A distributed memory parallel multibody Contact Dynamics code

Tomasz Koziara, Nenad Bićanić

Department of Civil Engineering
University of Glasgow, Glasgow G12 8LT, UK

Abstract

Solfec is a computational code aimed at simulation of constrained multibody systems on distributed memory parallel computers. It implements an instance of the Contact Dynamics (CD) method by Moreau and Jean, hence the constraints are handled implicitly. One of the main goals of the software is to provide a user friendly platform for testing formulations and solution methods for the (dynamic) frictional contact problem. It is also meant to serve as a development platform for other aspects of time-stepping methods (e.g. contact detection, time integration, kinematic models). The purpose of this communication is to outline the algorithmic ideas behind Solfec, with an emphasis on the Gauss-Seidel constraints solver, a classical element of CD, efficiently implemented on a distributed memory model. The remaining topics include a dynamic domain decomposition balancing the computational load of the overall algorithm, and an accuracy measure for the frictional contact constraints. Solfec is an open-source software and can be downloaded from http://code.google.com/p/solfec.

1. Introduction

The current paradigm in modelling large sets of interacting bodies dates back to Cundall and Strack [14] and the Discrete Element Method (DEM). In its simplest form, this is essentially the approach comprising rigid bodies and repulsive springs: when bodies overlap, repulsive springs prevent excessive interpenetration. This simple strategy proved sufficient in many practical applications (e.g. granular flow). It has been implemented in numerous commercial codes and it is now one of the standard tools in engineering applications.

In the meantime, several approaches have been developed where contact constraints were formulated through implicit (non-regularized) relations between the contact reactions and the relative contact velocities or displacements. A good review can be found in Brogliato et al. [8]. Accounting for non-smoothness

Email address: t.koziara@civil.gla.ac.uk, n.bicanic@civil.gla.ac.uk (Tomasz Koziara, Nenad Bićanić)

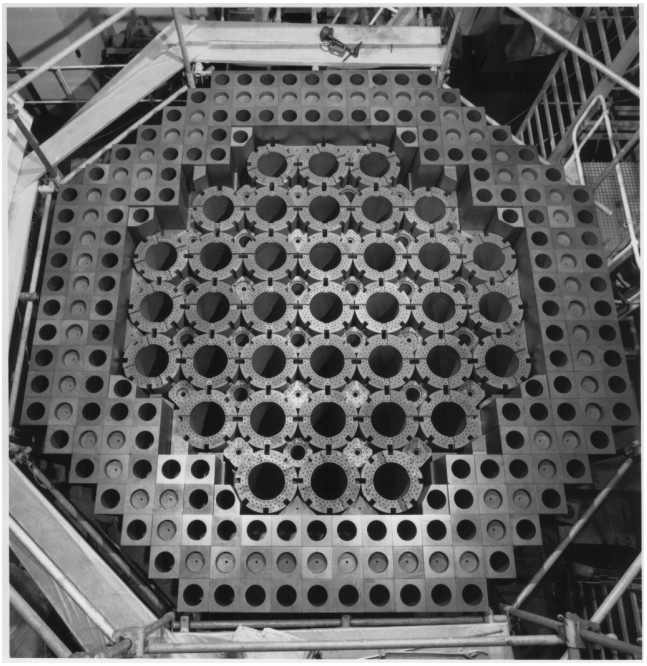

Figure 1: Graphite core of a test advanced gas-cooled reactor (AGR). The small test reactor core above has 37 fuel channels, while the actual AGR core has about 300 fuel channels.

(velocity jumps, stick-slip transitions), the implicit formulations differ from DEM both mathematically and algorithmically. At the same time, as a result of increased complexity, for large dynamic problems they usually fail to deliver computational efficiency comparable with DEM. Combined with the difficulty of implementation, this seems to be among the reasons why such formulations have not been embraced by the industry at large.

Our point of departure is the Contact Dynamics method (CD) by Moreau [31] and Jean [23]. The main features of the method are the use of a velocity level time-stepping, a non-regularised treatment of friction and contact, and a block Gauss-Seidel relaxation employed to resolve constraints. Applications range from granular flow [36], through statics of masonry [12], to deep drawing simulations [24]. In the current context, the interest is in modelling dynamics of large, densely packed assemblies of stiff, non-convex bodies. This is motivated by supporting a seismic integrity assessment of an Advanced Gas-cooled Reactor (AGR) nuclear plant core, composed of thousands of loosely keyed together hollow graphite bricks (Figure 1). Our goal is to deliver a user friendly, efficient and scalable time-stepping implementation, bringing the non-regularized approaches somewhat closer to the engineering mainstream.

Contact Dynamics has been parallelized only partially so far. The research has mostly been focused on the contact solution: the most time consuming element of CD. A shared-memory parallel Gauss-Seidel solver was developed in

[5, 36] and a good scaling was achieved on up to 16 processors. In the distributed memory context, an initial application of a static geometrical domain decomposition was made by Breitkopf and Jean [7]. Some results on the domain decomposition for non-smooth mechanical systems were given in [33, 22], although without an application in the context of frictional contact. Within this context, a static domain decomposition with per-domain nonlinear Gauss-Seidel solutions coupled through linear inter-domain compatibility conditions was presented by Iceta et al. [21]. Although a parallel implementation has not been demonstrated, the linearity of the compatibility conditions potentially facilitates a distributed memory implementation.

In the current paper we outline the algorithmic ideas behind Solfec: an open-source code implementing Contact Dynamics for distributed memory parallel computers. The code originates from our previous work [25]. The work presented here has two basic aspects. Firstly, we develop distributed data structures and algorithms necessary to balance the computational load of CD. Secondly, although we are aware of the limitations of a distributed Gauss-Seidel algorithm (requiring some sequential processing), we adapt and test the limits of a best known parallel implementation (borrowed [3] from the field of multigrid smoothing). It has to be noted, that although this baseline implementation of the classical CD framework "had to be done" for completeness, the purpose of Solfec is to seek new and more efficient ways of doing Contact Dynamics. Some of the ongoing work is commented on in Section 9.

Other codes as well implement the non-regularized formulation. The reference implementation of Contact Dynamics is LMGC90 [17]. Similarly as LMGC90, DynamY [40, 41] and MBSim [18, 42, 37] can both handle mechanical systems with unilateral contacts and friction. The Siconos project [1, 2] on the other hand provides an abstract interface for modeling, simulation and control of non-smooth dynamical systems (e.g. mechanical or electrical). To our knowledge, Solfec is currently the only distributed memory implementation of Contact Dynamics.

The basic time-stepping is outlined in Section 2. In Section 3 a brief account of two major data structures is given. The parallel version of the time-stepping is summarised in Section 4. Section 5 details the subsequently used interface law. Section 6 describes a constraints satisfaction accuracy measure. Section 7 focuses on a distributed memory implementation of the Gauss-Seidel solver. Finally, after presenting several examples in Section 8, we conclude and discuss future work in Section 9.

2. Basic time-stepping

It will be useful to introduce some basic notions. Let us have a look at Figure 2. There are four bodies in the figure. Placement of each point of every body is determined by a configuration $\mathbf{q}_i$. Velocity of each point of every body is determined by a velocity $\mathbf{u}_i$. Let $\mathbf{q}$ and $\mathbf{u}$ collect configurations and velocities

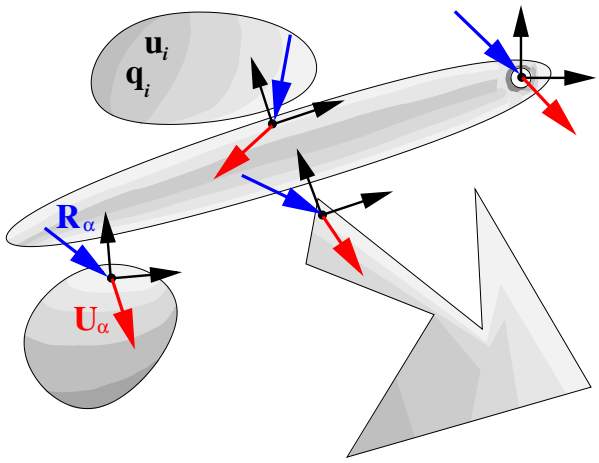

Figure 2: A multibody system.

of all bodies. If the time history of velocity is known, the configuration time history can be computed as

$$\mathbf{q}(t) = \mathbf{q}(0) + \int_0^t \mathbf{u} dt \tag{1}$$

The velocity is determined by integrating Newton's law

$$\mathbf{u}(t) = \mathbf{u}(0) + \mathbf{M}^{-1} \int_0^t \left(\mathbf{f} + \mathbf{H}^T \mathbf{R}\right) dt \tag{2}$$

where $\mathbf{M}$ is an inertia operator (assumed constant here), $\mathbf{f}$ is an out of balance force, $\mathbf{H}$ is a linear operator, and $\mathbf{R}$ collects some point forces $\mathbf{R}_\alpha$. While integrating the motion of bodies, we keep track of a number of local coordinate systems (local frames). There are four of them in Figure 2. Each local frame is related to a pair of points, usually belonging to two distinct bodies. An observer embedded at a local frame calculates the local relative velocity $\mathbf{U}_\alpha$ of one of the points, viewed from the perspective of the other point. Let $\mathbf{U}$ collect all local velocities. Then, we can find a linear transformation $\mathbf{H}$, such that

$$\mathbf{U} = \mathbf{H}\mathbf{u} \tag{3}$$

In our case local frames correspond to *constraints*. We influence the local relative velocities by applying local forces $\mathbf{R}_\alpha$. This can be collectively described by an implicit relation

$$\mathbf{C}(\mathbf{U}, \mathbf{R}) = \mathbf{0} \tag{4}$$

Hence, in order to integrate equations (1) and (2), at every instant of time we need to solve the implicit relation (4). Here is an example of a numerical approximation of such procedure

$$\mathbf{q}^{t+\frac{h}{2}} = \mathbf{q}^t + \frac{h}{2}\mathbf{u}^t \tag{5}$$

$$\mathbf{u}^{t+h} = \mathbf{u}^t + \mathbf{M}^{-1}h\mathbf{f}^{t+\frac{h}{2}} + \mathbf{M}^{-1}\mathbf{H}^T\mathbf{R} \tag{6}$$

$$\mathbf{q}^{t+h} = \mathbf{q}^{t+\frac{h}{2}} + \frac{h}{2}\mathbf{u}^{t+h} \tag{7}$$

where h is a discrete time step. As the time step h does not appear by $\mathbf{M}^{-1}\mathbf{H}^T\mathbf{R}$, $\mathbf{R}$ should now be interpreted as an impulse[1]. At a start we have

$$\mathbf{q}^0 \text{ and } \mathbf{u}^0 \text{ as prescribed initial conditions.} \tag{8}$$

The out of balance force

$$\mathbf{f}^{t+\frac{h}{2}} = \mathbf{f}\left(\mathbf{q}^{t+\frac{h}{2}}, t+\frac{h}{2}\right) \tag{9}$$

incorporates both internal and external forces. The symmetric and positive-definite inertia operator

$$\mathbf{M} = \mathbf{M}\left(\mathbf{q}^0\right) \tag{10}$$

is computed once. The linear operator

$$\mathbf{H} = \mathbf{H}\left(\mathbf{q}^{t+\frac{h}{2}}\right) \tag{11}$$

is computed at every time step. The number of rows of $\mathbf{H}$ depends on the number of constraints, while its rank is related to their linear independence. We then compute

$$\mathbf{B} = \mathbf{H}\left(\mathbf{u}^t + \mathbf{M}^{-1}h\mathbf{f}^{t+\frac{h}{2}}\right) \tag{12}$$

and

$$\mathbf{W} = \mathbf{H}\mathbf{M}^{-1}\mathbf{H}^T \tag{13}$$

which is symmetric and semi-positive definite. The linear transformation

$$\mathbf{U} = \mathbf{B} + \mathbf{W}\mathbf{R} \tag{14}$$

maps constraint reactions $\mathbf{R}$ into local relative velocities $\mathbf{U} = \mathbf{H}\mathbf{u}^{t+h}$ at time $t+h$. Relation (14) will be here referred to as the *local dynamics*. Finally

$$\mathbf{R} \text{ is such that } \mathbf{C}\left(\mathbf{U},\mathbf{R}\right) = \mathbf{C}\left(\mathbf{B}+\mathbf{W}\mathbf{R},\mathbf{R}\right) = \mathbf{C}\left(\mathbf{R}\right) = \mathbf{0} \tag{15}$$

where $\mathbf{C}$ is a nonlinear and usually nonsmooth operator. A basic Contact Dynamics algorithm can be summarised as follows:

[1] For simplicity, we shall often write "reaction" rather than "reaction impulse".

1. Perform first half-step $\mathbf{q}^{t+\frac{h}{2}} = \mathbf{q}^t + \frac{h}{2}\mathbf{u}^t$.
2. Update existing constraints and detect new contact points.
3. Compute $\mathbf{W}$, $\mathbf{B}$.
4. Solve $\mathbf{C}(\mathbf{R}) = \mathbf{0}$.
5. Update velocity $\mathbf{u}^{t+h} = \mathbf{u}^t + \mathbf{M}^{-1}h\mathbf{f}^{t+\frac{h}{2}} + \mathbf{M}^{-1}\mathbf{H}^T\mathbf{R}$.
6. Perform second half-step $\mathbf{q}^{t+h} = \mathbf{q}^{t+\frac{h}{2}} + \frac{h}{2}\mathbf{u}^{t+h}$.

It should be stressed that the above presentation exemplifies only a particular instance of Contact Dynamics. Let us refer the reader to [30, 32, 9, 31, 23, 19, 1, 29, 41] for more details.

3. Data structures

BODY	
$\mathbf{q}, \mathbf{u}, \mathbf{M}^{-1}$	configuration, velocity, inverse inertia
constraints	set of adjacent constraints
kind	parent, child or dummy

Table 1: Body structure.

A multibody domain is partitioned among processors in order to balance the computational load. The two basic computational objects are bodies and constraints. The body structure is summarised in Table 1. A body stores its configuration $\mathbf{q}$, velocity $\mathbf{u}$ and the inverse inertia operator $\mathbf{M}^{-1}$ (a generalised counterpart in case of implicit time integration). It also stores references to all adjacent *constraints*. The *kind* property is specific to the parallel implementation. There are three kinds of bodies:

1. Parent bodies take part in time integration, contact detection and $\mathbf{W}$ assembling. They can migrate during load balancing and belong to processors whose domains contain their characteristic points (e.g. spatial mass centers). They are the bodies comprising the multibody domain (i.e. they have been defined by the user).
2. Child bodies are copies of parent bodies on the non-parent processors whose domains are overlapped by the parents. They take part in contact detection and $\mathbf{W}$ assembling. They need to be updated by the parents.
3. Dummy bodies are only used during $\mathbf{W}$ assembling. They are attached to constraints, whose one body is either a parent or a child and another body (which is neither a child nor a parent) must be brought in (and becomes a dummy).

Table 3 enumerates parent, child and dummy bodies in Figure 3. It has to be noted, that maintaining this kind of structure requires communication. That is to say, during load balancing copies of bodies need to be sent across a computer network. As the communication bandwidth is the most expensive resource during distributed computing, our current design choice in Solfec is to maintain

the complete sets of bodies on all processors, while sending only the necessary minimum of data. Hence, although each processors stores all bodies involved in the analysis, at a particular time only a fraction of them is actively used as parents, children and dummies. This, of course, is an optimization suitable in our context, where tens of thousands of relatively complex bodies need to be analysed. In case of simulations involving millions of bodies, it might be more efficient to avoid such redundancy.

CONSTRAINT α	
$\mathbf{R}_\alpha, \mathbf{U}_\alpha$	constraint reaction and relative velocity
$\mathbf{B}_\alpha, \mathbf{W}_{\alpha\beta}$	corresponding row of local dynamics
master	master body involved in the constraint
slave	slave body involved in the constraint
kind	internal or external

Table 2: Constraint structure.

The constraint structure is summarised in Table 2. It contains the constraint reaction $\mathbf{R}_\alpha$, the relative velocity $\mathbf{U}_\alpha$ as well as the corresponding row $\mathbf{B}_\alpha, \mathbf{W}_{\alpha\beta}$ of local dynamics (14). Similarly to $\mathbf{R}_\alpha$ and $\mathbf{U}_\alpha$, $\mathbf{B}_\alpha$ is a 3-vector while $\mathbf{W}_{\alpha\beta}$ are the non-zero 3×3 blocks of the corresponding block-row of $\mathbf{W}$ (the parallel domain partitioning implies then a row-partitioning of local dynamics). The *master* and *slave* pointers reference the attached bodies. The *kind* property is specific to the parallel implementation. There are two kinds of constraints:

1. Internal constrains, which take part in load balancing and belong to specific processor domains. They are established between pairs of parent and child bodies (parent-parent, parent-child, child-child), both present on a given processor.
2. External constraints, needed to compute a row of local dynamics corresponding to an internal constraint. They are established between dummy bodies and parent or child bodies.

Table 3 enumerates internal and external constraints in Figure 3. Unlike in the case of bodies, communication cost related to constraints cannot be optimized by a redundant data storage. Contact points are frequently added or removed which results in the need for constant updates of the constraints data. Nevertheless, although this is not pictured in Figure 3, the necessary data flow happens only in the vicinity of sub-domain boundaries, while large inner parts of the processors domains require no communication.

We use the Recursive Coordinate Bisection (RCB) [6] implemented in Zoltan [16] in order to maintain the computational balance. RCB takes as an input a set of weighted points and outputs their assignments to processors. The space is partitioned along axis aligned planes, so that the resultant point clusters have well balanced total weights. Rebalancing after a change of the input is fast and incremental - consecutive partitionings are similar to each other. This helps to minimise communication. It should be noted, that in our context both geometrical (e.g. body shapes) and topological (e.g. $\mathbf{W}$ matrix) data needs to be

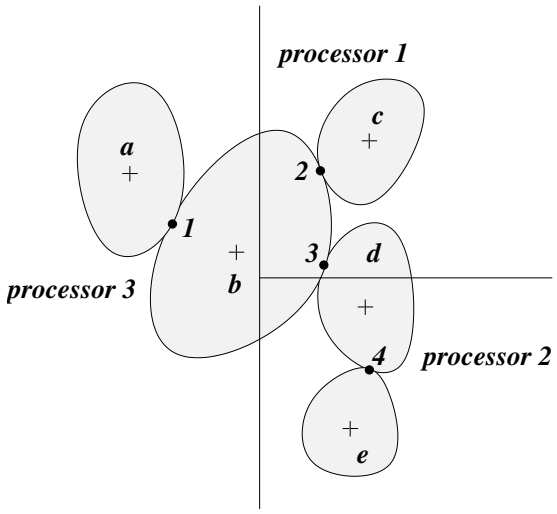

Figure 3: A partitioned multibody domain. Bodies are marked by letters from *a* to *e*, while constraints are numbered from *1* to *4*.

	processor 1	processor 2	processor 3
parent bodies	*c*	*d*, *e*	*a*, *b*
child bodies	*b*, *d*	*b*	-
dummy bodies	*a*, *e*	-	*c*, *d*
internal constraints	*2*, *3*	*4*	*1*
external constraints	*1*, *4*	*3*	*2*, *3*

Table 3: Kinds of bodies and constraints for each processor from Figure 3.

balanced. Although we did test topological (graph based) strategies, the need for frequent rebalancing with considerably varying input graphs rendered them inefficient. Motivated by PRONTO3D [10], where separate partitionings are maintained for various computational tasks, after many tests including maintaining separate partitionings for time integration, contact detection and **W** assembling we concluded, that the most efficient approach (at least in our application context) is based on just one, well adjusted geometrical partitioning. This is because we deal with dense assemblies of interacting, simply deformable bodies for which the cost of time integration is small, while the spatial locality of tasks related to contact detection and resolution is similar.

Figure 3 illustrates a partitioned multibody domain. There are five bodies in the domain, marked from *a* to *e*. There are also four constraints numbered from *1* to *4*. Bodies and constraints are associated with nine points: plus signs represent bodies while dots represent constraints. These are the input points used by the RCB algorithm. The actual locations of points representing bodies can be chosen in a variety of ways. In our case these are the mid-points of their axis aligned bounding boxes. The locations of points representing constraints

are simpler to chose (e.g. contact points). Clearly, as far as load balancing is concerned, we do not discriminate between bodies and constraints. We use a common point set and, in order to obtain a good balance, suitably adjust the weights. A quite simple approach works well in practice. Let us have a quick look at the constraint structure detailed in Table 2. A constraint *con* is adjacent to at most two bodies: *con.master* and *con.slave*. In case of a single-body constraint (e.g. prescribed displacement) only *con.master* is valid while $con.slave = nil$. In any case, the constraint weight is obtained as follows

$$d_m = \dim\left(con.master.\mathbf{u}\right)$$

$$d_s = \begin{cases} \dim\left(con.slave.\mathbf{u}\right) \text{ if } con.slave \neq nil \\ 0 \text{ otherwise} \end{cases}$$

$$s_m = \left|con.master.constraints\right|$$

$$s_s = \begin{cases} \left|con.slave.constraints\right| \text{ if } con.slave \neq nil \\ 0 \text{ otherwise} \end{cases}$$

$$weight\left(con\right) = d_m + d_s + factor \cdot \left[d_m\left(s_m - 1\right) + d_s\left(s_s - 1\right)\right] \qquad (16)$$

where d_m and d_s denote dimensions of master and slave body velocities, s_m and s_s denote sizes of master and slave body constraint sets, and $factor \in [0, 1]$. Above, we use the dimension of a body velocity as a basic indicator of the computational complexity. The meaning of expression (16) is easier understood when we first look at the case $factor = 1$. For a given constraint *con*, the corresponding row of local dynamics will have at most as many non-zero blocks as there are constraints adjacent to *con.master* and *con.slave*. Hence $d_m s_m + d_s s_s$ approximates computational cost of dealing with a single row of contact dynamics. This choice produces well balanced local dynamics, but can in some cases lead to poor balancing of time integration or contact detection. By selecting $factor < 1$ one can improve the overall balance. Looking now at the body structure in Table 1, we can see that it contains the set of *constraints* adjacent to the body. We can then define the weight of a body as

$$weight\left(bod\right) = \dim\left(bod.\mathbf{u}\right) + \sum_{\substack{con \in bod.constraints \\ \text{and } con.slave = nil}} weight\left(con\right) \qquad (17)$$

The weight of a body equals to its configuration (tangent space) dimension plus the sum of weights of the attached single-body constraints. The last contribution corresponds to the fact, that in our implementation the single-body constraints migrate together with bodies. As a result only the bodies and the two-body

Algorithm 1 Parallel time-stepping.

1 for all parent bodies update $\mathbf{q}^{t+\frac{h}{2}} = \mathbf{q}^t + \frac{h}{2}\mathbf{u}^t$
2 send $\mathbf{q}^{t+\frac{h}{2}}$ from parents to children
3 update internal constraints and remove ceased constraints
4 update RCB partitioning
5 insert new children into sub-domains overlapped by their parents
6 remove children from sub-domains no longer overlapped by their parents
7 migrate parent bodies and their attached internal single-body constraints
8 migrate internal two-body constraints
9 migrate internal constraints to external processors where their attached parent or child bodies reside; these are external constraints now
10 detect new constraints
11 migrate new constraints as in step 9
12 post-process constraints and remove redundancies
13 assemble local dynamics $\mathbf{B}$ and $\mathbf{W}$
14 find $\mathbf{R}$ such that $\mathbf{C}(\mathbf{R}) = \mathbf{0}$
15 for all parent bodies update $\mathbf{u}^{t+h} = \mathbf{u}^t + \mathbf{M}^{-1}h\mathbf{f}^{t+\frac{h}{2}} + \mathbf{M}^{-1}\mathbf{H}^T\mathbf{R}$
16 for all parent bodies update $\mathbf{q}^{t+h} = \mathbf{q}^{t+\frac{h}{2}} + \frac{h}{2}\mathbf{u}^{t+h}$

constraints (e.g. contacts) migrate independently. The constraint weights are usually larger than those of bodies. This corresponds to the fact, that time integration and contact detection are much less time consuming, when compared to an implicit solution of constraint equations. Processing constraints drives the computational load. When it comes to balancing the load of contact detection, the above approach works well as long as the distribution of body shapes is sufficiently uniform in terms of their spatial complexity. This is the case in our context, as well as in many other scenarios comprising large numbers of interacting bodies.

4. Parallel time-stepping

The major steps of the parallel time-stepping are summarised in Algorithm 1. The time integration is performed only for the parent bodies. In the first line, the mid-configuration of the parent bodies is computed. Then, in the second line, it is sent to child bodies. This way updated shapes of parent and child bodies can be used when updating constraints in line 3. The domain partitioning is updated next. As a result, the sub-domain boundaries are adjusted to the new positions of bodies. This allows to redistribute child bodies in lines 5 and 6. At the same time, after the RCB update, new processors are assigned to parent bodies and internal two-body constraints. This information is used when migrating bodies and constraints in lines 7 and 8. In the next step (line 9) the set of external constraints is created. At this stage parent and child bodies store subsets of adjacent internal and external constraints. When detecting new

constraints (e.g. contact points) in line 10, this information is used to discriminate newly detected, but redundant constraints (i.e. already existing). The set of new constraints is then migrated out in the form of external constraints (line 11). In line 12, a last post-processing step is executed on the complete constraint sets. This allows to eliminate duplicated contact points, previously skipped owing to the lack of information from external processors. In line 13 the local dynamics operator $\mathbf{W}$ and vector $\mathbf{B}$ are assembled. Each processor assembles a subset of rows of local dynamics corresponding to its own set of internal constraints. Since constraints are balanced, the amount of work needed to compute $\mathbf{W}$ and $\mathbf{B}$ is also balanced among the processors. The implicit constraint equations $\mathbf{C}(\mathbf{R}) = \mathbf{0}$ are solved next (line 14). This aspect of the time-stepping is discussed in Section 7. Once the reactions have been computed, the velocities and configurations of parent bodies are updated in lines 15 and 16.

5. Interface law

In the examples of Section 8 we utilize a minimalistic interface law, reflecting the core difficulties related to the lack of smoothness and the non-associated character of friction. In terms of normal behaviour we employ the velocity Signorini condition, $U_{\alpha N} \geq 0,\ R_{\alpha N} \geq 0,\ U_{\alpha N} R_{\alpha N} = 0$, which implies ideally plastic impacts. In terms of shearing behavior we utilize the Coulomb law, $\mathbf{R}_\alpha \in \text{interior}\,(FC_\alpha) \Rightarrow \mathbf{U}_{\alpha T} = 0$ and $\mathbf{R}_\alpha \in \text{boundary}\,(FC_\alpha) \Rightarrow \mathbf{U}_{\alpha T} = -\gamma \mathbf{R}_{\alpha T}$, where $\gamma > 0$. The friction cone FC_α is defined as

$$FC_\alpha = \{\mathbf{R}_\alpha : \|\mathbf{R}_{\alpha T}\| \leq \mu_\alpha R_{\alpha N}, R_{\alpha N} \geq 0\} \tag{18}$$

where μ_α is the coefficient of friction. It has been shown by De Saxcé and Feng [15], that the Signorini-Coulomb law can be expressed in a compact form

$$-\begin{bmatrix} \mathbf{U}_{\alpha T} \\ U_{\alpha N} + \mu_\alpha \|\mathbf{U}_{\alpha T}\| \end{bmatrix} \in N_{FC_\alpha}(\mathbf{R}_\alpha) \tag{19}$$

where N_{FC_α} stands for the normal cone of the set FC_α. For a convex set A the normal cone $N_A(\mathbf{R})$ at point $\mathbf{R} \in A$ is defined as the set of all vectors $\mathbf{V}$ such that $\langle \mathbf{V}, \mathbf{S} - \mathbf{R} \rangle \leq 0$ for all $\mathbf{S} \in A$. Let

$$\mathbf{F}(\mathbf{R}) = \begin{bmatrix} \dots \\ \mathbf{U}_{\alpha T}(\mathbf{R}) \\ U_{\alpha N}(\mathbf{R}) + \mu_\alpha \|\mathbf{U}_{\alpha T}(\mathbf{R})\| \\ \dots \end{bmatrix} \tag{20}$$

and

$$FC = \bigcup_\alpha FC_\alpha \tag{21}$$

where the dependence $\mathbf{U}_\alpha(\mathbf{R})$ is defined in (14). Formula (19) states, that the frictional contact constraints are satisfied if $-\mathbf{F}(\mathbf{R})$ belongs to the normal cone of the friction cone at $\mathbf{R}$. Hence

$$-\mathbf{F}(\mathbf{R}) = \mathbf{R} - \mathbf{F}(\mathbf{R}) - \mathrm{proj}_{FC}(\mathbf{R} - \mathbf{F}(\mathbf{R})) \tag{22}$$

which can be reduced to the usual projection formula $\mathbf{R} = \mathrm{proj}_{FC}(\mathbf{R} - \mathbf{F}(\mathbf{R}))$. Let us not do it though, but rather define a vector field

$$\mathbf{m}(\mathbf{S}) = \mathbf{S} - \mathrm{proj}_{FC}(\mathbf{S}) = \mathbf{n}(\mathbf{S}) \langle \mathbf{n}(\mathbf{S}), \mathbf{S} \rangle \tag{23}$$

where

$$\mathbf{n}_\alpha(\mathbf{S}_\alpha) = \begin{cases} \mathbf{0} & \text{if} \quad \|\mathbf{S}_{\alpha T}\| - \mu_\alpha S_{\alpha N} \leq 0 \\ \mathbf{S}_\alpha / \|\mathbf{S}_\alpha\| & \text{if} \quad \mu_\alpha \|\mathbf{S}_{\alpha T}\| + S_{\alpha N} < 0 \\ \frac{1}{\sqrt{1+\mu_\alpha^2}} \begin{bmatrix} \mathbf{S}_{\alpha T} / \|\mathbf{S}_{\alpha T}\| \\ -\mu_\alpha \end{bmatrix} & \text{otherwise} \end{cases} \tag{24}$$

We can rewrite (19) as

$$\mathbf{C}(\mathbf{R}) = \mathbf{F}(\mathbf{R}) + \mathbf{m}(\mathbf{R} - \mathbf{F}(\mathbf{R})) = \mathbf{0} \text{ and } \mathbf{R} \in FC \tag{25}$$

In a forthcoming paper [28] the above formulation serves as a basis for a projected quasi-Newton method. In Sections 6 and 7 we utilize it in order to measure the accuracy of the satisfaction of constraints.

6. Constraints accuracy

As already explained, at every time step an implicit equation $\mathbf{C}(\mathbf{R}) = \mathbf{0}$ is solved. For frictional contact constraints this can be (25) or any other formulation like in [4, 15, 20]. Ideally, when for some $\mathbf{R}$ there holds $\mathbf{C}(\mathbf{R}) = \mathbf{0}$ we have an exact solution. Of course, in numerical terms this is not possible. For very large problems, and especially for problems where the amount of constraints exceeds the amount of kinematic freedom, obtaining very accurate solutions is hard and often impractical. In any case, it is useful to have an accuracy measure that has some physical interpretation. Because in (25) $\mathbf{C}(\mathbf{R})$ is expressed in terms of velocity, the following quantity

$$g(\mathbf{R}) = \sum_\alpha \left\langle \mathbf{W}_{\alpha\alpha}^{-1} \mathbf{C}_\alpha(\mathbf{R}), \mathbf{C}_\alpha(\mathbf{R}) \right\rangle / \sum_\alpha \left\langle \mathbf{W}_{\alpha\alpha}^{-1} \mathbf{B}_\alpha, \mathbf{B}_\alpha \right\rangle \tag{26}$$

approximately measures the relative amount of spurious energy, due to an inexact satisfaction of constraints. The denominator corresponds to the kinetic energy of the relative free motion, hence $g(\mathbf{R})$ is the ratio of the spurious energy over the nominal amount of the energy available at the constraints. Since inverting $\mathbf{W}$ would be impractical or impossible due to singularity, we only use the diagonal blocks, which are always positive definite. Of course, $g(\mathbf{R}) \ll 1$ is desirable. The above formula can be used to compare the accuracy of solutions obtained by means of different numerical approaches.

The merit function $g(\mathbf{R})$ will be used in the next section as a termination criterion for the Gauss-Seidel scheme. This is only one among many possible

Algorithm 2 Serial Gauss-Seidel algorithm

SERIAL_GAUSS_SEIDEL ($Constraints, \epsilon$)
1 do
2 for each α in $Constraints$ do
3 find $\mathbf{R}_\alpha$ such that $\mathbf{C}_\alpha \left(\mathbf{B}_\alpha + \sum_\beta \mathbf{W}_{\alpha\beta}\mathbf{R}_\beta, \mathbf{R}_\alpha \right) = \mathbf{0}$
4 assuming $\mathbf{R}_\beta =$ constant for $\beta \neq \alpha$
5 while $g(\mathbf{R}) > \epsilon$

choices (not necessarily the best). It is used as a demonstration. Selecting a best convergence criterion for the ill-conditioned frictional contact problems generated by Contact Dynamics remains an open issue. Other termination criteria has been discussed for example in [35, 34].

7. Distributed Gauss-Seidel scheme

The equations of local dynamics (14) read

$$\mathbf{U}_\alpha = \mathbf{B}_\alpha + \sum_\beta \mathbf{W}_{\alpha\beta}\mathbf{R}_\beta \tag{27}$$

where $\mathbf{U}_\alpha$ are the relative velocities and $\mathbf{R}_\alpha$ are the reactions at constraint points. $\mathbf{U}_\alpha, \mathbf{R}_\alpha, \mathbf{B}_\alpha$ are 3-vectors, while $\mathbf{W}_{\alpha\beta}$ are 3×3 matrix blocks. Each constraint equation can be formulated as

$$\mathbf{C}_\alpha (\mathbf{U}_\alpha, \mathbf{R}_\alpha) = \mathbf{0} \tag{28}$$

or in other words

$$\mathbf{C}_\alpha \left(\mathbf{B}_\alpha + \sum_\beta \mathbf{W}_{\alpha\beta}\mathbf{R}_\beta, \mathbf{R}_\alpha \right) = \mathbf{0} \tag{29}$$

Algorithm 2 is quite simple: diagonal block problems are solved[2] until the accuracy measure is small enough. The Gauss-Seidel paradigm corresponds to the fact, that the most recent off-diagonal reaction values are used when solving the diagonal problems. Of course, because of that, a perfectly parallel implementation is not possible. After all, reactions are updated in a sequence. We can nevertheless relax the need for sequential processing. Perhaps the most scalable Gauss-Seidel approach to date was devised by Adams [3]. Although originally used as a multigrid smoother, the core idea can be as well applied in our context. Each processor owes a subset of (internal) constraints Q_i, where $i = 1, 2, ..., n$ are the processors indices. Therefore the local velocity update can be rewritten as

[2] The semismooth Newton method from [26, 20] is used to solve the diagonal block problems.

Algorithm 3 Simple processor coloring.

COLOR ()
1 for i= $1, ..., n$ do $color\,[i] = 0$
2 for i= $1, ..., n$ do
3 do
4 $color\,[i] = color\,[i] + 1$
5 while for any $j \in adj\,(i)$ there holds $color\,[i] = color\,[j]$

$$\mathbf{U}_\alpha = \mathbf{B}_\alpha + \sum_{\beta \in Q_i} \mathbf{W}_{\alpha\beta}\mathbf{R}_\beta + \sum_{\beta \notin Q_i} \mathbf{W}_{\alpha\beta}\mathbf{R}_\beta \tag{30}$$

Some of the $\mathbf{W}_{\alpha\beta}$ blocks and reactions $\mathbf{R}_\beta$ correspond to the (external) constraints stored on other processors ($\beta \notin Q_i$). Let us denote the set of corresponding reaction indices by P_i. That is

$$P_i = \{\beta : \exists \mathbf{W}_{\alpha\beta} \neq \mathbf{0} \text{ and } \alpha \in Q_i \text{ and } \beta \notin Q_i\} \tag{31}$$

For each $\beta \in P_i$ we know an index of processor $cpu\,(\beta)$ storing the constraint with index β. For processor i we can then define a set of adjacent processors as follows

$$adj\,(i) = \{cpu\,(\beta) : \beta \in P_i\} \tag{32}$$

When updating reactions, a processor needs to communicate only with the other adjacent processors. We are going to optimise a pattern of this communication by *coloring* the processors. We shall then assign to each processor a color, such that no two adjacent processors have the same color. A simple coloring method is summarised in Algorithm 3. We try to assign as few colors as possible. We then split the index sets Q_i as follows

$$Top_i = \{\alpha : \forall \mathbf{W}_{\alpha\beta} : \beta \in P_i \wedge color\,[cpu\,(\beta)] < color\,[i]\} \tag{33}$$

$$Bottom_i = \{\alpha : \forall \mathbf{W}_{\alpha\beta} : \beta \in P_i \wedge color\,[cpu\,(\beta)] > color\,[i]\} \tag{34}$$

$$Middle_i = \{\alpha : \forall \mathbf{W}_{\alpha\beta} : \beta \in P_i \wedge \alpha \notin Top_i \cup Bottom_i\} \tag{35}$$

$$Inner_i = Q_i \setminus \{Top_i \cup Bottom_i \cup Middle_i\} \tag{36}$$

The top constraints require communication only with processors of lower colors. The bottom constraints require communication only with processors of higher colors. The middle constraints require communication with either. The inner constraints require no communication. The inner reactions are further split in two sets

$$Inner_i = Inner1_i \cup Inner2_i \tag{37}$$

such that

$$|Bottom_i| + |Inner2_i| = |Top_i| + |Inner1_i| \tag{38}$$

The parallel Gauss-Seidel scheme is summarised in Algorithm 4. The presented version is simplified in the respect, that alternate forward and backward runs are not accounted for (in terms of constraints ordering). We first process the Top_i set: a single sweep over the corresponding diagonal block problems is performed in line 3. Next we send the computed top reactions to the processors with lower colors. We try to overlap communication and computation, hence we sweep over the $Inner2_i$ set (line 5) while sending. We then receive the top reactions. It should be noted that all communication is asynchronous: we only wait to receive reactions immediately necessary for computations. In line 7 we enter the loop processing the $Middle_i$ set. This is the location of the computational bottleneck. Middle nodes communicate with processors of higher and lower colors and hence they need to be processed in a sequence. The sequential processing is further relaxed by using processor coloring. In the LOOP part of the algorithm we first sort the constraints according to the descending order of maximal colors of their adjacent processors (line 1). We then maintain this ordering while processing constraints. As the top reactions were already sent, some of the constraints from the middle set will have their external reactions from higher colors fully updated. These will be processed first in line 5 of LOOP and then sent to lower and higher (by color) processors in line 7. This way some processors with lower colors will have their higher color off-diagonal reactions of middle set constraints fully updated and they will proceed next. And so on. At the end (line 8), we need to receive all remaining reactions that have been sent in line 7 of LOOP. Coming back to PARALLEL_GAUSS_SEIDEL, after the bottleneck of the LOOP, in lines 8-11 we process the $Bottom_i$ and $Inner1_i$ sets in the same way as we did with the Top_i and $Inner2_i$ sets. The condition (38) attempts to balance the amount of computations needed to hide the communication (e.g. the larger the Top_i set is, the larger the $Inner2_i$ set becomes). It should be noted, that the convergence criterion in line 12 is global across all processors.

8. Examples

The first example is meant to illustrate an analysis with large relative motion: a masonry bridge is backfilled with randomly generated stones. The second example is closer to our application context: a simplified graphite core undergoes a seismic excitation. The examples involve an undamped dynamic analysis with contact states changing at every time step. Dissipation of energy corresponds to the ideally plastic impacts as detailed in Section 5. The material properties were: 15E9, 0.3, 1.8E3, 0.5 for respectively the Young modulus, the Poisson ratio, the mass density and the Coulomb friction. In all of the tests a "symmetrized" version of the Gauss-Seidel solvers was employed: alternate forward and backward runs were made over the set of contact points. In all tests the Gauss-Seidel tolerance was 1E-5 and the iterations bound was 200. The tests

Algorithm 4 Parallel Gauss-Seidel algorithm.

SWEEP (Set)
1 for each $\alpha \in Set$ do
2 find $\mathbf{R}_\alpha$ such that $\mathbf{C}_\alpha \left(\mathbf{B}_\alpha + \sum_\beta \mathbf{W}_{\alpha\beta}\mathbf{R}_\beta, \mathbf{R}_\alpha \right) = \mathbf{0}$
3 assuming $\mathbf{R}_\beta =$ constant for $\beta \neq \alpha$

LOOP (Set)
1 descending sort of $\alpha \in Set$ based on $\max\left(color\left[cpu\left(\beta\right)\right]\right)$ where $\exists \mathbf{W}_{\alpha\beta}$
2 for each ordered α in Set do
3 for each β such that $\exists \mathbf{W}_{\alpha\beta}$ and $color\left[cpu\left(\alpha\right)\right] < color\left[cpu\left(\beta\right)\right]$ do
4 if not received $(\mathbf{R}_\beta)$ then receive $(\mathbf{R}_\beta)$
5 find $\mathbf{R}_\alpha$ such that $\mathbf{C}_\alpha \left(\mathbf{B}_\alpha + \sum_\beta \mathbf{W}_{\alpha\beta}\mathbf{R}_\beta, \mathbf{R}_\alpha \right) = \mathbf{0}$
6 assuming $\mathbf{R}_\beta =$ constant for $\beta \neq \alpha$
7 send $(\mathbf{R}_\alpha)$
8 receive all remaining $\mathbf{R}_\beta$

PARALLEL_GAUSS_SEIDEL (ϵ)
1 COLOR ()
2 do
3 SWEEP (Top_i)
4 send (Top_i)
5 SWEEP ($Inner2_i$)
6 receive (Top_i)
7 LOOP ($Middle_i$)
8 SWEEP ($Bottom_i$)
9 send ($Bottom_i$)
10 SWEEP ($Inner1_i$)
11 receive ($Bottom_i$)
12 while $g\left(\mathbf{R}\right) > \epsilon$

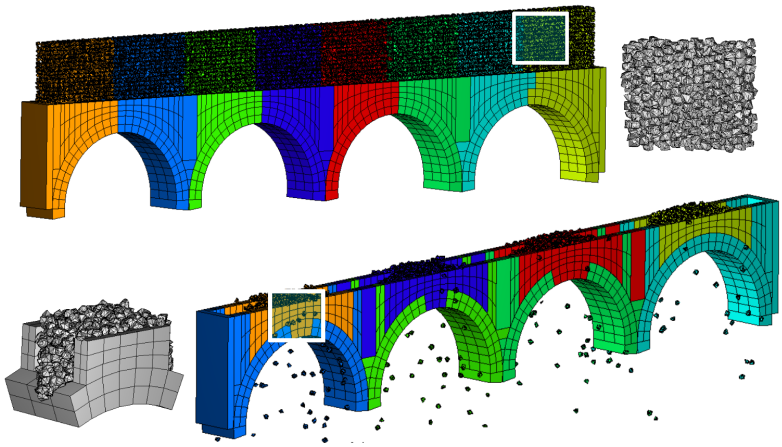

Figure 4: The bridge backfilling example. The upper figure pictures the beginning and the lower figure pictures the end of the analysis. The colors represent the assignment of bodies to processors for the 8 processor run.

were run on an Intel Xeon 5472 3.00GHz based cluster with Infiniband network. We used 8, 16, 32 and 64 processors in order to investigate the parallel scaling. In all tests an imbalance tolerance 1.3 was used for the dynamic domain decomposition. This means, that the RCB partitioning was updated once the the ratio of the maximal subdomain weight overt the minimal subdomain weight reached 1.3 (imbalance tolerance > 1.0 avoids frequent updates of RCB and alleviates the cost of related data migration). The $factor$ in (16) was 1.0 during all tests.

8.1. Bridge backfilling

The bridge backfilling model is presented in Figure 4. 10030 rigid bodies were used to set up the geometry (60180 degrees of freedom). The backfilling stones were convex, with 8 to 64 vertices and they were randomly generated without initial interactions. The rigid motion was integrated as detailed in [27]. 3 second runs were computed with the time step 1E-4 in order to test the parallel scaling on 8 to 64 processors. Additionally, the 64 processor run was repeated with time steps 2E-5, 5E-4 and 1E-3 in order to exemplify the influence of the step size on the convergence of the Gauss-Seidel solver and on total runtimes.

Figure 5 illustrates total runtimes. The scaling is linear up to 32 processors and then it flattens. With 10E3 bodies and 25E3 contact points (cf. Figure 8) the bridge example is too small to scale linearly up to 64 processors. With the increase of the time step though, the 64 processor run shows quite substantial reduction of the total runtime. This is because the system gradually comes

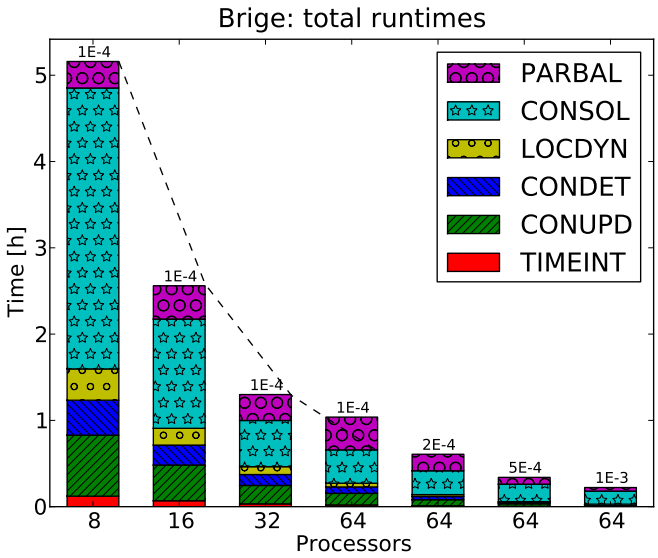

Figure 5: The bridge example: total runtimes. TIMEINT stands for time integration. CONUPD stands for constraints update. CONDET stands for contact detection. LOCDYN stands for local dynamics assembling. CONSOL stands for the Gauss-Seidel constraints solution. PARBAL stands for parallel load balancing and communication not related to constraints solution. The dashed line indicates linear scaling for total runtimes.

to a rest and the Gauss-Seidel solver takes advantage of the fact that contact interactions stabilize. The obvious observation from Figure 5 is that the constraint solution (CONSOL, Gauss-Seidel) takes up most of the time. Another significant part, at least for smaller processor counts is contact detection and update (CONDET, CONUPD). This is due to the fact, that Solfec employs a volumetric intersection method in order to derive contact points and normals [25], which is robust for nonsmooth geometry but comes at an additional cost. The PARBAL time corresponds to the maintenance of the dynamic domain decomposition (updates of external constraints and child bodies take up most of it, while RCB updates are negligible). In Figure 5 the PARBAL time is moderate and it does not show a clear growth or decrease trend. This is quite typical, as it is a resultant of the domain geometry combined with the geometry of the RCB partitioning (dependent on the number of processors), further combined with the imbalance tolerance value and the value of $factor$ in (16). In practice, one can always choose the RCB parameters such that the PARBAL time remains moderate. The LOCDYN runtime corresponds to the assembling of local dynamics (14). It normally scales well and for simple kinematic models (e.g. rigid, pseudo-rigid) it remains quite small. The time integration runtimes (TIMEINT) are clearly not an issue.

The Gauss-Seidel solver scaling in Figure 6 is initially superlinear, although for 64 for processors it becomes linear again. One reason for this irregular be-

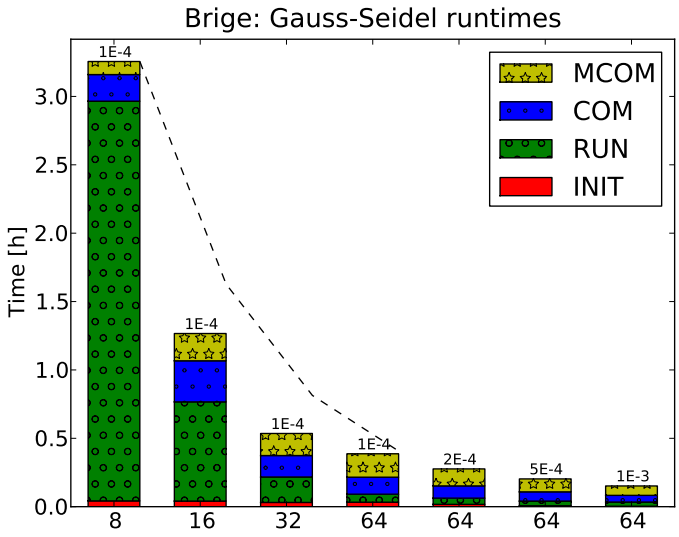

Figure 6: The bridge example: Gauss-Seidel runtimes. INIT stands for initialization. RUN stands for floating point computations. COM stands for communication not related to the middle set of constraints. MCOM stands for the communication of the middle set of constraint. The dashed line indicates linear scaling for Gauss-Seidel runtimes.

haviour is that the geometrical domain decomposition is not always capable of providing a good partitioning of $\mathbf{W}$ in (14). As a result, the off-diagonal terms of $\mathbf{W}$ can be non-uniformly distributed across the subdomains. This is additionally stipulated by the fact, that in order to compute $g(\mathbf{R})$ in (26), an extra matrix-vector product must be evaluated for each Gauss-Seidel iteration. This corresponds to the high values of RUN in Figure 6, which sums up the Gauss-Seidel sweep and $g(\mathbf{R})$ calculation runtimes. A typical tendency, observed in Figure 6, is the domination of the communication time with the growing number of processors (e.g. COM and MCOM for 64 processors). This is unavoidable if we chose to maintain the strict Gauss-Seidel update. Our solver implements as well simplifications, e.g. the Jacobi update for the middle nodes or for all of the boundary nodes. Nevertheless our experience is such, that using the Jacobi update, even for the middle nodes alone, often breaks down the convergence and results in unphysical artifacts.

The Gauss-Seidel iteration counts are pictured in Figure 7. The runs with the time step 1E-4 on 8 to 64 processors do not show much of a difference in terms of the convergence rate, until $g(\mathbf{R}) <$ 1E-5. This is expected, since they only correspond to different orderings of contact points, while the Gauss-Seidel implementation is consistent. Only in case of very frequent re-partitioning the convergence rate could deteriorate (for any number of processors), as the altered contact point orderings might disturb "hot restarts". For the 64 processor run

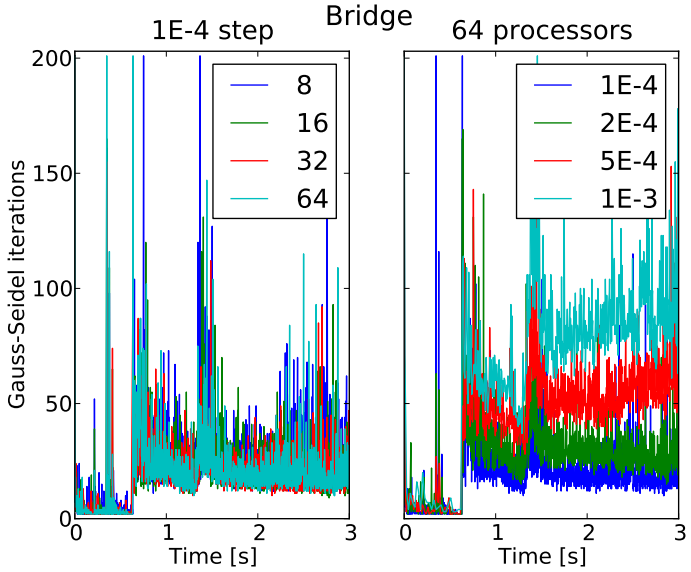

Figure 7: The bridge example: Gauss-Seidel iterations histories for 8, 16, 32 and 64 processors for the time step 1E-4 (left), and the 64 processor run histories for the time steps 1E-4, 2E-4, 5E-4 and 1E-3 (right).

with the growing step sizes, the number of the necessary Gauss-Seidel iterations grows as well. Clearly, the larger steps lead to more variable $\mathbf{C}(\mathbf{R}) = \mathbf{0}$ problems and the advantage of time coherence (similarity of consecutive solutions) is smaller. For this particular example it still "pays off" to use larger steps, which is not always the case.

Finally, Figure 8 shows, that the variability of the total number of contact points is quite minor with respect to the number of processors or step size. From the initial number of contact points of about 2600 (bridge skeleton), their number gradually grows to the order of 27000. This evolution is gradual and since the elastic energy is not stored in the model, the structure of contact point interactions is building up rather than rapidly changing. This might be one of the reasons why the Gauss-Seidel solver performs well also for the larger time steps for this example.

8.2. Simplified core

The simplified graphite core model is presented in Figure 9. The model is based on a repetition of a pattern of loosely keyed together bricks and it is meant to approximate the computational complexity of a more elaborate model. 4000 pseudo-rigid[3] and 1333 rigid bodies (55998 degrees of freedom) are subject

[3]See [13, 39, 11] for the introduction to pseudo-rigid bodies.

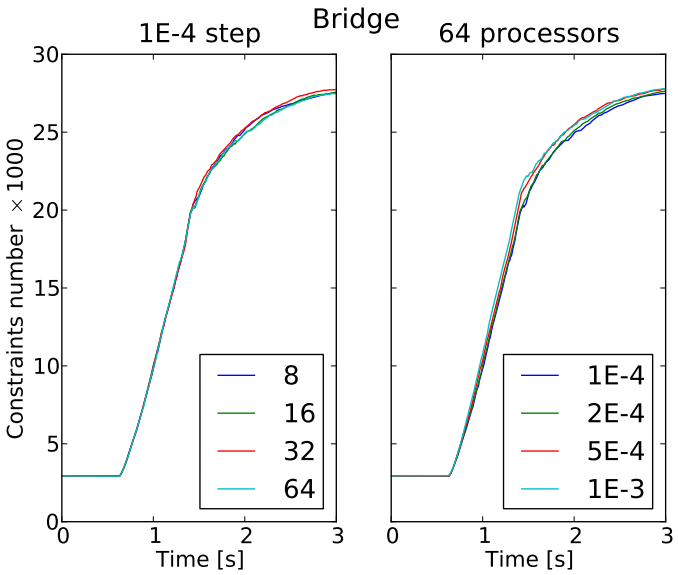

Figure 8: The bridge example: Constraints number histories for 8, 16, 32 and 64 processors for the time step 1E-4 (left), and the 64 processor run histories for the time steps 1E-4, 2E-4, 5E-4 and 1E-3 (right).

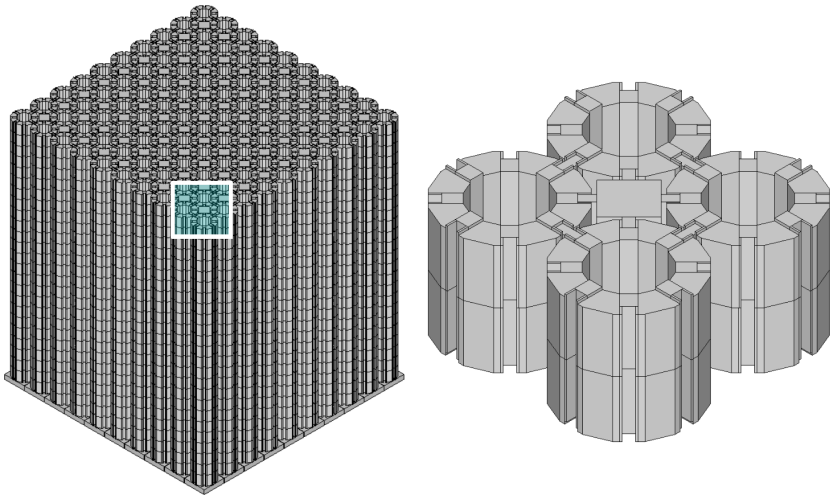

Figure 9: The simplified graphite core example. The model comprises twelve layers of a repeated pattern, detailed on the right. The layers are connected through convex-concave features of the top and bottom surfaces of the hollow bricks.

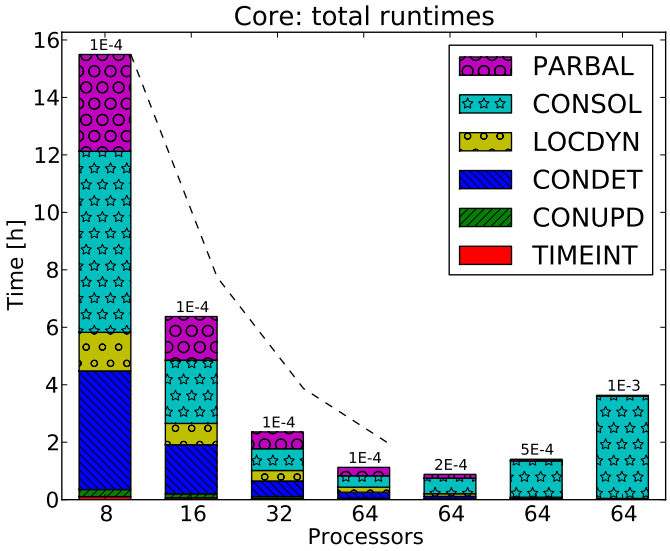

Figure 10: The core example: total runtimes. TIMEINT stands for time integration. CONUPD stands for constraints update. CONDET stands for contact detection. LOCDYN stands for local dynamics assembling. CONSOL stands for the Gauss-Seidel constraints solution. PARBAL stands for parallel load balancing and communication not related to constraints solution. The dashed line indicates linear scaling for total runtimes.

to gravity and rest on a rigid foundation. The horizontal acceleration of the foundation equals to a seismic signal. Rather than using equations (5-7), an implicit time integrator similar to Simo and Tarnow [38] was employed for the pseudo-rigid model, while the rigid motion was integrated as detailed in [27]. 1 second runs were computed with the time step 1E-4 in order to test the parallel scaling on 8 to 64 processors. Additionally, the 64 processor run was repeated with time steps 2E-5, 5E-4 and 1E-3 in order to exemplify the influence of the step size on the convergence of the Gauss-Seidel solver and on total runtimes.

In Figures 10 and 11 for both the total runtimes and for the Gauss-Seidel solver the superlinear parallel scaling is observed. The same comments as in the case of the bridge example apply. Here however, the behaviour of the 64 processor run for growing time steps is clearly different. Because in this example the base excitation is constantly injecting the energy, while the pseudo-rigid bodies deform and redistribute it further, the contact point interactions considerably change across short time intervals. Hence, while the time coherence can be exploited for the smallest time steps, the convergence quite rapidly deteriorates for larger steps. This is clearly visible in Figure 12, where for the step 1E-3 the Gauss-Seidel solver is constantly hitting the iterations bound. The total runtimes grow for larger time steps and are Gauss-Seidel dominated. Leaving out physical considerations, there is than an advantage for the dynamic simulations of this kind, in finding an "optimal time step" for which a minimum runtime

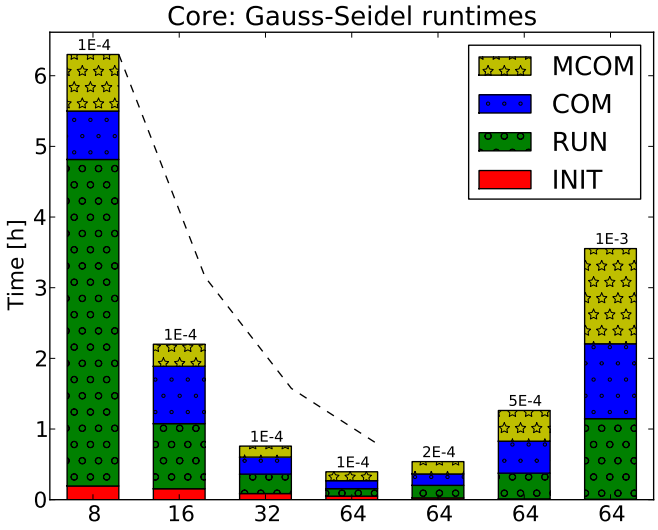

Figure 11: The core example: Gauss-Seidel runtimes. INIT stands for initialization. RUN stands for floating point computations. COM stands for communication not related to the middle set of constraints. MCOM stands for the communication of the middle set of constraint. The dashed line indicates linear scaling for Gauss-Seidel runtimes.

can be achieved (decreasing the time step would again cause the growth of the total runtime).

In Figure 12 on the left, there are bumps in the iterations number history for the 8-processor run. The PARBAL time in Figure 10 is considerable for this run. As already discussed in the previous section, this suggests that for this particular combination of geometry, processors number and domain decomposition parameters, the number of re-partitionings might be unnecessarily high. This could have been alleviated by a hand tuning of the RCB parameters.

The strongly dynamic character of this example is visible in Figure 13, where the contact point number histories show some variability with respect both the processor number and step size. This is not a surprise, since the dynamics of the considered system is highly nonlinear and sensitive with respect to the simulation circumstances. Nevertheless, the general growth trend of the number of contact points is well represented for all runs.

9. Conclusions and outlook

We have demonstrated that a good parallel efficiency can be achieved by a multibody time-stepping implemented on a distributed memory computer. The Gauss-Seidel solver (GS) performs reasonably well in this context, although its implementation requires much care. The Gauss-Seidel method is used in Contact Dynamics as a solver, mostly because of the ill-conditioning and strongly

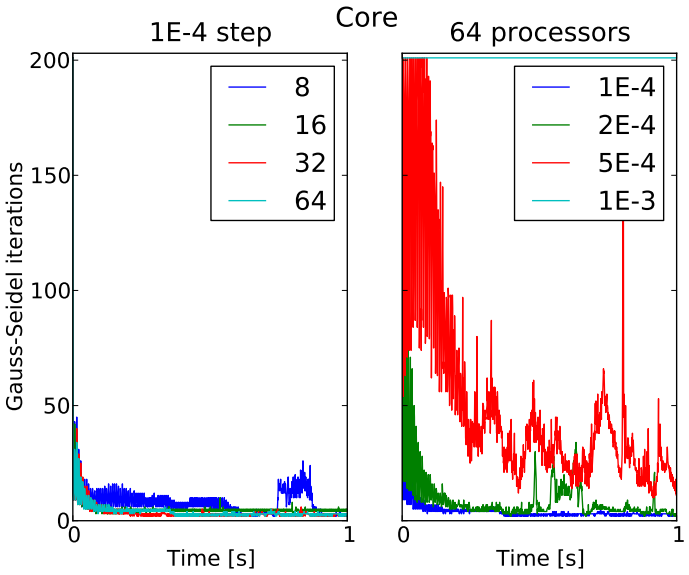

Figure 12: The core example: Gauss-Seidel iterations histories for 8, 16, 32 and 64 processors for the time step 1E-4 (left), and the 64 processor run histories for the time steps 1E-4, 2E-4, 5E-4 and 1E-3 (right).

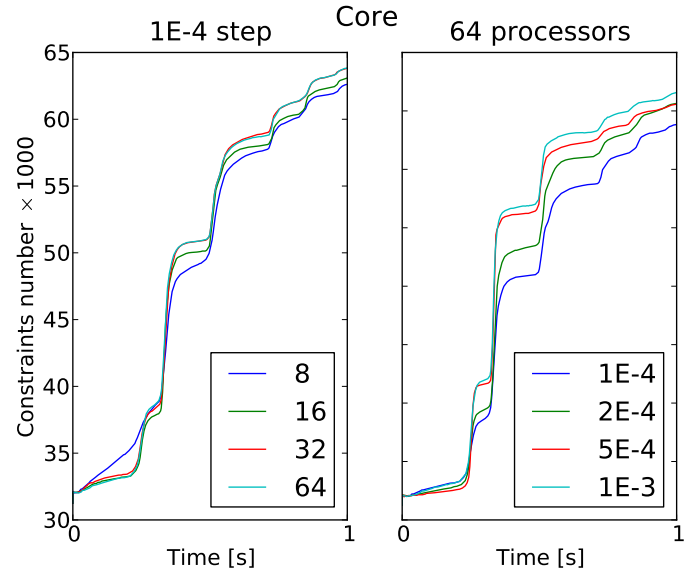

Figure 13: The core example: Constraints number histories for 8, 16, 32 and 64 processors for the time step 1E-4 (left), and the 64 processor run histories for the time steps 1E-4, 2E-4, 5E-4 and 1E-3 (right).

nonlinear nature of the constraints equations. While the solution of the diagonal block problems is manageable in GS, more sophisticated solvers (e.g. Newton method or optimization based) usually fail on three-dimensional frictional contact problems, derived from large multi-body systems. The Gauss-Seidel solver remains then, until a better method is found, the most robust approach in the CD context. Of course, GS occasionally fails as well. There are instances for which it "gets stuck", a phenomenon confirmed in our numerical practice and in communication with other researchers, and perhaps owing to the fact, that GS is in fact more of a nonlinear smoother rather than a solver: it may fail to resolve longer range force interactions.

In the simplified core example of the previous section we have used a rather small time step in order to better exploit the time coherence. The consecutive frictional contact problems had similar solutions and, in consequence, the numbers of per-step GS iterations were moderate. Should we use a one order of magnitude larger step, the Gauss-Seidel solver would have required hundreds of iterations to converge (if at all) to the assumed accuracy. In terms of practicality of runtimes, this would not be acceptable. But even for the smaller time step, we could observe that the Contact Dynamic specific tasks took up over 80% of the total computational time. Clearly, if we would wish to explicitly integrate a large, stiff deformable problem, this constant per-step cost of CD would make such computation impractical. More so, when compared with the corresponding per-step cost of the force $\cdot$ gap computation of the classical DEM. Surely, the necessary choice of the penalty parameter in DEM usually additionally lowers the stability limit, while in CD it corresponds solely to the material properties. But even then, for very small steps the picture is no good for CD and in a sense, it cannot be improved (force $\cdot$ gap computation is faster than any implicit solution). Of course, the whole point of using Contact Dynamics is that the very small time steps can be avoided: for large multibody structures we are usually interested in the overall, gross motion, rather than in fast dynamics. Hence, like in the simplified core example, although we are aware of the fact that fast dynamics plays a role in the transfer of shock waves, we use CD together with simplified deformability in order to capture some of the dynamics at a lower resolution. It is a pragmatic choice, since an explicit FE model fully resolving the geometry would have required a much larger computational effort.

Current work on Solfec centers around lowering the constant per-step cost of Contact Dynamics. In particular, a projected quasi-Newton solver is under development, for which the communication cost is smaller than that of the Gauss-Seidel scheme. Another stream of work corresponds to simplified deformable models. Apart from the rigid and pseudo-rigid kinematics, Solfec implements Total Lagrangian and co-rotational finite elements, allowing for an arbitrary shape to be submerged in a coarse, non-conforming mesh. This, together with a family of semi-implicit time integrators, facilitates modeling of variable deformability. Finally, readers are reminded that Solfec can be downloaded from http://code.google.com/p/solfec.

Acknowledgements

The support from British Energy is gratefully acknowledged. The calculations were performed on a cluster of the Institute of Computer Modelling at the Krakow University of Technology. We would like to thank the anonymous reviewers for their feedback, which contributed to a substantial improvement of the paper.

References

[1] V. Acary and B. Brogliato. *Numerical Methods for Nonsmooth Dynamical Systems*, volume 35 of *Lecture Notes in Applied Lecture Notes in Applied and Computational Mechanics*. Springer Verlang, 2008.

[2] Vincent Acary and Franck Pérignon. An introduction to Siconos. Technical Report RT-0340, INRIA, 2007.

[3] Mark F. Adams. A distributed memory unstructured Gauss-Seidel algorithm for multigrid smoothers. In *Supercomputing '01: Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, pages 4–4, New York, USA, 2001. ACM Press.

[4] P. Alart and A. Curnier. Mixed formulation for frictional contact problems prone to Newton like solution methods. *Computer Methods in Applied Mechanics and Engineering*, 92:353–375, 1991.

[5] P. Alart, Barboteu M., and M. Renouf. Parallel computational strategies for multicontact problems: Applications to cellular and granular media. *Internat. J. Multiscale Comput. Engrg*, pages 419–430, 2003.

[6] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Comput.*, 36(5):570–580, 1987. ISSN 0018-9340.

[7] P. Breitkopf and M. Jean. Modélisation parallèle des matériaux granulaires. In *4ème colloque national en calcul des structures*, Giens, 1999.

[8] B Brogliato, AA ten Dam, L Paoli, F Genot, and M Abadie. Numerical simulation of finite dimensional multibody nonsmooth mechanical systems. *Applied Mechanics Reviews*, 55(2):107–150, 2002.

[9] Bernard Brogliato. *Nonsmooth Mechanics*. Communications and Control Engineering. Springer Verlang, 1999.

[10] Kevin Brown, Steve Attaway, Steve Plimpton, and Bruce Hendrickson. Parallel strategies for crash and impact simulations. *Computer Methods in Applied Mechanics and Engineering*, 184(2-4):375–390, Apr 2000.

[11] J. Casey. Pseudo-rigid continua: Basic theory and a geometrical derivation of lagrange's equations. *Proceedings of the Royal Society - Mathematical, Physical and Engineering Sciences (Series A)*, 460:2021–2049, 2004.

[12] B. Chetouane, F. Dubois, M. Vinches, and C. Bohatier. Nscd discrete element method for modelling masonry structures. *International Journal for Numerical Methods in Engineering*, 64:65–94, 2005.

[13] H. Cohen and R. G. Muncaster. *The Theory of Pseudo-rigid Bodies*. Springer, New York, 1988.

[14] P. A. Cundall and O. D. L. Strack. A distinct element model for granular assemblies. *Geotechnique*, 29:47–65, 1979.

[15] G. De Saxcé and Z. Q. Feng. The bipotential method: a constructive approach to design the complete contact law with friction and improved numerical algorithms. *Mathematical and Computer Modelling*, 28:225–245, 1998.

[16] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[17] F. Dubois and M. Jean. The non smooth contact dynamic method: recent lmgc90 software developments and application. In *Analysis and Simulation of Contact Problems*, volume 27 of *Lecture Notes in Applied and Computational Mechanics*, pages 375–378. Springer Berlin / Heidelberg, 2006.

[18] M. Förg, R. Zander, and H. Ulbrich. A framework for the efficient simulation of spatial contact problems. In *Proceedings of the ECCOMAS Conference on Multi-Body Systems*, Milano, Italy, 2007.

[19] Ch. Glocker. *Set-Valued Force Laws*, volume 1 of *Lecture Notes in Applied and Computational Mechanics*. Springer Verlang, 2001.

[20] S. Hüeber, G. Stadler, and B. I. Wohlmuth. A primal-dual active set algorithm for three-dimensional contact problems with Coulomb friction. *SIAM Journal on Scientific Computing*, 30(2):572–596, 2007.

[21] D. Iceta, D. Dureisseix, and P. Alart. Mixed versus impulse-oriented domain decomposition method for granular dynamics. *European Journal of Computational Mechanics*, 18(5-6):429–443, 2009.

[22] Damien Iceta, Pierre Alart, and David Dureisseix. A multilevel domain decomposition solver suited to nonsmooth mechanical problems. In *Domain Decomposition Methods in Science and Engineering*, pages 113–120. Springer Berlin Heidelberg, 2009.

[23] M. Jean. The non-smooth contact dynamics method. *Computer Methods in Applied Mechanics and Engineering*, 177(3-4):235–257, 1999.

[24] F. Jourdan, P. Alart, and M. Jean. A Gauss-Seidel like algorithm to solve frictional contact problems. *Computer Methods in Applied Mechanics and Engineering*, 155:31–47, 1998.

[25] Tomasz Koziara. *Aspects of computational contact dynamics*. PhD thesis, University of Glasgow, http://theses.gla.ac.uk/429/, 2008.

[26] Tomasz Koziara and Nenad Bićanić. Semismooth Newton method for frictional contact between pseudo-rigid bodies. *Computer Methods in Applied Mechanics and Engineering*, 197:2763–2777, 2008.

[27] Tomasz Koziara and Nenad Bićanić. Simple and efficient integration of rigid rotations suitable for constraint solvers. *Journal for Numerical Methods in Engineering*, 81(9):1073 – 1092, 2009.

[28] Tomasz Koziara and Nenad Bićanić. A projected quasi-Newton method for dynamic multibody frictional contact problems. *In preparation*, 2011.

[29] R.I. Leine and N. van de Wouw. *Stability and Convergence of Mechanical Systems with Unilateral Constraints*, volume 36 of *Lecture Notes in Applied and Computational Mechanics*. Springer Verlang, 2008.

[30] J. J. Moreau. *Unilateral Contact and Dry Friction in Finite Freedom Dynamics*, volume 302 of *Non-smooth Mechanics and Applications, CISM Courses and Lectures*. Springer, Wien, 1988.

[31] J. J. Moreau. Numerical aspects of the sweeping process. *Computer Methods in Applied Mechanics and Engineering*, 177(3-4):329–349, 1999.

[32] J. J. Moreau. Some basics of unilateral dynamics. In F. Pfeiffer and C. Glocker, editors, *Unilateral Multibody Contacts*. Kluwer, Dordrecht, 1999.

[33] Shéhérazade Nineb, Pierre Alart, and David Dureisseix. Domain decomposition approach for non-smooth discrete problems, example of a tensegrity structure. *Computers & Structures*, 85(9):499–511, May 2007. ISSN 0045-7949.

[34] Farhang Radjai and Vincent Richefeu. Contact dynamics as a nonsmooth discrete element method. *Mechanics of Materials*, 41(6):715–728, 2009. ISSN 0167-6636.

[35] M. Renouf and P. Alart. Conjugate gradient type algorithms for frictional multi-contact problems: Applications to granular materials. *Computer Methods in Applied Mechanics and Engineering*, 194:2019–2041, 2005.

[36] M. Renouf, F. Dubois, and P. Alart. A parallel version of the non smooth contact dynamics algorithm applied to the simulation of granular media. *Journal of Computational and Applied Mathematics*, 168:375–382, 2004.

[37] Thorsten Schindler, Martin Förg, Markus Friedrich, Markus Schneider, Bastian Esefeld, Robert Huber, Roland Zander, and Heinz Ulbrich. Analysing dynamical phenomenons: Introduction to mbsim. In *The 1st Joint International Conference on Multibody System Dynamics*, Lappeenranta, Finland, 2010.

[38] J. C. Simo and N. Tarnow. The discrete energy-momentum method. Conserving algorithms for nonlinear elastodynamics. *Zeitschrift für Angewandte Mathematik und Physik (ZAMP)*, 43:757–792, September 1992.

[39] J. M. Solberg and P. Papadopoulos. A simple finite element-based framework for the analysis of elastic pseudo-rigid bodies. *International Journal for Numerical Methods in Engineering*, 45:1297–1314, 1999.

[40] Christian Studer. The DynamY software. In *Numerics of Unilateral Contacts and Friction*, volume 47 of *Lecture Notes in Applied and Computational Mechanics*, pages 129–159. Springer Berlin / Heidelberg, 2009.

[41] Christian Studer. *Numerics of Unilateral Contacts and Friction*, volume 47 of *Lecture Notes in Applied and Computational Mechanics*. Springer Verlang, 2009.

[42] Robert Zander, T. Schindler, M. Friedrich, R. Huber, M. Förg, and H. Ulbrich. Non-smooth dynamics in academia and industry: recent work at TU München. *Acta Mechanica*, 195(1):167–183, Jan 2008.