한국동력기계공학회지 제16권 제4호 pp.12~16 2012년 8월

평면 다물체 동역학 해석에서 GPU 병렬 프로그래밍의 계산효과
Calculation Effect of GPU Parallel Programing for Planar Multibody System Dynamics

전철웅* · 손정현**†
C. W. Jun* and J. H. Sohn**†

(접수일 : 2012년 01월 27일, 수정일 : 2012년 05월 07일, 채택확정 : 2012년 05월 08일)

Key Words : Multibody dynamics(다물체동역학), CUDA Programing(CUDA 프로그래밍), Parallel Computing(병렬 컴퓨팅), GPU(그래픽 처리 장치)

Abstract : In this paper, the equations of motions for planar multibody dynamics are established for considering the parallel programming based on GPU. Cartesian coordinates are used to formulate the equations of motion and implicit integration method called HHT-alpha is employed. Open chain multibody system is considered for computer simulation. CUDA toolkit is employed for establishing the GPU parallel programming. The exactness of the analysis is verified from the comparison with ADAMS. The results from parallel computing based on GPU are compared with the results from the sequential programming based on CPU in terms of calculation time. The multiple pendulum with bodies and joints is employed for the computer simulation. In the pendulum system that has 290 bodies, the parallel program indicates an improved efficiency of about 25.5 second(15.5% improvement). It is noted that the larger the size of system is, the time efficiency is better.

1. 서 론

오늘날의 GPU는 CPU에 비해 부동 소수점 처리를 비롯한 여러 부분의 연산 능력과 데이터 대역폭 면에서 훨씬 앞선 성능을 발휘하게 되었다. 기존 GPU의 용도는 단순한 그래픽 작업 처리의 가속화였지만, 현재에는 그래픽 연산 처리를 목적으로 개발된 프로그래밍 가능한 GPU를 본래의 용도가 아닌 범용 연산의 용도로 활용하는데 목적을 둔 GPGPU(General Purpose computing on GPU)로 많은 개발이 이루어지고 있다. 많은 처리장치와 넓은 대역폭을 가지고, 빠른 연산 처리에 뛰어나다는 GPU의 장점을 병렬처리로 연산 횟수를 크게 줄일 수 있는 연산 또는 빠른 정보 처리를 요구하는 실시간 연산처리에 적용시킨다면 효율적인 결과를 얻을 수 있다. NVIDIA는 GeForce 8800 GTX에서 계산 및 그래픽스 모두를 위한 제품을 소비자들에게 제공하기 위해 CUDA 구조 기반 설계를 시작으로 이후의 NVIDIA 그래픽 장치에는 CUDA 구조를 기반으로 생산되고 있다. NVIDIA는 장치 생산뿐만 아니라 GPU를 이용한 범용적인 프로그램을 개발 할 수 있도록 '프로그램 모델', '프로그램 언어', 컴파일러', 라이브러리', '디버거', '프로파일러'를 제공하는 통합 환경을 구축하였다. CUDA를 기존 CPU 프로그램에 적용하면 기존 성능보다 수십에서 수백 배에 달하는 놀라운 성능 향상 효과를 얻을 수 있다. CUDA를 적용하는 연구는 산업, 설계, 의학, 교육, 금융 등 다양한 분야에서 연구가 이루어지고 있으며 성공 사례 또한 수없이 많다.

다물체 동역학의 순차 해석은 많은 반복문의 실행이 불가피하다. 이런 반복들은 시스템의 크기가 커질

**† 손정현(교신저자) : 부경대학교 기계자동차공학과
E-mail : jhsohn@pknu.ac.kr, Tel : 051-629-6166
*전철웅 : 부경대학교 대학원

**† J. H. Sohn(corresponding author) : Department of Mechanical & Automotive Engineering, Pukyong National University.
E-mail : jhsohn@pknu.ac.kr, Tel : 051-629-6166
*C. W. Jun : Department of Mechanical & Automotive Engineering, Pukyong National University.

수록 계산 시간의 증가로 실시간 해석의 어려움을 드러내었다. GPU 병렬 계산은 순차 해석의 반복문들의 병렬성을 활용하여, 필요한 계산을 동시에 수행함으로써 계산 시간의 효율을 상승 시킬 수 있다. 또한, 기존의 긴 해석 시간의 필요로 인해 접근하기 어려웠던 문제들에 대해서 GPU를 활용한 병렬 계산은 새로운 대안법이 될 수 있다. 현재 병렬 계산의 장점을 활용하여 다물체 동역학 부문에서의 GPU를 활용하고자 하는 연구가 활발하게 이루어지고 있다. NVIDIA에서는 GPU를 활용한 많은 예제들을 SDK로 제공하여 있으며, 이연희[1]는 CUDA 컴퓨팅을 이용한 충돌검사 및 동역학 시뮬레이션 연구를 수행하였고, Dan Negrut[2]은 병렬 계산을 통해 복잡한 다물체 시스템과 다물체 간의 접촉을 연구하였다. 하지만, 다물체 시스템의 동역학 해석에 있어서 어떠한 부분이 GPU에서 효율적으로 병렬화 될 수 있는지에 관하여서는 발표되지 않고 있다.

본 연구에서는 구속된 평면 다물체 동역학 해석에서 물체 수 및 구속의 증가에 따라 어떠한 계산 과정이 GPU를 통해 효율적으로 프로그래밍 될 수 있는지를 연구하였다. 선형 방정식 해석과 적분기 전체 과정에 대해서 CPU에서 기존 순차 해석을 수행했을 때와 병렬 프로그래밍을 통한 GPU에서의 계산 속도를 비교하였다.

2. GPU 프로그래밍의 개요

CUDA 프로그래밍은 CPU와 GPU장치를 각각 호스트(Host)와 디바이스(Device)로 나눈다. GPU 프로그래밍의 계산 과정에서 호스트 메모리와 디바이스 메모리가 서로 정보를 교환하며 계산을 수행하기 때문에 디바이스 메모리의 할당과 호스트에서 디바이스로의 메모리 복사과정이 필요하다. 또한 디바이스에서 계산된 결과를 호스트 메모리로 가져오기 위한 복사 과정이 필요하다. Fig. 1은 CUDA를 이용한 데이터 처리 과정을 나타낸다. 데이터 처리 과정에서 호스트와 디바이스간의 데이터 전송은 비용이 큰 과정이다.

본 연구에서는 시스템의 초기상태가 결정되었을 때 호스트의 모든 정보를 디바이스 메모리에 넘겨줌으로써 효율을 높였다. GPU의 코어는 연산을 위한 최소한의 단위로 구성하여 스레드를 할당하고 실행한다.

CUDA의 스레드는 계층의 구조로 되어 있는데,

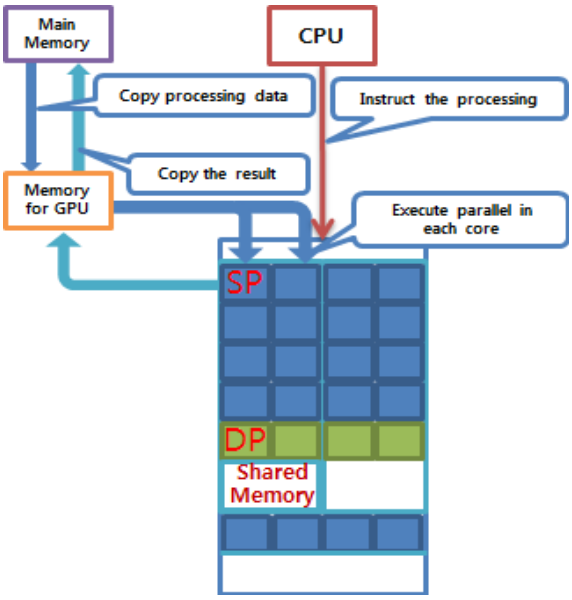

Fig. 1 Data processing flow on CUDA[3]

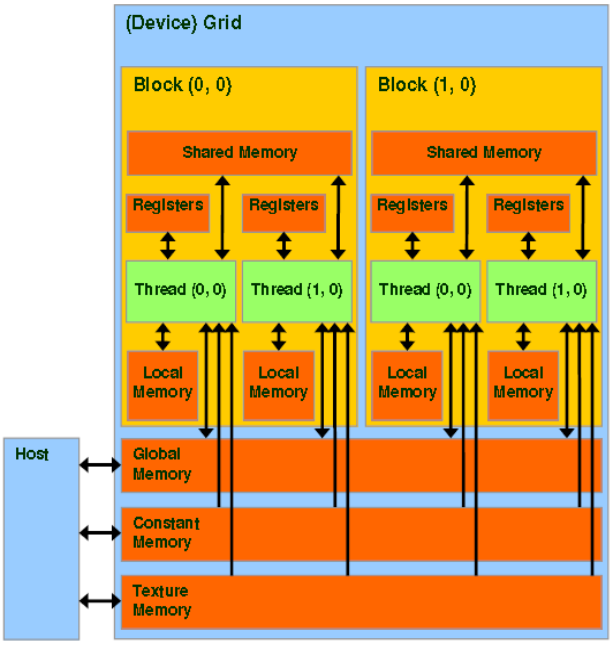

Fig. 2 Overview of CUDA device memory[4]

스레드-블록-그리드(Thread - Block - Grid)로 이루어져 있다. 스레드가 모여 블록을 이루고 블록이 모여 그리드를 이루게 된다. 각 블록은 하나의 커널함수를 가지고 블록내의 각 스레드가 같은 커널함수를 실행함으로써 병렬 계산이 이루어지게 된다. 행렬 연산이나 복잡한 데이터 연산을 돕기 위해 그리드의 차원은 1, 2, 3차원이 모두 가능하고, 각각의 블록과 스레드는 유일한 ID를 가지며, 커널 함수 내에서 blockIdx, threadIdx로 내장 변수(built-in variable)로 블록과 스레드의 위치를 결정한다.

Table1 Salient features of device memory[5)]

Memory	Location on/off chip	Cached	Access	Scope
Register	On	No	R/W	1 thread
Local	Off	No	R/W	1 thread
Shared	On	No	R/W	all threads in block
Global	Off	No	R/W	all threads + host
Constant	Off	Yes	R	all threads + host
Texture	Off	Yes	R	all threads + host

Fig. 2는 디바이스의 메모리와 스레드 사이의 데이터 교환 경로를 보여주고 있으며, Table1에 메모리 특성을 나타내었다. 실제 계산을 수행하는 스레드는 레지스터(Register)를 비롯한 여러 메모리에 접근하여 주어진 계산을 수행하게 된다. 레지스터는 직접 연산을 수행하는 가장 빠른 메모리로 개당 32비트 크기이고, GPU 1사이클 이내의 속도로 읽고 쓰기가 가능하다. 로컬(Local) 메모리는 읽고 쓰는 속도가 GPU 400사이클에서 600 사이클까지의 시간이 소요되는 저속이다. CUDA 프로그래밍의 큰 장점 중 하나라 할 수 있는 것이 공유(Shared) 메모리이다. 16KB의 용량을 L1 캐시와 동등한 속도로 사용할 수 있다. 글로벌(Global) 메모리는 비디오 카드에 장착된 DRAM 메모리를 의미하고, 상수(Constant)메모리는 데이터를 읽기 전용으로 사용하며 캐시(cache)를 지원한다. 텍스쳐(Texture) 메모리는 원래 그래픽 전용 기능을 위해 제공되는 메모리이기에 텍스쳐 캐시는 2D 데이터 사용에 최적화되어 있다.

3. 다물체동역학 해석을 위한 프로그래밍

3.1 순차 해석 프로그래밍

본 연구에서는 구속이 있는 평면 다물체 동역학을 고려하였으며[6)], 절대 좌표계를 사용하였고 계의 일반좌표벡터는 식(1)~(3)과 같다.

$$q = [q_1, q_2, q_3, \cdots, q_{N_q}]^T \tag{1}$$

$$\dot{q} = [\dot{q}_1, \dot{q}_2, \dot{q}_3, \cdots, \dot{q}_{N_q}]^T \tag{2}$$

$$\ddot{q} = [\ddot{q}_1, \ddot{q}_2, \ddot{q}_3, \cdots, \ddot{q}_{N_q}]^T \tag{3}$$

구속된 기계 시스템의 속도와 가속도는 식(4)~(6)과 같이 표현된다.

$$\Phi_q \dot{q} = 0 \tag{4}$$

$$\Phi_q \ddot{q} - \gamma = 0 \tag{5}$$

$$\gamma = -[(\Phi_q \dot{q})_q \dot{q} + 2\Phi_{qt}\dot{q} + \Phi_{tt}] \tag{6}$$

구속에 의한 구속반력 $g^{(c)}$와 일반력 g를 고려한 시스템의 운동방정식은 식(7)~(9)와 같이 표현된다.

$$g^{(c)} = \Phi_q^T \lambda \tag{7}$$

$$M\ddot{q} - \Phi_q^T \lambda = g \tag{8}$$

$$\begin{bmatrix} M & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \end{bmatrix} = \begin{bmatrix} g \\ \gamma \end{bmatrix} \tag{9}$$

식(4)~(9)에서 M은 질량 행렬, Φ_q는 구속 자코비안, λ는 라그랑지 곱수를 나타낸다. 적분기는 $Implicit\,HHT-\alpha$[7)] 적분기를 사용하였다. HHT 법의 적분공식은 식(10)~(11)과 같다.

$$q_{n+1} = q_n + h\dot{q}_n + \frac{h^2}{2}\left[(1-2\beta)\ddot{q}_n + 2\beta\ddot{q}_{n+1}\right] \tag{10}$$

$$\dot{q}_{n+1} = \dot{q}_n + h\left[(1-\gamma)\ddot{q}_n + \gamma\ddot{q}_{n+1}\right] \tag{11}$$

미지수 β와 γ는 미지수 $\alpha \in \left[-\frac{1}{3}, 0\right]$에 의해 결정되어지는데 그 관계는 식(12)와 같고, h는 시간 간격을 나타낸다.

$$\gamma = \frac{1-2\alpha}{2}, \qquad \beta = \frac{(1-\alpha)^2}{4} \tag{12}$$

$HHT-\alpha$법의 시스템 자코비안은 식(13)과 같다.

$$\begin{bmatrix} \widehat{M} & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix} \begin{bmatrix} \triangle\ddot{q} \\ \triangle\lambda \end{bmatrix}^{(k)} = \begin{bmatrix} -e_1 \\ -e_2 \end{bmatrix} \tag{13}$$

$$e_1 = \frac{1}{1+\alpha}(M\ddot{q})_{n+1} + \left(\Phi_q^T\lambda - Q\right)_{n+1} - \frac{\alpha}{1+\alpha}(\Phi_q^T\lambda - Q)_n \tag{14}$$

$$e_2 = \frac{1}{\beta h^2}\Phi(q,t) \tag{15}$$

$$\widehat{M} = \frac{\partial e_1}{\partial \ddot{q}} = \frac{1}{1+\alpha}(M\ddot{q})_{n+1} - h\gamma\frac{\partial Q}{\partial \dot{q}} + \left[\frac{1}{1+\alpha}(M\ddot{q})q + \left(\Phi_q^T\lambda\right)_q - \frac{\partial Q}{\partial q}\right]\beta h^2 \tag{16}$$

3.2 병렬 프로그래밍

일반적인 기계 시스템을 프로그래밍을 이용하여 해석하는데 있어 크게 두 가지 부분으로 나눌 수 있는데, 여러 계산과정을 통해 행렬과 벡터를 구성하는 부분과 선형 방정식(Ax=B)의 해를 도출하는 것이다. 본 연구에서는 계산을 통해 행렬과 벡터에 값을 할당하는 과정 외에 벡터-벡터, 행렬-벡터, 행렬-행렬 간의 사칙연산은 CUBLAS 라이브러리[8]를 활용하였다. 모든 계산 과정을 GPU에서 수행하기 위해서는 행렬과 벡터에 값을 계산하고 할당하는 과정 또한 GPU에서 이루어져야 한다. 블록과 스레드의 할당을 통해 바디의 수와 조인트의 수에 비례되는 반복 계산과정을 각 스레드가 하나의 바디와 조인트의 정보를 가지고 병렬적으로 계산함으로써 계산의 효율을 높였다.

4. 시뮬레이션 결과

4.1 해의 정확성 검증

본 연구에서는 구속 다물체 시스템의 CPU와 GPU 간의 계산 효율을 검증하기 위해서 Fig. 3과 같이 여러 개의 강체가 회전조인트로 연결되어 있는 진자(pendulum) 형식의 시스템을 고려하였다. 강체는 질량이 2kg이며, 관성모멘트는 0.005(kg·m^2)이다. 시뮬레이션 시간은 1초이며, 이때 적분 간격은 0.001초이다. 총 30개의 물체로 이루어진 진자형태의 모델을 대상으로 GPU 기반 프로그램(CAMAS)과 상용 다물체 동역학 해석 프로그램인 ADAMS와 결과를 비교하였다. 진자 제일 끝단에 위치한 물체의 수평방향위치(x)와 수직방향위치(y)의 결과를 Fig. 4와 Fig. 5에 각각 나타내었고, 수치해석 결과는 상용 프로그램의 결과와 거의 일치하였다.

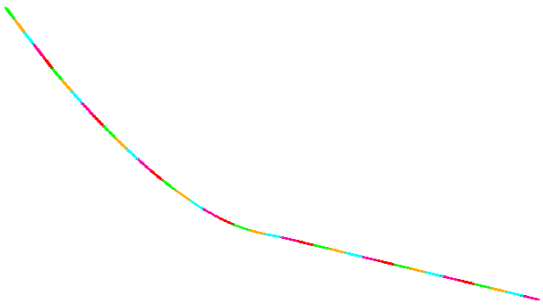

Fig. 3 The multibody pendulum model

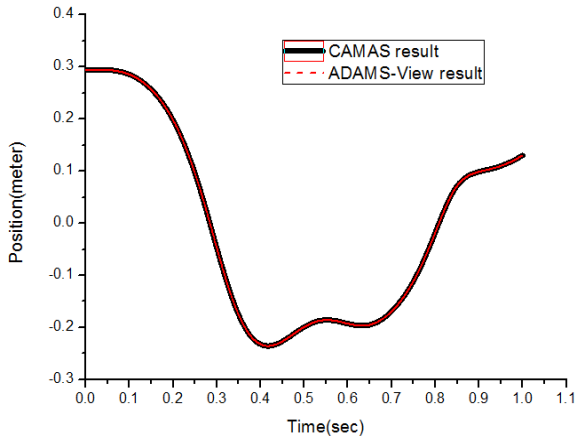

Fig. 4 Vertical displacement of the end body in the pendulum system

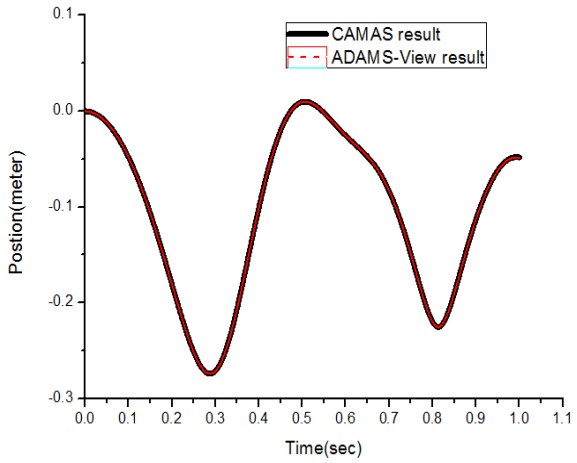

Fig. 5 Horizontal displacement of the end body in the pendulumn system

4.2 계산 효율 비교

적분과정과 선형방정식 해법 등에 GPU를 활용한 경우(case1)와 선형 방정식 해법에만 GPU를 활용한 경우(case2)에 대해서 CPU와 GPU간의 해석 시간을 비교하였다.

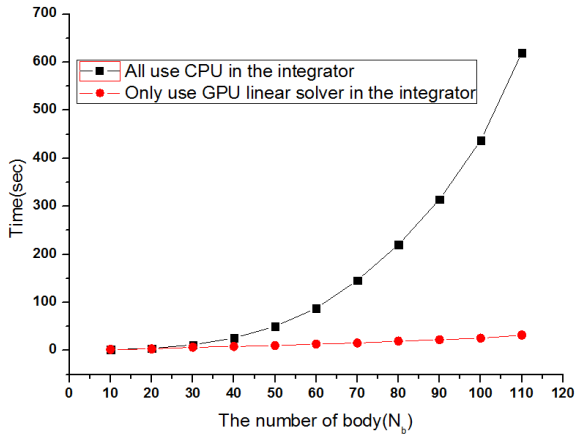

Fig. 6 Comparison of computing time between all CPU use and only GPU use for linear solver

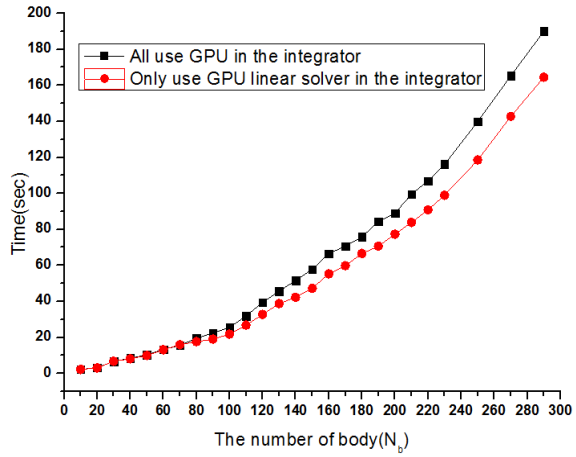

Fig. 7 Comparison of computing time between all GPU use and only GPU use for linear solver

Table 2 Specification of CPU and GPU

CPU	GPU
core i5 2500	Geforce GTX560Ti
Quard core	384 core
3.3GHz clock	880MHz core clock

본 연구에서 사용한 CPU와 GPU의 사양을 Table 2에 나타내었다. Case 1의 결과는 Fig. 6에 나타내었으며 물체의 수를 10개에서 110개까지 10개씩 증가시키며 계산 효율을 비교하였다. CPU만을 사용한 프로그램 해석은 물체가 증가할수록 지수적으로 해석시간이 증가하였다. Case 2의 결과는 Fig. 7에 나타내었으며 물체를 10~290까지 10개씩 증가 시켜가며 비교하였다. Case 2의 경우에는 증가율이 상당히 작은 것을 볼 수 있다. 물체 수가 50개 이상부터 어느 정도 효율 차이를 보이는 것을 확인 하였으며, 물체의 수가 290개일 때 약 25.5초의 시간차이가 났으며 15.5% 효율성이 향상되었다.

4. 결 론

본 연구에서는 CPU 해석기반 및 GPU 해석기반의 평면 다물체동역학 해석 프로그램을 구성하였다. GPU를 이용한 병렬 계산의 효과를 분석하기 위해서 선형 방정식 해석 루틴만 병렬화 했을 경우와 적분기 전체 루틴에 적용한 두 가지 경우에 대해서 CPU와 GPU간의 계산 효율을 비교하였다. 전체 루틴을 병렬화하여 GPU를 이용할 때는 CPU를 이용할 때에 비해 월등한 계산효과를 보여주었으며, 선형방정식만을 GPU로 계산했을 때 보다 전체 루틴을 GPU로 계산했을 때가 약 15%의 계산효과를 보였다. 다물체 동역학 해석에 있어서 GPU를 활용한 병렬 프로그래밍이 CPU의 순차 해석과 비교하여 계산 시간 효율 측면에서 많은 효과가 있음을 확인하였다.

후 기

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임(No. 2011-0011845)

참고 문헌

1. Lee, Y. H., 2009, "Parallel collision detection and dynamics simulation study with CUDA programming", Seoul, Ewha womans graduate school.
2. Tasora, A., Negrut, D. and Anitescu, M., 2011, "GPU-based Parallel Computing for the Simulation of Complex Multibody Systems with Unilateral and Bilateral Constraints" Computational Methods in Applied Science, Volume 23, pp. 283-307.
3. "http://en.wikipedia.org/wiki/CUDA"
4. Negrut, D., 2011, "High Performance computing for Engineering Applications" ME964 UW-Madison, lecture reference, pp. 14
5. Jung, Y. H., 2011, "Parallel Programming with CUDA", GyeongGi-Do, Freelec, pp. 105-113.
6. Parviz E.Nikravesh, 1988, "Computer-Aided Analysis of Mechanical Systems", Prentice-Hall International Inc., Arizona, pp. 227-248
7. D. Negrut, et. al., 2006, "On an implementation of the Hilber-Hughes-Taylor Method in the Context of Index 3 Differential- Algebraic Equations of Multibody Dyna- mics" American Society of Mechanical Engineers, Volume 2, Issue 1, pp. 73-86.
8. NVIDIA, 2011, "CUDA Toolkit 4.0 CUBLAS Library", PG-05326-040_v01, pp. 1-85.