A Parallel Simulator for Multi-Body Systems Based on Group Equations

José-Carlos Cano · Javier Cuenca · Domingo Giménez · Mariano Saura-Sánchez · Pablo Segado-Cabezos

Abstract Multibody systems consist of a set of components connected through some joints, where the movement of the system is determined by those of its components. Their design is computationally demanding, and the Group Equations formulation facilitates the application of parallelism to reduce the simulation time. A simulator for the kinematic analysis of multibody systems on up-to-date computational nodes (multicore CPU+GPU) is presented. The movement of the components is simulated by repeatedly solving independent linear systems working on sparse matrices. The appropriate selection of the linear algebra library to be used and the degree of parallelism at each level (explicit with OpenMP and implicit with multithread libraries) helps obtain important reductions in the simulation time.

Keywords Multicore · GPU · Linear Algebra Libraries · Multilevel Paralellism · Multibody Systems

1 Introduction

This paper presents a simulator for the kinematic analysis of multibody systems (MBS) in today's computational systems composed of multicore CPUs

José-Carlos Cano, Domingo Giménez
Department of Computing and Systems, University of Murcia, Spain
E-mail: {josecarlos.canol, domingo}@um.es

Javier Cuenca
Department of Engineering and Technology of Computers, University of Murcia, Spain
Tel.: +34-868-884821
Fax: +34-868-884151
E-mail: jcuenca@um.es

Mariano Saura-Sánchez, Pablo Segado-Cabezos
Department of Mechanical Engineering, Technical University of Cartagena, Spain
E-mail: {msaura.sanchez, pablo.segado}@upct.es

plus one or more GPUs. MBS are mechanical systems composed of rigid and flexible bodies connected through mechanical joints which determine the dependencies between the individual bodies and their contribution to the movement of the whole system. The kinematics analysis of a MBS consists of the study of the relation of its components' movements. The position and orientation of the bodies are defined by a vector of coordinates, $\mathbf{q} \in \Re^n$. If the MBS has g degrees of freedom (DOF), the values of $\mathbf{z} \in \Re^g$ independent coordinates out of n are known, and the values of $m = n - g$ dependent coordinates out of n are calculated by means of a set of m constraint equations, $\boldsymbol{\Phi}(\mathbf{q}, \boldsymbol{t}) = \mathbf{0}$. The kinematic analysis of MBS deals with the study of the position $\mathbf{q}$, velocity $\dot{\boldsymbol{q}}$ and acceleration $\ddot{\boldsymbol{q}}$ of the MBS when the position $\mathbf{z}$, velocity $\dot{\boldsymbol{z}}$ and acceleration $\ddot{\boldsymbol{z}}$ of the DOF are prescribed by solving the corresponding position $\boldsymbol{\Phi} = \mathbf{0}$, velocity $\dot{\boldsymbol{\Phi}}(\boldsymbol{q}, \boldsymbol{t}) = \mathbf{0}$ and acceleration $\ddot{\boldsymbol{\Phi}}(\boldsymbol{q}, \boldsymbol{t}) = \mathbf{0}$ constraint equations.

A simulator for a particular type of MBS was presented in [3]. It has been improved with new functionalities and a desktop application has been developed. Those improvements are discussed here, and its application to optimize the simulation time through the efficient exploitation of the parallelism in today's computational systems composed of multicore CPUs and GPUs is experimentally analyzed.

The remainder of the paper is organized as follows. Section 2 presents the Group Equations formulation for kinematic analysis and discusses the possibilities of parallelism. The MBS simulator is described in Section 3, and experimental results obtained with its parallel implementation are summarized in Section 4. Section 5 concludes the paper.

2 The Group Equations Formulation and Possibilities of Parallelism

Kinematic formulations that use a number of coordinates to define the position of each body independently of the position of the other bodies are known as Global formulations. Constraint equations, due to the rigid body condition and to the degrees of freedom restricted by each type of joint between bodies, are systematically imposed, and the number of constraint equations increases with the complexity of the topology of the MBS. So, a large system of constraint equations has to be solved simultaneously.

On the other hand, kinematic formulations like the Group Equations formulation, which takes into consideration the topology of the MBS, are known as Topological formulations. One advantage of the Group Equations formulation is that each of the subsystems that defines the kinematic structure of the MBS is defined by a subset of group coordinates $\boldsymbol{q_G} \in \Re^k$, $k < n$, and the corresponding subset of constraint equations $\boldsymbol{\Phi_G}(\boldsymbol{q_G}, \boldsymbol{t}) = \mathbf{0}$. So, the MBS is decomposed into a set of subsystems whose kinematics can be solved in a specific order [9,10]. As an example, a MBS and its decomposition into three subsystems are shown in Figure 1 (a and b). The kinematic structure of the MBS in Figure 1.c shows the division of the MBS into subsystems.

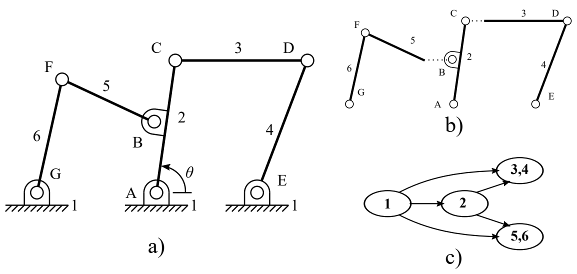

Fig. 1 a) MBS, b) Decomposition into Groups, c) Kinematic structure

Parallelism can be exploited at the implicit and explicit level. The term *explicit parallelism* is used to indicate that different subsystems are solved with shared-memory [5] or GPU [8] parallelism, and *implicit parallelism* refers to the use of multithread libraries which are available for the solution of the constraint equations with both dense [2] and sparse matrices [7] and for multicore CPU or GPUs [1,4].

A simulator for the Stewart Platform [11] (Figure 2) was presented in [3], where an analysis of the exploitation of parallelism and energy consumption for the real-time control in multicore CPU and Raspberry Pi was carried out, together with an initial experimental study of the exploitation of parallelism in the simulation of larger MBS in larger computational systems. There are many other real life MBS with highly paralellizable kinematic structures, in which their subsystems undergo planar or spatial movements, the joints between their bodies are of different kinds (spherical, cylindrical, prismatic) and the number of group coordinates varies from only a few (e.g. 2) to a typical range of (15−50) in 3D rigid body subsystems, and above those values for subsystems in which flexible bodies are considered. For example, [10] analyzes the application of the Group Equations formulation to a scalable four-bar linkage and to the suspension of a truck with a variable number of axles (Figure 3).

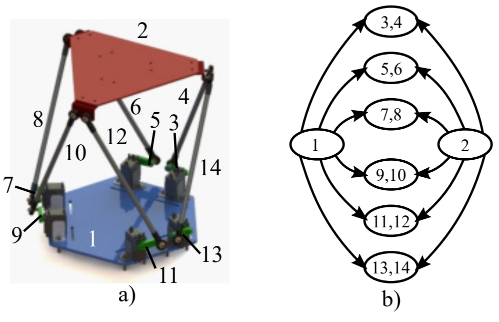

Fig. 2 Schema of a Stewart Platform and its kinematic structure

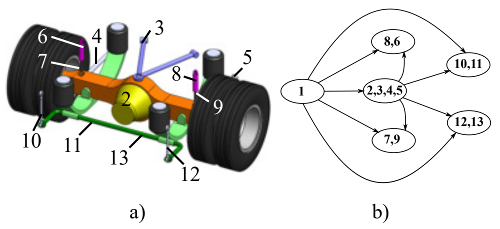

Fig. 3 Schema of a trunk suspension system and its kinematic structure

We use the structure shown by the Stewart Platform as a case study, but our previous studies are extended for the optimization of the exploitation of parallelism when larger systems are simulated, for which we use larger, heterogeneous and hybrid systems composed of multicore CPUs plus one or more GPUs with a variety of architectures. An application to facilitate simulation and optimization for general MBS has been developed. It allows us to work with a wide range of input scenarios (sparsity degree, matrix sizes and number of groups) and of linear algebra libraries in computational systems with variable complexity.

A schema for the simulation of the kinematics of a MBS with the topological structure of the Stewart Platform is shown in Algorithm 1. There are two loops corresponding to iterations to determine the position of intermediate, independent bodies (line 4) and of the body whose position is being established (line 1). These bodies correspond in the Stewart Platform to the six structural groups $(3, 4)$, $(5, 6)$ and so on, and to the terminal (2). The solution of the kinematics of the terminal (line 2) and of each structural component (line 5) is obtained from the solutions of systems of linear equations, with implicit parallelism through basic linear algebra routines. Because there are several independent structural components, they can be processed with explicit parallelism in the **for all** loop.

Algorithm 1 Schema of the Group Equations method for the Stewart Platform

```
1: for number of external iterations do
2:     Solve kinematics of terminal //implicit parallelism
3:     for all structural components //explicit parallelism do
4:         for number of internal iterations do
5:             Solve kinematics of structural component //implicit parallelism
6:         end for
7:     end for
8: end for
```

Both multicore CPUs and GPUs can be used for the exploitation of implicit and explicit parallelism:

– Parallelism can be exploited at some points without explicitly programming it. This implicit parallelism can be exploited through multithread libraries,

for example the MKL implementations of LAPACK and PARDISO on multicore CPUs [2,7]. Some libraries run on GPU [4], while others exploit the heterogeneity of multicore CPU+GPU systems [1].
– The explicit parallelism can be exploited in multicore CPUs with OpenMP. If several GPUs are available, one thread from a set of OpenMP threads can be assigned to each GPU in such a way that each structural component is solved by a GPU. Heterogeneity of a multicore CPU+multiGPU system can be explicitly exploited with a subset of threads working on the multicore CPU and each of the remaining threads sending work to be done to its associated GPU.

With two-level parallelism the two types of parallelism are exploited. Explicit parallelism is used to process the structural components (**for all** loop) and implicit parallelism is used in the solution of the systems associated to the terminal (line 2) and to the structural components (line 5). So, to fully exploit the parallelism offered by up-to-date computational nodes composed of multicore CPU+multiGPU, the number of threads to work with in the **for all** loop must be selected together with the number of structural components to be solved in multicore CPU or GPU and the number of threads used in the computation of the linear algebra libraries.

3 MBS Simulator

How the MBS simulator works is briefly explained here, with topological structures similar to that of the Stewart Platform. The size and shape of the matrices and the number of structural groups can be varied to simulate more complex structures and various types of coordinates. Some parameters can be varied to analyze the kinematics of the system (Table 1). There are iteration and MBS parameters:

Table 1 Iteration and MBS parameters

parameter	description
	Iteration parameters
`tEnd`	maximum time of simulation
`tIncr`	time increment between iterations
`nIntIter`	number of internal iterations
	MBS parameters
`numSG`	number of Structural Groups
`dimT`	dimension of the matrix of the final body (`dimT`×`dimT`)
`dimSG`	dimension of the matrix of each structural component (`dimSG`×`dimSG`)
`sparsity`	sparsity percentage of the matrices generated

– The iteration parameters are used to control the iteration properties of the analysis of the MBS. The number of iterations of the external loop in Algorithm 1 indicates how many times the whole MBS is solved (each

solution corresponds to different values of the $\mathtt{z}$ independent coordinates), and is determined by parameters `tEnd` and `tIncr` (number of iterations= `tEnd`/`tIncr`). `nIntIter` establishes the number of iterations in line 4. Normally, the number of external iterations is large, while that of internal iterations is very low (between one and five), but the parameters can be varied for the simulation of different convergence speeds.
– The MBS parameters determine the structure and size of the MBS. The solution of the kinematics of the terminal or the structural components has a cubic cost, so the total cost is

$$O\left(\frac{\mathtt{tEnd}}{\mathtt{tIncr}}\left(\mathtt{dimT}^3+\mathtt{tIntIter}*\mathtt{numSG}*\mathtt{dimSG}^3\right)\right)$$

for dense problems. For the Stewart Platform, `numSG`=6, `dimT`=12 and `dimSG`=15, and the sparsity degree is around 70% (the percentage of non-zero elements in the matrices is 30%). So, the possibilities of exploitation of parallelism are very limited. This could render the application of implicit parallelism of little interest. On the other hand, only six threads can be exploited simultaneously with explicit parallelism. But the parameters enable us to use the simulator for the study of larger systems.

There are additional parameters to determine the way in which the parallelism is applied. These parameters establish the basic linear algebra library and the number of computational elements (OpenMP threads and number of GPUs) to be used. They need to be selected properly for low simulation time.

3.1 Desktop application

The simulator worked initially with the information on the parameters, the scenarios and the decomposition of the MBS in groups provided by the user in files. But the users of the simulator will be engineers, and a more friendly environment is advisable. So, a desktop application has been developed, both for Windows and Linux. A screenshot of the application is shown in Figure 4. The graphic interface comprises three parts:

– The `Toolbar` gives access to the functionalities of the simulator. An existing model can be loaded or a new model can be created. The actual model can be deleted and its coherence can be checked. When a model is being saved, its coherence is checked. If it is valid, it is marked for simulation, otherwise it continues in the edition state. Once a model is marked as coherent it can be simulated (`Execute model`). The simulation generates a database which contains the possible combinations in the execution of the nodes of the directed graph, and the execution times obtained for each group and the functions they comprise. This information can be used to determine how to run future simulations.

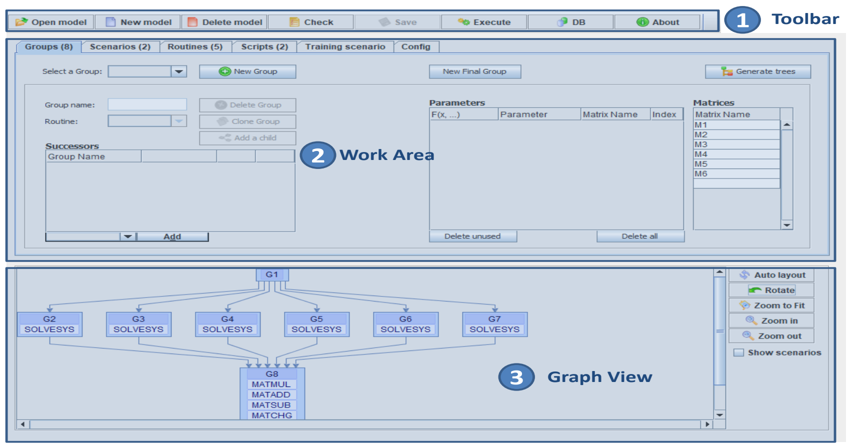

Fig. 4 Screenshot of the desktop application of the simulator (group representation of the Stewart Platform on the right)

- The `Work Area` maintains the information of the elements managed by the user for a particular model. The groups (nodes) of the MBS being simulated and their connections are established. The Figure shows the graph for the Stewart Platform, with eight groups. The routines executed in each group are also introduced (for example, `G8` comprises routines `MATMUL`, `MATADD`, etc, while linear systems are solved in groups `G2` to `G7`). The scenarios represent the input for the simulation (matrix sizes and sparsity degree). The values of the algorithmic parameters the simulator will work with to obtain the best configuration are established through the `Scripts` editor. These parameters are the maximum number of OpenMP threads, the number of GPUs and the basic libraries to experiment with. The general configuration (number of simulations, model to be simulated, location of the simulator and the database, etc) of the simulator is edited through `Config`.
- The `Graph View` shows the graph of the model that is being edited by the user. Is is updated dynamically while new scenarios, groups, basic routines, etc, are being included.

The simulator has four execution modes, which can be established in the configuration:

- In normal mode, executions are carried out for each possible combination of the parameters specified in the script. It supposes a long execution time, but the results are stored in the database, and the user can consult it to decide how to run similar simulations in the future.
- In the simple mode the user provides the values of the algorithmic parameters to experiment with.

– In training mode each of the basic functions in the model are executed, for each scenario and script, and the results are stored in an autotuning database.
– In autotuning mode the simulation is carried out with the algorithmic parameters which give the lowest execution time according to the results obtained in the training. This mode aims to obtain low simulation times without the huge execution time the normal mode has, even for small DAGs.

4 Experimental Results

The simulator can be used to analyze the application of parallelism in the kinematic analysis of MBS, helping to obtain the best combination of the algorithmic parameters for low simulation time. We summarize the results of some experiments varying the size and the structure of the problem, with the smallest configuration corresponding to the Stewart Platform, and with larger configurations to analyze the scalability for larger MBS. The matrices' sizes range from 12 and 15 (`dimT`=12 and `dimSG`=15) to 4000, sparse and dense matrices are considered (sparsity degrees of 85%, 50%, 30% and 10%), and configurations with more than six structural groups are experimented with. The number of combinations is very large, so the results of the experiments are summarized in the following subsections, beginning with the simplest parallelism considered (implicit exploitation of parallelism by calling routines of multithread libraries) and gradually increasing the parallelism complexity, considering nested implicit-explicit parallelism, CPU-GPU heterogeneity and multiGPU computation. Experiments are carried out with two multicore CPU+ GPUs configurations:

– **24C+GPU** is a node with 4 hexa-cores Intel Xeon E7530 with 32 GB of shared-memory at 1.87GHz, and a GPU Tesla K20c (Kepler architecture) with 4800 MBytes in Global Memory and 2496 CUDA cores (13 Streaming Multiprocessors and 192 Streaming Processors per Multiprocessor).
– **12C+6GPU** is a node with 12 cores and 6 GPUs. The multicore has two hexa-cores Intel Xeon E5-2620 with 32 GB of shared-memory at 2.00GHz. The GPU cards are two Nvidia Fermi Tesla C2075 with 5375 MBytes in Global Memory and 448 cores (14 Streaming Multiprocessors and 32 Streaming Processors per Multiprocessor) and four Nvidia GeForce GTX 590 with 1536 MBytes in Global Memory and 512 CUDA cores (16 Streaming Multiprocessors and 32 Streaming Processors per Multiprocessor).

For both systems the operating system is Linux (kernel 3.13.0-33-generic #58-Ubuntu), the CUDA version is 7.5, and the compiler used is Intel FORTRAN version 17.0.1, compilation 20161005.

4.1 Experiments with implicit parallelism on multicore CPU

The exploitation of implicit parallelism is analyzed with the MKL multithread implementation. The implementations of the dense library LAPACK and of the sparse library PARDISO are considered to analyze the sparsity degree at which each library is preferable. The results are compared with those with the routine MA27 of the Harwell Subroutine Library [6], which is sequential, for symmetric, sparse matrices and works especially well for small problems. So, the linear algebra libraries established in the editor of scripts are LAPACK, PARDISO and MA27.

Figure 5 compares the execution time (in logarithmic scale) for small and large problems, and with sparsity degrees 30% and 85%, and Table 2 summarizes the preferred library for different configurations, with sequential and parallel execution with 24 threads in **24C+GPU** (the GPU is not used). For small problems, the preferred routine is MA27 (which is sequential) independently of the sparsity degree. For larger matrices LAPACK outperforms MA27, even for sparse matrices if the parallelism is exploited. PARDISO is best only for very large, sparse matrices, but the parallelism is exploited worse than with LAPACK.

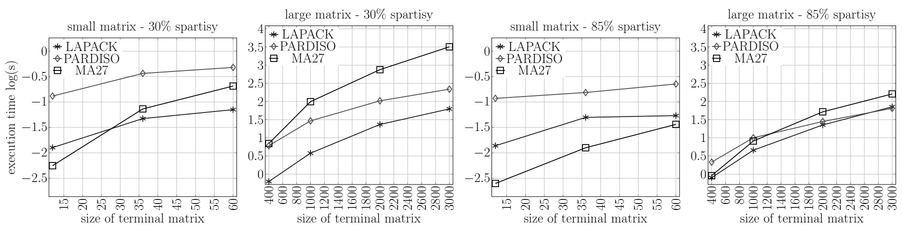

Fig. 5 Execution time (seconds in logarithmic scale) with LAPACK, PARDISO and MA27, for small and large matrices, and for sparsity degrees 30% and 85%, in **24C+GPU**

Table 2 Most efficient library when varying the matrix size and the sparsity degree, for sequential and parallel execution with 24 threads, in **24C+GPU**

	sequential				parallel 24 threads			
`dimT`/spar	10%	30%	50%	85%	10%	30%	50%	85%
12	MA27	MA27	MA27	MA27	MA27	MA27	MA27	MA27
36	LAP	LAP	LAP	MA27	MA27	LAP	LAP	MA27
60	LAP	LAP	LAP	MA27	LAP	LAP	LAP	MA27
400	LAP	LAP	LAP	MA27	LAP	LAP	LAP	LAP
1000	LAP	LAP	LAP	MA27	LAP	LAP	LAP	LAP
2000	LAP	LAP	LAP	PARD	LAP	LAP	LAP	LAP
3000	LAP	LAP	PARD	PARD	LAP	LAP	LAP	PARD

4.2 Exploitation of two-level parallelism on multicore CPU

The granularity of parallelism is low when only that of the linear algebra operations is exploited, but the combination of multithread libraries with explicit parallelism enables the exploitation of coarser parallelism. So, by using shared-memory with OpenMP in the **for all** loop of Algorithm 1, parallelism can be exploited at two levels: explicit OpenMP parallelism and implicit parallelism with multithread libraries. The number of threads at each level needs to be tuned for a particular computational system as a function of the problem size, the sparsity degree and the number of structural groups. The speed-up obtained with two-level parallelism is shown in Figure 6, for small and large problems and when using the LAPACK or PARDISO implementations of MKL for the basic linear algebra problems. The sparsity degree is 85% and the system is **12C+6GPU** (the GPUs are not used). The behavior with the two libraries is different:

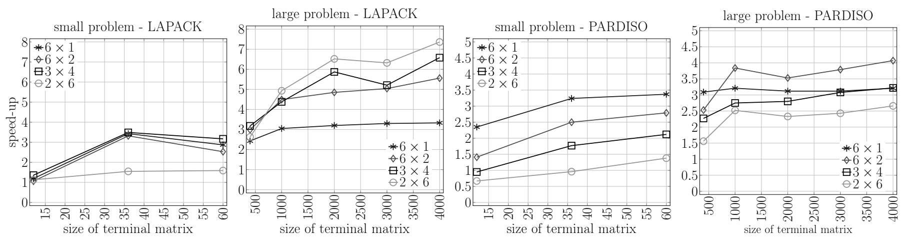

Fig. 6 Speed-up with two-level parallelism for 6 structural groups and 85% sparsity degree, for several threads combinations, with MKL implementations of LAPACK (top) and PARDISO (bottom), in **12C+6GPU**

- For small problems the use of OpenMP parallelism alone is a satisfactory option, especially for PARDISO, with a worse exploitation of the implicit parallelism.
- The combination of the two types of parallelism increases the performance, but with the preferred number of threads being dependent on the library. For large problems, 2×6 is preferred for LAPACK, while for PARDISO the best combination is 6×2. This again is due to the better exploitation of parallelism with LAPACK.
- The two-level parallelism allows us to surpass the limit of the number of structural groups (six in the example).

Exhaustive experiments were carried out in the two computational systems considered, with two-level parallelism with LAPACK and PARDISO and with explicit parallelism with MA27, for six structural groups and several sparsity degrees, and the number of threads at each level was varied to find the best combination. Table 3 summarizes the results for sparsity degree 85%. Only configurations which give the best results for some of the sizes are shown. Explicit parallelism with MA27 as basic library is the preferred option for small and medium problems. For large problems, the exploitation of sparsity with PARDISO together with two-level parallelism with a number of OpenMP threads equal to the number of structural groups gives the best results. If the number of structural groups is larger than the number of cores in the system, it may be preferable to use only explicit parallelism with PARDISO. This is seen in Figure 7, which shows that in **12C+6GPU** and for a sparsity degree of 85% the preferred combination is sequential PARDISO with a number of OpenMP threads equal to the number of cores. For larger matrices the difference decreases.

Table 3 Summary of the lowest execution times obtained with two-level parallelism in two computational systems, for `numSG`=6, 85% sparsity degree and varying the problem size

`numSG`=6		**12C+6GPU**		**24C+GPU**	
`dimT`	`dimSG`	MA27 6×1	PARD 6×2	MA27 6×1	PARD 6×4
12	15	**0.0009**	0.0234	**0.0030**	0.0571
36	54	**0.0043**	0.0288	**0.0060**	0.0475
60	90	**0.0089**	0.0430	**0.0132**	0.0682
400	400	**0.1559**	0.4945	**0.2728**	0.6976
1000	1000	**1.6476**	2.4988	**2.8928**	3.5935
2000	2000	10.3387	**7.7590**	17.5241	**10.7895**
3000	3000	32.8752	**16.732**	54.6981	**24.8652**
4000	4000	76.5881	**31.6888**	123.6253	**52.3393**

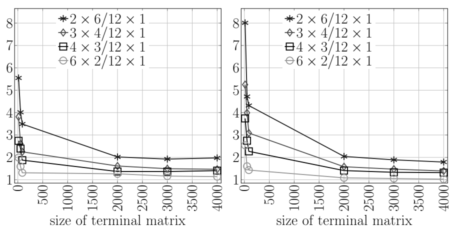

Fig. 7 Speed-up of OpenMP+PARDISO parallelism with 12×1 threads with respect to other combinations of OpenMP and PARDISO threads, with 21 (left) and 26 (right) structural groups, in **12C+6GPU**

4.3 Experiments with parallelism on GPUs

Today's standard nodes normally include GPUs. One of the systems experimented with includes six GPUs of two types, and so there is heterogeneity in the coprocessors in addition to that of CPU vs. GPU. There are linear algebra libraries for GPU [1,4], which can be used to implicitly exploit the massive parallelism of the coprocessors, and CPU+GPU hybrid parallelism can be exploited with explicit parallelism with OpenMP threads starting multithread or GPU routines to work on the solution of the underlying linear systems. But starting kernels in GPU comes at a high cost, which means the use of GPUs for the problem in hand is only of interest for large problems.

The results with MA27 and PARDISO for sparse matrices (sparsity degree 85%) are compared with those obtained with MAGMA in **12C+6GPU** (Table 4). The number of structural groups is six, and so 6×1 and 6×2 threads are used for MA27 and PARDISO, respectively. For MAGMA, because there are six GPUs in the node, the best results are obtained with each GPU working on a structural group, but results with 2 and 3 GPUs are also shown to see the reduction achieved with the inclusion of more coprocessors. The exploitation of GPUs is advisable only for large problems and with a large number of GPUs, and the reduction in execution time is not proportional to the number of GPUs, which is due to the high management and transference cost of the GPUs.

Table 4 Comparison of the execution time with CPU (MA27 and PARDISO) and with GPU (MAGMA), for `numSG`=6, 85% sparsity degree and varying the problem size

`numSG`=6		th. OMP×MA27	th. OMP×PARD	MAGMA		
`dimT`	`dimSG`	6×1	6×2	2	3	6
12	15	**0.0009**	0.0234	0.0073	0.0064	0.0064
36	54	**0.0043**	0.0288	0.0073	0.0066	0.0067
60	90	**0.0089**	0.0430	0.0326	0.0269	0.0122
400	400	**0.1559**	0.4945	1.2254	1.1266	1.2620
1000	1000	**1.6476**	2.4988	2.7287	2.6829	2.2188
2000	2000	10.3387	7.7590	9.2946	7.8085	**5.9685**
3000	3000	32.8752	16.732	23.7600	18.9474	**14.0435**
4000	4000	76.5881	31.6888	48.3116	38.1400	**28.2109**

4.4 Autotuning preliminary results

The experiments in the previous subsections show that the preferred configuration for low execution time depends not only on the problem size but also on the basic library, the number of threads at each level and the computational system. The training and autotuning modes of the simulator are thought to help the user in the efficient execution of simulations. The simulator can be trained with all the computational components in the system (CPUs and GPUs) and the corresponding linear algebra libraries (LAPACK, PARDISO and MAGMA), and possible combinations of them if CPU and GPU compute together. MAGMA has two interfaces, CPU and GPU, which differ in the implementation of the routines. In the CPU interface, the LU is computed with a hybrid algorithm which combines CPU and GPU, but the solution of the equations system is not implemented for GPU, and is solved through LAPACK in CPU. In the GPU interface both the LU factorization and the solution of the system are implemented with a hybrid algorithm for CPU+GPU. Training is done with the scenario parameters `numSG`=$\{8, 14, 20, 26\}$, `dimT`=`dimSG`= $\{1000, 2000, 3000, 4000\}$, `sparsity`=$\{50, 70, 85\}$.

Table 5 compares the execution times with autotuning mode with the lowest experimental times for other problem sizes (`numSG`=$\{11, 17, 23\}$, `dimT`= `dimSG` =$\{1500, 2500, 3500\}$, `sparsity`=$\{60, 80\}$). MAGMA in its GPU (MA-G) or CPU (MA-C) interface is always the best option, in some cases in combination with PARDISO (P+MA). The number of threads at each level changes, with a larger number of threads for OpenMP, and with the total number of threads close to the number of cores in the CPU. The autotuning methodology always selects a combination of PARDISO with MAGMA for large problems or for large sparsity. The combination selected with autotuning coincides with the best experimental ones in less than half of the cases, which produces a mean deviation of the lowest experimental time with respect to that with autotuning of around 15%. This deviation is admissible considering that it is obtained without user intervention.

Table 5 Comparison of the parallelism parameters, library and execution time (in seconds) with the autotuning methodology and those which give the lowest experimental time for different validation configurations, in **12C+6GPU**

configuration			autotuning			Optimum experimental		
			threads			threads		
`numSG`	`dim`	`spar.`	Ex×Im	library	time	Ex×Im	library	time
11	1500	60	11×1	MA-G	5.47	6×2	MA-G	5.15
11	2500	60	8×1	MA-G	13.24	12×1	MA-G	13.21
11	3500	60	12×1	MA-G	30.05	5×2	P+MA-G	29.28
17	1500	60	6×2	MA-G	7.36	9×1	P+MA-G	6.50
17	2500	60	8×1	MA-G	18.82	10×1	MA-G	18.29
17	3500	60	6×2	P+MA-G	41.96	5×2	MA-G	40.68
23	1500	60	9×1	MA-G	10.11	10×1	MA-C	8.82
23	2500	60	10×1	MA-G	23.88	6×2	P+MA-G	23.25
23	3500	60	6×2	P+MA-G	53.45	12×1	MA-G	52.37
11	1500	80	6×2	MA-C	5.30	6×2	P+MA-G	5.05
11	2500	80	6×2	MA-G	13.32	6×2	MA-G	13.32
11	3500	80	8×1	MA-G	31.35	6×2	MA-G	29.72
17	1500	80	11×1	P+MA-C	7.06	10×1	P+MA-G	6.66
17	2500	80	12×1	P+MA-G	28.07	10×1	MA-G	18.33
17	3500	80	12×1	P+MA-G	68.93	5×2	MA-G	41.49
23	1500	80	12×1	P+MA-C	12.40	9×1	MA-C	9.00
23	2500	80	12×1	P+MA-G	46.16	9×1	MA-G	23.61
23	3500	80	12×1	P+MA-G	126.39	12×1	MA-G	52.38

5 Conclusions and Future Work

This paper presents a simulator for the kinematics of MBS. It can be used to determine satisfactory configurations of the parallelism parameters (number of OpenMP threads and of GPUs, and basic linear algebra library) for low simulation time. An autotuning mode is included to help non-expert users in the selection of the simulation' configuration. Experiments with a MBS with the structure of the Stewart Platform show the usefulness of the simulator with different configurations: computational systems composed of multicore CPU+multiGPU and several basic linear algebra libraries (LAPACK, PARDISO, MA27, MAGMA), together with their combinations with OpenMP through two-level parallelism.

The simulator should be improved in some points. In particular, the autotuning engine needs to be improved and the training time be reduced. Once the simulator includes appropriate functionality, it will become freely available. The simulator and the autotuning engine can be adapted to other computational systems, for example, nodes including Xeon Phi coprocessors and clusters of multicore CPU+multicoprocessors. The simulator is being applied to other problems whose computation can be decomposed in DAGs.

Acknowledgements This work was supported by the Spanish MINECO and European Commission FEDER funds under grant TIN2015-66972-C5-3-R.

References

1. Emmanuel Agullo, Jim Demmel, Jack Dongarra, Bilel Hadri, Jakub Kurzak, Julien Langou, Hatem Ltaief, Piotr Luszczek, and Stanimire Tomov. Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects. *Journal of Physics: Conference Series*, 180(1), 2009.
2. E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. J. Dongarra, J. Du Croz, A. Grenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1995.
3. Gregorio Bernabé, José-Carlos Cano, Javier Cuenca, Antonio Flores, Domingo Giménez, Mariano Saura-Sánchez, and Pablo Segado-Cabezos. Exploiting hybrid parallelism in the kinematic analysis of multibody systems based on group equations. In *International Conference on Computational Science*, pages 576–585, 2017.
4. CUBLAS. `http://docs.nvidia.com/cuda/cublas/`, 2018.
5. Leonardo Dagum and Ramesh Menon. OpenMP: an industry standard API for shared-memory programming. *Computational Science & Engineering, IEEE*, 5(1):46–55, 1998.
6. HSL. A collection of Fortran codes for large scale scientific computation, `http://www.hsl.rl.ac.uk/`, year=2013,.
7. Intel© MKL PARDISO. `https://software.intel.com/en-us/node/470282`, 2018.
8. John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.
9. M. Saura, A. I. Celdrán, D. Dopico, and J. Cuadrado. Computational structural analysis of planar multibody systems with lower and higher kinematic pairs. *Mechanism and Machine Theory*, 71:79–92, 2014.
10. M. Saura, P. Segado, B. Muñoz, and D. Dopico. Multibody kinematics. A topological formulation based on structural-group coordinates. In *ECCOMAS Thematic Conference on Multibody Dynamics*, pages 88–99, June 2015.
11. D. Stewart. A platform with Six Degrees of Freedom. In *Institution of Mechanical Engineers UK*, volume 180, pages 371–386, 1965.