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

Louis Guigon[1], Benjamin Boudon[1], Andres 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

EXTENDED ABSTRACT

1 Introduction

In computer technologies, the Graphic Processing Unit (GPU) has proved its efficiency in the last years, dealing with high range of parallel computations at the same time. With its main application in image processing and displaying, GPU manufacturers improve continuously their hardware to get the best performance for users. The GPU architecture is manufactured with hundreds of processors in parallel to perform many independent calculations at the same time while Central Processing Units (CPU) performs it serially. A famous example to prove the GPU performance is the dot product of two matrices (GEMM) [1].

Solving the Kinematics and Dynamics of complex multibody systems requires a lot of processing resources due to the amount of equations to solve. To answer this issue, Prof A.Kecskeméthy designed SymKin, a Wolfram Mathematica package to generate symbolically the equations of motions (EoMs) [2, 3]. 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. In order to pursue this optimization, a numerical solver has to be designed to calculate the solution of the symbolic equations set. A research work has been done on a FPGA parallel architecture using CORDIC algorithm [4] to speed up trigonometric calculations. The CORDIC emulator board provided a speed-up coefficient of 15 with a traditional numerical solution. The current project pursues the objective of minimizing the equations solving time using modern parallel architectures such as GPUs.

2 Problem description

The aim of the project is to design and automate the generation of the system of equations into GPU code corresponding to any multibody system kinematic and dynamics. In other words, the contribution mobilises the potential of parallelism in a system of equations to process a maximum calculations at the same time on a GPU. In addition, the GPU code must be automatically generated and further optimized from any set of equations. This generation process is an interface between the Wolfram source language and the targeted GPU language. In the context of this project, the CUDA environment is used with C and PTX assembly languages. However, the generation process must be flexible to incorporate other output environments such as OpenCL.

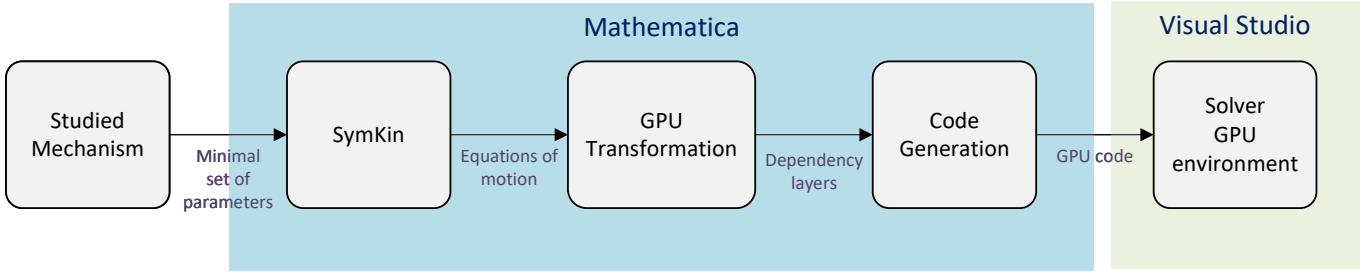

Figure 1: Generation of the GPU code from a given mechanism

3 Applied methods

The first point is to select the correct environment to develop the GPU code. The choice is focused on a low level language to access the primitives of the GPU. In this research, the code is tested on an NVIDIA GTX 1060 architecture with the CUDA environment [5]. Since manufacturers lock their architecture for different reasons such as competition or compatibility between different architectures, the lowest level of the code is not accessible by the user. However, in case of NVIDIA GPUs, an Instruction Set Architecture (ISA) is available and it is called Parallel Thread Execution (PTX). At this level of the code, it is possible to code with an assembly like language in order to design kernels by hand. The designed code is also compared with other native libraries such as NVIDIA cuBlas for linear algebra. This permits the comparison with higly optimized codes in order to get the best improvements of the code.

Then, symbolic equations are optimized with respect to their independence. The objective is to perform the maximum independent calculations at the same time. This step is done with Wolfram language to benefit from the symbolical processing that speed up calculations. Afterwards, the optimized equations are converted in a GPU compatible language. In this project, the code is a hybrid combination of CUDA C and PTX languages. The code is automatically generated thanks to built-in Wolfram libraries such as SymbolicC. The generation is done for NVIDIA environment but it is also flexible for other ones, such as OpenCL.

Finally, the result is a readable, writable and executable Visual Studio project which solves the equations of the chosen system on the GPU. The sum up of the project is presented on the figure 1.

4 Results

The first implementations have been done on the dynamics of a 3R spatial robot (it corresponds to the 3 first degree of freedom (DoF) of a classic serial robot with the Dofs associated to the wrist blocked) with the function *GenerateEqm* implemented in SymKin. This function calculates the global kinematics of the input system and then generates the equations of motion with the combined use of d'Alembert's principle and backward recursive Newton-Euler algorithm [3]. The numerical results obtained with the GPU solver are currently in process to compare them with a traditional CPU solver.

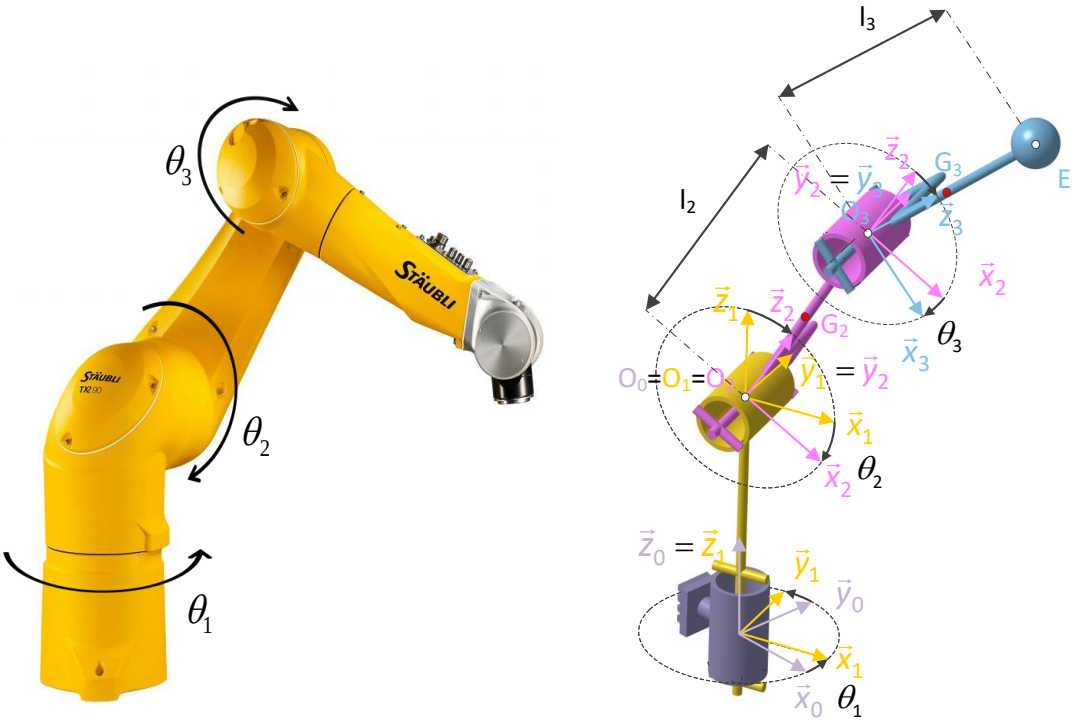

Figure 2: Parameterized kinematic scheme of the spatial 3R robot.

References

[1] Kazuya Matsumoto, Naohito Nakasato, and Stanislav G. Sedukhin. Performance tuning of matrix multiplication in opencl on different gpus and cpus. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 396–405, 2012.

[2] Andrés Kecskeméthy and Thorsten Krupp. Application of symbolical kinematics to real-time vehicle dynamics., 8 1995.

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

[4] Christoph Krieger, B. Hosticka, Thorsten Krupp, Manfred Hiller, Andrés Kecskeméthy, and Bedrich J. Hosticka. A combined hardware/software approach for fast kinematic processing. *Microprocessors and Microsystems*, 22:263–275, 9 1998.

[5] NVIDIA. *CUDA Toolkit documentation*, 2022.