

# Maple as an automatic code generator ?

Brian Moore<sup>1</sup>, Jean-Claude Piedbœuf  
Space Technologies, Canadian Space Agency <sup>2</sup>  
6767 Route de l'Aéroport, St-Hubert,  
Québec, Canada, J3Y 8Y9  
Brian.Moore@space.gc.ca, Jean-Claude.Piedboeuf@space.gc.ca

Laurent Bernardin  
Waterloo Maple Inc.  
57 Erb Street W., Waterloo, Ontario, Canada, N2L 6C2  
lbernardin@maplesoft.com

## Abstract

This paper discusses the use of Maple as a code generator for multi-body systems. First, an overview of Symofros, a multibody system modeling and simulation environment developed at the Canadian Space Agency is given. Symofros uses Maple for symbolic modeling and code generation. Improvements of the Maple subs function and Maple LinearAlgebra package are presented. We then show how these improvements greatly simplify the handling of complex expressions and as a consequence, simplify the modeling approach used in Symofros. Using a model of the SSRMS, the Space Station Remote Manipulator System, comparison between the efficiency of the previous and new modeling approach is done in terms of modeling time and code efficiency. We conclude by a summary of the improvements done in Maple and the improvements to be done in order to be able to use Maple as a code generator for multibody systems.

---

<sup>1</sup>Opal-RT Technologies on secondment to Canadian Space Agency

<sup>2</sup>©Canadian Space Agency 2002

# 1 Introduction

This presentation discusses the application of Maple as a code generator for multibody system dynamics. As an example, we present Symofros, a modeling, simulation and real-time implementation software for rigid and flexible mechanical structures developed at the Canadian Space Agency since 1994. Symofros is used at the Canadian Space Agency for different projects, including SPDM Tasks Verification Facilities [9] and astronauts training using BORIS [3]. Symofros is based on commercial tools like Maple and Matlab/Simulink. It is composed of 3 main modules: the graphical model editor, the Symbolic Model Generator (SMG) and the simulation environment. The simulation environment is composed of the model querying module, the non-real time simulation environment and the real-time simulation environment.

The Symbolic Model Generator (SMG) built within Maple is the root of Symofros. The SMG is composed of several Maple modules: topology, preprocessing, kinematics, non-linear dynamics, linear dynamics, parallel structures kinematics and analysis. It automatically derives a set of over 90 functions, called basic functions, which represent the kinematics and dynamics. These functions are symbolic expressions written in terms of variables (time dependent) and parameters (constant with time). Variables comprise positions, velocities, accelerations and system inputs while parameters can be the mass and the length of the links. The functions are translated in C for simulation purpose.

The main issue for a symbolic code generator is to do efficient modeling and generate efficient code. Efficient modeling means that the time and memory required to generate the code should be reasonable. It depends on the code produced by the programmer or included in Maple but also on Maple memory management. The problem of efficiency was encountered in Symofros because we compute large but simple expressions that have many common subexpressions. The solution implemented in 1996 was to replace common subexpressions by temporary variables. This solution allowed Symofros to model multibody systems with more than a few degree-of-freedom. However, the manipulation of the expressions became cumbersome because the actual content of the temporary variables had to be taken into account for many operations. Basically, it meant that we had to rewrite many Maple procedures in order to deal with these temporary variables. In practice, we had to recreate most of the variable management normally done by Maple. In Fall 2001, Waterloo Maple and CSA reviewed the code to evaluate why the Maple memory management was inefficient for large but simple expressions. We found that by modifying some Maple procedures a drastic improvement can be achieved making the temporary variables unnecessary. In particular, improvements to the subs function and to the LinearAlgebra package to avoid duplication when computing expressions with common subexpressions, helped improve significantly the efficiency of Maple in the context of code generation. These improvements are implemented in Maple release 8, and allow us to let Maple operate on the unmodified expressions as opposed to having to replace common subexpressions with temporary variables. The efficiency of the code generated is also an important issue, since the code

is used in our case for real-time simulation. Being able to use Maple data management also allows Symofros to use the full functionalities of the C code optimization.

As an example, we model a 7 degrees of freedom manipulator with elastic joints with Symofros. A comparison of the results for different Maple releases, with and without the use of "artificial" temporary variables is then presented. We compare the efficiency of the modeling process in terms of memory management and time, and the efficiency of the C code is compared for different Maple optimization functions and options. We show in this presentation that Maple can be used as a code generator.

## 2 Symofros

Symofros is based on commercial tools and is composed of three main modules for mechanical system description, modeling and simulation (see Figure 2).

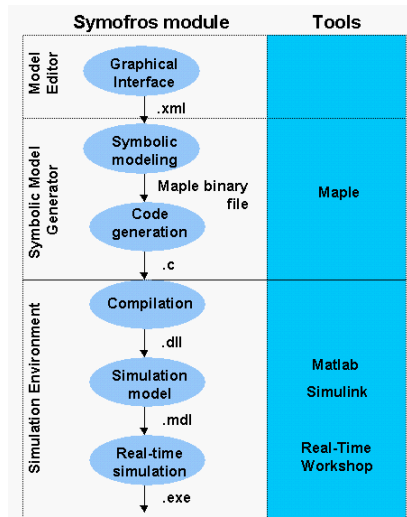


Figure 1: Overview of Symofros modules

Creating a model of a mechanical system consists of describing the bodies, the joints and the topology of the system. This model description is based on the XML language<sup>3</sup>, a standardized language used to describe any kind of data and used for many applications. For mechanical system description, this language is also used by researchers in Spain [10].

The Symbolic Model Generator (SMG) comprises modules written in the Maple language to perform the symbolic modeling. The input of the module is an XML or Maple file describing the properties of the mechanical system. This

<sup>3</sup>[www.w3.org](http://www.w3.org)

file is used by the module to compute the kinematic and dynamic quantities of the bodies and the joints. From the input file, the topology of the mechanical system is analyzed to generate a graph model. Using the topology with the body and joint data, the SMG develops the kinematic equations. Using the kinematic formulation, the SMG builds the dynamic equations in various forms, for simulation (forward dynamics), control (inverse dynamics), and parameter identification (currently in development). Special kinematic quantities are also generated for parallel mechanisms based on the approach proposed in [5]. The SMG is normally used as an automatic model generator, but it is also a powerful tool to analyze the dynamic equations and to develop models on-line. More details on the symbolic modeling part of Symofros can be found in [8].

For simulation and real-time implementation, the SMG generates C code to represent the multibody system. The code generation requires optimization tools to break the complex expressions down to smaller expressions. This also helps improving the code efficiency for simulation since sub-expressions appearing several times need to be computed only once. The C functions are the links between the modeling part of Symofros, and the simulation/real-time implementation parts. Therefore, using the model in an advanced simulation or in the real-time environment is straightforward.

To allow an efficient and convenient use of the mathematical model derived, and to enable the numerical simulation, Symofros is directly linked to the Matlab/Simulink environment. The Simulink environment allows to create complex models and generate complex simulation systems in only a few simple steps without the need of advanced programming skills. Special blocks are available in the library in order to call the functions generated symbolically and written in the .c file. As an example, Figure 2 shows how the forward dynamics can be computed. In this example, the dark blocks (*Mnl*, *gnl*) are used to call the functions written in the .c file. Then, using standard Simulink blocks, the system of equations is solved to obtain the accelerations, and integrated to obtain the generalized velocities and generalized coordinates. This block (Forward Dynamics) can then be found in the Symofros library and re-used with other models.

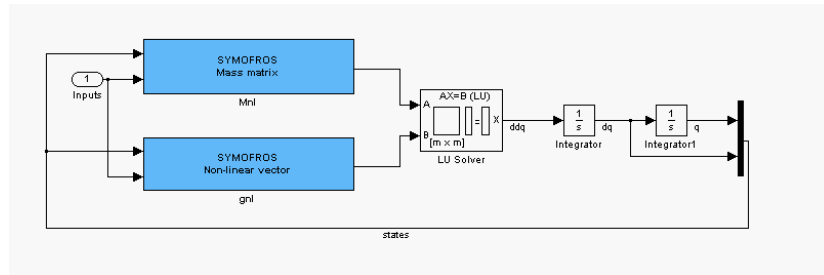


Figure 2: Simulation model within Simulink

Real-time simulation and hardware-in-the-loop simulation can be achieved

by using complementary tools like the Real-Time Workshop and RT-Lab for generating real-time simulation code and distributing the computations on several computers. More details on this topic can be found in [4], [2] and [9].

## 3 Symbolic modeling and code generation using Maple

### 3.1 Modeling approaches for multibody systems

Modeling and simulation can be achieved using symbolic or numerical approaches. In the symbolic approach, the mathematical model is developed in symbolic form prior to the simulation and is done only once. It consists of deriving symbolically the expressions in terms of parameters (mass, length, inertia, etc.) and time dependent variables (coordinates, velocities, acceleration, forces, etc.). These expressions can then be used in simulation by replacing the parameters at the beginning of the simulation and the time variables at each simulation time step by their numerical values. Contrarily to the symbolic approach, modeling and simulation are closely linked when a numerical approach is used. In the numerical approach, the mathematical model is developed at each simulation time step.

Symbolic approaches for modeling have several advantages over numerical approaches:

- Simplification of the equations is possible. For example, removing null expressions, replacing expressions equal to 1 and simplifying trigonometric expressions can reduce significantly the amount of computation required during the simulation.
- The development of the mathematical model is done only once before the simulation stage while numerical approaches require the development of the mathematical model at each simulation time step. In other terms and contrarily to the numerical approach, the modeling is decoupled from the simulation.
- Analytical analysis can be achieved for simple case.

However, numerical approaches are more convenient in some cases:

- Handling of complex constrained systems has been primarily investigated based on numerical considerations, for example coordinate partitioning for generic multibody systems. The symbolic implementation of these methods can be difficult.
- For mechanical systems with many degree of freedom ( $> 50$  dof), the complexity of the symbolic expressions grow significantly leading to memory management problems and difficulty of handling complex expressions.

The use of a symbolic approach for modeling requires the use of symbolic tools to manipulate the symbolic expressions. A first possibility is to use a symbolic manipulation package dedicated to a given application. This approach is used in Robotran [1] where a symbolic tool has been created using the C language. In Symofros and Dynaflex [11], symbolic manipulations are achieved using a general purpose symbolic tool, Maple.

The use of dedicated symbolic tools for code generation has the advantage of being more efficient in terms of time to generate the code compared to a general purpose symbolic tools. However, since the code has to be generated only once prior to simulation, the time spent in the symbolic modeling is not critical, but should be reasonable. General-purpose symbolic tools have some advantages over dedicated symbolic code generator:

- There is no time and maintenance required for creating and maintaining the symbolic tool.
- The use of a general purpose symbolic tool like Maple allows to handle the expressions within Maple in order to get some physical insight on the physical phenomena and to use it as a powerful teaching tool.

### 3.2 Symbolic code optimization

In this paper, we use the term optimization to refer to the process of breaking down a symbolic expression in order to write it as code to be computed numerically. This process consists of replacing common subexpressions by a temporary variables, which will be assigned to the corresponding sub-expressions. In a later step, when the expression will be evaluated numerically, this temporary variable will be computed and replaced in the expressions avoiding the repeated computation of the same sub-expressions.

In Symofros, most of the symbolic expressions derived, result from the addition or multiplication of matrices and vectors. In this case, two different approach can be used for optimization:

- Optimization during the modeling process
- Optimization after the modeling process

These two approaches have been used in Symofros as described below.

### 3.3 Previous approach

From the beginning, huge efforts have been put in Symofros to improve the modeling and code generation phase. The reason is that it was only possible to generate simple model (a few degree of freedom) and not more complex model. Moreover, it was difficult to model flexibility. These problems were coming from the fact that the modeling and code generation processes were time consuming and were requiring a large amount of memory within Maple.

The solution proposed by Piedboeuf[7] to overcome these difficulties was to do the optimization during the symbolic modeling process instead of after. This was done by modifying the Maple `optimize` procedure to make it global [6].

Using this approach, it was possible to generate more complex model. However, new difficulties arised from this new approach:

- Postprocessing was required to remove temporary variables which were not used and to substitute the variables which were used only once. This postprocessing had to be done after the symbolic modeling to reduce the number of temporary variables used.
- It was not possible to use the advanced Maple optimization capabilities. Therefore, the code obtained after the code generation was not efficient.
- Manipulation of the expressions was cumbersome because we had to handle expressions with temporary variables. We had to recreate most of the variables management to handle the temporary variables. For example, we had to rewrite a procedure for derivation using the chain-rule.
- We lost the advantage of on-line manipulation of the expressions because the expressions were not readable because they were written in terms of temporary variables.
- Since the optimization of the equations was achieved during the modeling process, the optimization was taking place on the equations known up to a given point, loosing the possibility of efficient simplification on the final equations. In other words, a lot of symbolic simplification were difficult to achieve like trigonometric simplification.

It is clear from all these drawbacks, that this approach should be avoided if possible. Using the profiling option of Maple, we observed that it would be possible to improve the efficiency of the modeling process in order to be able to do the optimization after the symbolic modeling process as shown in the next section.

### 3.4 New approach

Maple uses directed acyclic graphs (DAGs) as its internal representation for (symbolic) expressions. This means that common sub-expressions are shared; they are stored only once in the system and are automatically re-used whenever they occur. Very large expressions that have many common sub-expression, such as those occuring in Symofros can thus be stored in a very efficient manner.

When computing with such data-structures, it is important to realize that common sub-expressions are re-used. When writing a Maple program, `option remember` can be used to avoid re-computing and re-evaluating common sub-expressions. Some internal Maple operations, like for example `indets`, also take advantage of the DAG structure when operating recursively on large expressions. Prior to Maple 8, the `subs` command was not DAG-aware, which

means that it always traversed the expression tree, which can be exponentially more expensive than traversing the DAG. Since Maple 8, both the `subs` and the `eval` command use DAG traversal; each sub-expression is operated on exactly once, no matter how often it occurs in the expression. This has resulted in a tremendous efficiency gain (see below).

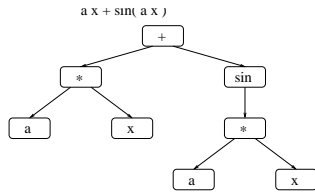


Figure 3: Tree representation

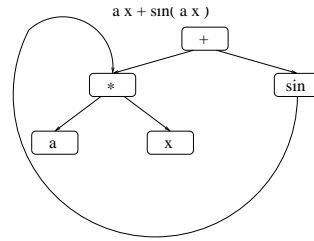


Figure 4: DAG representation

Another bottleneck in Symofros turned out to be linear algebra operations. The `LinearAlgebra` package in Maple has been optimized to be highly efficient for large, numeric matrices. The matrices occurring in the symbolic modeling process have a different structure: The matrices are small but the entries are very complex, symbolic expressions. It turned out that more of the time in `LinearAlgebra` was actually spent on type checking and full evaluations of the matrix entries. While some of this behaviour could be traced back to bugs in the `LinearAlgebra` package, which were subsequently fixed, a large part of the extra time spent is due to the generic nature of the `LinearAlgebra` package, which needs to deal with a large number of different datatypes. In order to get efficient computations with the kind of data that occurs in Symofros, we developed the `FastLA` package, which achieves maximal efficiency for symbolic matrices by bypassing type checking and avoiding any full evaluations of the matrix entries.

With these improvements, it is now possible to do the optimization only after the modeling process. The new approach reduces significantly the complexity of Symofros, allow the use of Maple procedures (including optimization and derivation procedures) and improve the code efficiency.

## 4 Efficiency analysis

To evaluate the efficiency of the new modeling approach without the use of temporary variables during the symbolic modeling within Symofros and to evaluate the new Maple `subs` function and the `LinearAlgebra` module, the model of the SSRMS will be modelled in Symofros. The SSRMS model represents the Canadarm2, the Space Station Remote Manipulator System presently installed on the International Space Station. In this model, 7 rigid degree-of-freedom and 4 elastic joints are considered. The flexibility of the beams is not considered. The tests presented below have been achieved using a Pentium 1.8 GHz

Maple version	Time spent in subs	Number of calls
Maple 7 (tree data structure)	83.6 s	10471
Maple 8 (DAG data structure)	2.8 s	

Table 1: Comparison of the `subs` function

Procedure	Time	Number of calls
LinearAlgebra:-VectorAdd FastLA:-FastVectorAdd	602.6 s 29.6 s	294
LinearAlgebra:-MatrixAdd FastLA:-FastMatrixAdd	98.5 s 5.5 s	117
LinearAlgebra:-MatrixMatrixMultiply FastLA:-FastMatrixMatrixMultiply	53.2 s 0.180 s	354
LinearAlgebra:-MatrixVectorMultiply FastLA:-FastMatrixVectorMultiply	125.4 s 0.180 s	408
LinearAlgebra:-Transpose FastLA:-FastTranspose	13.4 s 0.0 s	258

Table 2: Comparison of the linear algebra package

with 260 M of RAM under Windows 2000.

#### 4.1 Efficiency of the `subs` function

Table 1 shows the total time spent in the symbolic modeling process (without the code generation) and in the Maple `subs` function using Maple 7 (data stored in a tree) or Maple 8 (data stored in a DAG). In this case, the time spent in the `subs` function has been reduced by a factor of almost 30.

#### 4.2 Efficiency of the LinearAlgebra package

As discussed above, improvement to the LinearAlgebra package has been done. Table 2 shows the time spent for different linear algebra functions during the symbolic modeling process. The comparison is done between the function of the LinearAlgebra module and the new functions of the FastLA package.

#### 4.3 Efficiency of the symbolic modeling process

Table 3 below shows the time spent for the symbolic modeling and the code generation. The first test has been done with Maple 5 using the temporary variables and the `linalg` module. We see that this approach doesn't required

	Maple release	Time symbolic modeling
With temporary variables	5	180 s
Without temporary variables	7	641 s
With <code>FastLA</code> module	7	154 s
With Maple8 <code>subs</code> and <code>FastLA</code> module	8	72 s

Table 3: Modeling time

		Add.	Mult.	Subs.	Ass.	Fcn.
Maple 5		5083	4318	12966	2520	12
Maple 8	Basic optimization	2492	2196	7809	2072	12
	Maple <code>optimize</code>	2019	1901	5699	1247	12
	Maple <code>optimize tryhard</code>	1147	1376	3367	741	12

Table 4: Code efficiency for the mass matrix

much time, but has all the inconvenient described previously because the optimization is done during the modeling process. Then, using Maple 7 and the Maple `LinearAlgebra` module, the symbolic modeling takes more time. We observe that the improvement done to the `FastLA` module and the Maple 8 `subs` has reduced by a factor of 9 the time required to do the symbolic modeling.

#### 4.4 Efficiency of the code generator

As discussed above, the most critical measure is the efficiency of the code generated because the expressions written in the C file will be computed thousands of time in simulation. In this section, we compare the efficiency in terms of the number of additions, multiplications, assignments, subscripts and functions of the code generated by Symofros.

In the new approach where the optimization takes place after the symbolic modeling, it is possible to do an optimization on the final expressions. Therefore, it is possible to use different optimization procedures to break the expressions down to simpler expressions. In the following tests, we have used three different optimization methods. First, a basic optimization is used where the expressions are simply break down into smaller expressions, without any redundant terms. The other optimization processes are based on the Maple `codegen/optimize` function, with and without the `tryhard` option.

Comparison of the time, memory and computational cost required to compute the mass matrix and the non-linear force vector for different versions of Symofros and Maple are given in table 4 and 5. The mass matrix and the non-linear force vector are the basic expressions used for simulation and control.

The time and memory required to do the optimization and write the code in the file with different optimization methods are shown in table 6.

		Add.	Mult.	Subs.	Ass.	Fcn.
Maple 5	No reduction of TV	4099	4625	11898	2147	12
Maple 8	Basic optimization	2252	2090	7279	1962	12
	Maple <code>optimize</code>	1930	1760	5569	1288	12
	Maple <code>optimize tryhard</code>	1002	1285	3001	614	12

Table 5: Code efficiency for the non-linear force vector

	Time(s)	Memory (Meg)
Basic optimization	99.8 s	12.9 M
Maple <code>optimize</code>	152.6 s	58.6 M
Maple <code>optimize tryhard</code>	1087.9	12.8 M

Table 6: Time and memory required to generate the code

The results clearly show the advantage of using Maple optimization procedure for generating the code and the advantage of optimizing the code at the end of the symbolic modeling process instead of doing the optimization during the symbolic modeling process. However, we also see that the time required for the optimization process can be quite long when using tryhard option.

## 5 Conclusion

In this paper, we have shown that it is possible to model multibody systems without creating artificial temporary variables during the modeling process with the improvement done to the `subs` functions and the optimization that can be achieved by modifying the `LinearAlgebra` module. We have also shown the improvement of the code efficiency in terms of the basic number of operations which is the critical factor when doing simulation. The efficiency has been considerably improved by being able to use Maple optimization capabilities.

## References

- [1] P. Fiset, T. Postiau, L. Sass, and J.-C. Samin. Fully symbolic generation of complex multibody models. *Mechanism of Structures and Machines*, 30(1):31–82, 2002.
- [2] Michel Lambert, Brian Moore, and Mojtaba Ahmadi. Essential real-time and modeling tools for robot rapid prototyping. In *The 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS 2001*, June 2001.
- [3] R. L’Archevêque, Z. Joukakelian, and P. Allard. BORIS: A simulator for generic robotic training. In *The 6th International Symposium on Artificial*

*Intelligence and Robotics & Automation in Space: i-SAIRAS 2001*, June 18-22 2001.

- [4] Regent L'Archevêque, Michel Doyon, Jean-Claude Piedbœuf, and Yves Gonthier. SYMOFROS: Software architecture and real time issues. In *DA-SIA 2000 - Data Systems in Aerospace*, Montreal, Canada, 22-26 May 2000.
- [5] Bruno Monsarrat and Clément Gosselin. Jacobian matrix of general parallel and hybrid mechanisms with rigid and flexible links: A software-oriented approach. In *Proceedings of the 2002 ASME Design Engineering Technical Conferences*, 2002.
- [6] B. Moore. Optimisation de la génération symbolique du code et estimation des paramètres dynamiques de structures mécaniques dans SYMOFROS. Master's thesis, Department of Mathematics, Université du Québec à Trois-Rivières, 2000.
- [7] J.-C. Piedbœuf. Modelling flexible robots with maple. *Maple Tech: The Maple Technical Newsletter*, 3(1):38-47, 1996.
- [8] Jean-Claude Piedbœuf. Recursive modelling of flexible manipulators. *The Journal of Astronautical Sciences*, 46(1), January-March 1998.
- [9] Jean-Claude Piedbœuf, F. Aghili, M. Doyon, Y. Gonthier, E. Martin, and W.-H. Zhu. Emulation of space robot through hardware-in-the-loop simulation. In *The 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space: i-SAIRAS 2001*, Canadian Space Agency, St-Hubert, Quebec, Canada, June 18-22 2001.
- [10] José Ignacio Rodríguez, José Manuel Jiménez, Francisco Javier Funes, and Javier García de Jalón. Dynamic simulation of multi-body systems on internet using corba, java and XML. In *USACM, Sixth U.S. National Congress on Computational Mechanics*, page 398, 2001.
- [11] Pengfei Shi and John McPhee. Symbolic programming of a graph-theoretic approach to flexible multibody dynamics. *Mechanism of Structures and Machines*, 30(1):123-155, 2002.