METHOD FOR DISTRIBUTED FORWARD DYNAMIC SIMULATION OF CONSTRAINED MECHANICAL SYSTEMS

Roland Kasper, Dmitry Vlasenko

Otto-von-Guericke-University Magdeburg
Institute of Mechatronics and Drives (IMAT)
Universitätsplatz 2, D-39106 Magdeburg, Germany
Roland.Kasper@mb.Uni-Magdeburg.DE
Tel: +49 391 67 18 606
Fax: +49 391 67 12 656
Dmitri.Vlasenko@Masch-Bau.Uni-Magdeburg.DE

ABSTRACT

This paper presents a recursive algorithm for calculating the forward dynamics of general rigid-body systems using a subsystem approach that is well suited for parallel processing. It is an exact, non-iterative algorithm and is applicable to mechanisms with any joint type and any topology, including branches and kinematic loops. The calculation of accelerations has *O(log(n))* time complexity on *O(n)* processors, that is comparable with the fastest available parallel algorithms. We developed an object-oriented simulation tool that is based on our method. The test simulation of a car suspension system with the closed-loop structure shows the stability of our algorithm.

KEYWORDS

Modular modelling, Dynamics Simulation, Multibodies

1 INTRODUCTION

The development of a tool for simulation of mechanical systems is a sophisticated problem. Simulating software should satisfy the wide set of conditions: numerical efficiency, stability, distributivity, flexibility, interaction with other tools, distributed development, etc.

Trying to satisfy all demands, the modern simulation tools use the object-oriented method. But though the object-oriented approach has a huge number of advantages, this type of modularization in most cases is given up during simulation, especially for mechanical systems because common modelling formulations use access to the complete system to calculate all accelerations needed. From the other hand, there are big advantages of a simulation on the basis of subsystems:

1. Subsystems can be modelled, tested and compiled. Then they can be used in a way similar to software components that encapsulate their internal structure and can be connected via interfaces.
2. Critical effects like coulomb friction, backslash etc. can be encapsulated inside a subsystem.
3. Subsystems are ideal candidates for the partitioning of large systems on multiple processors.

Our goal is to create an object-oriented method for the distributed simulation of multibodies with variable number of degrees of freedom. Unlike of a huge number of other methods, we keep the

block-module concept during simulation. Based on our method we develop the simulation software that can be implemented for the simulation of mechanical parts of mechatronic systems.

2 DESCRIPTION OF METHOD

In our method we perform the simulation using the hierarchy of submodels. The submodels of the first level consist of connected bodies. The submodels of next levels (called *children*) consist of connected submodels (called *parents*). Since the main number of calculations proceeds inside submodels, it follows that we can easily distribute the simulation on several processors.

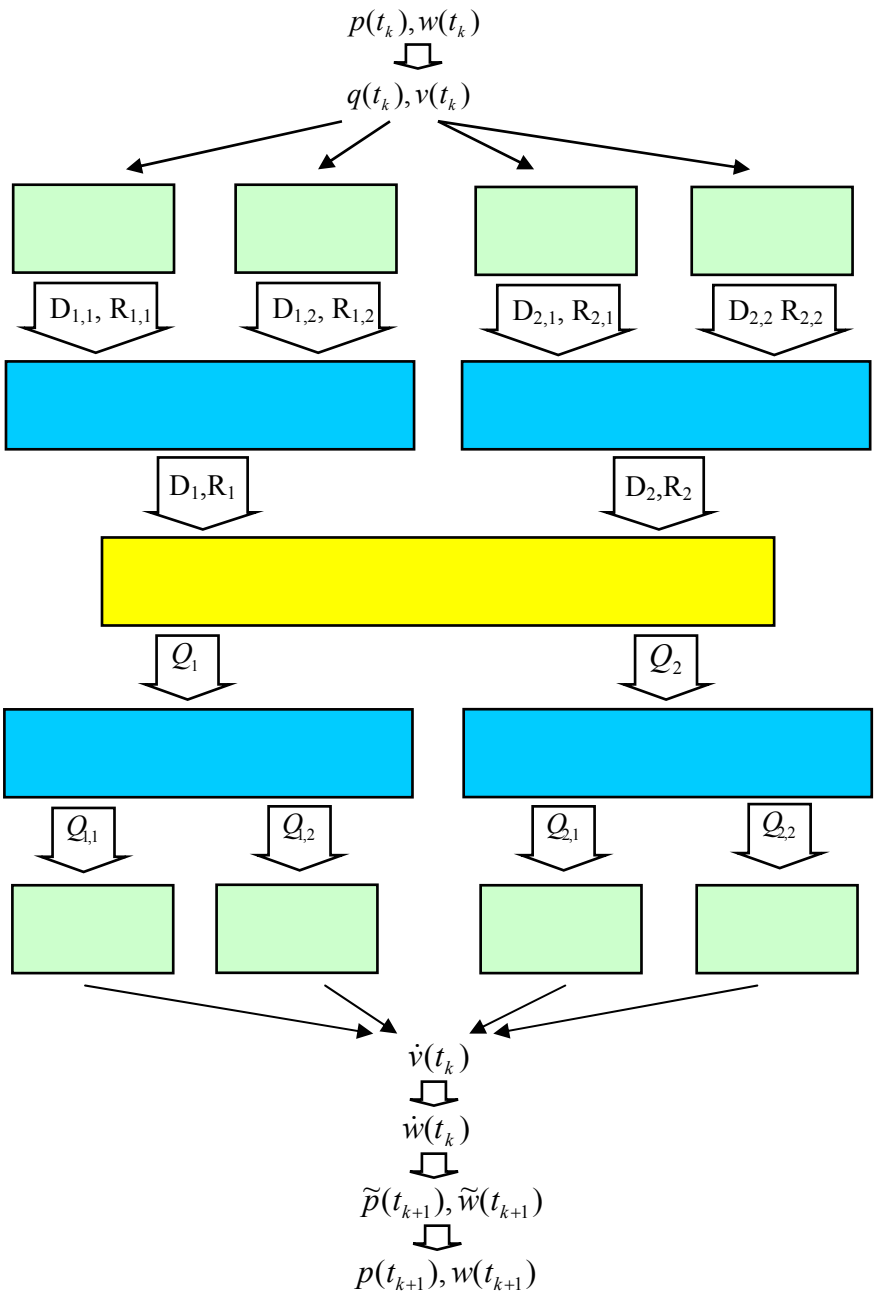

Figure 1: Simulation steps

During the simulation on each time step we perform several operations, shown in Fig. 1:

1. **Calculation of absolute coordinates and velocities**. Using the current value of generalized velocities *w* and coordinates *p* we consequently calculate the absolute coordinates *q* and velocities *v*.

2. **Hierarchical generations of equations of motion.** A subsystem gets from its parents their dependency matrices $D^{(i)}$ and $R^{(i)}$:

$$\dot{v}_e^{(i)} = D^{(i)} Q^{(i)} + R^{(i)}$$

where

$\dot{v}_e^{(i)}$ is the vector of accelerations of *i*-th parent's bordering bodies,

$Q^{(i)}$ is the vector of forces acting in *i*-th parent's external links.

Here we call a body *bordering to a subsystem* if it is connected with the external joints and it is called *internal* if it does not have external constraints.

The subsystem generates matrices *D* and *R* using the equation of constraints connecting the parents. Here *D* and *R* are the dependency matrices:

$$\dot{v}_e = DQ + R$$

where

$\dot{v}_e$ is the vector of accelerations of subsystem's bordering bodies,

Q is the vector of forces in subsystem's external links.

Then the subsystem transmits *D* and *R* to its children.

3. **Backward hierarchical calculation of absolute accelerations**. A subsystem gets the current value of *Q* from its child. Using *Q* we calculate $\dot{v}_e$. Then for each parent *i* the subsystem calculates $Q^{(i)}$ and transforms it to the parent. After we finish, we obtain the absolute acceleration of all bodies.

4. **Calculation of generalized accelerations**. Using the current value of absolute accelerations $\dot{v}$, we consequently calculate the generalized accelerations $\dot{w}$.

5. **Calculation of the generalized coordinates and velocities on the next time step**. Using a favourite ODE integration scheme (e.g. Runge-Kutta or multistep) we obtain the value of $(\tilde{q}_{n+1} \quad \tilde{v}_{n+1})$ on the new time step.

6. **Post-stabilization of generalized coordinates and velocities**. We use the stabilization technique proposed by Ascher, Chin [1]. Their method guarantees the asymptotic stability of solution. Our simulation example shows the stability of the technique.

The calculation of accelerations has $O(n/k)$ time complexity on $k<<n$ processors or $O(\log(n))$ time complexity on $O(n)$ processors, that is comparable with the fastest available parallel algorithms [3, 2].

As you see, we use the combination of generalized and absolute coordinates trying to maximise the effectiveness of the method. Using absolute coordinates we can performe distributive calculation of forces and accelerations. But using generalized coordinates we perform numericaly efficient integration and stabilization.

The global computation complexity of our method is $O(n \cdot D^3 + t^2 \cdot s)$, where *n* is the total number of bodies, *D* is the upper limit of constraints in a subsystem, *t* is the number of closed loops and *s* is the total number of bodies in loops.

3 IMPLEMENTATION OF METHOD

Our tool is based on a strictly capsulated block-module concept proposed by Kasper [4]. This approach has some significant advantages: flexibility, top-down design, distributed and quick development. Using this approach we implemented our software in Visual Basic 6.0.

3.1 Basic objects

In our tools are used eight basic objects:

1. *Timer* object is used for the identification of the current time inside of a simulating model.
2. *Ground* object simulates is the body whose motion is predefined. There is no restriction on the number of ground objects inside of the complete model.
3. *Body* object simulates the body whose motion is not predefined.
4. *Body output* is used for the working with a body outside of the submodel where the body was defined. This is possible because each output object (called *child*) inherits the parameters (i.e. absolute coordinates, velocity etc.) of its *parent* (the body or the other output). If output's parameters change then the output object automatically changes corresponding parameters of its parent.
5. *Generalized force* object describes is an external force or external torque acting on bodies.
6. *Constraint object* describes a holonomical constraint connecting several bodies.
7. *Basic subsystem* object is a subsystem of the lowest level of hierarchy, that can include body objects, ground objects, force objects and output objects.
8. *Derived subsystem* object is a subsystem of a high level of hierarchy that can include other subsystems, ground objects, force objects and output objects.

3.2 Derived objects

The basic objects are parents of derived objects: different types of joints, forces, bodies and subsystems. Using *Constraint* class we developed the four most frequent types of joints: *Revolute joint Prismatic joint, Ball joint* and *Rigid connection*.

Using *Generalized force* class we develop three types of generalised forces: *Gravity force* and *Spring Damper*.

4 EXAMPLE OF SIMULATION

We have performed a number of calculations for the problem of the car suspension shown in Fig. 2. This example perfectly illustrates all advantages of our method: the object-oriented simulation of multibodies, the stabilization of a closed-loop system, the numerical efficiency of the combination of absolute and generalized coordinates.

From the physical point of view the car system consists of the car body (marked in Fig. 2 by grey) connected with two suspensions (marked by purple and by yellow) by revolute joints with *y*-axes of rotation, two wheels (marked by red) connected with suspensions by revolute joints with *x*-axes of rotation. Each wheel is connected with a ground by a string. A suspension consists of the damper and the beam, where damper's piston is connected with the beam by the revolute joint (the axe of the joint is perpendicular to the frontal plane).

While the construction of the car model we partition the complete system on several subsystems:

1. **WheelSubsystem** is a basic subsystem consists of Body (the steel ring), Ground, String and WheelOutput.
2. **DamperSubsystem** is a basic subsystem that consists of two Body objects (Cylinder and Piston), two Output objects (CylinderOutput and PistonOutput), PrismaticJoint and SpringDamper.
3. **SuspensionSubsystem** is a derived subsystem consisting of Damper, Beam, RevoluteJoint, and two Outputs (CylinderOutput and BeamOutput).
4. **CompleteSystem** is a derived subsystem that consists of Beam subsystem (car body), two WheelSubsystem objects, RightSuspension, LeftSuspension, six RevoluteJoint objects and Gravity.

We keep this partitioning during the forward dynamic simulation of the car system.

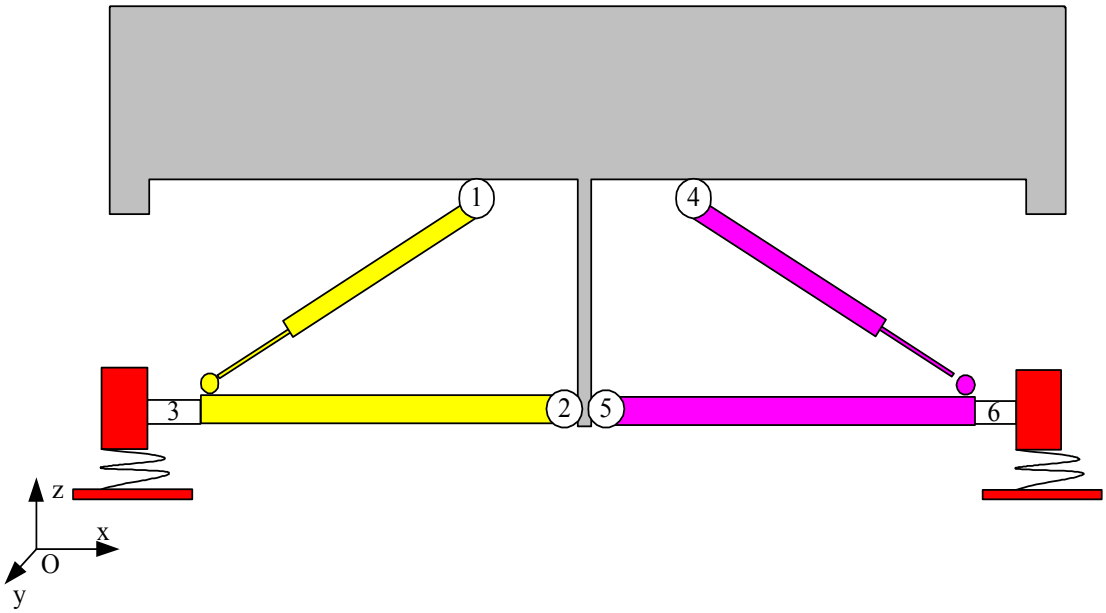

Figure 2: Car system

We choose the time interval to be [0,1.3]. Simulation was performed with Runge-Kutta method of the second order with the fixed timestep equal to 0.01*s*.

In Fig. 3 is shown the oscillation of the car trunk.

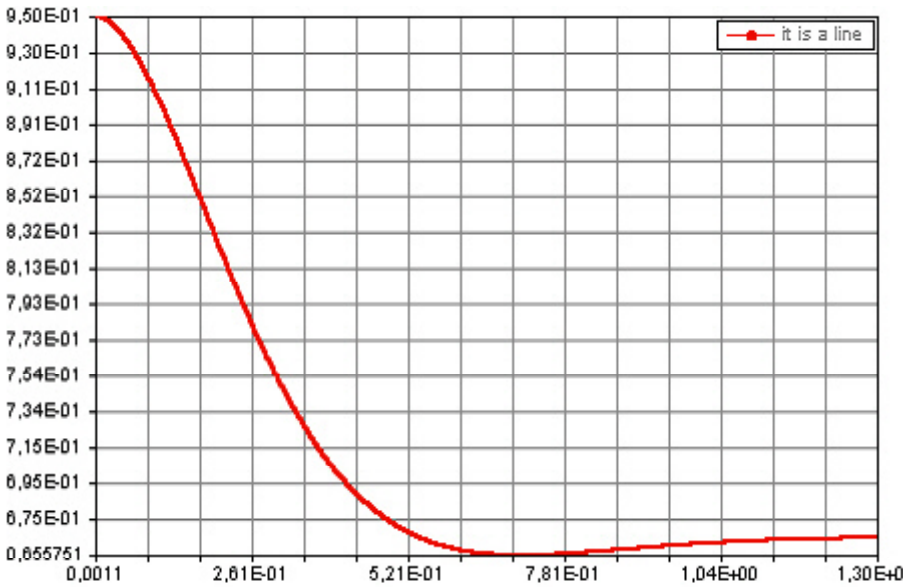

Figure 3. *Z*-coordinate of CarBody

In Fig. 4 shows the drift of the model. The simulation data shows that the algorithm is stable and the model's drift is constant and has the order of the computation accuracy.

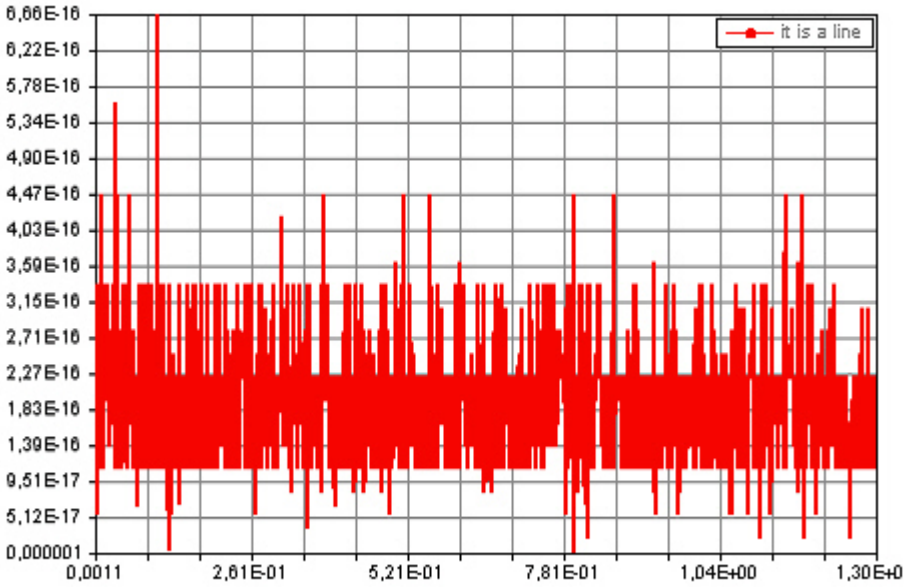

Figure 4: Drift of the model

CONCLUSION

We developed the recursive object-oriented algorithm for calculating the forward dynamics of general rigid-body system using a subsystem approach that is well suited for parallel processing. It is an exact, non-iterative algorithm, and is applicable to mechanisms with any joint type and any topology, including branches and kinematic loops. The calculation of accelerations has *O(log(n))* time complexity on *O(n)* processors, that is comparable with the fastest available parallel algorithms.

We performed the implementation of our method and developed the object-oriented tool for simulation of multibody systems. We created objects that simulate some frequent types of joints and generalized forces:

The experimental data shows the stability of our method. The drift of closed-loop structures is limited for a long period of time. Thus, we obtain the experimental proof that our tool can be implemented for the simulation of large constrained multibody systems.

REFERENCES

[1] U. Ascher, H. Chin, L. Petzold and S. Reich. *Stabilization of constrained mechanical systems with DAEs and invariant manifolds*, the Journal of Mechanics of Structures and Machines, 23(2), 135-157(1995).

[2] Anderson, K. and Duan, S., 2000, *Highly parallelizable low-order dynamics simulation algorithm for multi-rigid-body systems*, AIAA Journal on Guidance, Control and Dynamics 23, no. 2, pp. 355-364.

[3] Featherstone, R., 1999, *A divide-and-conquer articulated-body algorithm for parallel O(log(n)) calculation of rigid-body dynamics, part 2: trees, loops and accuracy*, Int. Journal of Robotics Research 18, no. 9, pp. 876-892.

[4] Kasper, R.; Koch, W.; Kayser, A.; Wolf, A. *Integrated design environment for mechatronic components and systems of automotive industry*, VDI Bericht 1374, pp. 451-465, 1997