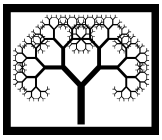
Paper 5



Proceedings of the Eighth International Conference
on Engineering Computational Technology,
B.H.V. Topping, (Editor),
Civil-Comp Press, Stirlingshire, Scotland

A Parallelization Algorithm for Non-Smooth Multibody Dynamics

J. Clauberg and H. Ulbrich
Institute of Applied Mechanics
Technische Universität München, Garching, Germany

Abstract

This paper proposes an algorithm for parallelization of non-smooth multibody simulations. In recent decades simulations have become one of the most important methods in product development. It allows to investigate, analyze and optimize complicated technical systems by methods like parameter variation and numerical optimization. But with simulation models getting more and more complex, the computational costs are steadily rising up, leading to unacceptable long simulation times. One of the most promising ways to reduce computational time are parallelization methods. Today, nearly every desktop computer comprises more than one core per CPU, which can only be used gainfully by developing new parallel algorithms for multibody dynamics. Beside existing algorithms like parallel co-simulation, parallelization within the numerical integrator and parallel O(log(N)) algorithms for tree-structured multibody systems, the method presented in this paper proposes the parallel calculation of the system update (gap-distances, gap-velocities, right-hand-sides, Jacobian-matrices, etc.). Efficiency is shown by applying the algorithm on one rigid multibody system and on two flexible multibody systems comprising up to 288 degrees of freedom and 4752 set-valued force parameters.

Keywords: parallel processing, multibody dynamics, parallelization, numerical simulation.

1 Introduction

The efficient and detailed simulation of complex technical multibody systems is a very important method in todays research and product development. Current multibody systems comprise a high number of degrees of freedom as well as uni- and bilateral constraints which can basically be modeled smooth (single-valued) or non-smooth

(set-valued) (see Figure 1).

Non-smooth multibody dynamics [2] thereby plays a special role in two main points of view. First, because of its ability to depict contacts and friction in a physically motivated way. Set-valued force laws are used to describe the physical behavior of the system and require special numerical methods for their solution. Second, the computational time using set-valued force laws is lower than using single-valued force laws in the majority of cases. Using a high number of single-valued force laws with appropriate high stiffness and damping values, leads to numerically stiff equations of motion. Numerically stiff equations of motion need special time-consuming integration schemes or small time step sizes. This drawback can be avoided by using non-smooth force laws in combination with special integration schemes. The probably most important development in non-smooth multibody dynamics is the solution of the set-valued force laws by means of the proximal point to a convex set - a mathematical method from the convex algebra - instead of the formulation as linear complementarity problems [2]. This new approach caused a significantly speedup of the solution of the set-valued force laws of a factor of about 50 or more. A comparison of different single- and set-valued force laws can be found in [1]. FOERG ET. AL. compares different single- and set-values force laws on the example of a large valve-train of a combustion engine. Using set-valued force laws in combination with a time-stepping integration scheme is about three times faster as using single-valued force-laws in combination with a DOPRI5 integration scheme.

	State-Updates	Set-Valued Laws	Plotting	Rest
Spheres in Cup	67,23 %	17,23 %	12,56 %	2,98 %
Triple Woodpecker	94,12 %	0,07 %	1,41 %	4,4 %
Rotorsystem	80,8 %	11,32 %	4,17 %	3,71 %

Table 1: Distribution of Computational Costs

But now, not the solution of the force laws itself is the main time consuming part of the simulation, but rather the calculations of the system states and variables. In each time step the system state (gap-distances, gap-velocities, right-hand-sides, Jacobian-matrices, state-dependent variables, etc.) has to be calculated once or more. This process is called ”system-update” further on. The distribution of the computational costs between the system-update, the solution of the set-valued force laws, the data plotting and the rest is shown in Table 1 for the examples taken into account in this paper (see Section 3). The algorithm proposed in this paper allows to calculate the system-update in parallel leading to significantly lower computational times. There are also several other possibilities to take advantage of multicore architectures. Parallel co-simulations allow to split technical simulation models into different submodels and to calculate them in parallel [3]. Furthermore, Huber [4] presents a method to use parallelization within the integration scheme. He proposes to simulate the same system in parallel with different step-sizes in order to adjust the step-size of the inte-

gration scheme or to generate schemes of higher order without dramatically increasing computational costs. Featherstone [5] proposes a parallel O(log(N)) method for tree-structured multibody systems called "divide-and-conquer"-algorithm. This algorithm is very fast if the number of involved bodies and the number of available CPU cores are very high. Simulating "normal" technical systems, ordinary O(N) or even methods that require the inversion of the mass-matrix (O(N^3)) are more adequate.

2 Non-Smooth Multibody Dynamics

In the first subsection the mathematical formulation of non-smooth multibody systems with uni- and bilateral contacts and friction is outlined. For the understanding of the parallelization method presented in this paper it is not required to describe the set-valued force laws and their solution in detail, therefore only a brief outline of their formulation is given in the second subsection.

2.1 Mathematical Formulation

Point of departure of the mathematical description of a uni- and bilateral constrained non-smooth multibody system is the *Measure Differential Equation* [2, 6, 7]

$$\vec{M}\,\mathrm{d}\vec{u} = \vec{h}(\vec{u},\vec{q},t)\,\mathrm{d}t + \vec{W}\,\mathrm{d}\mathbf{\Lambda}. \tag{1}$$

In Equation (1) $\vec{M}$ denotes the mass-matrix, $\vec{h}$ contains all external and gyroscopical forces depending on the generalized velocities $\vec{u}$, generalized positions $\vec{q}$ and the time t. $\mathbf{\Lambda}$ describes the force reactions of the set-valued force laws with $\vec{W}$ containing their directions.

The acceleration measure $\mathrm{d}\vec{u} = \dot{\vec{u}}\,\mathrm{d}t + (\vec{u}^{+} - \vec{u}^{-})\,\mathrm{d}\eta$ consists of a continuous part $\dot{\vec{u}}\,\mathrm{d}t$ and an atomic part $(\vec{u}^{+} - \vec{u}^{-})\,\mathrm{d}\eta$, with the left and right limit $\vec{u}^{-}$, $\vec{u}^{+}$ and the Dirac point measure $\mathrm{d}\eta$. Analogously, the measure for impulses $\mathrm{d}\mathbf{\Lambda} = \boldsymbol{\lambda}\,\mathrm{d}t + \mathbf{\Lambda}\,\mathrm{d}\eta$ can be split into a continuous part $\boldsymbol{\lambda}\,\mathrm{d}t$ and an atomic part $\mathbf{\Lambda}\,\mathrm{d}\eta$. Integrating Equation (1) under the consideration of the *DIRAC* delta function [4] yields the equations of motion of a smooth constrained system

$$\vec{M}\,\dot{\vec{u}} = \vec{h}(\vec{u},\vec{q},t) + \vec{W}\,\boldsymbol{\lambda}. \tag{2}$$

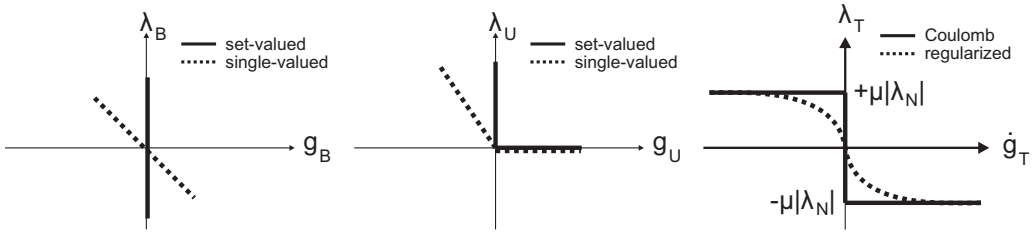
as well as the impact equations

$$\vec{M}_i\left(\vec{u}_i^{+} - \vec{u}_i^{-}\right) = \vec{W}_i\,\mathbf{\Lambda}_i \qquad \forall i \in N \tag{3}$$

being valid at times t_i of impact.

2.2 Set-Valued Force-Laws

In order to calculate the unknown accelerations $\dot{\vec{u}}$ in Equation (2) and the post-impact velocities $\vec{u}_i^{+}$ in Equation (3) it is necessary to know the reactions $\boldsymbol{\lambda}$ and $\mathbf{\Lambda}_i$ governed



(a) Bilateral constraint. (b) Unilateral constraint. (c) Planar COULOMB friction.

Figure 1: Force Laws for Bi- and Unilateral Contacts and Friction.

by the set-valued force laws $(\vec{q},\, \vec{u},\, \boldsymbol{\lambda},\, \boldsymbol{\Lambda}_i,\, t) \in \mathcal{N}$. The main advantage of the set-valued force laws is that they are physically motivated in contrast to single-valued force-laws (e.g. spring-damper combinations). It is possible to define contacts with impact laws (e.g. Newton impacts) as well as to model stick-slip effects. On the other hand, set-valued force laws require special integration schemes such as time-stepping schemes or event-driven schemes [4].

Figure 1 shows the basic three set-valued force laws used in mechanical multibody dynamics. A bilateral force law (Figure 1(a)) implies a bilateral constraint of the form

$$g_B = 0, \qquad \lambda_B \in \mathbb{R} \tag{4}$$

with g_B the normal distance between the interacting bodies and λ_B the corresponding force. The second force law represents a contact in the mechanical point of view (Figure 1(b)). It is given by the *Signorini-Fichera*-condition

$$g_U \geq 0, \quad \lambda_U \geq 0, \quad g_U\, \lambda_U = 0, \tag{5}$$

with g_U the normal distance between the interacting bodies and λ_U the corresponding force. Furthermore, *COULOMB*-friction (Figure 1(c)) is taken into account which can be mathematically formulated by Equation (6) with the relative tangential velocity $\dot{\vec{g}}_T$ and the friction coefficient μ. The force in the contact point can be decomposed in a part $\lambda_N \in \{\lambda_B,\, \lambda_U\}$ normal to the contact plane and a part tangential $\boldsymbol{\lambda}_T$ in friction direction.

$$\begin{aligned} \dot{\vec{g}}_T = \vec{0} \quad &\Rightarrow \quad |\boldsymbol{\lambda}_T| \;\leq \mu\, |\lambda_N| \\ \dot{\vec{g}}_T \neq \vec{0} \quad &\Rightarrow \quad \boldsymbol{\lambda}_T \;= -\frac{\dot{\vec{g}}_T}{|\dot{\vec{g}}_T|}\, \mu\, |\lambda_N| \end{aligned} \tag{6}$$

For the numerical integration of the system, the impact laws are formulated on velocity level leading to the substitution of g by $\dot{g}^+$ and λ by Λ. For the solution of the set-valued force laws the proximal point to a convex set, a method from the convex algebra, is used [2].

3 Computational Framework and Examples of Use

The parallelization method is implemented in the C++ multibody simulation framework MBSim (developed at the Institute of Applied Mechanics, TU München under

the GPL License [8]). All simulations are done under OpenSuse Linux 11.4 running on a desktop computer with two Intel Xeon E5620 processors (each with four real cores) and 12 GB RAM. The realization of the parallel code is done with OpenMP (Open Multiprocessing [9]).

To show the efficiency of the parallelization method, two academic and one industrial example are taken into account. These examples can be seen in Figure 2.

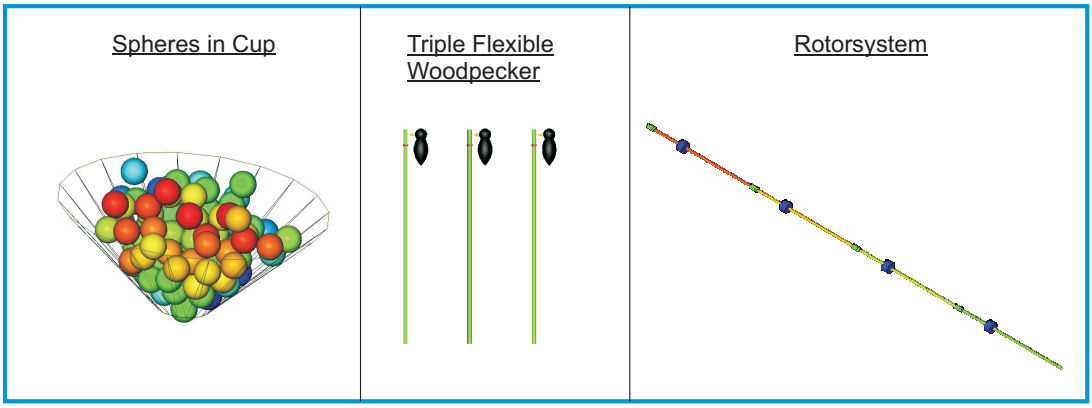


Figure 2: Spheres in Cup, Triple Flexible Woodpecker, Rotorsystem

This paper has two main objectives. One the hand hand the efficiency of the proposed parallelization method will be demonstrated on the three examples taken into account and on the other hand the difference between using single- and set-valued force laws will be shown by comparing the necessary computational times of the example ”Spheres in Cup” using the two kinds of contact mechanics.

3.1 Spheres in Cup

This academic example consists of 96 spheres that are falling into a cup. This configuration leads to 96 degrees of freedom and 4752 gap calculations. In each time step the contact kinematic between 96 spheres and a frustum (cup) and the contact kinematic between the spheres themselves must be solved once or more. On the one hand, the contact kinematic between two spheres is very simple leading to low calculation times and on the other hand, the contact kinematic between a sphere and a frustum is little more expensive in the computational point of view leading to little longer computational times. In Section 6 the update-g method is further examined regarding the parallelization method. In addition to that, the difference between modeling the contacts smooth and non-smooth is shown.

3.2 Triple Flexible Woodpecker

This academic example consisting of three woodpeckers each on an elastic pole is often used in the field of teaching stick-slip effects. By varying the friction coefficient

between the woodpecker and the elastic pole, the influence on the dynamic of the system can be shown. The contacts between the woodpecker and the pole are modeled non-smooth. The system comprises 96 degrees of freedom, 24 gap calculations and 54 force parameters. Two update-methods are especially interesting in this example. The update-g method (calculation of gap distances) and the update-h method (calculation of the right hand sides). The contact kinematic between a flexible beam and a point contour is time consuming, because the contact point must be searched by methods like Newton-solvers. The calculation of the right hand side of this system is time consuming because of the flexible beam, that is based on a finite element model. Therefore these two update methods will be further examined in Section 6.

3.3 Rotorsystem

This industrial example from the field of rotor dynamics comprises three flexible rotors (model based on finite elements) that are mounted with bearing clearance and friction leading to 192 degrees of freedom, 28 gap calculations and 36 force parameters. Taking this example into account, the efficiency of the parallelization method will be demonstrated by applying it to the update-Jac method (calculation of Jacobian matrices), the update-SDV method (calculation of variables depending on the system state), the update-h method (calculation of the right hand sides) and the update-M method (calculation of the mass-matrices of each object within the system).

The specifications of each example are summarized in Table 2, where **size of** $\vec{g}$ denotes the number of gap-calculation, **size of** $\vec{q}$ denotes the number of degrees of freedom and **size of** $\boldsymbol{\lambda}$ denotes the number of force-parameters.

Table 2: Example Specifications

	Spheres in Hopper	**Trip. Flex. Woodpecker**	**Rotorsystem**
Contact Mechanics	single- & set-valued	set-valued	single- & set-valued
Size of q	96	96	192
Size of g	4752	24	28
Size of λ	4752	54	36

4 Parallelization Method

As depicted in the introduction, the proposed algorithm allows to calculate the system-update in parallel.

The method itself is explained on the example of the ”update-g” method (update-g calculates all gap distances within the system). In each integration step all gap

distances must be calculated once or more, which can be done in parallel. The main problem of all parallelization methods is the necessary overhead for the management of the parallelization, which is growing with the number of used cores.

For this reason only operations that need enough computational time should be parallelized. Otherwise the parallelized simulation would slow down due to the overhead of parallelization.

This implies that the decision which gap-functions are calculated in parallel should depend on the *used number of cores* and the *individual computational cost* of the gap-functions. Equation (7) and (9) describe the update-g functions with n the number of gap-functions, $\vec{g}$ the gap functions and s, p the indices for serial and parallel calculation. The algorithm dynamically measures the computational time for each gap-calculation g_i in Equation (7) during the first steps of the simulation and compares the measured time with a defined border between serial and parallel computation (*BorderTime*). If the needed time is lower than the *BorderTime* then the considered gap-function g_i is put in the sequential container $\vec{g}_s$, otherwise in the parallel container $\vec{g}_p$ (Equation (9)).

$$g_i = f(\vec{q}, \vec{u}, t), \quad i = 1...n. \tag{7}$$

$$g_{i,s} = f(\vec{q}, \vec{u}, t), \quad i = 1...n_s \quad \text{and} \quad g_{i,p} = f(\vec{q}, \vec{u}, t), \quad i = 1...n_p \tag{8}$$

$$n = n_s + n_p. \tag{9}$$

The method is summarized in Table 4.

Parallelization Algorithm
1. time measurement: measure computational time t_i for $i = 1...n$ of $g_i = f(\vec{q}, \vec{u}, t)$
2. decision between serial and parallel computation
a) if t_i of $g_i \leq$ *BorderTime*: add g_i to $\vec{g}_s$ (serial computation)
b) if t_i of $g_i >$ *BorderTime*: add g_i to $\vec{g}_p$ (parallel computation)
3. perform calculation
a) compute $g_{i,s}$ for $i = 1...n_s$ sequentially
a) compute $g_{i,p}$ for $i = 1...n_p$ in parallel

Table 3: Parallelization Algorithm

In contrast to other areas of simulation like finite elements or CFD (computational fluid dynamics), multibody dynamics is characterized by very small and fast individual calculations, for example gap calculations. For this reason, the main purpose of the proposed parallelization algorithm is to keep the overhead for the parallelization as low as possible as mentioned before. The partitioning into a sequential and a parallel container is just the first step of the whole parallelization method. This step ensures that only calculations that need enough computational time are parallelized. But due to the fact that the individual calculations in the parallel container need different calculation times (the calculation times can vary by several powers of ten), some kind

of load balancing is needed. Normal dynamic load balancing algorithms like the ones provided by OpenMP are not completely suitable. Therefore, new semi-dynamic load-balancing algorithms based on Karmarkar-Karp Heuristics and Greedy Algorithms are under current research by the author.

5 Discussion

As depicted in Section 3 the parallelization method is applied to the three examples ”Spheres in Cup”, ”Triple Flexible Woodpecker” and ”Rotorsystem”.

For closer examination of the influence of the criteria *number of cores* and *Border-Time* to the efficiency of the parallelization method, the update-g method (calculation of gap distances) of the example ”Spheres in Cup”, the update-g method (calculation of gap distances) of the example ”Triple Flexible Woodpecker” and the update-M (calculation of the individual mass matrix distributions to the global mass matrix $\vec{M}$) of the example ”Rotorsystem” are taken into account.

5.1 Spheres in Cup: Update-g Method

As depicted in Section 3, the gap calculation of the example ”Spheres in Cup” are characterized by the very fast solution of the contact kinematic between two spheres and the little slower contact kinematic between a sphere and a frustum.

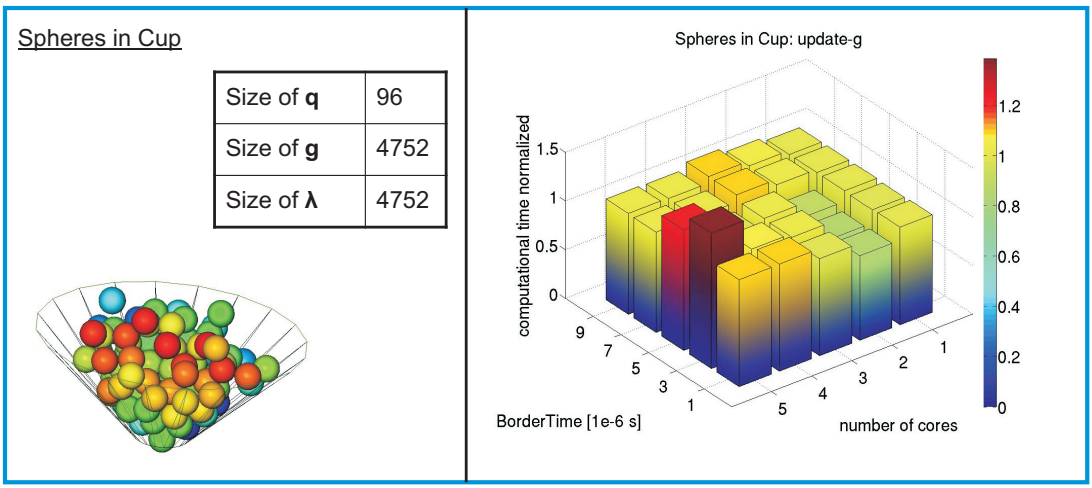


Figure 3: Specifications and Analysis of the ”Spheres in Cup”

These fast gap calculation leads to the assumption that the overhead due to parallelization should be relatively small. This assumption can be confirmed by Figure 3. It shows the normalized calculation time of the simulation (only the update-g method is

parallelized) depending on the *BorderTime* and the *number of cores*. The calculation time is normalized to the sequential simulation time on one core.

The optimum lies at a *BorderTime* of about $3 \cdot 10^{-6}$sec and three cores. Using more than two cores leads to a too large overhead for the management of the parallelization. Using a *BorderTime* larger than about $7 \cdot 10^{-6}$sec leads to an empty parallel calculation container, because nearly all gap calculation are faster than $7 \cdot 10^{-6}$sec.

5.2 Triple Flexible Woodpecker: Update-g Method

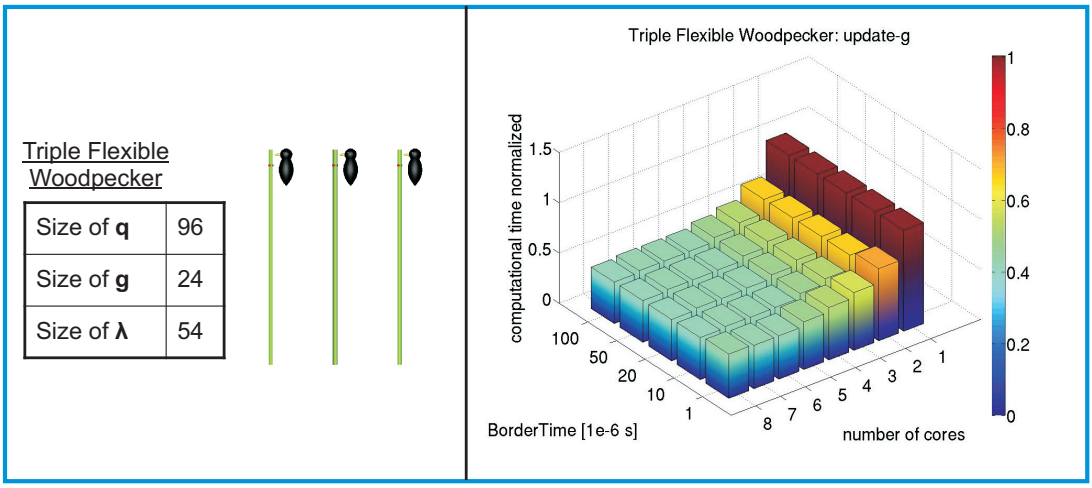


Figure 4: Specifications and Analysis of the "Triple Flexible Woodpecker"

Figure 5 shows the same information for the simulation of the "Triple Flexible Woodpecker".

The gap calculation within the example "Triple Flexible Woodpecker" can be divided in two categories. The contact kinematic between the woodpecker and the flexible beam is certainly time consuming because the contact point must be numerically searched on the flexible contour. All other gap calculation within the system are relatively easy leading to low computational times.

Figure 4 shows that the *number of cores* should be seven and the *BorderTime* should be about $7 \cdot 20^{-6}$sec. This can be explained by the mentioned to classes of contact kinematics. Choosing the *BorderTime* smaller than $20 \cdot 10^{-6}$sec would mean that also the easy gap calculations within the system would be calculated in parallel. This would lead to a lower speedup because the necessary overhead due to the parallelization would be larger than the saved time.

But the parallelization overhead is negligible compared to the necessary calculation time for the contact kinematics between the flexible beam and the woodpecker. Therefore, a high number of cores is usable. Using seven cores rather than eight cores is

better, because the used computer comprises only eight cores and the operating system also needs computational resources.

5.3 Rotorsystem: Update-M Method

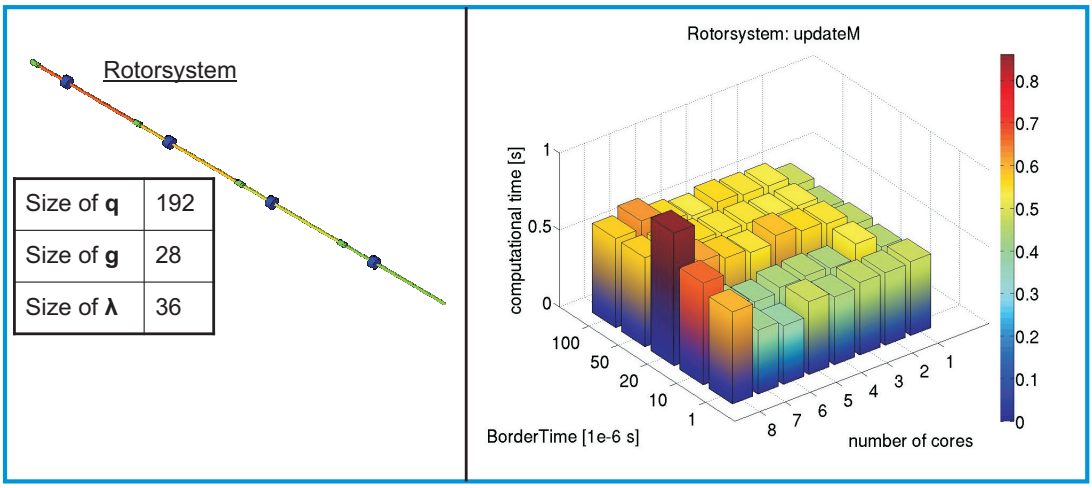


Figure 5: Specifications and Analysis of ”Rotorsystem”

Figure 5 shows the normalized computational time for the calculation of the update-M method of the ”Rotorsystem” depending on the *BorderTime* and the *number of cores*. The computational time is normalized to the sequential calculation time of the update-M method.

This figure expresses that the speedup due to parallelization of the update-M process is essentially depending on the *BorderTime* and the *number of cores*. For example, if the *number of cores* is to high, the overhead due to parallelization may be too large and if the *BorderTime* is chosen too high, too little gap calculations are done in parallel.

It can be concluded that the possible speedup is essentially depending on the *BorderTime* and the *number of cores*. The optimal choice of the two parameters should be done by the algorithm itself which is topic of current research.

6 Results

In this section the results of applying the method to different update-processes of the three examples are shown. The optimal choice of the parameters *BorderTime* and the *number of cores* is chosen and the computational time for purely sequential calculation and parallel calculation is measured. The results are recapitulated in Table 4, 5 and 6.

6.1 Spheres in Cup: Parallelization Method

considered update-process	optimal parameters (BT[a],cores[b])	part of whole[c]	seq. time	par. time	speedup
update-g[d]	(3,2)	74,78%	10,09s	7,12s	27,26%

[(a)]: *BorderTime* $[10^{-6}s]$, [(b)]: *number of cores for computation*

[(c)]: Part of the computational time of the considered update-process of the whole computational time

[(d)]: calculating each gap distance within the system

Table 4: Results of Applying the Parallelization Algorithm to the "Spheres in Cup"

Table 4 shows that the parallelization method is able to reduce the computational time of the update-g method about 26,27% using two cores and a *BorderTime* of $3 \cdot 10^{-6}$sec.

6.2 Triple Flexible Woodpecker: Parallelization Method

considered update-process	optimal parameters (BT[a],cores[b])	part of whole[c]	seq. time	par. time	speedup
update-g[d]	(20,8)	70,8%	11,57s	2,13s	81,60%
update-h[e]	(1,4)	14,62%	2,12s	0,76s	64,15%

[(a)]: *BorderTime* $[10^{-6}s]$, [(b)]: *number of cores for computation*

[(c)]: Part of the computational time of the considered update-process of the whole computational time

[(d)]: calculating each gap distance within the system, [(e)]: calculating right-hand-sides h for each object

Table 5: Results of Applying the Parallelization Algorithm to the "Triple Flexible Woodpecker"

Table 5 shows that with the proposed algorithm it is possible to reduce the computational time of the update-g process of the "Triple Flexible Woodpecker" by about 81,60% and of the update-h process by about 64,15%. The update-g methods uses 8 cores and a *BorderTime* of $20 \cdot 10^{-6}$sec, the update-h method uses 4 cores and a *BorderTime* of $1 \cdot 10^{-6}$sec.

6.3 Rotorsystem: Parallelization Method

The results of for the system "Rotordynamics" are depicted in Table 6. The computational time for the update-StateDependentVariables process (update-SDV) can be reduced by about 64%, of the update-Jacobians (updateJac) by about 42%, of the update-h by about 27% and of the update-M by about 52%.

considered update-process	optimal parameters (BT[a],cores[b])	part of whole[c]	seq. time	par. time	speedup
update-Jac[f]	(50,7)	10,8%	1,01s	0,59s	41,58%
update-SDV[g]	(1,4)	14,62%	2,12s	0,76s	64,15%
update-h[e]	(10,3)	7,28%	0,44s	0,32s	26,80%
update-M[h]	(1,6)	11,4%	0,39s	0,19s	51,28%

$^{(f)}$: calculating the necessary Jacobian matrices

$^{(g)}$: calculating variables depending on the system state

$^{(h)}$: calculating the mass-matrix of each object within the system

Table 6: Results of Applying the Parallelization Algorithm to the "Rotorsystem"

It can be summarized that the efficiency of the parallelization method depends on the system type. Some systems are more suitable for this method, others only show average speedup results. Even more potential can be seen in combining the proposed parallelization algorithm with the mentioned parallelization within the numerical integrator and parallel co-simulation. This is also topic of future research.

6.4 Spheres in Cup: Smooth- and Non-Smooth Mechanics

Furthermore, the system "Spheres in Cup" is well suited to shows the difference between smooth- and non-smooth contact mechanics regarding the computational time.

Therefore, the system "Spheres in Cup" is first modeled using smooth contact mechanics (single-valued force laws) and second modeled using non-smooth contact mechanics.

The version using smooth contact mechanics is integrated by a common "state of the art" ode solver (LSODE) and the non-smooth version is integrated by a fixed step-size, half-explicit time-stepping integration scheme. The half-explicit time-stepping integration scheme has no step-size control. Therefore, the system was integrated with a little lower time step size than the average time step size a time-stepping integration scheme with step size adjustment ([4]) would suggest.

The results concerning the computational time are summarized in Table 7.

These results do not claim to provide a complete comparison of smooth- and non-smooth contact mechanics. Its only aim is to show that non-smooth contact mechanics is one promising way to reduce computational time. The parallelization method proposed in this paper can be applied to both, smooth- and non-smooth multibody dynamics and can therefore help to further reduce the computational time within multibody dynamics.

The result of this small comparison is very similar to the one from FOERG ET. AL. [1] (see Section 1). The non-smooth version of the "Spheres in Cup" in combination with a half-explicit time-stepping integration scheme is about three times faster as the

	LSODE (non-stiff)	LSODE (stiff)	Time-Stepping$^{(f)}$
time norm.$^{(h)}$	1,00	n.p.$^{(i)}$	0,36
speedup	1,00	n.p.$^{(i)}$	2,80

$^{(f)}$: a step size of $5e^{-5}$sec was used

$^{(h)}$: normalized to the computational time of the LSODE (non-stiff)

$^{(i)}$: LSODE with "stiff"-option stuck during the integration

Table 7: Comparison of Computational Times of Smooth- and Non-Smooth Contact Mechanics

smooth version in combination with a common "state of the art" LSODE integrator.

This fact can be explained by the numerical stiffness of the equations of motion using smooth contact mechanics. The contacts are modeled with a contact stiffness of $1 \cdot 10^7 \frac{N}{m}$. It is for this reason that the LSODE integrator needs smaller step sizes compared to the time-stepping integration scheme.

7 Summary

Beginning with a brief outline of non-smooth multibody dynamics and set-valued force-laws, different parallelization methods are shortly described. Furthermore, an internal parallelization algorithm for multibody dynamics is proposed which allows an effective parallelization of the system-update (calculation of gap-distances, gap-velocities, Jacobian matrices, state-dependent variables, etc.). It is explained that it is very important that not all calculations can be done in parallel due to the inevitable overhead of parallelization methods. Efficiency is shown on three examples. Two academic systems comprising up to 4752 contacts and 96 degrees of freedom and an industrial system from the field of rotor dynamics comprising four flexible rotors mounted with bearing clearance and friction. The results show that by applying the parallelization method is possible to significantly reduce the computational time for the update processes. Taking one academic example into account the effects of the set-valued force laws on the computational costs are explained.

References

[1] M. Förg et. al., "Contacts within Valve Train Simulations: a Comparison of Models", JSME Technical Journal, 1(1), 2006.

[2] F. Pfeiffer, "Mechanical System Dynamics", in "Lecture Notes in Applied and Computational Mechanics", Springer, Berlin, 2005.

[3] M. Friedrich, H. Ulbrich, "A Parallel Co-Simulation For Multibody Systems", in "Proceedings of the 2nd South-East European Conference on Computational Mechanics", Rhodos, Greece, 2009.

[4] R. Huber, H. Ulbrich, ”Integration of Non-Smooth Systems using Time-Stepping based Extrapolation Methods and DAE Solver Combined with Time-Stepping”, in ”Proceedings of the 2nd South-East European Conference on Computational Mechanics”, Rhodos, Greece, 2009.

[5] R. Featherstone, ”A Divide-and-Conquer Articulated-Body Algorithm for Parallel O(log(n)) Calculation of Rigid-Body Dynamics. Part 1: Basic Algorithm”, in ”The International Conference on Robotic Research”, Vol. 18, 867-875, 1999.

[6] M. Förg, ”Mehrkörpersysteme mit mengenwertigen Kraftgesetzen - Theorie und Numerik”, Dissertation, Technische Universität München, 2008.

[7] J. Clauberg et. al., ”Simulation of a Non-Smooth Continuous System”, in ”Vibration Problems ICOVP - Proceedings in Physics”, 978-94-007-2068-8, Springer, Berlin, 2011.

[8] MBSim - Multi-Body Simulation Software. GNU Lesser General Public License, http://code.google.com/p/mbsim-env/

[9] OpenMP - Open Multiprocessing, http://openmp.org