Benchmarking of Multibody System Simulations: Points to Consider

Christian Lange[†], József Kövecses[*], Yves Gonthier[†]

[†] Canadian Space Agency (CSA), 6767 Route de l'Aeroport, St-Hubert, Quebec, Canada, J3Y 8Y9, {Christian.Lange|Yves.Gonthier}@space.gc.ca

[*]Department of Mechanical Engineering, McGill University, 817 Sherbrooke St. West, Montreal, Quebec, Canada, H3A 2K6, Jozsef.Kovecses@mcgill.ca

Abstract

A great number of multibody system (MBS) simulation packages exist today (e.g., ADAMS, ALASKA, Autolev, DADS, Dymola, DynaFlexPro, Dynawiz, SAMCEF MECANO, MOBILE, NewEul, RecurDyn, Robotran, SimMechanics, SIMPACK, Visual Nastran 4D). Each package comes with its own implementation of one or several algorithms for the formulation of the kinematics, statics and dynamics formulations used to model, analyze and simulate rigid and flexible multibody systems. Furthermore, one can find several publications on improvements, extensions or completely new derivations of MBS model formulations.

For users and developers of MBS software it is of great importance to validate the correctness of simulation results. It is also critical to ascertain that they were obtained in the most efficient manner. In short, it is necessary to benchmark them. This is especially true for real-time simulations with hardware or a human operator in the loop or for parametric and sensitivity studies or design optimizations.

As already pointed out in [1], there is a lack of standard benchmarks to do so. They found that benchmarking is typically done on an individual basis, i.e., different sets of problems are used as well as different procedures to measure computational efficiency, making it almost impossible to compare the performance of different available simulation methods in an objective and quantitative way. Hence, they propose a benchmarking system for different problem categories, including documentation and validated reference solutions in a standard format and a procedure to measure the computational efficiency of a given simulation software.

The contribution of this paper is a more complete list of key factors that should be considered for a benchmarking system. The following factors should at least be specified:

(1) Hardware factors: execution platform, i.e., same number and types of processors with clock speed;

(2) Software factors: (a) operating system, e.g., Windows, QNX; (b) selected compiler, compiler flags; (c) code implementation language, e.g., FORTRAN, C, Java, or Matlab m-script, S-Function, Real-Time-Workshop (RTW) compiled code; (d) code generation method, e.g., hand-coded, auto-generated, auto-optimized; (e) code implementation quality;

(3) Accuracy factors: (a) computation of the variables or quantities compared, e.g., accelerations or integrated values in case of forward dynamics (type and settings of integrator); (b)

computation of absolute and relative errors and thresholds, e.g., for rotation matrices, Jacobians, mass matrices;

(4) Efficiency factors: (a) operations count; (b) execution time; (c) possible function or (sub-)expression re-use or data sharing in code/diagram;

(5) Model factors: (a) topology of the model; (b) size of the model, i.e., number of degrees-of-freedom (DoF); (c) number of hard-coded and non-hard-coded parameters;

As suggested in [1], it, hence, seems to make sense to develop a benchmarking system with a library of models categorized into problem classes containing a full and clear model description.

We found that this system should be actually a whole benchmarking infrastructure providing additionally a dedicated server space as execution platform and a general MBS software with interfaces to common software packages as well as the possibility to upload single model functions, (e.g., to compute the mass matrix, written in different languages, e.g., FORTRAN, C and Java). This would make it possible to reduce the complexity of the benchmarking process drastically. To underline the importance of the factors mentioned above a few key findings are given below:

(1) Automatic code generation: For a six DoF serial manipulator the number of multiplications and additions varied for the same symbolic mass matrix function in three code optimization runs using Maplesoft's Maple code optimization tool by 7% and 3.5%, respectively.

(2) Numerical result: the largest relative error between two corresponding elements of those three mass matrices was 6e-9%.

(3) Code implementation quality: In a simple test program, the assignment of a matrix with values was about twice as fast when using a variable defined as a single array (`n[19*3*3]`) instead of twice de-referenced variable (`m[19][3][3]`).

(4) Model parameters, compiler flags: The CPU times necessary to compute the forward dynamics of a serial 3-dimensional chain mechanism with 21 DoF are shown in Table 1. A symbolic O(n^3) (highly optimized C-code using Maple) and a numeric O(n) algorithm (hand-coded C-code) [2] under Windows are used with either hard-coded (hcp) or non-hard-coded (nhcp) model parameters. The RTW compiled code and the C-code functions wrapped with an S-function are called from a Matlab/Simulink diagram. For the O(n^3) case, Simulink's LDL Solver is used to compute the accelerations based on the mass matrix and the non-linear force vector.

	RTW	S-function
O(n^3) nhcp	61	80
O(n^3) hcp	72	139
O(n) nhcp	57	74
O(n) hcp	54	71

Table 1: CPU time comparison in μ seconds.

The Multibody Toolbox (MuT), a CSA in-house modelling tool, was used for the dynamics modelling and served as benchmarking infrastructure.

References

[1] M. González, D. Dopico, U. Lugrís, and J. Cuadrado. A benchmarking system for MBS simulation software: Problem standardization and performance measurement. *Multibody System Dynamics*, 16(2):179–190, September 2006.

[2] P. F. Wong. Algorithms for efficient dynamics simulation of space robotic systems. Master's thesis, Department of Mechanical Engineering, McGill University, 2006.