Enhancing the Performance of the Divide-and-Conqure Algorithm When Forming and Solving the Equations of Motion for Multibody Systems

Jeremy J. Laflin[*], **Kurt S. Anderson**[#], **Michael Hans**[*]

[*]Computational Dynamics Lab
Rensselaer Polytechnic Institute
110 8th Street, Troy, NY, USA
laflij@rpi.edu, hansm@rpi.edu

[#]Professor of Aerospace Engineering
Rensselaer Polytechnic Institute
110 8th Street, Troy, NY, USA
anderk5@rpi.edu

Abstract

Since computational performance is critically important for simulations to be used as an effective tool to study and design dynamic systems, the computing performance gains offered by GPUs cannot be ignored. The GPU has been used to increase the computational performance of many tasks necessary to simulate multibody systems [1–5]. Since the GPU is designed to execute a very large number of simultaneous tasks (nominally Single Instruction Multi-Data (SIMD)), recursive algorithms in general, such as the DCA, are not well suited to be executed on GPU-type architecture. This is because each level of recursion is dependent on the previous level. Therefore, all tasks associated with the algorithm cannot be executed independently. The primary issue is the large amount of data transfer that must occur when moving from one level of recursion to the next. However, there are some ways that the GPU can be leveraged to increase computational performance when using the DCA to form and solve the equations of motion for articulated multibody systems with a very large number of degrees-of-freedom.

Computational performance of dynamic simulations is highly dependent on the nature of the underlying formulation and the number of generalized coordinates used to characterize the system. If not done intelligently, multibody formulations, due to their kinematic coupling between the equations, require cubic ($O(N^3)$ with N generalized coordinates) computational expense per temporal integration step to form the equations of motion and subsequently solve for the system state derivatives. Therefore, algorithms that scale in a more desirable (lower order) fashion with the number of degrees-of-freedom are generally preferred when dealing with large ($N \gg 10$). However, the utility of using simulations as a scientific tool is directly related to actual compute time. The DCA, and other top performing methods, have demonstrated desirable scaling properties of the compute time required being linear ($O(N)$) with increasing number of degrees-of-freedom (N) and sublinear ($O(\log(N))$ performance when implemented in parallel. However for the DCA, total compute time could be further reduced by exploiting the large number of independent operations involved in the first few levels of recursion.

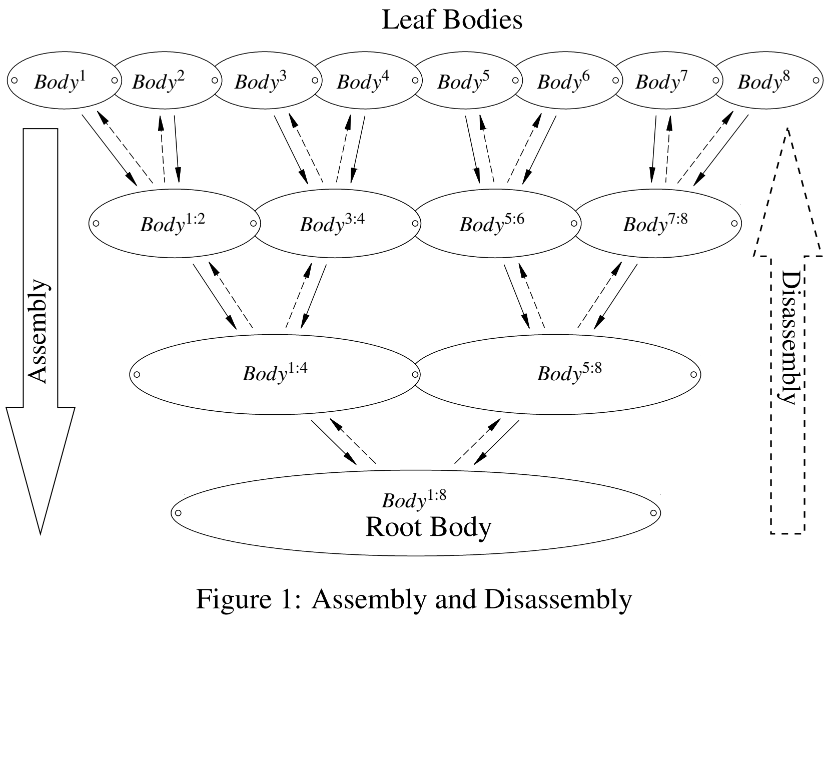

Figure 1: Assembly and Disassembly

For articulated multibody systems, the DCA is often implemented in a binary tree structure, see Fig. 1, and consists of two key tasks, assembly and disassembly. In the assembly sweep, the inertial properties and applied loads of adjacent bodies are assembled to produce the inertial properties and applied loads of a fictitious assembled-body. In the disassembly portion of the algorithm, the constraint forces in the system are determined, which requires that the inertial properties of the assembled-bodies be disassembled. Since the assembly (disassembly) processes for one pair of bodies is uncoupled from the assembly (disassembly) process of another pair of bodies (at the same level of recursion), the DCA would benefit from implementing the assembly and disassembly processes on GPU architecture for those levels that contain a relatively high number of bodies. For systems with a very large number of degrees-of-freedom, a much larger number of uncoupled assembly and disassembly processes can be executed in parallel on a GPU-type device than can be done in parallel using a traditional Central Processing Unit (CPU), even with many cores. However, after the assembly and disassembly operations have been performed for these levels, and the number of bodies per level decreases rapidly as $\left(\frac{1}{2}\right)^L$ (where L is the number of levels traversed by the algorithm) so, these operations are more efficiently executed on the CPU.

A simple chain-type pendulum example is used to explore the feasibility of using the GPU to execute the assembly and disassembly operations for the levels of recursion that contain enough bodies for this process to be computationally advantageous. A multi-core CPU is used to perform the operations in parallel using Open MP for the remaining levels. The number of levels of recursion that utilizes the GPU is varied from zero to all levels. The data corresponding to zero utilizing the GPU provides the reference compute-time in which the assembly and disassembly operations necessary at each level are performed in parallel using Open MP. The computational time required to simulate the system for one time-step where the GPU is utilized for various levels of recursion is compared to the reference compute time also varying the number of bodies in the system. A decrease in the compute-time when using the GPU is demonstrated relative to the reference compute-time even for systems of moderate size.

References

[1] N. Khude, I. Stanciulescu, D. Melanz, and D. Negrut. Efficient Parallel Simulation of Large Flexible Body Systems With Multiple Contacts. *Journal of Computational and Nonlinear Dynamics*, 8(4):041003, Mar. 2013.

[2] H. Mazhar, T. Heyn, and D. Negrut. A scalable parallel method for large collision detection problems. *Multibody System Dynamics*, 26(1):37–55, 2011.

[3] D. Melanz, N. Khude, P. Jayakumar, and D. Negrut. A Matrix-Free Newton–Krylov Parallel Implicit Implementation of the Absolute Nodal Coordinate Formulation. *Journal of Computational and Nonlinear Dynamics*, 9(1):011006, Oct. 2013.

[4] D. Negrut, R. Serban, H. Mazhar, and T. Heyn. Parallel Computing in Multibody System Dynamics: Why, When, and How. *Journal of Computational and Nonlinear Dynamics*, 9(4):041007, July 2014.

[5] D. Negrut, A. Tasora, H. Mazhar, T. Heyn, and P. Hahn. Leveraging parallel computing in multibody dynamics. *Multibody System Dynamics*, 27(1):95–117, 2012.