ECCOMAS Thematic Conference on Multibody Dynamics
July 24 - 28, 2023, Lisbon, Portugal

Dynamics of a 3R spatial robot based on a GPU approach

Louis Guigon[1], **Benjamin Boudon**[1], **Andrés Kecskeméthy**[2]

[1] Université Clermont Auvergne, CNRS,
SIGMA Clermont, Institut Pascal
F-63000 Clermont-Ferrand, France
[guigon.louis, boudon.benjamin]@sigma-clermont.fr

[2] Chair of Mechanics and Robotics
University Duisburg-Essen
Lotharstr. 1, 47057 Duisburg, Germany
andres.kecskemethy@uni-due.de

ABSTRACT

This paper presents an approach to solve the dynamics of a mechanism using parallel processing on Graphic Processing Units (GPU). Starting from the SYMKIN package [1, 2] which generates the kinematics and the dynamics of any system using the symbolical software Mathematica, a paralleling process has been designed to maximize the number of equations which can be performed at the same time on the GPU. To do so, several techniques have been used to group equations into blocks of parallel calculations. Then, blocks are translated and exported to a CUDA environment in order to be processed on the GPU. Finally, an automated generation of the parallel blocks is made to ease the use of the package, permitting to the user to obtain a working Visual Studio project simply by entering the system parameters. The implementation of the method is currently underway and concrete execution times will be presented at the conference. These could not be included here due to not yet finished coding but it is hoped that the concepts presented here are suitable to understand procedure for efficient GPU kinematical and dynamical simulation.
Keywords: Symbolic equations, Parallel processing, Dynamics, Hard-Real Time.

1 INTRODUCTION

In modern robotics, the use of complex models is often required to obtain better performances. However, those models result in a heavy load of calculations for processors due to their complexity. A first example can be the use of a digital twin for immersive simulations where the simulated scenario and the robot motion should be synchronized in Real-Time to obtain the "immersive effect" as does the Kuka Robocoaster [3]. Another example is the processing of complex controls such as computed torque for robot dynamics which results in a gain of precision but an heavy load of calculation on processors [4].

Since the beginning of the 1980's, Graphic Processing Units became famous in arcade games first, and then for displaying images on computer screens. Designed as a support for the Central Processing Unit (CPU) to process graphics, the GPU is capable to calculate hundreds of calculations at the same time thanks to its multi-processor architecture. Nowadays, the GPU is mainly developed for video games to simulate detailed environments and also for general processing tasks thanks to the General Purpose on Graphic Processing Units (GPGPU) [5]. GPGPU is a major contribution to open GPU architectures to research and general applications. For instance, linear algebra and matrix multiplication are a common example to prove the benefits of a parallel architecture [6]. A hard competition between GPU manufacturers results in a fast evolution of this parallel architecture, with an average of one new architecture commercialized every two years. In other words, the GPU is a modern tool which is accessible as each computer has one (integrated or dedicated) and the development of new architectures pushes back the limits of parallel processing. Finally, the GPU is not bound to decline soon as it is the main organ of multimedia advertisement.

2 DEVELOPMENT

The sum-up of the project is presented in Fig. 1. The developments focus on different points to automate the generation of GPU code given a system description as an input. The process

generates the equations of motion from the description, thanks to the SYMKIN package, and then parallelize them in different levels and techniques. Afterwards, Parallel calculations are grouped into blocks which are translated and exported to a GPU-compatible code. In the context of the project, CUDA is the chosen environment as it is a well documented and efficient interface but the algorithm is flexible enough to adapt to other environments such as OpenCL. Finally, the code is automatically exported to a Visual-Studio project which is ready for execution.

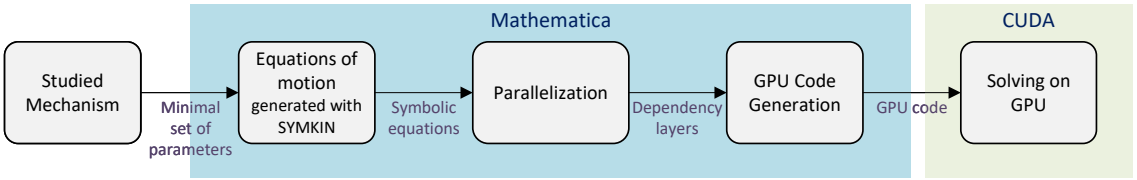

Figure 1. Generation of the GPU code from a given mechanism.

2.1 Generation of dynamic equations

Prof A.Kecskeméthy designed SYMKIN, a Wolfram Mathematica package to generate symbolically the equations of motions (EoMs) [1, 2]. SimKin helps the designer to obtain a minimal coordinate formulation of the EoMs with as an input a minimal set of parameters and closed form solutions of the geometry equations. The result is an ordered set of symbolic equations of the EoMs in a ODE form.

A first approach of parallel processing has been considered in the SYMKIN context, using a CO-ordinate Rotation in DIgital Computer (CORDIC) FPGA board [7]. With an optimization of the pipeline and the processing in-parallel of arrays, the processing time has been reduced by a factor of 15 with respect to a numerical solver.

2.2 Parallel processing

The current work keeps the idea of paralleling calculations, now using GPUs. GPGPU proposes many environments to code this parallel architecture. As an example, NVIDIA developed CUDA which let the users program their GPU using a concept of pointers to parallel their algorithms. As shown on Fig. 2, a hierarchy of fictive elements is designed to control the threads of a GPU. A thread is an independent sequence of instructions placed at the bottom of this hierarchy as they are parallelized into blocks and blocks into a grid. This distribution is designed to spread memory and avoid bottlenecks in memory accesses as each element in the hierarchy has its dedicated memory:

- Local memory for threads
- Shared memory for blocks
- Global memory for the grid

In order to point threads in the algorithm, pointers have been introduced. In this sense, the selection of a thread through a block is done by `threadIdx` and a block through the grid is done by `blockIdx`. Those pointers are extended to the dimension used in the code with a suffix `x`, `y` or `z` for the three dimensions.

Thus, the use of pointers in the algorithm informs the CUDA compiler that operations have to be processed in parallel. To illustrate this concept, the `for` loop, often used in many programming languages, uses an incremental pointer which evolves for each iteration of the loop. In parallel processing, the incremental pointer is replaced by the parallel pointers presented above as described in Tab. 1.

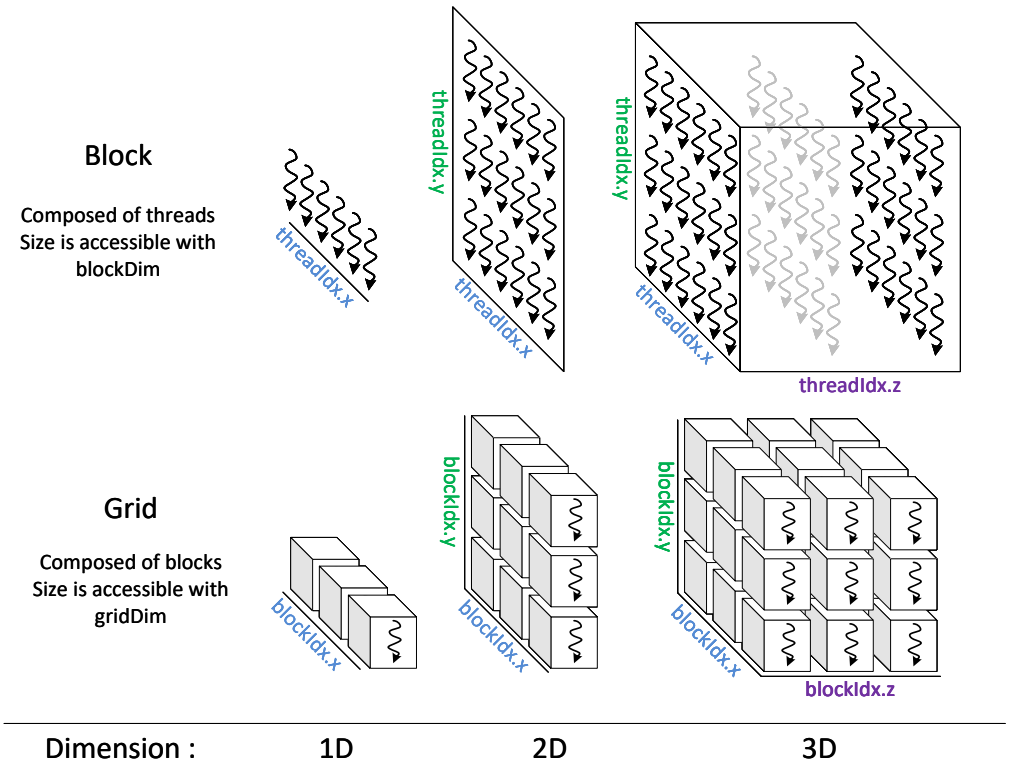

Figure 2. Description of CUDA pointers for GPGPU.

Table 1. Example of a vector addition using serial and parallel programming.

	Algorithm
Serial	`for(int i = 0; i < 10; i++) { C[i] = A[i] + B[i] };`
Parallel	`int i = threadId.x; C[i] = A[i] + B[i];`

Parallel processing is efficient for processing data in parallel but the main constraint is the dependency between this data. Consequently, linear algebra is easily parallelizable as each element of a matrix can be calculated independently from the others. However, the concept of parallelism becomes hard to figure out for serial processes. The parallelization can be operated on different levels. The first level concerns generated equations which have to be sorted with respect to their dependency and the second level parallelizes the mathematics to perform an equation in a minimal amount of steps.

2.2.1 Equation dependancy

Concerning the equations, SYMKIN outputs a set of minimal equations in processing order. Using this set, the paralleling issue is to sort independent equations and to group them in parallel blocks. A tree is built on Fig 3a to get linkages between equations, named V_i. It also results in a better overview of the parallel layers, corresponding to the lines of the tree. In the work of Postiau [8], the notions of processing "as soon as possible" and "as late as possible" are introduced to describe two methods of paralleling. In the first method Fig. 3a, equations are processed as late as possible. The second method, "as soon as possible", is presented on Fig. 3b. Using this method, the process shows several benefits to solve the set of equations:

- Elementary calculations are sorted on the first layer (bottom of the tree) which highlights redundant calculations.

- Calculations including system variables are grouped on the first layer which allows trigonometric simplifications as a processor calculates using the CORDIC method which calculates the sine and cosine simultaneously.
- The GPU workload to calculate this set is known as the first layer will have the most equations to proceed and then the workload will decrease with steps as shown on graphs of Fig. 3b

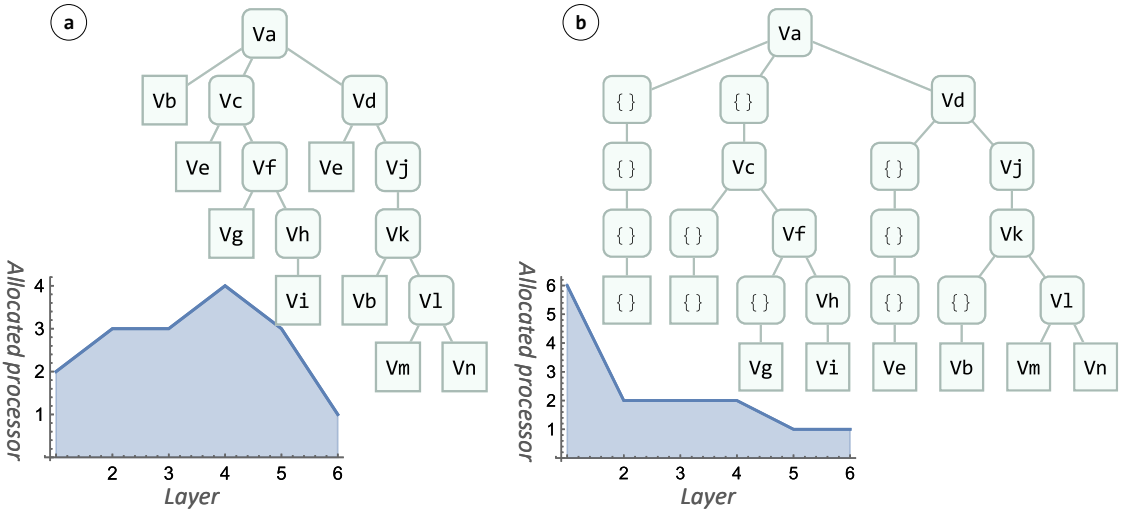

Figure 3. Trees of dependancy between equations, sorted for calculation *as late as possible* (a) and *as soon as possible* (b).

The method used in the parallelization process is "as soon as possible" due to the benefits it occurs for algorithm simplifications. After paralleling equations, it is important to study the mathematics which defines each equation.

2.2.2 Mathematical operation

Mathematics are regulated with strict rules which ensure a good result for a specific calculation. To parallelize mathematics, it is important to define clearly the priority of calculations. This is where the brackets justify their importance. As an example, the addition of seven variables as presented in Fig. 4 is a simple modification of priority in the sum:

$$sum = a+b+c+d+e+f+g \tag{1}$$

$$Serial : sum = (((((a+b)+c)+d)+e)+f)+g \tag{2}$$

$$Parallel : sum = ((a+b)+(c+d))+((e+f)+g) \tag{3}$$

In Eq. 2, the brackets are placed to show how a serial processor sees and sums the seven elements. Each step requires to calculate the current addition, save it and iterate to next addition until the seven variables have been added. However, Eq. 3 is designed for a parallel execution. It requires more resources as the first step process three additions and save the three results but this method divides by two the number of steps required to calculate the sum.

Paralleling mathematics is a case by case study as all equations are different but some properties remain identical between different equations:

- Each equation is an operation of one or two elements, which are consecutively an operation of two sub-elements until obtaining a constant or variable.
- Successive operations of same type can be grouped by two to calculate them in parallel.

Using the properties, the paralleling scheme for mathematics is first the determination of priorities and then the process can be assimilated to the scheme applied to the set of equations.

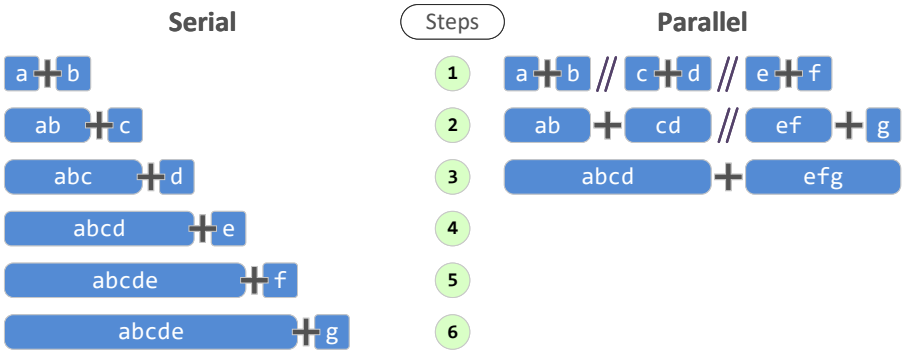

Figure 4. Parallelization and step reduction using parallel processing for a sum.

3 EXPORT

Most GPGPU environments are designed on the C language with bindings to the GPU dedicated language. In other words, launching a GPU kernel is to call it through a C script using structures imposed by the chosen environment. For instance, Tab. 2 shows the syntax for two different environments, CUDA and OpenCL, for two operations. This example shows that switching from an interface to another can be done by choosing another syntax. Thanks to the concept of rules introduced in Mathematica software, this switching is possible with the definition of rules for each environment. Those rules are then used to replace the desired function for the chosen environment when generating the code.

Table 2. GPU environment bindings

Environment	Kernel call	Pointer
CUDA	`__global__ name(){}`	`threadIdx`
OpenCL	`__kernel void name(){}`	`get_local_id()`

The code is generated in blocks of parallel calculations which are then added to kernel files linked to visual studio project. As a result, The exported code is a working Visual Studio project containing the function which calculates the equations of the system on the GPU.

4 APPLICATION TO THE 3R SPATIAL ROBOT

To process the algorithm, a serial chain has been chosen with an example of a spatial 3R robot (it corresponds to the 3 first degree of freedom (DoF) of a classic serial robot with the Dofs associated to the wrist blocked) described in Fig. 5. The robot is actuated on three revolute joints around the following directions and parameters : $\{\theta_1, \vec{z}_0\}$, $\{\theta_2, \vec{y}_1\}$ and $\{\theta_3, \vec{y}_2\}$.

The equations of motion have been generated with the function *GenerateEqm* implemented in SymKin. This function calculates the global kinematics of the input system and then generates

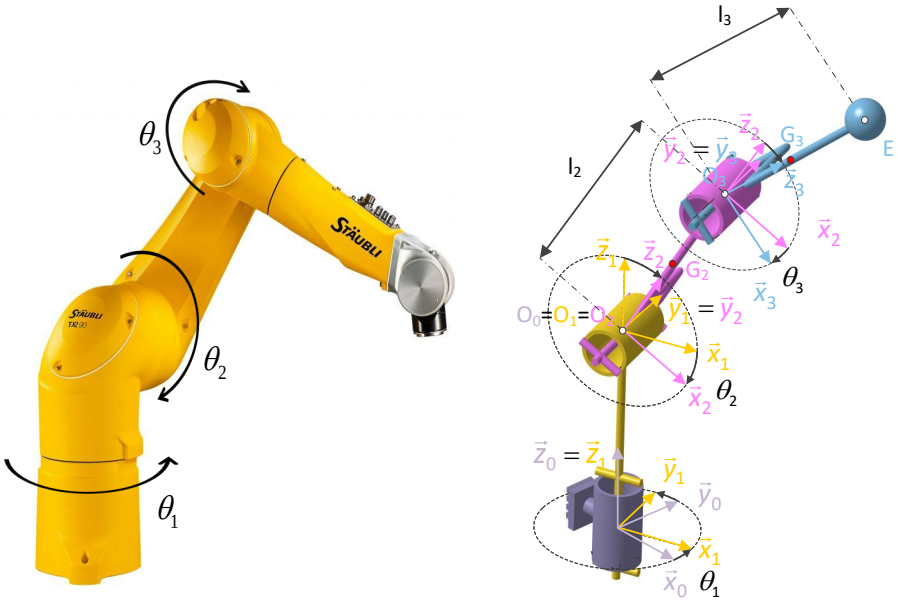

Figure 5. Parameterized kinematic scheme of a commercial spatial 3R robot.

the equations of motion with the combined use of d'Alembert's principle and backward recursive Newton-Euler algorithm [2].

The Tab. 3 shows preliminaries results concerning the parallelism of the 3R equations of motion with different processing methods. The minimal set generated by SYMKIN is in this case study composed of 68 equations. The first method shows the serial execution which corresponds to an execution of the equations one after each other. In this case 68 equations are proceeded which represents 306 steps of calculation for the mathematics. On the other hand, the parallelization process groups the 68 equations with respect to their dependency, which results in 9 blocks of equations to be executed serially. After the mathematical parallelization, the resolution of the equations takes only 40 steps of execution.

Table 3. Step reduction for the processing of a 3R robot equations of motion.

Processing method	Steps for equations	total number of operations processed serially
Serial computation	68	306
Parallel computation	9	40

5 CONCLUSION

The present work shows the step reduction in solving the equations of motion for a serial chain case. Results have shown that the parallelization of the equations and the mathematics generated by SYMKIN reduces the number of steps to solve them by more than 7. The implementation of the method is currently underway and concrete execution times will be presented at the conference. These could not be included here due to not yet finished coding but it is hoped that the concepts presented here are suitable to understand procedure for efficient GPU kinematical and dynamical simulation.

This first result is also promising as the parallelization process is considering only the set of equations and the mathematics. A further research on dynamic formulations (Kinetostatic transmission

elements [9], Composite-Rigid-Body [10], Articulated-Rigid-Body [11] reduced in different reference points and expressed in different frames) for different topology will be analyzed, taking into account paralleling issues. Parallelization of specific chains with an adapted formulation should improve the results for a better step reduction of the equations of motion.

REFERENCES

[1] Kecskeméthy, A., Krupp, T.: Application of symbolical kinematics to real-time vehicle dynamics. Technical report, European Research Office of the U.S. Army (1995)

[2] Kecskeméthy, A., Krupp, T., Hiller, M.: Symbolic processing of multiloop mechanism dynamics using closed-form kinematics solutions. MSD **1**(1) (1997) 23–45

[3] Kecskeméthy, A., Masic, I., Tändl, M.: Workspace fitting and control for a serial-robot motion simulator. In Ceccarelli, M., ed.: Proceedings of EUCOMES 08, Dordrecht, Springer Netherlands (2009) 183–190

[4] Kingsley, C., Poursina, M., Sabet, S., Dabiri, A.: Logarithmic complexity dynamics formulation for computed torque control of articulated multibody systems. Mechanism and Machine Theory **116** (2017) 481–500

[5] Wu, E., Liu, Y.: Emerging technology about gpgpu. In: APCCAS 2008 - 2008 IEEE Asia Pacific Conference on Circuits and Systems. (2008) 618–622

[6] Matsumoto, K., Nakasato, N., Sedukhin, S.G.: Performance tuning of matrix multiplication in opencl on different gpus and cpus. In: 2012 SC Companion: High Performance Computing, Networking Storage and Analysis. (2012) 396–405

[7] Krieger, C., Hosticka, B., Krupp, T., Hiller, M., Kecskeméthy, A., Hosticka, B.J.: A combined hardware/software approach for fast kinematic processing. Microprocessors and Microsystems **22** (9 1998) 263–275

[8] Postiau, T.: Génération et Parallélisation des Équations du Mouvement de Systèmes Multicorps par l'Approche Symbolique. PhD thesis, Université catholique de Louvain (2004)

[9] Angeles, J., Kecskeméthy, A.: Dynamics Modelling. Springer (1995)

[10] Walker, M.W., Orin, D.E.: Efficient Dynamic Computer Simulation of Robotic Mechanisms. Journal of Dynamic Systems, Measurement, and Control **104**(3) (09 1982) 205–211

[11] Featherstone, R.: The calculation of robot dynamics using articulated-body inertias. The international journal of robotics research **2**(1) (1983) 13–30