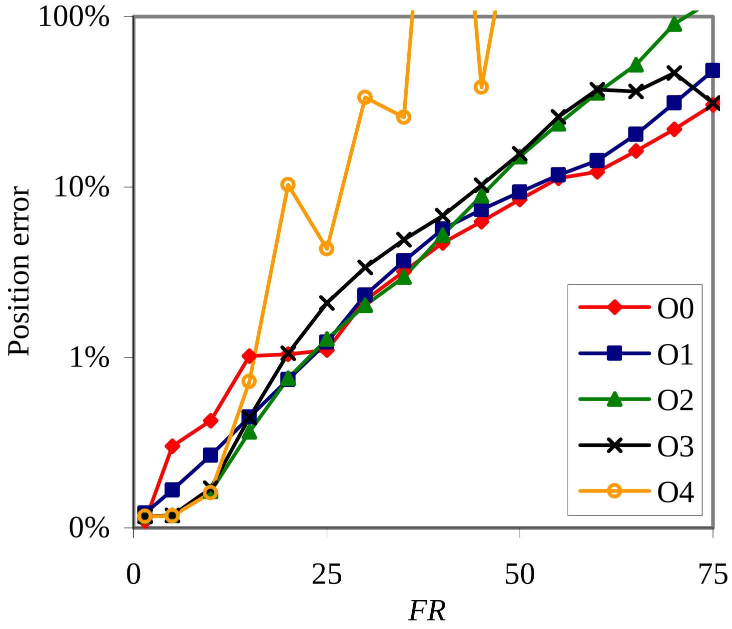
- **Different co-simulation strategies evaluated in a sweep of *FR***
  - *FR* varies from 1.5 to 100

$$FR = \omega_1 / \omega_2 \approx h_2 / h_1$$

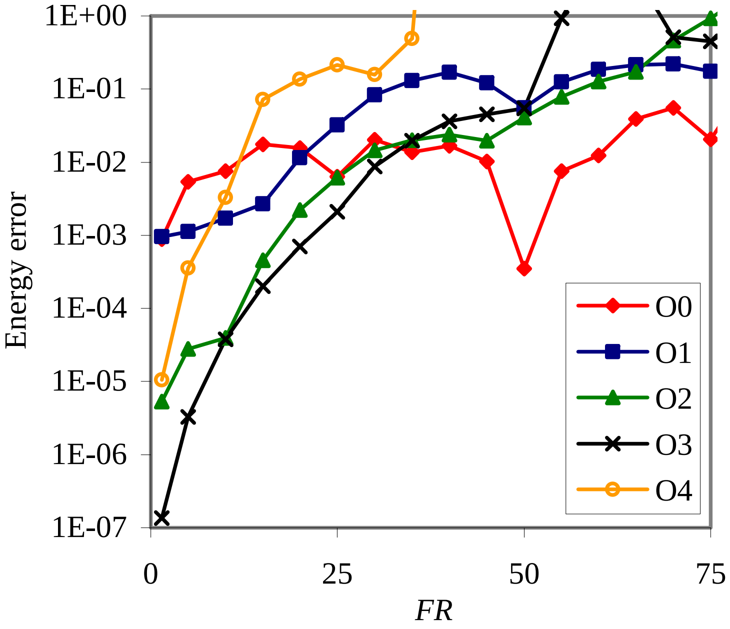
Test problem: results

- There is not a 'general purpose' technique valid for every ***FR***
- ***SF*** is suitable for ***FR*** < 50
  - With cubic interpolation (*O3*) for *FR* < 25
  - Without interpolation (*O0*) for 25 < *FR* < 50

Position error for x_1



Energy error



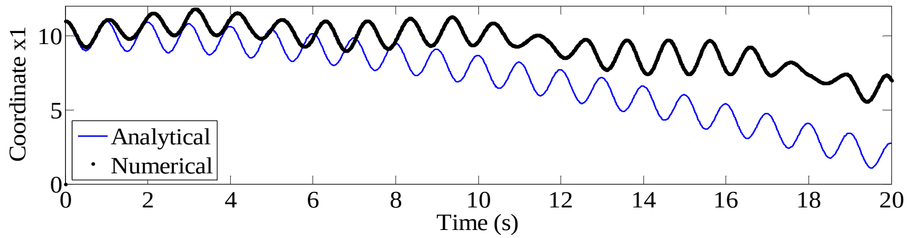
Test problem: results

- **For *FR* > 50**
  - *SF* scheme increases position error (phase error)
  - *FF* scheme increases energy error (amplification/attenuation)

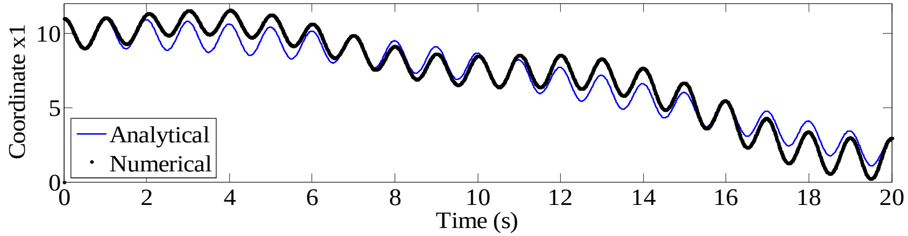
- **Smoothing techniques can reduce error for certain combinations of *FR* and interpolation order**

***FR* = 90**

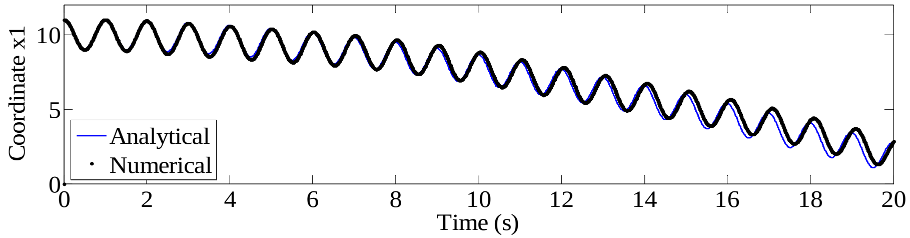
***SF* + *O*0**



***FF* + *O*3**

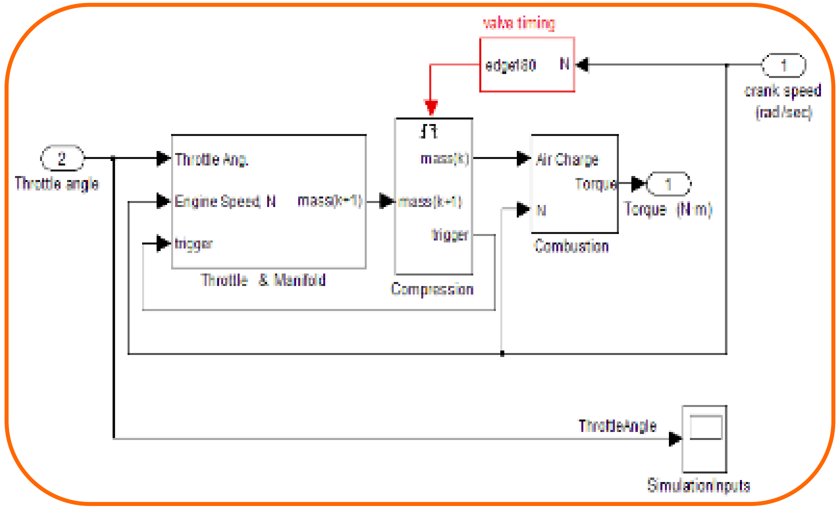


***Smoothing* + *O*3**

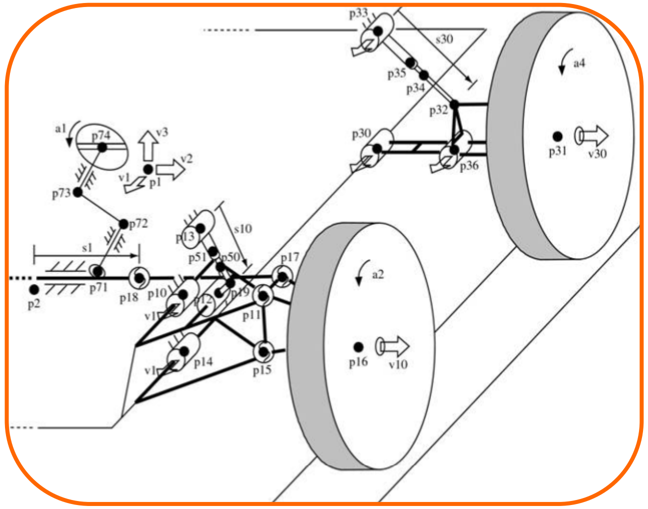


Application to a multiphysics problem

- **Simulink model of a thermal engine + MBS model of a kart**



$h_1 = 0.1$ ms

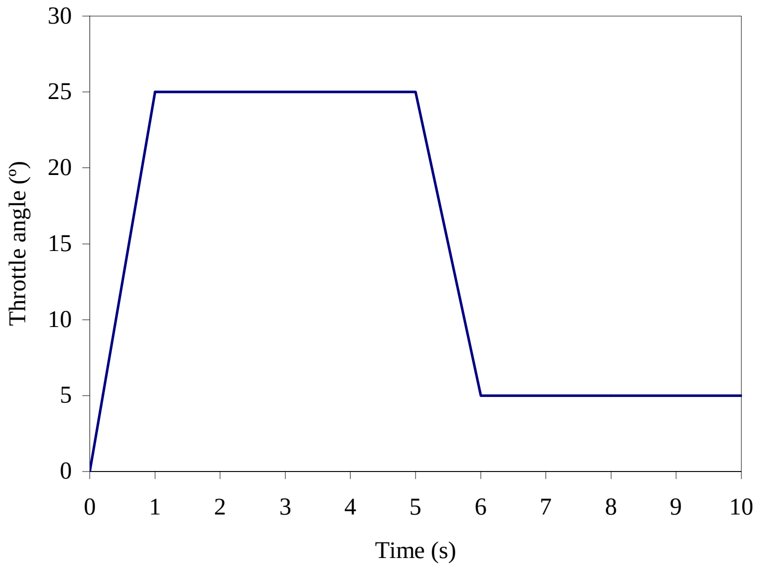


$h_2 = 10$ ms

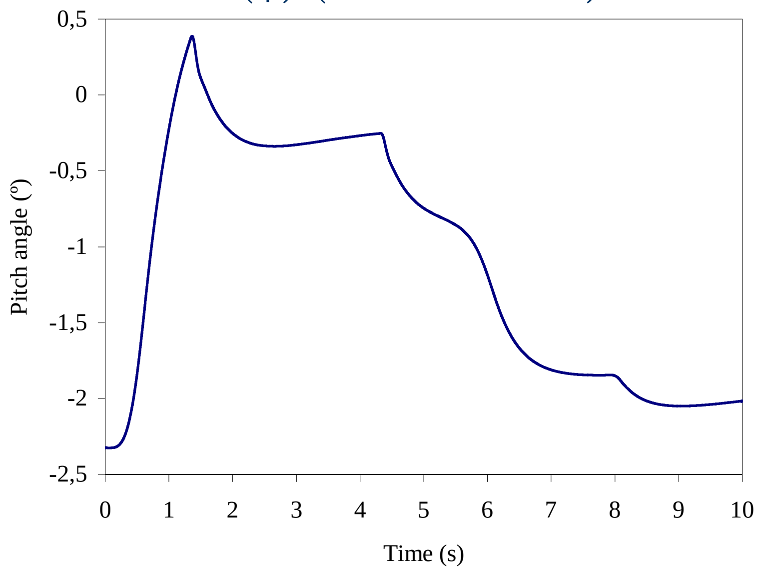
Application to a multiphysics problem

- **10 s simulation**

Input: angle law of throttle pedal (Simulink)



Output: pitch angle of vehicle (ψ) (MBS software)



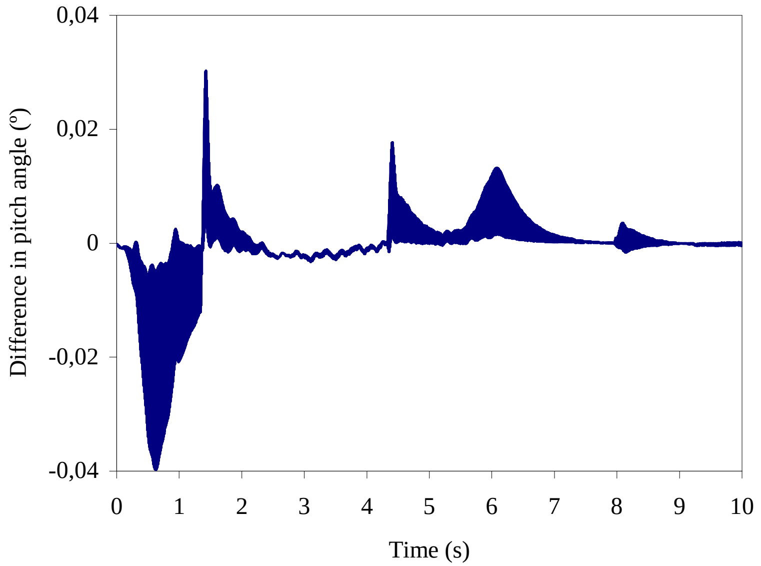
- **Reference simulation: $h_1 = h_2 = 0.1$ ms ($FR = 1$; $O0$)**
  - Elapsed time: 158.4 s

Application to a multiphysics problem

- Simulation with multirate techniques (increase of h_2 up to 10 ms)
- Measurement of deviations with respect to reference pitch ($\Delta\psi$)

	FR = 1	FR = 5	FR = 10	FR = 50	FR = 100
Elapsed time (s)	158.4	44.8	30.4	19.0	17.1
$\Delta\psi$ (º)	0	0.0031	0.0055	0.0252	0.0398

FR = 100



SF

*O*0 (Simulink)

*O*0 (MBS)

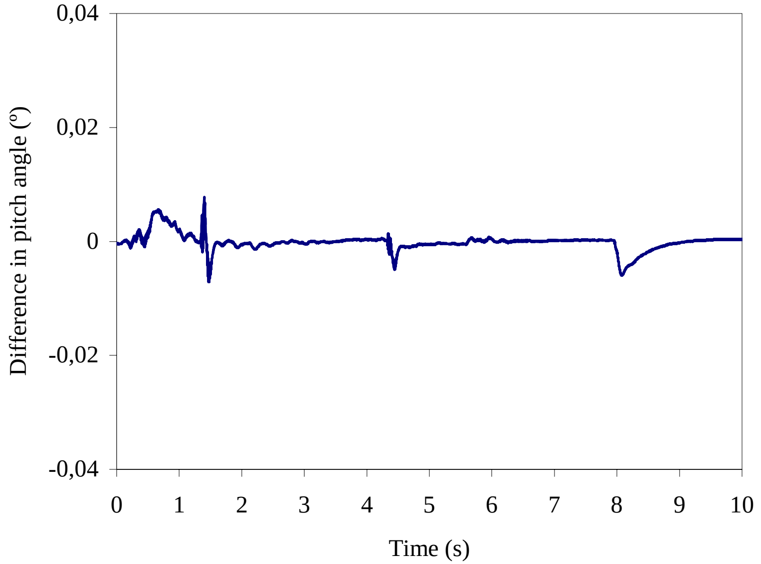
$\Delta\psi_{max}$ = 0.0398

Application to a multiphysics problem

- Simulation with multirate techniques (increase of h_2 up to 10 ms)
- Measurement of deviations with respect to reference pitch ($\Delta\psi$)

	FR = 1	FR = 5	FR = 10	FR = 50	FR = 100
Elapsed time (s)	158.4	44.8	30.4	19.0	17.1
$\Delta\psi$ (º)	0	0.0031	0.0055	0.0252	0.0398

***FR* = 100**



FF

***O*0 (Simulink)**

***O*1 (MBS)**

$\Delta\psi_{max}$ **= 0.0078**

Conclusions

- **A multirate co-simulation interface has been implemented and tested, which allows the use of**
  - Different interpolation/extrapolation polynomial orders
  - Fastest-first and slowest-first integration schemes
  - Smoothing
- **Use of the interface demonstrated**
  - In a simple example with analytical solution
  - In a complex multiphysics model
- **Multirate techniques enable reductions in simulation time (up to a factor of 9, in the shown example) with acceptable derived errors**
- **A way of finding the best co-simulation strategy beforehand is desirable**

Outline

Introduction

Software Architecture for MBS Simulation

Linear Algebra Implementations

Parallelization

Integration with MATLAB/Simulink

Multirate Co-simulation Methods

Conclusions and Future Research

Conclusions and future research

- **A modular software tool for MBS dynamic simulation has been built**
  - Open-source, object-oriented, implemented in C++
  - Extensible, through the addition of new modules

- **Optimization of MBS simulation codes explored through**
  - Streamlining of linear algebra routines
  - Non-intrusive parallelization
  - Communication with math software and block diagram simulators
  - Multirate integration

Conclusions and future research

- **Future research lines will focus on**

  - Assessment of the validity of the tested optimization techniques in recursive and semi-recursive formulations

  - Co-simulation of complex multiphysics systems
    - Research on a way to determinate beforehand the optimal co-simulation scheme when multirate techniques are introduced
    - Definition of general purpose indicators of the quality of the results of the co-simulation

Publications

- **The research conducted in this thesis has yielded the following papers**
  - M. González, F. González, D. Dopico and A. Luaces. On the effect of linear algebra implementations in real-time multibody system dynamics. *Computational Mechanics*, 41(4):607-615. 2008.
  - F. González, A. Luaces, U. Lugrís and M. González. Non-intrusive parallelization of multibody system dynamic simulations. *Computational Mechanics*, 44(4):493-504. 2009.
  - F. González, M. González and A. Mikkola. Efficient coupling of multibody software with numerical computing environments and block diagram simulators. *Multibody System Dynamics*, online first. 2010.
  - F. González, M.A. Naya, A. Luaces and M. González. On the effect of multirate co-simulation techniques in the efficiency and accuracy of multibody system dynamics. Submitted to *Multibody System Dynamics* in March, 2010 (undergoing revision process).

EFFICIENT IMPLEMENTATIONS AND CO-SIMULATION TECHNIQUES IN MULTIBODY SYSTEM DYNAMICS

Francisco Javier González Varela

Doctoral thesis

University of A Coruña

Ferrol, May 3rd, 2010