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

Open

Platform-independent

Object-oriented paradigm

C++

Flexibility - Efficiency
Structure of the simulation software

- Modular structure

**Additional functionality**
- Generation of equations of motion
- I/O routines
- Flexibility
- ...

**Basic MBS functionality**
- Model
  \[ M\ddot{q} + \Phi_q^T\lambda = Q \]
- Interface
- Dynamic formulation
- Numerical integrator
  \[ q_{n+1} = f(q_n, \dot{q}_n) \]

- Evaluation of dynamic terms
- Storage of generalized positions
- Global, natural coordinates
- Storage of dynamic terms
- Evaluation of accelerations
- Integration of the motion
Outline

Introduction

Software Architecture for MBS Simulation

Linear Algebra Implementations

Parallelization

Integration with MATLAB/Simulink

Multirate Co-simulation Methods

Conclusions and Future Research
Linear algebra operations in MBS simulation

- MBS codes make intensive use of linear algebra operations
  - Low-level scalar-matrix-vector operations \( B = A^T A \)
  - Solution of linear systems of equations \( Ax = b \)

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

<table>
<thead>
<tr>
<th>Fraction of elapsed time in computations</th>
<th>Dense</th>
<th>Sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>Residual and tangent matrix</td>
<td>48%</td>
<td>15%</td>
</tr>
<tr>
<td>Factorization and back-substitutions</td>
<td>44%</td>
<td>51%</td>
</tr>
<tr>
<td>Velocity and acceleration projections</td>
<td>4%</td>
<td>13%</td>
</tr>
<tr>
<td>Other</td>
<td>4%</td>
<td>21%</td>
</tr>
</tbody>
</table>
Benchmark setup: test problem

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

\[ g = 9.81 \text{ N/kg} \]

Diagram:
- Loop 1
- Loop L
- \( A_0 \rightarrow A_1 \rightarrow B_1 \rightarrow B_0 \)
- \( A_{L-1} \rightarrow A_L \rightarrow B_L \rightarrow B_{L-1} \)

Coordinate axes:
- \( x \)
- \( y \)
Benchmark setup: dynamic formulation

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

  \[
  M\ddot{q} + \Phi_q^T \alpha \Phi + \Phi_q^T \lambda^* = Q \\
  \lambda_{i+1}^* = \lambda_i^* + \alpha \Phi_{i+1}; \quad i = 0,1,2,...
  \]

- **Trapezoidal rule as integrator (implicit)**

- **Newton-Raphson iteration with approximate tangent matrix (SPD)**

  \[
  f(q) = Mq_{n+1} + \frac{\Delta t^2}{4} \Phi_{n+1}^T \left( \alpha \Phi_{n+1} + \lambda_{n+1} \right) - \frac{\Delta t^2}{4} Q_{n+1} + \frac{\Delta t^2}{4} M\dot{q}_n
  \]

  \[
  \left[ \frac{\partial f(q)}{\partial q} \right] \approx M + \frac{\Delta t}{2} C + \frac{\Delta t^2}{4} \left( \Phi_q^T \alpha \Phi_q + K \right)
  \]

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

  \[
  \left[ \frac{\partial f(q)}{\partial \dot{q}} \right] \ddot{q} = \left[ M + \frac{\Delta t}{2} C + \frac{\Delta t^2}{4} K \right] \ddot{q}^* - \frac{\Delta t^2}{4} \Phi_q^T \alpha \Phi_t
  \]

  \[
  \left[ \frac{\partial f(q)}{\partial \ddot{q}} \right] \dddot{q} = \left[ M + \frac{\Delta t}{2} C + \frac{\Delta t^2}{4} K \right] \dddot{q}^* - \frac{\Delta t^2}{4} \Phi_q^T \alpha \left( \dot{\Phi}_q \dot{q} + \Phi_t \right)
  \]
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
  - \( A + B \)
  - \( A^TA \)
  - Access to Jacobian matrix
- Different solvers have been tested (all CCS)

<table>
<thead>
<tr>
<th>Sparse linear solver</th>
<th>Matrix type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cholmod</td>
<td>Symmetric positive definite</td>
</tr>
<tr>
<td>KLU</td>
<td>General</td>
</tr>
<tr>
<td>SuperLU</td>
<td>General</td>
</tr>
<tr>
<td>Umfpack</td>
<td>General</td>
</tr>
<tr>
<td>WSMP</td>
<td>Symmetric indefinite</td>
</tr>
</tbody>
</table>
Performance of sparse linear solvers

- As a function of problem size

![Graph showing performance of sparse linear solvers as a function of problem size](image-url)
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)

![Graph showing CPU time as a function of percentage of non-zeros in the tangent matrix for different implementations: KLU, SuperLU, CHOLMOD, and WSMP. The graph illustrates how CPU time increases with the percentage of non-zeros.]
Best linear equation solver

- As a function of problem size and % of non-zeros in leading matrix
- 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 ($NNZ$)

<table>
<thead>
<tr>
<th>Type of leading matrix</th>
<th>$N \cdot (NNZ - 10)$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>&lt; 900</td>
</tr>
<tr>
<td>Symmetric positive definite</td>
<td>KLU (smooth problems)</td>
</tr>
<tr>
<td>Cholmod (rough problems)</td>
<td></td>
</tr>
<tr>
<td>Symmetric</td>
<td>KLU</td>
</tr>
<tr>
<td>Unsymmetric</td>
<td>KLU</td>
</tr>
</tbody>
</table>

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

GM
### Profiling of the initial implementation, for N variables

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

**Integration time step**
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 $\frac{NNZ}{N}$ ($NNZ =$ number of non-zeros in the leading matrix)

<table>
<thead>
<tr>
<th>Type of problem and dynamic formulation</th>
<th>$\frac{NNZ}{N}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A) Rigid bodies – Global formulations</td>
<td>$&lt;10$</td>
</tr>
<tr>
<td>B) Rigid bodies – Recursive formulations</td>
<td></td>
</tr>
<tr>
<td>Flexible bodies – Component mode synthesis</td>
<td>$10 – 30$</td>
</tr>
<tr>
<td>C) Flexible bodies – Finite element mesh</td>
<td>$30 – 100$</td>
</tr>
</tbody>
</table>

- 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

![Graph showing speedup of Pardiso versus best sequential solver](image)

\[ S = \frac{\text{elapsed time}_{\text{sequential}}}{\text{elapsed time}_{\text{parallel}}} \]

- 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

```c
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

\[ S = \frac{\text{elapsed time}_{\text{sequential}}}{\text{elapsed time}_{\text{parallel}}} \]

- 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 $\frac{NNZ}{N} > 10$

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

Introduction

Software Architecture for MBS Simulation

Linear Algebra Implementations

Parallelization

Integration with MATLAB/Simulink

Multirate Co-simulation Methods

Conclusions and Future Research
Multibody dynamics often needs multiphysics modelling
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
- **MATLAB Engine**
  - Inter-process communication

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

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

- 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)

[Diagram of the system with labels for angular speed ($\omega_1$), torque ($T$), and various components connected by arrows indicating flow.]
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

![Graph showing throttle angle and angular velocity](image)
Comparison of CPU-times

- For $\Delta t = 1$ 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**
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)*

  - **Block diagram**
    - Simulator
    - Multibody software
  - **Co-simulation interface**
    - **eval_slave** \((t_1, y_1)\)
    - **Time-history of** \([y_1, y_2]\)
    - **eval_master** \((t_2, y_2)\)
**Coupling strategy**

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

  - **Block diagram simulator**
    - $t_1^i$ to $t_1^{i+1}$
    - $h_1$

  - **Needs** $u_i|_{t_1^{i+1}}$

  - **Co-simulation interface**

  - **eval_slave** $(t_1, y_1)$

  - **Time-history of** $y_1$

  - **eval_master** $(t_2, y_2)$

  - **Multibody software**
    - $t_2^j$

  - **Block diagram simulator**
    - $t_1$

  - **Multibody software**
    - $t_2$

**Laboratorio de Ingeniería Mecánica**
**Universidade da Coruña**

http://lim.ii.udc.es
Coupling strategy

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

Block diagram simulator

<table>
<thead>
<tr>
<th>eval_slave</th>
<th>$(t_1, y_1)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time-history of</td>
<td>$\begin{cases} y_1 \ y_2 \end{cases}$</td>
</tr>
<tr>
<td>eval_master</td>
<td>$(t_2, y_2)$</td>
</tr>
</tbody>
</table>

Needs $u_2 |_{t_2^{j+1}}$ (extrapolation)
Coupling strategy

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

![Block diagram simulator](image)

Co-simulation interface

- **eval_slave** $(t_1, y_1)$
- **Time-history of**
  \[
  \begin{cases}
  y_1 \\
  y_2
  \end{cases}
  \]
- **eval_master** $(t_2, y_2)$

Store $t_2^{j+1}, y_2^{j+1}$
The block diagram simulator can resume its time-step

Co-simulation interface

- **eval_slave**: \((t_1, y_1)\)
- **Time-history of** \(y_1\)
- **eval_master**: \((t_2, y_2)\)
Coupling strategy

- And more time-steps can be taken

Block diagram simulator

\[ t^i_1 \rightarrow t^{i+1}_1 \rightarrow h_1 \rightarrow t^{i+2}_1 \]

Store \( t^{i+2}_1, y^{i+2}_1 \)

Co-simulation interface

\[
\begin{align*}
\text{eval_slave} & \quad (t_1, y_1) \\
\text{Time-history of} & \quad \begin{cases} y_1 \\ y_2 \end{cases} \\
\text{eval_master} & \quad (t_2, y_2)
\end{align*}
\]

Multibody software

\[ t^j_2 \rightarrow t^{j+1}_2 \]
Coupling strategy

- And the process starts again…

Block diagram simulator

Co-simulation interface

Multibody software

- eval_slave \((t_1, y_1)\)
- Time-history of \(\begin{cases} y_1 \\ y_2 \end{cases}\)
- eval_master \((t_2, y_2)\)
Test problem: modelling approach

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

- Purely mechanical

\[ \dddot{x}_1 = \frac{(k_1 x_1 + k_2 (x_1 - x_2))}{m_1} \]

\[ x_1(0), \quad \dot{x}_1(0), \quad \ddot{x}_1(0) \]

\[ m_1: \text{Simulink} \]

\[ m_2: \text{MBS software embedded in } S\text{-function block} \]
Test problem

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

- **Simulation:** 100 cycles of the fast subsystem
  
  \[
  x_1(t) = C_{11} \cdot \cos(\omega_1 t) + C_{13} \cdot \cos(\omega_2 t) \\
  x_2(t) = C_{21} \cdot \cos(\omega_1 t) + C_{23} \cdot \cos(\omega_2 t)
  \]

- **Different co-simulation strategies evaluated in a sweep of FR**
  - \( FR \) varies from 1.5 to 100
  
  \[ FR = \frac{\omega_1}{\omega_2} \approx \frac{h_2}{h_1} \]
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 $25 < FR < 50$
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

---

$FR = 90$

$SF + O0$

$FF + O3$

Smoothing + $O3$
Application to a multiphysics problem

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

\[ \dot{b}_1 = 0.1 \text{ ms} \]

\[ \dot{b}_2 = 10 \text{ ms} \]
Application to a multiphysics problem

- 10 s simulation

**Input:** angle law of throttle pedal (Simulink)

**Output:** pitch angle of vehicle ($\psi$) (MBS software)

- **Reference simulation:** $h_1 = h_2 = 0.1$ ms ($FR = 1; C0$)
  - 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 ($\Delta \psi$)

<table>
<thead>
<tr>
<th>FR</th>
<th>Elapsed time (s)</th>
<th>$\Delta \psi$ (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>158.4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>44.8</td>
<td>0.0031</td>
</tr>
<tr>
<td>10</td>
<td>30.4</td>
<td>0.0055</td>
</tr>
<tr>
<td>50</td>
<td>19.0</td>
<td>0.0252</td>
</tr>
<tr>
<td>100</td>
<td>17.1</td>
<td>0.0398</td>
</tr>
</tbody>
</table>

$FR = 100$

$\Delta \psi_{\text{max}} = 0.0398$

$SF$

$O0$ (Simulink)

$O0$ (MBS)
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 ($\Delta \psi$)

<table>
<thead>
<tr>
<th>FR</th>
<th>Elapsed time (s)</th>
<th>$\Delta \psi$ (°)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>158.4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>44.8</td>
<td>0.0031</td>
</tr>
<tr>
<td>10</td>
<td>30.4</td>
<td>0.0055</td>
</tr>
<tr>
<td>50</td>
<td>19.0</td>
<td>0.0252</td>
</tr>
<tr>
<td>100</td>
<td>17.1</td>
<td>0.0398</td>
</tr>
</tbody>
</table>

$FR = 100$

- $O0$ (Simulink)
- $O1$ (MBS)
- $\Delta \psi_{\text{max}} = 0.0078$
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
The research conducted in this thesis has yielded the following papers


- 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