MULTIBODY DYNAMICS 2011, ECCOMAS Thematic Conference
J.C. Samin, P. Fisette (eds.)
Brussels, Belgium, 4-7 July 2011

HIGH PERFORMANCE ALGORITHMS AND IMPLEMENTATIONS USING SPARSE AND PARALLELIZATION TECHNIQUES ON MBS

Andrés F. Hidalgo[*], Javier García de Jalón[†] and Santiago Tapia[†]

[*] INSIA
Universidad Politécnica de Madrid, 28031 Madrid, Spain
e-mail: andres.francisco.hidalgo@upm.es,

[†] ETSII and INSIA
Universidad Politécnica de Madrid, 28031 Madrid, Spain
e-mails: javier.garciadejalon@upm.es, santiago.tapia@upm.es

Keywords: Multibody dynamics, Parallelization, Sparse, Real-time.

Abstract. *In this paper we will see how the efficiency of the MBS simulations can be improved in two different ways, by considering both an explicit and implicit semi-recursive formulation. The explicit method is based on a double velocity transformation that involves the solution of a redundant but compatible system of equations. The high computational cost of this operation has been drastically reduced by taking into account the sparsity pattern of the system. Regarding this, the goal of this method is the introduction of MA48, a high performance mathematical library provided by Harwell Subroutine Library. The second method proposed in this paper has the particularity that, depending on the case, between 70 and 85% of the computation time is devoted to the evaluation of forces derivatives with respect to the relative position and velocity vectors. Keeping in mind that evaluating these derivatives can be decomposed into concurrent tasks, the main goal of this paper lies on a successful and straightforward parallel implementation that have led to a substantial improvement with a speedup of 3.2 by keeping all the cores busy in a quad-core processor and distributing the workload between them, achieving on this way a huge time reduction by doing an ideal CPU usage.*

1 INTRODUCTION

Too many things can be said about the increasing interest that multibody systems have in different fields like automobiles, machinery, robotics, biomechanics, etc. The work of this paper is mainly focused on achieving high performance algorithm implementation on large and realistic multibody systems. Regarding this, two semi-recursive formulations will be used.

It is well known that one of the most important decisions to make when a dynamic simulation program is going to be developed is to choose an appropriate set of coordinates that univocally determine the position of each body of the system. Taking into account that we are concerned about real time simulations applied to large and complex multibody systems, and considering that the Cartesian coordinates can lead to large matrices sizes involving very expensive computation times, in this paper two formulations based on the usage of relatives coordinates will be seen.

According to some authors [1] and [2], using relative coordinates can significantly improve the efficiency of the formulation due to the reduction of the numerical problems to be solved. Although the relative coordinates usage in fully recursive formulations have been proved to be very efficient on large open-chain systems, it is even possible to take advantage of this fact working with closed chain systems too. This can be done by opening the closed loops of the system and working at this stage as it would be an open chain system. The relative coordinates in an open chain system are independent, in that way, the kinematic and dynamic of the system can be easily solved. Once the open chain equations have been obtained, the closure of them is performed. The two formulations used on this paper introduce the closure of open chain in two different ways; the first one uses a coordinate partitioning method [3] which has outstanding stabilization properties, and the second one enforces the fulfillment of the constraint equation by means of penalization and projections methods [4], [5] , [6] and [7].

Both formulations have its hotspots and bottlenecks. For that reason, two significant enhancements have been proposed. The first one explores the introduction of sparse techniques by means of MA48 package on semi-recursive formulations with redundant but compatible linear system of equations. The second and most important contribution, introduces a simple and straightforward parallelization that achieves a *speedup* of 3.2 in a commercial quad-core processor.

Nowadays, multi-core processors have made parallel computing a subject of interest for programmer, engineer and research communities. As some experts say, the age of serial computing is over. Most of the currently efficient multibody applications run in a serial way, but in parallel environments, as any commercial multi-core processor are.

Controlling parallelism by thinking in threads, synchronization and other low level variables can be a hard thing if we keep in mind all the inherent difficulties the multibody algorithm and implementations have. The goal of a parallel application in a modern computing environment is to achieve scalability and take advantage of all processor cores. The scalability can be seriously compromised if the parallel package is not being used correctly by writing an inefficient code. Hence, the parallel library to chose is a major decision when we are looking for a simple and efficient parallel application. Regarding this, threading Building Blocks [8] is a library provided by Intel® that presents some many advantages with respect to other packages like POSIX threads, MPI and OpenMP included. Maybe the outstanding advantage is that Threading Building Blocks by focusing on tasks, and not in threads, avoid tedious and error prone low-level programming operations. An additional deep and illustrative explanation about the advantages of Threading Building Blocks usage can be found on the quoted reference.

2 TOPOLOGICAL SEMI-RECURSIVE FORMULATIONS

In this section two different formulations are presented. At the beginning we will see a semi-recursive formulation that involves a double velocity transformation. This formulation starts with the dynamic equations set in Cartesian coordinates, and then, by applying these velocities transformations, the differential equations of motion expressed as a function of *independent* relatives coordinates are obtained.

The second method proposed on this paper consists of a semi-recursive formulation whose dynamic equations set is based on the usage of *dependent* relatives coordinates. The fulfillment of the constraints is imposed by using penalization coefficients and projections methods.

As mentioned before, the first formulation starts with the dynamic equations set in Cartesian coordinates and then applies two velocity transformations that lead to the differential equations of motion using a set of *independent* relative coordinates.

At the beginning, when considering the open-loop system, the geometry of each moving body is defined in a reference frame attached to the moving body in the input joint by using natural coordinates, i. e., by defining a set of points and unit vectors that describe the geometry of the body and its joints.

The geometry of each moving body is defined in a local reference frame attached to it by using natural coordinates, i. e., by defining a set of points and unit vectors that describe the geometry of the body and its joints, as can be seen in Fig. (1). In this way, the geometry becomes simpler and clearer than using multiple “markers” or additional reference frames attached to the moving bodies. When needed, this geometric information is easily transformed into the global reference frame using the body *position variables*, that are the position vectors of the origin of the moving reference frame $\mathbf{r}_i$ and the transformation matrix $\mathbf{A}_i$.

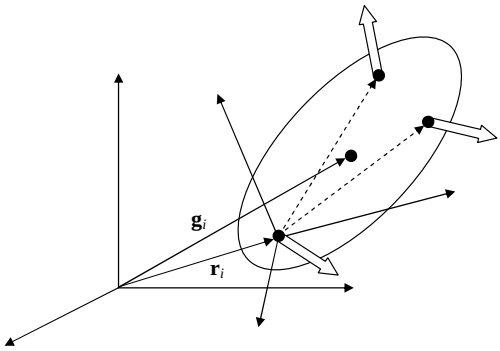

Figure 1: Body geometry defined with points and unit vectors

The Cartesian velocities and accelerations are defined by vectors

$$\mathbf{Z}_i \equiv \begin{Bmatrix} \dot{\mathbf{s}}_i \\ \boldsymbol{\omega}_i \end{Bmatrix}, \quad \dot{\mathbf{Z}}_i \equiv \begin{Bmatrix} \ddot{\mathbf{s}}_i \\ \dot{\boldsymbol{\omega}}_i \end{Bmatrix} \tag{1}$$

where $\dot{\mathbf{s}}_i$ and $\ddot{\mathbf{s}}_i$ are respectively the velocity and acceleration of the point attached to the body that instantaneously coincides with the origin of the inertial reference frame.

Vectors $\mathbf{Z}$ and $\dot{\mathbf{Z}}$ are respectively the vectors that contain the Cartesian velocities and accelerations of all bodies:

$$\mathbf{Z}^T = \begin{Bmatrix} \mathbf{Z}_1^T & \mathbf{Z}_2^T & \cdots & \mathbf{Z}_n^T \end{Bmatrix}, \quad \dot{\mathbf{Z}}^T = \begin{Bmatrix} \dot{\mathbf{Z}}_1^T & \dot{\mathbf{Z}}_2^T & \cdots & \dot{\mathbf{Z}}_n^T \end{Bmatrix} \tag{2}$$

Using points and unit vectors, joints between contiguous bodies are modeled very easily. For instance, in a revolute joint between bodies $i-1$ and i (see Fig. (2)), an output point and a unit vector of element $i-1$ coincide respectively with the input point and unit vector of element i. For a prismatic joint both elements share a unit vector, and the input point of element i is located on the line defined by the output point and unit vector of element $i-1$ (see Fig. (2)); in this case both elements share the same transformation matrix.

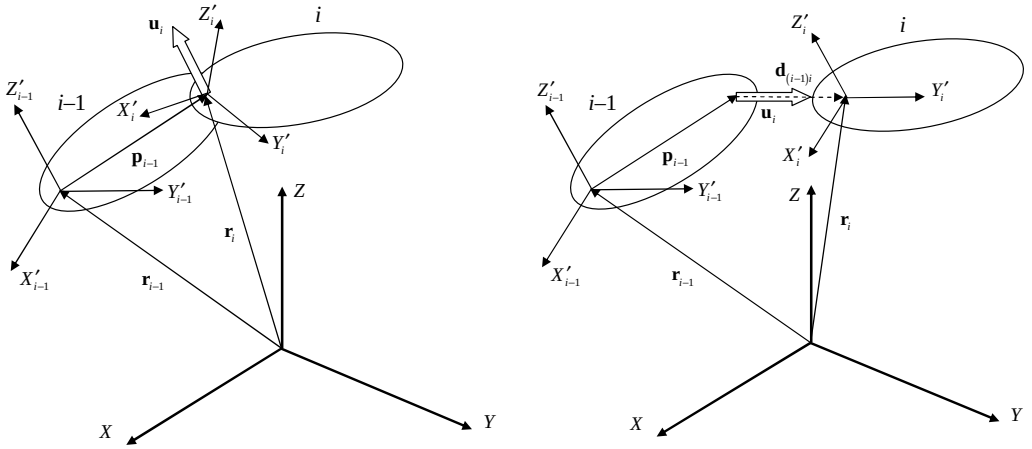

Figure 2: Revolute and Prismatic joints

In this way, all bodies share the same reference point coordinates, which has important advantages. For instance, the recursive formulas that give the Cartesian velocities and accelerations of body i in terms of those of body $i-1$ are very simple because they do not need transformation matrices:

$$\begin{aligned} \mathbf{Z}_i &= \mathbf{Z}_{i-1} + \mathbf{b}_i \dot{z}_i \\ \dot{\mathbf{Z}}_i &= \dot{\mathbf{Z}}_{i-1} + \mathbf{b}_i \ddot{z}_i + \mathbf{d}_i \end{aligned} \tag{3}$$

Vectors $\mathbf{b}_i$ and $\mathbf{d}_i$ depend on the joint type (Revolute or Prismatic) that joints bodies i and $i-1$.

$$\begin{aligned} \mathbf{b}_j^R &= \begin{Bmatrix} \mathbf{r}_i \times \mathbf{u}_i \\ \mathbf{u}_i \end{Bmatrix}, \quad \mathbf{d}_j^R = \begin{Bmatrix} \left(2\boldsymbol{\omega}_{i-1} + \mathbf{u}_i \dot{\varphi}_i\right) \times \left(\mathbf{r}_i \times \mathbf{u}_i \dot{\varphi}_i\right) \\ \boldsymbol{\omega}_{i-1} \times \mathbf{u}_i \dot{\varphi}_i \end{Bmatrix} \\ \mathbf{b}_j^P &= \begin{Bmatrix} \mathbf{u}_i \\ \mathbf{0} \end{Bmatrix}, \quad \mathbf{d}_j^P = \begin{Bmatrix} 2\boldsymbol{\omega}_{i-1} \times \mathbf{u}_i \dot{s}_i \\ \mathbf{0} \end{Bmatrix} \end{aligned} \tag{4}$$

The dynamic analysis is easier using Cartesian velocities $\mathbf{Y}$ and accelerations $\dot{\mathbf{Y}}$ based on the center of gravity, which are defined in the form:

$$\mathbf{Y}_i = \begin{Bmatrix} \dot{\mathbf{g}}_i \\ \boldsymbol{\omega}_i \end{Bmatrix} = \begin{bmatrix} \mathbf{I}_3 & -\tilde{\mathbf{g}}_i \\ \mathbf{0} & \mathbf{I}_3 \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{s}}_i \\ \boldsymbol{\omega}_i \end{Bmatrix} = \mathbf{D}_i \mathbf{Z}_i \tag{5}$$

$$\dot{\mathbf{Y}}_i = \begin{Bmatrix} \ddot{\mathbf{g}}_i \\ \dot{\boldsymbol{\omega}}_i \end{Bmatrix} = \begin{bmatrix} \mathbf{I}_3 & -\tilde{\mathbf{g}}_i \\ \mathbf{0} & \mathbf{I}_3 \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{s}}_i \\ \dot{\boldsymbol{\omega}}_i \end{Bmatrix} + \begin{Bmatrix} \tilde{\boldsymbol{\omega}}_i \tilde{\boldsymbol{\omega}}_i \mathbf{g}_i \\ \mathbf{0} \end{Bmatrix} = \mathbf{D}_i \dot{\mathbf{Z}}_i + \mathbf{e}_i \tag{6}$$

Hence, by using the equations that have been seen before, the virtual power principle leads to the following expression:

$$\sum_{i=1}^{n} \mathbf{Y}_i^{*T} \left(\mathbf{M}_i \dot{\mathbf{Y}}_i - \mathbf{Q}_i \right) = \sum_{i=1}^{n} \mathbf{Z}_i^{*T} \mathbf{D}_i^T \left(\mathbf{M}_i \mathbf{D}_i \dot{\mathbf{Z}}_i + \mathbf{M}_i \mathbf{e}_i - \mathbf{Q}_i \right) = \\ = \sum_{i=1}^{n} \mathbf{Z}_i^{*T} \left(\bar{\mathbf{M}}_i \dot{\mathbf{Z}}_i - \bar{\mathbf{Q}}_i \right) = 0 \tag{7}$$

where the asterisk (*) indicates the virtual velocities. The matrices from the Eq. (7) are:

$$\mathbf{M}_i = \begin{bmatrix} m_i \mathbf{I}_3 & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_i \end{bmatrix}$$

$$\bar{\mathbf{M}}_i = \mathbf{D}_i^T \mathbf{M}_i \mathbf{D}_i = \begin{bmatrix} m_i \mathbf{I}_3 & -m_i \tilde{\mathbf{g}}_i \\ m_i \tilde{\mathbf{g}}_i & \mathbf{J}_i - m_i \tilde{\mathbf{g}}_i \tilde{\mathbf{g}}_i \end{bmatrix} \tag{8}$$

$$\bar{\mathbf{Q}}_i = \mathbf{D}_i^T \left(\mathbf{M}_i \mathbf{e}_i - \mathbf{Q}_i \right)$$

By implementing a matrix notation for the whole system, where $\bar{\mathbf{M}}$ is the inertia matrix, $\bar{\mathbf{Q}}$ is the force vector and $\dot{\mathbf{Z}}$ the acceleration vector:

$$\bar{\mathbf{M}} \equiv \operatorname{diag}\left(\bar{\mathbf{M}}_1, \bar{\mathbf{M}}_2, ..., \bar{\mathbf{M}}_n \right) \tag{9}$$

$$\bar{\mathbf{Q}}^T = \left[\bar{\mathbf{Q}}_1^T, \bar{\mathbf{Q}}_2^T, ..., \bar{\mathbf{Q}}_n^T \right] \tag{10}$$

$$\dot{\mathbf{Z}}^T = \left[\dot{\mathbf{Z}}_1^T, \dot{\mathbf{Z}}_2^T, .., \dot{\mathbf{Z}}_n^T \right] \tag{11}$$

the dynamic Eq. (7) take the form:

$$\mathbf{Z}^{*T} \left(\bar{\mathbf{M}} \dot{\mathbf{Z}} - \bar{\mathbf{Q}} \right) = 0 \tag{12}$$

In order to eliminate the dependent virtual velocities $\mathbf{Z}^*$ in Eq. (12), it is possible to introduce a velocity transformation between Cartesian and open-loop relative velocities, in the form:

$$\mathbf{Z} = \mathbf{R}_1 \dot{z}_1 + \mathbf{R}_2 \dot{z}_2 + ... + \mathbf{R}_n \dot{z}_n = \mathbf{R} \dot{\mathbf{z}} \tag{13}$$

The *j*-th column of matrix $\mathbf{R}$ means the velocities of all the bodies that are upwards of joint *j* in which a unit relative velocity is introduced by keeping null all the remaining relative velocities.

It is assumed that the input joint of a body has the same number than this body, and as suggested by Negrut, Serban and Porta [1], that open-loop bodies and joints have been numbered from the leaves to the root of the spanning tree in such a way that each body has a number lower than its parent. This numbering avoids the later filling-in in the Gauss elimination process. At this point it is very useful to introduce the path matrix $\mathbf{T}$ that defines the connectivity of the mechanism. Its rows correspond to bodies and its columns to joints. Element T_{ij} is 1 if body *i* is upwards of joint *j* and 0 otherwise. Then matrix $\mathbf{T}$ is upper triangular; its column *i* defines the bodies that are ahead of joint *i*, its row *j* define the joints that are behind body *j*, between the root body and it. Then, Eq. (13) can be written in the form:

$$\mathbf{R} = \mathbf{T} \operatorname{diag}\left(\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n \right) \equiv \mathbf{T} \mathbf{R}_d \tag{14}$$

where $\mathbf{b}_i$, according to Eq. (3), is the Cartesian velocity of all bodies ahead of joint i when $\dot{\mathbf{z}}_i = 1$ and $\dot{\mathbf{z}}_j = 0$, $j \neq i$. Remember that vector $\mathbf{b}_i$ represents the velocity of the point that coincides with the inertial frame origin, which is the same for all bodies.

When Eq. (14) is used to put Cartesian velocities and accelerations in terms of their relative counterparts, this is obtained:

$$\begin{aligned} \mathbf{Z} &= \mathbf{R}\dot{\mathbf{z}} = \mathbf{T}\mathbf{R}_d\dot{\mathbf{z}} \\ \dot{\mathbf{Z}} &= \mathbf{T}\mathbf{R}_d\ddot{\mathbf{z}} + \mathbf{T}\dot{\mathbf{R}}_d\dot{\mathbf{z}} \end{aligned} \tag{15}$$

By substituting Eq. (15) into Eq. (12) multiplying by $\mathbf{R}_d^T\mathbf{T}^T$ and taking into account that the relative open-loop virtual velocities are independent, a new set of dynamic equations is obtained:

$$\mathbf{R}_d^T\left(\mathbf{T}^T\bar{\mathbf{M}}\mathbf{T}\right)\mathbf{R}_d\ddot{\mathbf{z}} = \mathbf{R}_d^T\mathbf{T}^T\left(\bar{\mathbf{Q}} - \bar{\mathbf{M}}\mathbf{T}\dot{\mathbf{R}}_d\dot{\mathbf{z}}\right) \tag{16}$$

The symmetric matrix $\mathbf{T}^T\bar{\mathbf{M}}\mathbf{T}$, which is commonly expressed as $\mathbf{M}^\Sigma$,represents the accumulated inertia matrix of the whole system and where each component matrix $\mathbf{M}_i^\Sigma$ are the *composite inertia matrix* described by many authors. They represent the accumulation of the inertia matrices of all the elements that are ahead of joint *i*. It is important to keep in mind that $\mathbf{R}_d$ is a diagonal matrix and that $\mathbf{M}^\Sigma$ is a symmetric one with a well defined structure given by the path matrix $\mathbf{T}$, therefore this product can be evaluated in a very efficient way. Additionally, the advantage of numbering the bodies and joints from the leaves to the root becomes visible during the Gaussian elimination or the *LU* factorization of the LHS term of Eq. (16) because its symmetric pattern and thus, the pattern of zeros is not changed by the algorithm avoiding the filling and therefore, some additional arithmetic operations. It would be useful to see how this matrix looks like. So, for an example, see reference [9].

The Eq. (16) corresponds to the open-loop dynamics and can be written more compactly in the form:

$$\mathbf{M}\ddot{\mathbf{z}} = \mathbf{Q} \tag{17}$$

The vector $\mathbf{Q}$ on Eq. (17) represents all the external and velocity inertia accumulative forces as shown in the following expression:

$$\mathbf{Q} = \mathbf{R}_d^T\left(\mathbf{Q}^\Sigma + \mathbf{P}^\Sigma\right) \tag{18}$$

The equation above shows the vectors $\mathbf{Q}^\Sigma$ and $\mathbf{P}^\Sigma$ which represent the accumulated external forces and the accumulated velocity dependent inertia forces respectively. For a hypothetical open-loop system with *n* elements the expressions of $\mathbf{Q}^\Sigma$ and $\mathbf{P}^\Sigma$ are:

$$\mathbf{Q}^\Sigma \equiv \mathbf{T}^T\bar{\mathbf{Q}} = \begin{Bmatrix} \mathbf{Q}_1^\Sigma \\ \mathbf{Q}_2^\Sigma \\ \mathbf{Q}_3^\Sigma \\ \vdots \\ \mathbf{Q}_n^\Sigma \end{Bmatrix}, \quad \mathbf{P}^\Sigma \equiv -\mathbf{M}^\Sigma\dot{\mathbf{R}}_d\dot{\mathbf{z}} = -\mathbf{T}^T\bar{\mathbf{M}}\mathbf{T}\dot{\mathbf{R}}_d\dot{\mathbf{z}} = \begin{Bmatrix} \mathbf{P}_1^\Sigma \\ \mathbf{P}_2^\Sigma \\ \mathbf{P}_3^\Sigma \\ \vdots \\ \mathbf{P}_n^\Sigma \end{Bmatrix} \tag{19}$$

The Eq. (16) or (17) constitute a system of ODEs whose coefficient matrix and right-hand side vector can be computed recursively in a very efficient way.

2.1 Coordinate partitioning method

The dynamics of closed-loop multibody systems can be formulated by adding the constraint equations to the dynamic equations corresponding to the open-chain system obtained previously. In that sense, the differential equation of motion in a descriptor form takes the form:

$$\mathbf{M}(\mathbf{z})\ddot{\mathbf{z}} - \mathbf{\Phi}_{\mathbf{z}}^{T}\lambda = \mathbf{Q}(\mathbf{z}, \dot{\mathbf{z}}) \tag{20}$$

where $\mathbf{\Phi}_z$ is the Jacobian matrix of the kinematic constraints equations and λ the vector of Lagrange multipliers. The position, velocity and acceleration vectors in Eq. (20) must satisfy the corresponding constraint equation:

$$\mathbf{\Phi}(\mathbf{z})=\mathbf{0} \tag{21}$$

$$\dot{\mathbf{\Phi}}(\mathbf{z}) = \mathbf{\Phi}_{\mathbf{z}}\dot{\mathbf{z}} + \mathbf{\Phi}_{t} = \mathbf{0} \tag{22}$$

$$\ddot{\mathbf{\Phi}}(\mathbf{z}) = \mathbf{\Phi}_{\mathbf{z}}\ddot{\mathbf{z}} + \dot{\mathbf{\Phi}}_{\mathbf{z}}\dot{\mathbf{z}} + \dot{\mathbf{\Phi}}_{t} = \mathbf{0} \tag{23}$$

Eq. (20)-(23) constitute a system of index 3 DAEs. If only Eq. (20) and (23) are considered, the following index 1 DAE system – equivalent to an ODE system – is obtained:

$$\begin{bmatrix} \mathbf{M}(\mathbf{z}) & \mathbf{\Phi}_{\mathbf{z}}^{T} \\ \mathbf{\Phi}_{\mathbf{z}} & \mathbf{0} \end{bmatrix} \begin{Bmatrix} \ddot{\mathbf{z}} \\ \lambda \end{Bmatrix} = \begin{Bmatrix} \mathbf{Q}(\mathbf{z}, \dot{\mathbf{z}}) \\ -\dot{\mathbf{\Phi}}_{\mathbf{z}}\dot{\mathbf{z}} - \dot{\mathbf{\Phi}}_{t} \end{Bmatrix} \tag{24}$$

The matrix in this system of linear equation is known as the *augmented matrix* (Negrut, Serban and Porta [1]) or a matrix with *optimization structure* (Serban, Negrut, Porta and Haug [10], Von Schwerin [11]). The system of differential equations Eq. (24) suffers from the constraint stabilization problems. As only the acceleration constraint equations have been imposed, the positions and velocities provided by the integrator suffer from the "drift" phenomenon. On this paper two solutions to this problem have been used. The first one, which is introduced below, consists in eliminating the Lagrange multipliers from Eq. (24) by calculating a basis for the null space of the Jacobian matrix $\mathbf{\Phi}_{\mathbf{z}}$. This basis have been determined by using a coordinate partitioning method [3]-[12], that divides the coordinates $\mathbf{z}$ (and the columns of $\mathbf{\Phi}_{\mathbf{z}}$) into dependent and independent ones. The second approach to the stabilization problem, which is used on the next section of this paper, is based on the usage of mass-orthogonal projections of velocities and accelerations vectors (Bayo and Ledesma [5] and Cuadrado et al. [6]).

Following with the partitioning method, it is possible to select an independent subset of relative coordinates, in such a way that a set of ODEs will be obtained at the end. This is carried out by a new velocity transformation similar to the one introduced by Eq. (15). In this case, the transformation matrix $\mathbf{R}_{\mathbf{z}}$ will be obtained numerically as in the global formulation.

The closing-loop constraint equations are first formulated in Cartesian coordinates and then transformed into relative coordinates. In this paper two ways to set the closed-loop constraint equations will be considered. The first one is the cutting joint method, which is very common in the literature. The second method to open the loops consists in the elimination of the rods (slender bodies with two spherical joints and a negligible moment of inertia around the direction of the axis). This second procedure is very interesting in applications, as the car suspension system considered previously. Reference [9] shows how to take into account rod's inertia in an exact way. Here, only the kinematic constraints for closing the loops will be considered.

The kinematic constraint imposed by a rod is a constant distance condition that can be expressed as:

$$\left(\mathbf{r}_j - \mathbf{r}_k\right)^T \left(\mathbf{r}_j - \mathbf{r}_k\right) - l_{jk}^2 = 0 \tag{25}$$

On the other hand, the kinematic constraints corresponding to the removal of a Revolute can be defined with natural coordinates as (only Eq. (26) for a Spherical joint):

$$\mathbf{r}_j - \mathbf{r}_k = \mathbf{0} \quad \left(3 \text{ independent equations}\right) \tag{26}$$

$$\mathbf{u}_j - \mathbf{u}_k = \mathbf{0} \quad \left(\text{only 2 independent equations}\right) \tag{27}$$

The constraint equations (26)-(27) shall be expressed in terms of the relative coordinates $\mathbf{z}$. This is not difficult, because points $\mathbf{r}_j$ and $\mathbf{r}_k$, and unit vectors $\mathbf{u}_j$ and $\mathbf{u}_k$ can be expressed as functions of the relative coordinates of the joints in their respective branches of the open-chain system.

It is also necessary to compute the Jacobian matrix of constraints (25)-(27) with respect to relative coordinates $\mathbf{z}$. As the aforesaid constraints are expressed as a function of Cartesian coordinates, the chain derivative rule shall be used. For instance, for the constant distance constraint (25):

$$\mathbf{\Phi}_\mathbf{z} = \mathbf{\Phi}_{\mathbf{r}_j} \frac{\partial \mathbf{r}_j}{\partial \mathbf{z}} + \mathbf{\Phi}_{\mathbf{r}_k} \frac{\partial \mathbf{r}_k}{\partial \mathbf{z}} = \mathbf{\Phi}_{\mathbf{r}_j} \frac{\partial \dot{\mathbf{r}}_j}{\partial \dot{\mathbf{z}}} + \mathbf{\Phi}_{\mathbf{r}_k} \frac{\partial \dot{\mathbf{r}}_k}{\partial \dot{\mathbf{z}}} \tag{28}$$

The derivatives with respect to the coordinates $\mathbf{r}_j$ and $\mathbf{r}_k$ in Eq. (28) are easy to find:

$$\mathbf{\Phi}_{\mathbf{r}_j} = 2\left(\mathbf{r}_j^T - \mathbf{r}_k^T\right), \qquad \mathbf{\Phi}_{\mathbf{r}_k} = -2\left(\mathbf{r}_j^T - \mathbf{r}_k^T\right) \tag{29}$$

The derivatives of the position vectors $\mathbf{r}_j$ and $\mathbf{r}_k$ with respect to the relative coordinates $\mathbf{z}$ can be computed from the velocities of these points induced by unit relative velocities in the joints between the fixed body and bodies *j* and *k*, respectively. For instance, if the joint *i* is a revolute joint determined by a point $\mathbf{r}_i$ and a unit vector $\mathbf{u}_i$, located between the base body and point $\mathbf{r}_j$, the velocity of point *j* originated by a unit velocity in joint *i* can be set as:

$$\frac{\partial \dot{\mathbf{r}}_j}{\partial \dot{z}_i} = \mathbf{u}_i \times \left(\mathbf{r}_j - \mathbf{r}_i\right) = \tilde{\mathbf{u}}_i \left(\mathbf{r}_j - \mathbf{r}_i\right) \tag{30}$$

So it can be assumed that the closure of the loop constraint equations $\mathbf{\Phi}\left(\mathbf{z}\right)=\mathbf{0}$ and their Jacobian matrix $\mathbf{\Phi}_\mathbf{z}$ are known or easy to compute. Using the coordinate partitioning method based on Gaussian elimination with full pivoting, it is possible to arrive to the following partitioned velocity equation:

$$\begin{bmatrix} \mathbf{\Phi}_\mathbf{z}^d & \mathbf{\Phi}_\mathbf{z}^i \end{bmatrix} \begin{Bmatrix} \dot{\mathbf{z}}^d \\ \dot{\mathbf{z}}^i \end{Bmatrix} = \mathbf{0}, \qquad \dot{\mathbf{z}}^d = -\left(\mathbf{\Phi}_\mathbf{z}^d\right)^{-1} \mathbf{\Phi}_\mathbf{z}^i \dot{\mathbf{z}}^i \tag{31}$$

where it is assumed that matrix $\mathbf{\Phi}_\mathbf{z}^d$ is invertible. Eq. (31) allows an easy calculation of the transformation matrix $\mathbf{R}_\mathbf{z}$ that relates dependent and independent relative velocities:

$$\dot{\mathbf{z}} = \mathbf{R}_\mathbf{z} \dot{\mathbf{z}}^i, \quad \begin{Bmatrix} \dot{\mathbf{z}}^d \\ \dot{\mathbf{z}}^i \end{Bmatrix} = \begin{bmatrix} -\left(\mathbf{\Phi}_\mathbf{z}^d\right)^{-1} \mathbf{\Phi}_\mathbf{z}^i \\ \mathbf{I} \end{bmatrix} \dot{\mathbf{z}}^i, \quad \mathbf{R}_\mathbf{z} = \begin{bmatrix} -\left(\mathbf{\Phi}_\mathbf{z}^d\right)^{-1} \mathbf{\Phi}_\mathbf{z}^i \\ \mathbf{I} \end{bmatrix} \tag{32}$$

If this equation is differentiated with respect to time, the following equation is obtained:

$$\ddot{\mathbf{z}} = \mathbf{R}_{\mathbf{z}}\ddot{\mathbf{z}}^{i} + \dot{\mathbf{R}}_{\mathbf{z}}\dot{\mathbf{z}}^{i} \tag{33}$$

Introducing the velocity transformation defined by Eq. (32) and (33) in the equations of motion Eq. (16) allows obtaining the final set of ordinary differential equations in the independent relative coordinates:

$$\mathbf{R}_{\mathbf{z}}^{T}\mathbf{R}_{d}^{T}\mathbf{M}^{\Sigma}\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\ddot{\mathbf{z}}^{i} = \mathbf{R}_{\mathbf{z}}^{T}\mathbf{R}_{d}^{T}\mathbf{Q}^{\Sigma} - \mathbf{R}_{\mathbf{z}}^{T}\mathbf{R}_{d}^{T}\mathbf{M}^{\Sigma}\left(\dot{\mathbf{R}}_{d}\dot{\mathbf{z}} + \mathbf{R}_{d}\dot{\mathbf{R}}_{\mathbf{z}}\dot{\mathbf{z}}^{i}\right) \tag{34}$$

All the terms in this equation are known, except for the parenthesis with the derivatives of the transformation matrices. It is simpler to compute the two terms jointly. Considering Eq. (32), the parenthesis in Eq. (34) can be written as:

$$\dot{\mathbf{R}}_{d}\dot{\mathbf{z}} + \mathbf{R}_{d}\dot{\mathbf{R}}_{\mathbf{z}}\dot{\mathbf{z}}^{i} = \dot{\mathbf{R}}_{d}\mathbf{R}_{\mathbf{z}}\dot{\mathbf{z}}^{i} + \mathbf{R}_{d}\dot{\mathbf{R}}_{\mathbf{z}}\dot{\mathbf{z}}^{i} = \left(\dot{\mathbf{R}}_{d}\mathbf{R}_{\mathbf{z}} + \mathbf{R}_{d}\dot{\mathbf{R}}_{\mathbf{z}}\right)\dot{\mathbf{z}}^{i} = \frac{d\left(\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\right)}{dt}\dot{\mathbf{z}}^{i} \tag{35}$$

This derivative can be computed from the product of velocity transformations that relates Cartesian and independent relative velocities:

$$\mathbf{Z} = \mathbf{R}\dot{\mathbf{z}} = \mathbf{T}\mathbf{R}_{d}\dot{\mathbf{z}} = \mathbf{T}\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\dot{\mathbf{z}}^{i} \tag{36}$$

Taking the time derivative of this equation:

$$\dot{\mathbf{Z}} = \mathbf{T}\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\ddot{\mathbf{z}}^{i} + \mathbf{T}\frac{d\left(\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\right)}{dt}\dot{\mathbf{z}}^{i} \tag{37}$$

In this equation, the product of the path matrix $\mathbf{T}$ times the searched derivative, can be computed as the Cartesian accelerations $\dot{\mathbf{Z}}$ that would be produced by the true velocities $\dot{\mathbf{z}}$ and null relative independent accelerations ($\mathbf{z}^{i} = \mathbf{0}$). The dynamic equations Eq. (34) can be written in the form:

$$\mathbf{R}_{\mathbf{z}}^{T}\mathbf{R}_{d}^{T}\mathbf{M}^{\Sigma}\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\ddot{\mathbf{z}}^{i} = \mathbf{R}_{\mathbf{z}}^{T}\mathbf{R}_{d}^{T}\left(\mathbf{Q}^{\Sigma} - \mathbf{T}^{T}\bar{\mathbf{M}}\frac{d\left(\mathbf{T}\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}\right)}{dt}\dot{\mathbf{z}}^{i}\right) \tag{38}$$

So, a way to compute the terms in the ODEs set Eq. (38) has been completed. Two velocity transformations have been introduced. The first one, from Cartesian to open-chain relative velocities, is applied directly and leads to an accumulation of forces and inertias. The second one is applied in a fully numerical way to a (usually) smaller system.

The details about the inertia forces of the rods that have been removed to open the closed loops will not be given here. It is enough to point out that a rod introduces coupling terms between the inertia forces of the two tree branches connected by it. The topological information to compute these coupling terms is also contained in the path matrix.

2.2 Penalty method

The second method shown on this paper consists in a penalty formulation based on a slightly variant of the augmented Lagrangian originally proposed by [5] where relative coordinates are used instead of the Cartesian ones and only the position constraints have been directly introduced within the differential equations of motion. The open-loop kinematic and dynamic analysis remain the same with respect to those that have been used before. Hence, starting from Eq. (17), the closed-loop equations are introduced in the differential equation of

motion by means of a penalization coefficient that forces the fulfillment of the position constraints. Then, the differential equation of motion for a closed-loop system is:

$$\mathbf{M}\ddot{\mathbf{z}} + \boldsymbol{\Phi}_{\mathbf{z}}^{T}\alpha\boldsymbol{\Phi} = \mathbf{Q} \tag{39}$$

where α is the penalization coefficient.

The Eq. (39) represents a system of *m* differential equations with *m* unknowns. For the integration of these equations an alternative form of the trapezoidal rule has been used, which consists in expressing the relative velocity and acceleration for a particular instant of time $n+1$ as function of the relative position vector $\mathbf{z}_n$ on a time n :

$$\dot{\mathbf{z}}_{n+1} = \frac{2}{h}\mathbf{z}_{n+1} + \hat{\dot{\mathbf{z}}}_n; \qquad \hat{\dot{\mathbf{z}}}_n = -\left(\frac{2}{h}\mathbf{z}_n + \dot{\mathbf{z}}_n\right) \tag{40}$$

$$\ddot{\mathbf{z}}_{n+1} = \frac{4}{h^2}\mathbf{z}_{n+1} + \hat{\ddot{\mathbf{z}}}_n; \qquad \hat{\ddot{\mathbf{z}}}_n = -\left(\frac{4}{h^2}\mathbf{z}_n + \frac{4}{h}\dot{\mathbf{z}}_n + \ddot{\mathbf{z}}_n\right) \tag{41}$$

where $\dot{\mathbf{z}}_{n+1}$ is the relative velocity vector, $\ddot{\mathbf{z}}_{n+1}$ is the relative acceleration vector and h is the integration step. By adding Eq. (41) to Eq. (39) the following non-linear system equations are obtained:

$$\mathbf{f}(\mathbf{z}_{n+1}) = \mathbf{M}_{n+1}\mathbf{z}_{n+1} + \frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}_{n+1}}^{T}\alpha\boldsymbol{\Phi}_{n+1} - \frac{h^2}{4}\mathbf{Q}_{n+1} + \frac{h^2}{4}\mathbf{M}_{n+1}\hat{\ddot{\mathbf{z}}}_n = 0 \tag{42}$$

To solve Eq. (42) where the unknown is the vector $\mathbf{z}_{n+1}$, it is customary to use the Newton-Rahpson method, which has a quadratic convergence in the neighborhood of the solution and usually does not cause serious problems when starting with a good initial approximation. Using this iterative method implies the evaluation of a tangent matrix as indicated in the following expressions:

$$\left[\frac{\partial\mathbf{f}(\mathbf{z})}{\partial\mathbf{z}}\right]_{n+1}^{k} \Delta\mathbf{z}_{(n+1)}^{k} = -\left[\mathbf{f}(\mathbf{z})\right]_{n+1}^{k} \tag{43}$$

being k the iteration and $n+1$ the time-step. Then, the unknown vector $\mathbf{z}_{n+1}$ would be:

$$\mathbf{z}_{(n+1)}^{(k+1)} = \mathbf{z}_{(n+1)}^{k} + \Delta\mathbf{z}_{(n+1)}^{k} \tag{44}$$

The solution of Eq. (44) assumes the previous evaluation of an approximate tangent matrix from Eq. (43) that has the following form:

$$\left[\frac{\partial\mathbf{f}(\mathbf{z})}{\partial\mathbf{z}}\right]_{n}^{k} = \left(\mathbf{M}(\mathbf{z}) + \frac{h}{2}\mathbf{C} + \frac{h^2}{4}\left(\boldsymbol{\Phi}_{\mathbf{z}}^{T}\alpha\boldsymbol{\Phi}_{\mathbf{z}} + \mathbf{K}\right)\right)_{n}^{k} \tag{45}$$

where:

$$\mathbf{K} = -\frac{\partial\mathbf{Q}}{\partial\mathbf{z}}, \qquad \mathbf{C} = -\frac{\partial\mathbf{Q}}{\partial\dot{\mathbf{z}}} \tag{46}$$

This matrix shows ill-conditioning when the time step h is very small. According to Bayo and Ledesma the numerical condition of the tangent matrix is $O(h)^{-3}$, which sets a lower limit to the practical values of step-sizes.

Then, at time *n* and iteration *k*, the Eq. (42) can be expressed in a more compact form:

$$\mathbf{f(z)}_n^k = \frac{h^2}{4}\left(\mathbf{M}\ddot{\mathbf{z}} + \frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^T\alpha\boldsymbol{\Phi} - \frac{h^2}{4}\mathbf{Q}(\mathbf{z},\dot{\mathbf{z}})\right)_n^k \tag{47}$$

During the time integration process the numerical integration yields a set of velocities $\dot{\mathbf{z}}^*$ and accelerations $\ddot{\mathbf{z}}^*$ that do not satisfy the constraint conditions $\dot{\boldsymbol{\Phi}} = \mathbf{0}$ and $\ddot{\boldsymbol{\Phi}} = \mathbf{0}$ respectively, because both sets of vectors have been obtained numerically from the integrator and not by a derivation process of the positions. To overcome the stability problems that appear during the integration process, which are related to this lack of fulfillment on the constraints, a mass-orthogonal projection in velocities and accelerations have been used. Based on the projection methods proposed in (Bayo and Ledesma [5]) and the optimization introduced on that (Cuadrado et al. [6]), by using the same tangent matrix in the projection equations, the following expressions for the velocities are obtained:

$$\left(\mathbf{P} + \frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^T\alpha\boldsymbol{\Phi}_{\mathbf{z}}\right)\dot{\mathbf{z}} = \mathbf{P}\dot{\mathbf{z}}^* - \frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^T\alpha\boldsymbol{\Phi}_t \tag{48}$$

where the matrix $\mathbf{P}$ is the weighted matrix proposed on [6] and has the form:

$$\mathbf{P} = \mathbf{M}(\mathbf{z}) + \frac{h}{2}\mathbf{C} + \frac{h^2}{4}\mathbf{K} \tag{49}$$

The expression for the accelerations is:

$$\left(\mathbf{P} + \frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^T\alpha\boldsymbol{\Phi}_{\mathbf{z}}\right)\ddot{\mathbf{z}} = \mathbf{P}\ddot{\mathbf{z}}^* - \frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^T\alpha\dot{\boldsymbol{\Phi}}_{\mathbf{z}}\dot{\mathbf{z}} - \frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^T\alpha\boldsymbol{\Phi}_t \tag{50}$$

3 PERFORMANCE IMPROVEMENTS

This section shows the improvements that have been introduced in both of the two formulations developed above. The first implementation consists of the sparse techniques usage for the solution of a redundant but compatible linear system of equation. This formulation has been integrated with the 4th order explicit method of Runge-Kutta. The second improvement and most important contribution of this paper have consisted of a parallel implementation within the penalty method whose differential equation of motion has been solved with the implicit method of the Trapezoidal Rule. Finally, a very simple consideration has been used during the numerical forces derivatives that have allowed to achieve a significant reduction on the execution times.

3.1 Sparse implementation of the double velocity transformation

The coordinate partitioning method shown before is an efficient way to avoid the stability problems that arise during the integration of motion differential equations. However, in order to obtain the $\mathbf{R}_{\mathbf{z}}$ matrix of Eq. (32), it is necessary to carry out the calculation of the inverse of the partitioned Jacobian matrix. The solution to this problem has a prohibitive computational cost in large and complex multibody systems, due to the need of solving a *redundant* but *compatible* linear system of equations. This can be done by means of a least squares approximation, but this computational effort is completely out of bond for efficient applications.

There are very efficient dense matrix implementations that can be used to increase the performance of our numerical computation. In a previous work [13], we have shown how to introduce in our formulation the BLAS functions provided for MKL from Intel®. The main application of this library was in the most expensive numerical algorithms of this implementation, which are the computation of the matrix $\mathbf{R}_z$ from Eq. (32) and the product of matrices to arrive to Eq. (34) from Eq. (16).

However, when we are looking for increasing the efficiency, another alternative can be taken into account, which consists in taking advantage of the sparse pattern that some matrices have. Although saving arithmetical operations and computational costs by considering the non-zero structure of a matrix is not a new subject, in MBS dynamics, the sparse matrix techniques have mainly had success in global formulations or in topological ones with flexible bodies. Despite this, we will show that it is possible to exploit the sparsity too in semi-recursive formulation with relative coordinates when we are modeling large multibody systems. Actually, one of the goals of this paper consists in introducing the usage of the sparse function MA48 provided by Harwell [14] to obtain the partitioning coordinate matrix $\mathbf{R}_z$ in a topological semi-recursive formulation.

Regarding this, an interesting study and benchmark of various sparse functions can be seen on (González et al. [15]), where the most efficient sparse library have been KLU ("Clark Kent" LU), a solver specially designed for circuit simulation matrices [16]. The Jacobian matrix $\mathbf{\Phi}_z$ for large and complex MBS often introduces the appearance of *redundant* but *compatible* system of equations. This fact prevents from using the KLU subroutine because it only operates on square matrices [17]. Nevertheless, a subroutine able to deal with this kind of problems should be used.

The MA48, is a collection of Fortran 77 subroutines for the direct solution of a sparse unsymmetric set of linear equations $\mathbf{Ax} = \mathbf{b}$, where the matrix $\mathbf{A}$ could be square, rectangular or rank-deficient and $\mathbf{x}$ and $\mathbf{b}$ are vectors. The package basically consists of four subroutines, MA48I/ID, MA48A/AD, MA48B/BD, MA48C/CD. The first one, set default values for control parameters and would be called once prior to any calls MA48A/AD, MA48B/BD, or MA48C/CD. Then, MA48A/AD prepares data structure for factorization, MA48B/BD factorizes the matrix $\mathbf{A}$ and MA48C/CD uses the factors produced by MA48B/BD to solve $\mathbf{Ax} = \mathbf{b}$ or $\mathbf{A}^T\mathbf{x} = \mathbf{b}$.

Observing the Eq. (32), and calling $\mathbf{R}_z^d$ to those terms of the matrix $\mathbf{R}_z$ related with obtaining $\dot{\mathbf{z}}_d$, the system to be solved in our formulation is:

$$\mathbf{\Phi}_\mathbf{z}^d \mathbf{R}_\mathbf{z}^d = \mathbf{\Phi}_\mathbf{z}^i \tag{51}$$

The MA48C/CD does not solve systems with multiple RHS, therefore, the solution of Eq. (51) is reached by using MA48C/CD within a loop where each time a column of matrix $\mathbf{R}_z^d$ is obtained. The results of this implementation are shown on the numerical Results section. There we will be able to see that for large MBS, sparse techniques instead of dense ones could be the right option.

3.2 Parallel implementation (Implicit method)

One of the most expensive computational operations of the explained formulation, especially working on simulations of large and complex multibody systems, is the computation of the tangent matrix, specifically the calculation of forces derivatives with respect to relative position and velocity vectors (70-58% of the elapsed time). Instead, it is not necessary true in all examples; see [18], for the sake of accuracy, generality and robustness, its computation will not be approximate or neglected in our implementation. The matrices $\mathbf{K}$ and $\mathbf{C}$ from Eq.

(46) represent the influence that a variation on each element of vectors $\mathbf{z}$ and $\dot{\mathbf{z}}$ have on the forces $\mathbf{Q}$ of the mechanical system. At this point it is worth to mention that the accuracy on the evaluation of these forces must not be taken for granted, specially on those cases where the external forces acting on the system are not constrained to spring and dampers only, but more general and complex forces are involved on the dynamic behavior of the MBS. Modeling the tyres behavior by means of Pacejka's magic formula, a very important practical situation could be, for example, the evaluation of forces between the tyres and the soil when a vehicle is passing through an obstacle like a velocity reducer (speed bump) or a pothole. That kind of situation, when very accurate realistic multibody system models are used, prevents from neglecting or using huge approximation during the forces derivatives evaluation.

Another factor to take into account besides the accuracy is the generality that the implementation of a formulation should have attending to achieve a very robust program able to deal with the most assorted requirements. This fact could demand the appearance of external forces from different nature, what constitutes an important drawback for these general purpose applications. To work out the generalization problems that are likely to appear when a wide range of force types will be included in the dynamic analysis, a very general way to evaluate the forces derivatives must be used. Therefore, for the sake of generality, a good approach to this is to evaluate the derivatives numerically.

Despite the numerical evaluation of Eq. (46) have the drawback of being a very expensive computation, on the other hand it has two extremely important advantages. The first one, as mentioned before, is its generality; and the second one is its concurrent computation capability, in that sense that each column or groups of them could be simultaneously evaluated without interfering with each other.

The main goal of this paper is to propose a very efficient numerical evaluation of the forces derivatives that are focused on obtaining a high performance implementation, exploiting its parallelization properties by doing tasks in a concurrent way.

The form of achieving parallelism could be described in four fundamental decompositions, data parallelism, task parallelism, pipelining (task and data parallelism together) and mixed solutions. A pure data parallelism problem is a completely *thread-safe* operation called by programmers as *embarrassingly parallel*. Thinking in task, the numerical evaluation of matrices $\mathbf{K}$ and $\mathbf{C}$ in parallel is a very straightforward operation in such a way that each column or groups of them for both matrices can be concurrently obtained. Then, the basic *parallel_for* algorithm offered by Threading Building Blocks could be applied. This algorithm break the iteration space into chunks and run each one simultaneously on separated threads. Nevertheless, all the work required to perform a column as a unique entity and as a part of a whole (a matrix), should be carefully thought. Regarding this, supposing an instant of time *n* and that the column *i* of matrix $\mathbf{K}$ and $\mathbf{C}$ are being performed according with the numerical form:

$$\frac{\partial \mathbf{Q}^n}{\partial \mathbf{z}_i^n} \approx \frac{\mathbf{Q}^n(\mathbf{z}_i^n, \dot{\mathbf{z}}^n) - \mathbf{Q}^n(\mathbf{z}^n, \dot{\mathbf{z}}^n)}{\Delta z} \tag{52}$$

$$\frac{\partial \mathbf{Q}^n}{\partial \dot{\mathbf{z}}_i^n} \approx \frac{\mathbf{Q}^n(\mathbf{z}^n, \dot{\mathbf{z}}_i^n) - \mathbf{Q}^n(\mathbf{z}^n, \dot{\mathbf{z}}^n)}{\Delta z} \tag{53}$$

where $\mathbf{z}_i^n$ and $\dot{\mathbf{z}}_i^n$ are the relative position and velocity vectors at time *n* when a variation Δz on element *i* of each one is being performed. The Eq. (52) and (53) involve the evaluation of $\mathbf{Q}$ from Eq. (18) for each column of matrices $\mathbf{K}$ and $\mathbf{C}$. In addition to the high computational cost that should be invested on that, when looking for parallel patterns to apply, other basic important factors take place at this stage. Although Threading Building Blocks offers a

high level parallelism environment that relies on generic programming, allowing a useful abstraction of threads by concentrating in tasks, we need to know where our application could present problems like *not* being *thread-safe*, *mutual exclusion* and *locks*, and *correctness* (*deadlock* and *race conditions*). These considerations will be easily explained without getting into tedious details about the algorithm code implementation, we only need to see the expressions that are evaluated to obtain the vector $\mathbf{Q}$, which are explicitly shown for all the equations developed before than Eq. (18). Additionally, it is important to note that each column of matrices $\mathbf{K}$ and $\mathbf{C}$ must be evaluated using the same kinematic and dynamic information about all the bodies of the system, except for the variation on vectors $\mathbf{z}$ and $\dot{\mathbf{z}}$ as indicate Eq. (52) and (53). For that reason, before applying a *parallel_for* we need to see that during each column evaluation some kinematic and dynamic data (Cartesian velocities and acceleration, and forces) had changed their values and consequently, they need to be restored; on the other hand, the chunks must be supplied with its own block of data, what implies making a copy for each one. In that way, a simple form of safety and scalable parallelism in a loop has been described, because no threads and synchronization considerations were necessary, only a secure management of data have been demanded. Because not as simple parallel pattern as data parallelism is applied but this application consists in the simplest loop iteration, we can call this implementation like a *quasi embarrassingly parallel*.

Once the implementation features have been exposed, let see what we are really concerned about, the *scalability* and *speedup* able to be achieved by implementing parallelization on this algorithm.

The Fig. (3) shows two simplified flow diagrams about the algorithm that integrates the differential equations of motion with the trapezoidal rule as implicit integrator, the one on the left is a sequential program while the other represents its parallel version in a *quad-core* processor. Task 1 is the initial acceleration computation. Afterwards, a *for-loop* simulating the time advance starts. Task 2 is the computation of matrices $\mathbf{K}$ and $\mathbf{C}$, task 3 is the evaluation of the product $\mathbf{\Phi}_{\mathbf{z}}^{T}\alpha\mathbf{\Phi}_{\mathbf{z}}$ and task 4 basically involves the evaluation of the tangent matrix Eq. (45) together with vectors $\mathbf{z}$, $\dot{\mathbf{z}}^{*}$ and $\ddot{\mathbf{z}}^{*}$. Task 2, 3 and 4 are evaluated within the Newton-Raphson iteration. Finally, when the error in the constraints is sufficiently small, the velocity and acceleration projections are performed in task 5.

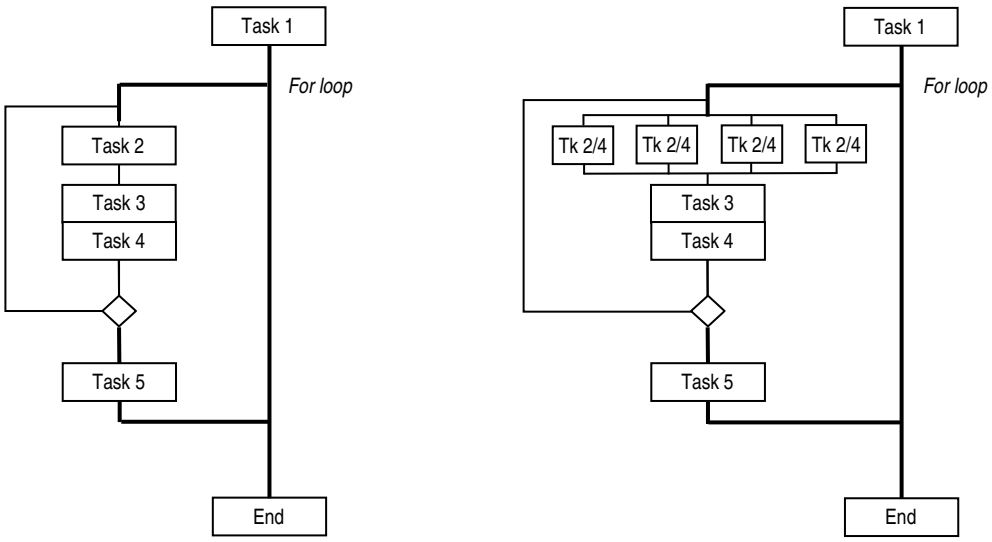

Figure 3: Adding parallelism within the iterative algorithm

The forces derivatives, represented in task 2 are the heaviest workload of the algorithm. Fortunately this operation is very suitable for parallelism; therefore this workload can be simultaneously distributed between all the processor cores. On the other hand, adding more processor cores does not imply that the total time of this step would decrease in a direct proportion. At this point we say that the program does not *scale*. This is because the overhead of distributing and synchronization begins to dominate the problem.

According to *Amdahl's Law*, it should be noted that the quantity of parallelism we can obtain from our application is limited by the serial portion of the program. In that sense, it is a huge advantage of this formulation that parallelizable portion of the code represents between the 70 and 85% of the elapsed time. Additionally, according to *Gustafson's Law*, it is possible to get more scalability by increasing the workload that the parallel portion of the code does. In such way that increases the problem size, the serial fraction decreases and according to *Amdahl's Law*, the scalability improves. Maybe there lies the main drawback of this method. In that sense that we can increase the workload by modeling larger and more complex multibody systems, but, if we are running a particular model, the only way to increase the workload that parallel portion of code does, is by means of using a smaller time step integration. This effectively would increase the step total time due to the growth of the number of evaluations not of the workload required for each step, therefore, the problem does not *scale*.

Finally, two more uses of Threading Building Blocks have been included. The first one is the usage of scalable memory allocation by replacing all the C++ memory routines like malloc, new and delete by the templates provided by the library. This basically avoid problems like *false sharing* [19], where for example two threads tend to access each one to different elements from two arrays that are located on the same *cache line*. The other Threading Building Blocks usage has been introduced to perform the product $\mathbf{\Phi}_{\mathbf{z}}^{T}\alpha\mathbf{\Phi}_{\mathbf{z}}$, that is the most expensive after the forces derivatives computation. Here, the contribution of the library has been the usage of a two- dimensional iteration space by means of the *blocked-range2d<T>* template with a parallel-for routine. This solution yields more parallelism and better cache behavior making a loop run faster than the sequential equivalent even on a single processor.

All the improvements that the parallelization has introduced are shown on the numerical results part.

3.3 Topology based improvement on forces derivatives

On this section, a simple but efficient improvement has been introduced taking into account the connectivity that exists between different branches from the spanning tree of the system. The example of the semi-trailer truck that is shown on the Results section consists of a tractor joined to the semi-trailer by means of a spherical joint. These are represented by means of two branches independent between them. This fact is illustrated if we observe the connectivity pattern of the matrix **K** as shown in Fig. (4).

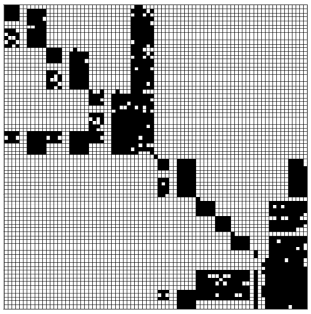

Figure 4: Connectivity Pattern of matrix **K.**

As we can see in this figure, the block from the upper-left corner and lower-right indicates that it is possible to evaluate two derivatives by covering the tree only once. Because of the same consideration can be done performing matrix $\mathbf{C}$, a huge reduction of time can be achieved by performing the derivatives on this way. Finally it is worth to mention that not all the multibody systems show a connectivity patter that allows to perform the derivatives as mentioned above, but it should always be taken into account that a new model would be defined.

4 RESULTS

In order to see the improvements, a large and complex example has been analyzed: a 40-dof 5-axle semi-trailer truck. This model is a real-life vehicle, with realistic suspension geometry and forces. Two simulations have been done, which consist of a 5s multiple slalom test along a flat road. For the sparse implementation, a 4th-order Runge-Kutta integrator with a 10^{-3}s time-step has been used, with 20,000 evaluations of the state vector derivative. On the other hand, the parallel implementation has been run using an implicit integrator as the trapezoidal rule with two time-step executions 5^{-3}s and 10^{-2}s.

The example is a multibody model of a complete truck with tractor and semi-trailer. It has 40 dof, 5 axles, 81 bodies (of which 34 are massless auxiliary), 89 joints and a driven coordinate for the steering system. All axles have two wheels. The tire-ground contact forces are modeled by Pacejka's magic formula. The semi-trailer suspension and the tractor rear suspension are made up of air springs and dampers, while the tractor front suspension is made up of leaf springs and dampers. The joint between the semi-trailer and the tractor has been considered as a spherical joint.

The semi-trailer truck has 282 Cartesian coordinates, 80 open-chain relative coordinates, 51 constraint equations (40 independent) and 40 degrees of freedom. The Fig. (5) shows two views of the truck MBS model.

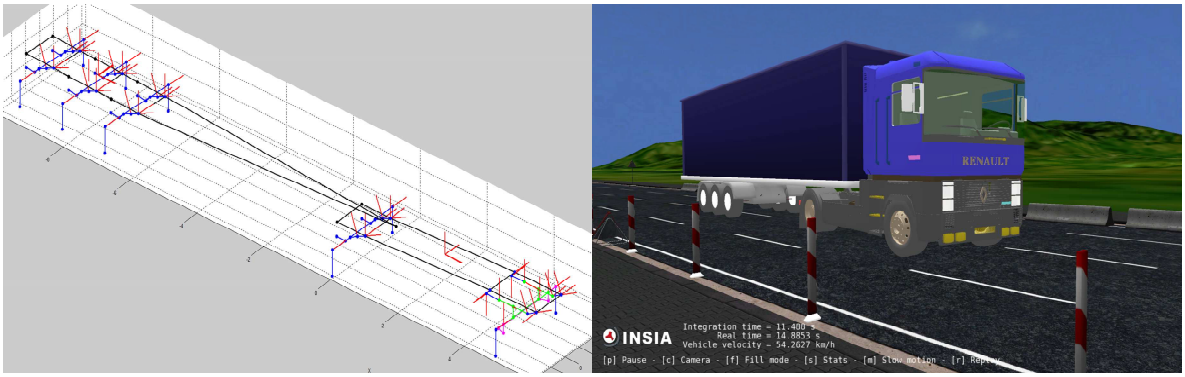

Figure 5: Schematic and CAD MBS model – Semi-trailer truck

Table 1 shows the CPU time spent for 20,000 evaluations of each algorithm step, considering three different implementations: standard C/C++, BLAS (MKL) and sparse (MA48). Steps 3 and 8 are the most expensive ones. Both BLAS/sparse routines improve the results considerably, being the latter the best one.

Algorithm Step		Semi-trailer truck		
		C/C++	BLAS	SPARSE
1	$\mathbf{b}_j,\ \mathbf{A}_i,\ \mathbf{J}_i,\ \bar{\mathbf{M}}_i,\ \mathbf{M}_i^{\Sigma}=\bar{\mathbf{M}}_i+\sum_{j<i}\mathbf{M}_i^{\Sigma}$	0.675	0.665	0.658
2	$\left[\boldsymbol{\Phi}_{\mathbf{z}}^{d}\quad\boldsymbol{\Phi}_{\mathbf{z}}^{i}\right],\quad[\mathbf{L},\mathbf{U}]=\mathrm{lu}\left(\boldsymbol{\Phi}_{\mathbf{z}}^{d}\right)$	0.425	0.927	0.721
3	$\mathbf{R}_{\mathbf{z}}=\begin{bmatrix}-\left(\boldsymbol{\Phi}_{\mathbf{z}}^{d}\right)^{-1}\boldsymbol{\Phi}_{\mathbf{z}}^{i}\\ \mathbf{I}\end{bmatrix}$	4.032	0.655	0.616
4	$\dot{\mathbf{z}}^{d}=-\left(\boldsymbol{\Phi}_{\mathbf{z}}^{d}\right)^{-1}\boldsymbol{\Phi}_{\mathbf{z}}^{i}\dot{\mathbf{z}}^{i},\quad$ recVel: $\mathbf{Z}_i,\ \mathbf{d}_i,\ \mathbf{d}_i^{\Sigma}$	0.289	0.361	0.195
5	$\mathbf{Q}_i,\ \bar{\mathbf{Q}}_i,\ \boldsymbol{\tau}_i$	0.273	0.275	0.275
6	$\mathbf{c}=-\dot{\boldsymbol{\Phi}}_{\mathbf{z}}\dot{\mathbf{z}},\quad\dot{\mathbf{R}}_{\mathbf{z}}\dot{\mathbf{z}}=-\boldsymbol{\Phi}_{\mathbf{z}}^{d}\setminus\dot{\boldsymbol{\Phi}}_{\mathbf{z}}\dot{\mathbf{z}}$	0.258	0.198	0.180
7	recAccel: $\mathbf{Z}_i,\ \bar{\mathbf{Q}},\ \mathbf{Q}^{\Sigma}$	0.193	0.194	0.193
8	$\mathbf{M}=\mathbf{R}_{\mathbf{z}}^{T}\mathbf{R}_{d}^{T}\mathbf{M}^{\Sigma}\mathbf{R}_{d}\mathbf{R}_{\mathbf{z}}$ $\mathbf{b}=\mathbf{R}_{\mathbf{z}}^{T}\left(\mathbf{R}_{d}^{T}\mathbf{Q}^{\Sigma}-\mathbf{R}_{d}^{T}\mathbf{M}^{\Sigma}\left(\dot{\mathbf{R}}_{d}\dot{\mathbf{z}}+\mathbf{R}_{d}\dot{\mathbf{R}}_{\mathbf{z}}\dot{\mathbf{z}}^{i}\right)\right)$	3.037	2.054	2.044
9	$\ddot{\mathbf{z}}_d=\mathbf{M}_{d,d}\setminus\left(\mathbf{M}_{d,g}\ddot{\mathbf{z}}_g\right),\quad\mathbf{b}_g=\mathbf{M}_{g,d}\ddot{\mathbf{z}}_d+\mathbf{M}_{g,g}\ddot{\mathbf{z}}_g$	0.247	0.258	0.255
Elapsed time [s]		**9.428**	**5.588**	**5.137**

Table 1: CPU times for 5s simulation – Explicit method

Table 2 shows the results that the parallelization, by means of the Threading Building Blocks usage, had in the semi-trailer truck model. The columns for each time-step means, the C++ serial execution (Serial), the Threading Building Blocks execution (TBB) and the topology based improvement with parallelization too (TBI). Here, we can see the *speedup* achieved and how this application does not *scale* when the workload is increased by using a small time-step size.

Algorithm Step		Semi-trailer truck					
		Time-step 5^{-3}s			Time-step 10^{-2}s		
		Serial	TBB	TBI	Serial	TBB	TBI
1	$\ddot{\mathbf{z}}_0$	0.001	0.001	0.001	0.002	0.001	0.001
2	$\mathbf{M(z)}\quad\boldsymbol{\Phi}\mathbf{(z)}\quad\boldsymbol{\Phi}_{\mathbf{z}}\mathbf{(z)}\quad\mathbf{Q(z,\dot{z})}\quad\mathbf{f(z)}$	0.192	0.224	0.221	0.137	0.154	0.156
3	$\mathbf{K}=-\left(\partial\mathbf{Q}/\partial\mathbf{z}\right)\quad\mathbf{C}=-\left(\partial\mathbf{Q}/\partial\dot{\mathbf{z}}\right)$	9.103	2.772	1.523	4.827	1.475	0.794
4	$\boldsymbol{\Phi}_{\mathbf{z}}^{T}\alpha\boldsymbol{\Phi}_{\mathbf{z}}$	1.814	0.232	0.221	1.309	0.167	0.168
5	$\left[\partial\mathbf{f(z)}/\partial\mathbf{z}\right]_{n+1}^{k}$	0.116	0.126	0.119	0.084	0.088	0.088
6	$\Delta\mathbf{z}_{(n+1)}^{k}=-\left[\partial\mathbf{f(z)}/\partial\mathbf{z}\right]_{n+1}^{k}\setminus\left[\mathbf{f(z)}\right]_{n+1}^{k}$	0.707	0.714	0.675	0.508	0.512	0.506
7	**Convergence evaluation**	0.002	0.002	0.003	0.001	0.001	0.001
8	$\dot{\mathbf{z}}=\left(\mathbf{P}+\frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^{T}\alpha\boldsymbol{\Phi}_{\mathbf{z}}\right)^{-1}\mathbf{P}\dot{\mathbf{z}}^{*}$	0.088	0.088	0.088	0.045	0.044	0.044
9	$\ddot{\mathbf{z}}=\left(\mathbf{P}+\frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^{T}\alpha\boldsymbol{\Phi}_{\mathbf{z}}\right)^{-1}\left(\mathbf{P}\dot{\mathbf{z}}^{*}-\frac{h^2}{4}\boldsymbol{\Phi}_{\mathbf{z}}^{T}\alpha\dot{\boldsymbol{\Phi}}_{\mathbf{z}}\dot{\mathbf{z}}\right)$	0.028	0.029	0.029	0.014	0.014	0.014
	Elapsed time [s]	**12.051**	**4.189**	**2.881**	**6.926**	**2.457**	**1.771**

Table 2: CPU times for 5s simulation – Implicit method

5 CONCLUSIONS

We can divide the contribution of this paper into two parts, the first one has been the introduction of sparse techniques for the direct solution of a redundant but compatible linear sys-

tem of equations and the second and most important goal of this paper has been the successful application of parallelization in a commercial multi core processor. Both methods have been applied on large, complex and very realistic multibody systems.

The coordinate partitioning method has outstanding stabilization problems but being the differential equation of motion solved with an explicit integrator like a 4^{th} order Runge-Kutta method the usage of very small time step integration demanding a big number of evaluations per unit of time has been required. The efficiency of this method has been significantly improved by including sparse techniques but this still prevents from obtaining real time simulations on a huge multibody system as the semi-trailer truck example.

To increase the time step integration size an implicit method has been used but once achieved this, the necessity of accuracy and generality have demanded the introduction of an expensive numerical forces differentiation. On the other hand, the numerical differentiation process is very suitable for parallelism, then an efficient CPU usage consisting in keeping busy all the processor cores, has given us a high performance algorithm reaching real time simulation by scaling the application with a *speedup* of 3.2 in a commercial quad-core processor. Finally, according to *Amdahl* and *Gustafson's Laws*, it is possible to scale with large multibody systems but it will not be the case if, for accuracy sake, we increase the workload of the parallel fraction of code by decreasing the time step size.

ACKNOWLEDGEMENTS

The authors acknowledge the support of the Ministry of Science and Innovation of Spain under Research Project TRA2009-14513-C02-01 (OPTIVIRTEST).

REFERENCES

[1] D. Negrut, R. Serban, and F. A. Potra. A Topology Based Approach for Ex-ploiting Sparsity in Multibody Dynamics. Joint Formulation. *Mechanics of Structures and Machines*, **25** (2), 1997.

[2] J. I. Rodriguez, J. M. Jiménez, F. Funes, and J. García de Jalón. Recursive and Residual Algorithms for the Efficient Numerical Integration of Multi-body Systems. *Multibody System Dynamics*, **11**, 295-320, 2004.

[3] R. Wehage and E. J. Haug. Generalized Coordinate Partitioning for Dimension Reduction in Analysis of Constrained Mechanical Systems. *ASME Journal of Me-chanical Design*, **104**, 247-255, 1982.

[4] E. Bayo, J. García de Jalón and M. A. Serna. A Modified Lagrangian Formulation for the Dynamic Analysis of Constrained Mechanical Systems. *Computer Methods In Applied Mechanics and Engineering*, **71**, 183–195, 1988.

[5] E. Bayo and R. Ledesma. "Augmented Lagrangian and Mass-Orthogonal Projection Method for Constrained Multibody Dynamics". *Journal of Nonlinear Dynamics*, **9**, 113-130, 1996.

[6] J. Cuadrado, J. Cardenal, P. Morer and E. Bayo. Intelligent simulation of multibody dynamics: space-state and descriptor methods in sequential and parallel computing enviroments. *Multibody System Dynamics*, **4**, 55-73, 2000.

[7] J. Cuadrado and D. Dopico. A hybrid global-topological real-time formulation for multibody Systems. Fourth Symposium on Multibody Dynamics and Vibration, at the

ASME Nineteenth Biennial Conference on Mechanical Vibration and Noise, Chicago, Illinois, USA, September 2-6, 2003.

[8] J. Reinders. *Intel Threading Building Blocks*. O'Reilly, 2007.

[9] J. García de Jalón, E. Álvarez, F. A. de Ribera, I. Rodríguez and F. J. Funes. A Fast and Simple Semi-Recursive Dynamic Formulation for Multi-Rigid-Body Systems. *Advances in Computational Multibody Systems*. J. Ambrósio, Springer-Verlag, 1-24, 2005.

[10] R. Serban, D. Negrut, F. A. Potra, and E. J. Haug. "A Topology Based Approach for Exploiting Sparsity in Multibody Dynamics. Cartesian Formulation". *Mechanics of Structures and Machines*, 25 (3), 1997

[11] R. Von Schwerin. *Multibody System Simulation*. Numerical Methods, Algorithms and Software. Springer, 1999.

[12] M. A. Serna, R. Avilés, and J. García de Jalón. Dynamic Analysis of Plane Mechanisms with Lower Pairs in Basic Coordinates. *Mechanisms and Machine Theory*, **17**, 397-403, 1982.

[13] J. García de Jalón, A. Callejo, A. F. Hidalgo and S. Tapia. Simple and Efficient Multibody Vehicle Dynamics using Matlab and C++. FISITA World Automotive Congress, Budapest, Hungary, 30 May-4 June, 2010.

[14] I. S Duff and J. K Reid. The Design of MA48: A code for the Direct Solution of Sparse Unsymetric Linear Systems of Equations. *ACM Transactions on Mathematical Software*, **22**, 187-226, 1996.

[15] M. González, F. González, D. Dopico and A. Luaces. On the effect of linear algebra implementations in real-time multibody system dynamics. *Computational Mechanics*, **41**, 607-615, 2008.

[16] T. A. Davis, K. Stanley. KLU: a Clark Kent Sparse LU Factorization Algorithm for Circuit Matrices. SIAM Conference on Parallel Processing for Scientific Computing, San Francisco CA, USA, February 25-27, 2004.

[17] T. A. Davis, E. Palamadai. User Guide for KLU and BTF. Dept. of Computer and Information Science and Enginnering, Univ. of Florida, Gainesville, FL, USA. http://www.cise.ufl.edu/~davis. March 24, 2009.

[18] F. González, A. Luaces, U. Lugrís, M. González. Non-intrusive parallelization of multibody system dynamic simulations. *Computational Mechanics*, **44**, 493-504, 2009.

[19] B. Chapman, G. Jost and R. Van Der Pas. *Using OpenMP: portable shared memory parallel programming*. The MIT Press, Cambridge, 2008.