creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공 학 석 사 학 위 논 문

GPGPU기반 공간 다물체 시스템과 대량 입자들 간의 접촉 해석 연구

2013년 2월

부 경 대 학 교 대 학 원

메카트로닉스공학과

전 철 웅

공 학 석 사 학 위 논 문

GPGPU를 이용한 공간 다물체 시스템과 대량 입자들 사이의 접촉 해석 연구

지도교수 손 정 현

이 논문을 공학석사 학위논문으로 제출함.

2013년 2월

부 경 대 학 교 대 학 원

메카트로닉스공학과

전 철 웅

전철웅의 공학석사 학위논문을 인준함

2013년 2월

주 심 공학박사 백 운 경 (인)

위 원 공학박사 정 영 석 (인)

위 원 공학박사 손 정 현 (인)

목 차

PUKYONG NATIONAL UNIVERSITY
부경대학교

List of table and figure

PUKYONG NATIONAL UNIVERSITY
부경대학교

Study on Contact Analysis between Spatial Multibody System and Many Particles using GPGPU

Chul-Woong Jun

Department of Mechanical Engineering,
The Graduate School,
Pukyong National University

Abstract

NVIDIA has been developing the GPGPU(General Purpose Graphics Processing Unit) for not only 3D graphics and image processing, but also signal processing, physical simulation and so on. Currently, a lot of studies carried out by using the parallel programing based GPU. A lot of time is required for solving the EOM of large system by using multi-body dynamics. If the parallel programing by GPU is applied to the multi-body dynamics, the analysis time can be improved.

In this study, parallel computing for collision analysis between 3D multi-body and many particles was carried out by using the CUDA programming. In order to detect collisions between multi-body and particle, grid-cell collision detection algorithm is employed. Hertzian normal and tangential sliding friction contact model are used to calculate the contact force.

The multi-body model consists of the particles and spring-damper force elements and spherical joints. To verify the accuracy, MSC ADAMS is used. To investigate the numerical efficiency for the multi-body dynamics, the sequential program and the parallel program were developed. In case of the multi-body dynamics, when the number of body and spring-damper is 280 and 360, respectively, computation time of the parallel program is faster about 2.17 times then the sequential program. In case of the DEM, when the number of particle is 10648, it shows the better efficiency about 100times. In case of the collision analysis, when the number of particle and spring-damper is 1280 and 360, respectively, it shows the better efficiency about 3times.

Keywords : GPU(그래픽 프로세서 유닛), Parallel computing(병렬 컴퓨팅), MBD(다물체 동역학), DEM(이산 요소법)

1. 서 론

1.1. 연구 배경과 동향

몇 년 전만 해도 주위 자연현상 및 기계들의 물리적인 상태를 수치적으로 해석하기 위한 산술 연산에는 CPU(Central Processing Unit)을 이용하였고, 해석 결과를 영상으로 표현하기 위해 GPU(Graphic Processing Unit)을 사용하였다. CPU에만 의존한 순차적인 산술 연산은 해석 모델이 복잡하거나 많은 반복 연산을 필요로 할 경우 시간적인 측면에서 매우 비용이 커지는 문제가 있다. 시간적인 비용 문제를 해결하기 위해 상호 독립적인 반복과정을 병렬화 시키는 방법을 생각할 수 있다. 최근까지 CPU는 단일 코어구조에서 성능을 향상시키기 위한 노력을 지속해왔다. 캐시 성능을 강화하여 메모리와 CPU간의 속도 차이를 줄이고, 클록(Clock)의 수행 능력을 높여 왔다. 최근에는 단일 코어에서의 성능 향상의 한계에 의해 다중코어의 CPU가 개발 되어 졌고 공급되고 있다. CPU를 사용한 병렬화는 단일 CPU가 가지는 코어의 개수 만큼의 병렬화가 가능하고 더 많은 데이터를 병렬화하기 위해서는 다중 CPU 장착이 가능한 시스템을 구축해야 하는 비용적인 문제가 있다. 반면에 GPU는 PC를 제어하는 모든 부가적인 작업을 CPU에 맡기고 GPU는 오직 계산을 위한 효율만을 높여 왔다. GPU가 수행하던 픽셀 좌표계, 렌더링, 비디오 인코딩, 화상 스케일링 같은 처리는 수많은 데이터를 멀티코어 또는 가속기를 이용하여 빠르게 계산한다는 점에서 범용적인 병렬처리와의 유사성을 이용하여 GPU 성능을 일반적인 신호 처리, 물리 시뮬레이션, 재무 예측, 생물학적 계산 같은 데이터 병렬처리에 이용하고자 하였고, 이를 GPGPU(General Purpose Graphics Processing unit)이라고 한다. NVIDIA는 GPGPU에 대한 흐름을 먼저 인식하고 그래픽 카드를 범용적인 용도로 사용할 때 발생하는 어려움을 해결하기 위한 노력을 시작하였다. 이전의 그래픽 카드는 3D 처리를 위해 다양한 기능이 추가되었지만 모두 그래픽 처리를 위한 것으로 범용적인 용도로 쓰이는 것들은 아니었다. 2006년 11월 NVIDIA사는 다목적 프로그램을 위한 그래픽 카드 GeForce 8000 시리즈를 출시하였고, 2007년 2월에 통합 개발 환경 'CUDA(compute Unified Device Architecture)'를 발표하였다. CUDA가 기존에 CPU코어에 의존하던 병렬 프로그래밍을 비교적 저렴한 GPU를 사용하여 가능하게 함으로써 국내외로 CUDA를 적용하는 많은 연구들이 수행되고 있다.

2007년 S.L Grand가 CUDA를 이용하여 광역충돌 검출 연구를 하였고,[(1)] 2012년 Daniel J. Melanz가 Flexible 바디들로 이루어진 큰 시스템에 GPU 병렬화를 연구하였다.[(2)] 국내에서는 이연희가 CUDA를 이용한 병렬형 충돌검사 및 동역학 시뮬레이션을 연구하였고,[(3)] 2012년 정혜영은 CUDA기반의 2D 평면상에서의 다물체 동역학과 DEM과의 충돌해석을 연구하였으며[(4)], 2012년 박지수외 3명이 구형 접촉

입자가 포함된 다물체 동역학 해석 연구를 한 사례가 있다.[5] 기존 연구의 대부분은 다물체 동역학과 입자동역학이 구분되어 연구가 이루어져 왔다. 통합해석에 있어서도 2D 평면상에서 연구가 이루어 졌거나, 비교적 단순한 다물체 동역학 모델을 사용하여 연구가 진행되었기 때문에, 좀 더 복잡하고 다양한 측면에서의 연구가 필요하다. 본 연구에서는 기존 연구의 한계를 인식하고 3D 공간상에서의 다물체 동역학과 입자동역학의 접촉문제를 다루었으며, 다물체 동역학 모델 또한 많은 강체 물체와 힘 요소로 구성되어 복잡한 형태의 모델에 대해서 연구를 진행하였다. 국내에서도 다물체 동역학 분야에서 CUDA를 이용한 연구가 활발하게 이루어지고 있지만, 여전히 해외에 비해 많이 뒤쳐져 있는 것 이 사실이다. 동역학 분야에서 CUDA를 이용한 연구는 구속이 없는 대량 입자들 간의 접촉에 관한 연구가 많았으며, 최근에 들어 구속된 시스템과 힘 요소를 고려한 병렬 프로그래밍 연구가 이루어지고 있다. 다물체 동역학 해석에서는 각 물체의 수와 조인트의 수 그리고 힘 요소의 수에 따른 계산은 각 계산과정이 다른 계산 과정에 영향을 미치지 않기 때문에 서로 독립적으로 계산이 가능하다는 점에서 병렬화의 가능성을 가늠해 볼 수 있다. 또한 구속이 없는 대량 입자들에 대해서는 그 수만큼의 반복연산을 병렬화 함으로써 상당한 시간적 효율을 기대할 수 있다.

PUKYONG NATIONAL UNIVERSITY
부경대학교

1.2. 연구 목적

최근까지 국내에서 GPGPU의 활용은 구속이 없는 입자들 간의 운동을 다룬 입자 동역학(particle dynamics)이 대부분이며, 구속과 힘 요소가 존재하는 다물체동역학 시스템에 대한 GPGPU를 적용한 연구 사례는 적다.

평면에 비해 공간상의 시스템은 더 많은 계산량과 메모리를 필요로 하고 복잡한 계산 알고리즘을 요구한다. 본 연구에서는 공간상에서의 조인트의 수와 힘 요소의 수에 따른 다물체 동역학 해석의 정확성을 검증하고, 순차 해석 프로그램과, GPGPU를 이용한 병렬 해석 프로그램과의 해석 시간 비교를 통해 전체 시스템의 효율성을 연구하였다. 또한 입자동역학에 대해서도 입자들의 수에 따른 GPGPU의 활용에 의한 효율성을 연구하여 최종적으로 다물체 동역학과 대량 입자 거동과의 상호 접촉문제에 대한 효율성 향상 연구에 목적이 있다.

PUKYONG NATIONAL UNIVERSITY
부경대학교

2. CUDA 병렬 프로그래밍

2.1. 그래픽 카드의 발전

GPU는 컴퓨터 그래픽을 전문으로 처리하는 연산장치를 말한다. 2000년대에 들어오면서 컴퓨터를 이용한 그래픽 분야는 영화와 게임을 중심으로 3D 영상을 중점적으로 다루게 되었다. 특히 3D 게임의 좀 더 현실적이고 화려한 영상을 위해 많은 연산과 자원을 요구하기 때문에 뛰어난 성능의 그래픽 카드의 수요가 생기게 되면서 많은 발전을 하게 되었다. 그래픽 카드의 중요성이 점차 대두되면서 시장에서 우위를 차지하고자 치열한 경쟁을 하기 시작하면서 많은 발전을 하게 된다. 3D 처리의 특성상 많은 객체에 대한 수학적 물리적 계산을 처리하게 되는데 이를 원활히 처리하기 위해 그래픽 칩은 대규모 파이프 라인 방식과 멀티코어로 발전하였다.

GPU는 트랜지스터를 집적시켜 연산을 처리하는 반도체라는 점에서 CPU와 유사한 점도 많이 있지만, 근본적으로 하는 일이 다르고 주어진 환경 또한 많이 다르다. GPU는 하나의 제품 안에 보드와 메모리, 프로세서를 갖추고 있기 때문에 내부의 규격을 자유롭게 변경할 수 있어 CPU에 비해 더 유연한 환경을 갖추고 있다. PC가 DDR1에서 DDR3로 규격을 변경하는데 오랜 시간이 소요됐지만, 그래픽 카드는GDDR5와 같은 빠른 메모리가 출시되면 바로 채택하고 자체 결정만으로 메모리 클록과 버스 규격을 자유롭게 변경하였다. 이런 이점과 더불어 GPU는 3D 그래픽 처리와 같은 대량의 병렬 연산을 처리하기 위해 많은 부분을 ALU에 배분하고 있기 때문에 CPU에 비해 강력한 계산 능력을 갖추게 되었다. 그림 1은 CPU와 GPU간의 연산 수행 능력의 차이를 나타내고 있다[11].

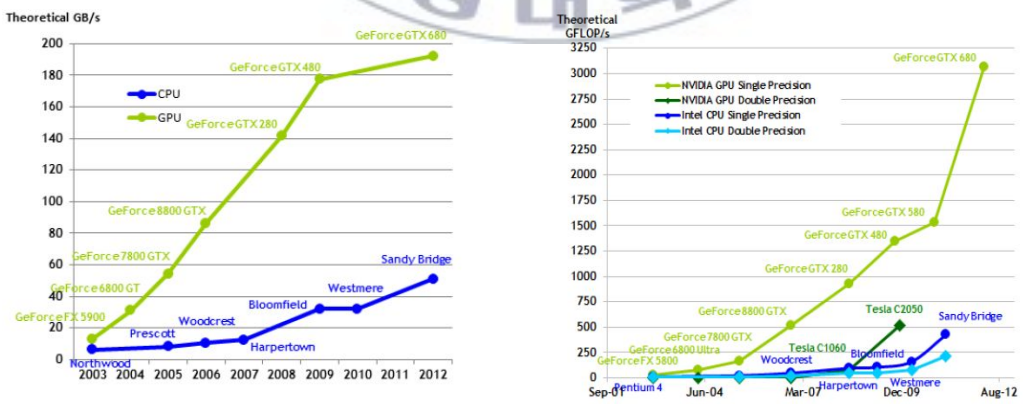


그림. 1 NVIDIA 그래픽 카드와 CPU의 연산 능력 차이

최근에는 3D그래픽, 이미지 영상 처리등에 사용되어져 오던 GPU를 기존에 CPU가 처리해오던 신호 처리, 물리 시뮬레이션, 재무 예측, 생물학적 계산 등을 병렬적

으로 빠르게 처리하기 위해서 GPGPU로 개발이 이루어지고 있다. 하지만 그래픽 카드를 일반적인 용도로 사용하기에는 무리가 있었는데, 그것은 그래픽을 위한 전용 API 함수와 그래픽 카드의 한계 때문이다. 기존에 제공되고 있던 그래픽 API함수는 그래픽 엔진을 다루는 프로그래머가 주로 사용하던 것으로, 일반적인 프로그래머나 초보자가 사용하기에는 많은 어려움이 있었다. 또 다른 어려움은 GPU의 연산 능력과 비교하면 DRAM 메모리 대역폭의 한계로 병목 현상(Bottleneck)이 발생하여 유연한 프로그램을 작성할 수 없었다는 점이다. 이러한 한계를 극복하고 효율적으로 GPGPU를 지원하기 위해 나온 것이 NVIDIA가 내놓은 CUDA이다[6].

2.2. CUDA

CUDA는 그래픽 카드를 이용한 GPGPU의 통합 개발 환경 제공을 목적으로 한다. NVIDIA사가 GPGPU에 대한 흐름을 먼저 인식하고 GPU를 이용한 범용적인 프로그램을 개발할 수 있도록 '프로그램 모델', '프로그램 언어', '컴파일러', '라이브러리', '디버거', '프로파일러'를 제공하는 통합 환경을 구축하였고, 2007년 2월에 통합 개발 환경 'CUDA(Compute Unified Device Architecture)'를 발표하였다. 그림 2는 CUDA가 지원하는 다양한 언어와 응용 프로그래밍 인터페이스를 보여주고 있다.

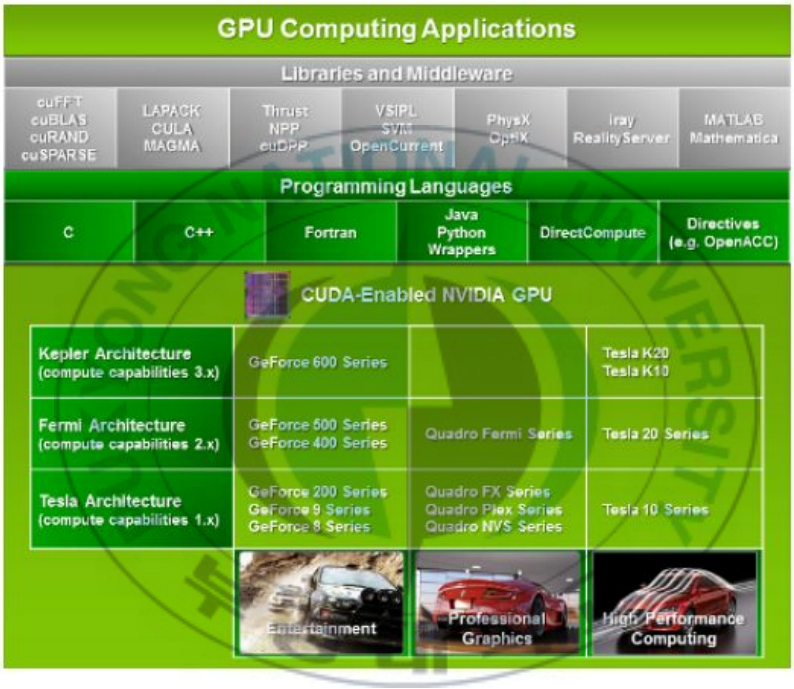


그림. 2 CUDA가 지원하는 다양한 언어와 응용 프로그래밍 인터페이스

CUDA의 장점은 보통 슈퍼 컴퓨터와 비교되곤 하는데 1TFLOP/s가 넘는 처리능력은 10년 전의 슈퍼 컴퓨터와 동등한 수준이다. 인텔 i7 CPU도 10개를 합쳐야 나오는 성능으로, 대규모 데이터를 처리하는데 적합하다. 그리고 잘 설계된 CUDA 프로그램은 GPU의 성능과 비례하여 연산 능력을 발휘하게 된다. 이는 GPU의 성능이 좋아지면 코어의 수가 많아지기 때문에 당연한 결과이다. 또한 저렴한 가격으로 인텔 i7 CPU를 장착한 PC 10대의 성능을 얻을 수 있다.

2.3. 병렬 프로그래밍

2.3.1. 병렬 프로그래밍 개발 환경

CUDA를 이용한 병렬 프로그래밍을 위해서는 NVIDIA 제품의 그래픽 카드가 필요하며, 본 연구에서 사용된 컴퓨터의 사양은 다음 표 1에 나타내었다.

표. 1 연구 개발에 사용된 컴퓨터 사양 및 개발 소프트웨어

컴퓨터 사양 및 개발 소프트웨어	
프로세서(CPU)	Intel(R) Core(TM) i7-2600K 3.40GHz
메모리(RAM)	8.00GB
그래픽카드	Tesla C2075
프로그래밍 개발 환경	Visual Studio Pro 2010

프로그램 개발은 Visual Studio 소프트웨어를 사용한 visual C++[7]를 기반으로 개발하였다. 일반 순차 프로그램은 프로그래밍 소프트웨어와 컴퓨터만 구비되면 가능하지만 CUDA를 이용한 병렬 프로그래밍의 경우 NVIDIA에서 제공하는 그래픽 카드와 CUDA toolkit이 필수적으로 요구되며 개발에 도움이 될 수 있는 라이브러리 또한 필요하다. 다음 표 2에서 연구에 사용된 버전과 라이브러리 종류를 명시하였다.

표. 2 병렬 프로그래밍 개발 환경

병렬 프로그래밍 개발 환경	
CUDA toolkit 버전	v5.0
Linear Algebra Library	CUBLAS
	CULADense
그래픽 표현	OpenGL

행렬-벡터, 벡터-벡터 계산에는 CUDA toolkit에서 제공하는 CUBLAS[8]를 사용하였으며, 선형 방정식의 해를 구하는 과정에는 CUDADense[9]의 함수가 사용되어졌다. 그리고 해석결과를 화면상에 표현하기 위해 OpenGL를 사용하였다.

2.3.2. 스레드-블록-그리드 구조

CUDA 스레드는 스레드-블록-그리드 구조로 이루어져 있다. 병렬 계산을 위해 호스트에서 커널을 요청하면 디바이스는 하나의 그리드를 생성한다. 그리고 커널 요청시에 명시한 블록과 스레드의 수에 의해서 그리드 내에 블록과 스레드를 생성하게 된다. 그림 3은 스레드-블록-그리드 구조를 나타내고 있다[10][11].

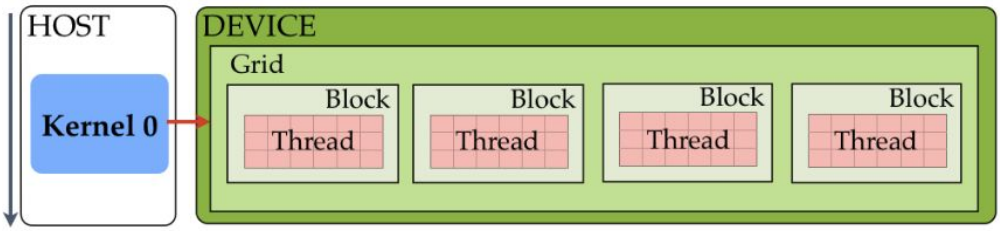


그림. 3 CUDA 스레드-블록-그리드 구조

CUDA의 커널 함수는 GPU에서 동작하는 명령어 세트의 조합으로 이루어진 함수로 수많은 코어에서 동시에 멀티 스레드로 동작하게 된다. 커널 함수를 정의하는 문법은 다음과 같다.

커널 함수 정의

```
__global__ void KernelFunc(int a, float b)
{
      ....
}
```

커널 함수는 일반 C에서의 함수 정의와 유사하다. 하지만 __global__이라는 지시어를 통해서 이 함수는 커널 함수임을 알리는 동시에 GPU에서 실행되는 함수라는 것을 정한다. 커널 함수는 몇 가지 제약사항을 가지는데 반드시 반환 값을 void로 설정을 해야 한다. 그래서 반환 하려는 값이 있다면 포인터를 사용하여 값을 받아야 한다. 또한 지정된 수만큼의 매개변수를 사용해야 하며 가변형 매개변수는 사용할 수 없다. 또 다른 지시어로는 __host__, __device__가 있는데, 전자는 PC에 장착된 CPU를 통해 계산을 하는 함수라는 것을 나타내고, 후자는 GPU에서만 실행되며 호출은 커널함수에서만 가능하다[11]. 그리드 내에 하나의 블록은 1, 2, 3차원으로 크기를 결정할 수 있으며, 하나의 그리드에 생성할 수 있는 블록의 수는 65535*65535*65535이며 gridDim 변수에 의해 그 크기를 얻을 수 있다. 각 블록은 빌트인 변수 blockIdx에 의해 고유한 인덱스를 가지고 있어 각 블록을 구분할 수 있으며 하나의 블록은 여러 개의 스레드들을 가질 수 있다. 스레드 또한 1,2,3차원으로 크기를 결정할 수 있으며, 각 블록에서 총 스레드의 수는 1024개를 넘길 수 없다. 스레드는 블록과 마찬가지로 blockDim 변수에 의해 그 크기를 얻을 수 있으

며, threadIdx 변수에 의해 각 스레드를 구분할 수 있다. 각 블록을 하나의 커널함수를 실행하는 기본 단위이며, 블록내의 각 스레드는 커널 함수에 정의된 명령어들을 계산하는 기본 계산 단위이다.

2.3.3. CUDA 데이터의 병렬처리

CUDA는 PC의 데이터를 전달받아 GPU프로세서의 많은 코어들을 통해 연산을 수행하게 된다. CUDA 에서도 다중코어 CPU와 같이 작업을 분활하고 스레드를 할당하여 병렬 처리를 수행하게 된다. 그림 4은 CUDA를 이용하여 데이터를 병렬처리하는 전 과정은 표현하고 있다.

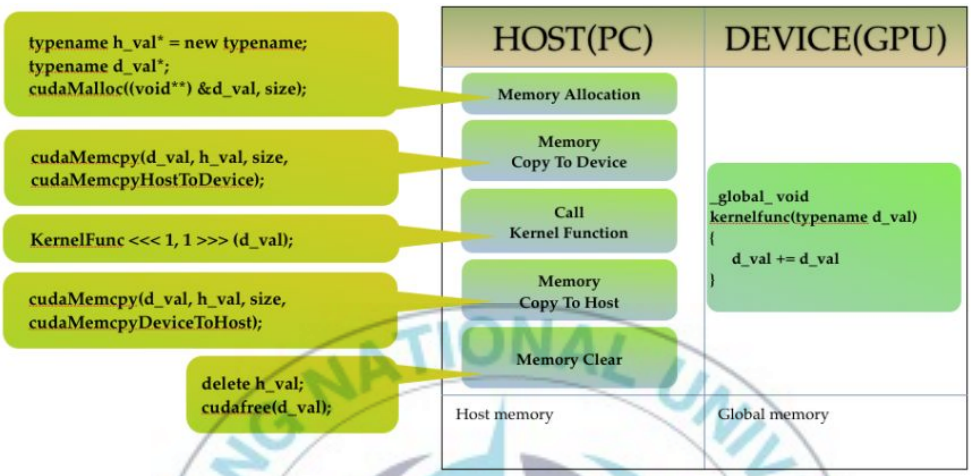


그림. 4 CUDA를 이용한 데이터 병렬 처리 과정

병렬 프로그래밍을 위해서 제일 먼저 호스트와 디바이스에 각각 저장 공간을 할당하는 것이다. 다음으로 디바이스의 메모리는 호스트 메모리로부터 정보를 전달받게 된다. CUDA의 병렬 계산은 커널함수에 의해서 수행되는데 함수의 호출은 호스트에서 이루어진다. 커널 함수는 __global__이라는 지시어를 붙여 이 함수는 GPU에서 실행하는 함수라는 것을 표현한다. 반대로 __host__는 CPU에서 실행하는 함수를 표현하는 것이다. CUDA는 커널함수를 호출 할 때에 계산에 사용될 스레드의 수가 결정되게 되는데, 이는 <<< >>>기호내부에 인자로 블록과 스레드의 수를 명시하게 된다. 또한 공유 메모리를 동적으로 할당할 시에 3번째 인자로 공유 메모리의 크기를 명시할 수 있다. 커널 함수의 실행 결과는 처리를 하기 위해 다시 호스트로 복사를 해주어야 한다. 호스트에서는 넘겨 받은 데이터를 처리하고 마지막으로 호스트와 디바이스에 각각 할당했던 메모리를 해제해야 한다.

2.3.4. CUDA 메모리

CUDA를 이용하여 병렬 프로그래밍을 하기 위해서는 CUDA 데이터 흐름과 메모리 구조를 알 필요가 있다. CUDA는 뛰어난 그래픽 카드의 연산 능력을 이용하여 처리하는 방법이다. 따라서 기존의 CPU처리 방법에 더 추가해야 하는 과정이 있다. 추가된 과정은 PC의 메모리에 있는 입력 데이터를 그래픽 카드의 메모리로 전달하고 GPU가 처리한 결과를 다시 그래픽 카드의 메모리에서 PC의 메모리로 가져오는 과정이다. 그림 5은 CPU와 GPU 메모리 사이의 데이터 처리 과정을 보여주고 있다.

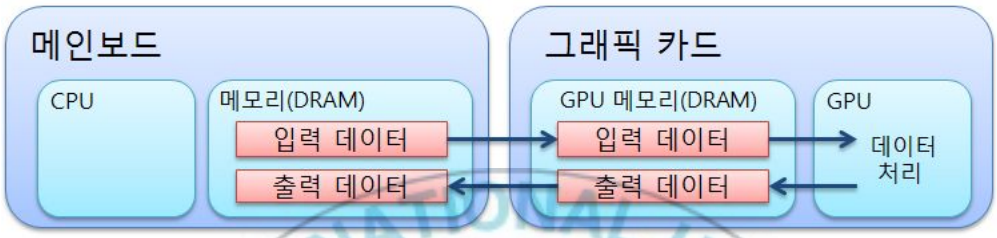


그림. 5 CPU와 GPU 메모리간의 데이터 처리 과정

그래픽 카드에는 PC와 달리 다양한 메모리를 가지고 있다. CUDA를 이용한 병렬 프로그래밍은 그래픽 카드의 메모리를 사용하기 때문에 어떤 종류의 메모리가 존재하며 각 메모리의 특성 또한 제대로 알 필요가 있다.

표. 3 그래픽 카드의 메모리 종류와 특성

메모리	위치	캐시사용	엑세스	사용범위
레지스터	온 칩	NO	읽기/쓰기	개별 스레드 내
로컬 메모리	오프 칩	NO	읽기/쓰기	개별 스레드 내
공유 메모리	온 칩	NO	읽기/쓰기	동일 블록 내 스레드
글로벌 메모리	오프 칩	NO	읽기/쓰기	모든 스레드+호스트
상수 메모리	오프 칩	YES	읽기	모든 스레드+호스트
텍스처 메모리	오프 칩	YES	읽기	모든 스레드+호스트
액세스 속도	느리다	——————————▶		빠르다
메모리 용량	작다	——————————▶		크다
액세스 범위	좁다	——————————▶		넓다

표 3은 그래픽 카드의 각 메모리와 그 특성을 보여주고 있다[6].

1) 레지스터

온 칩 프로세서에 있는 메모리로 직접 연산을 수행하는 가장 빠른 메모리이다. 레지스터 1개는 32bit 크기로, GPU 1사이클 이내의 속도로 읽고 쓰기가 가능하다.

커널 함수 안에서 사용하는 로컬 변수는 프로그램 실행 시 레지스터에 값이 저장된다. CPU와 GPU 프로그램 모두 최종 연산은 레지스터에서 이루어지지만, C 언어로 프로그램을 구현했을 때 어떤 레지스터를 쓰는지는 알 수 없다. 커널 함수에서 사용한 변수는 CUDA C에서도 레지스터를 통해 연산하게 되지만 몇 개를 사용하게 되는지 알 수는 없다. 어셈블리 언어와 같이 직접적으로 레지스터 사용을 지정한 경우에만 몇 개의 레지스터를 사용했는지를 알 수 있다.

2) 로컬 메모리

커널 함수 내에서 너무 많은 로컬 변수를 사용하거나, 배열형 변수로 큰 용량을 사용하면 프로세서 밖에 있는 DRAM에 메모리가 할당된다. 로컬 변수가 레지스터로 사용될지 로컬 메모리(Local Memory)에 할당될지 명확하지 않아 불편한 점이 있다. 로컬 메모리로 할당되는 경우는 다음과 같은 세 가지 경우이다.

① 너무 많은 레지스터 변수를 사용했을 때
② 너무 많은 로컬 변수를 사용했을 때
③ 로컬 변수로 배열을 사용했을 때

로컬 메모리를 읽고 쓰는 속도는 GPU 400 사이클에서 600 사이클까지의 시간이 소요되는 저속이다.

3) 공유 메모리

CUDA 프로그래밍의 큰 장점 중 하나라 할 수 있는 것이 공유 메모리(Shared Memory)이다. 공유 메모리는 온 칩 프로세서에 있는 메모리로, 16KB의 용량을 L1 캐시와 동등한 속도로 사용할 수 있다. 공유 메모리를 읽고 쓰는 속도는 GPU 1 사이클 이내이다. 공유메모리를 할당하는 방법에는 정적인 방법과 동적인 방법이 있다. 할당 방법은 다음과 같다.

```
공유 메모리 정적 할당 방법
__shared__ float a[512]; // 정적 할당

공유 메모리 동적 할당 방법
extern __shared__ float sdata[];
```

공유 메모리의 할당은 커널 함수 내에서 하게 되고, 동적 할당 방법에 경우 커널 함수를 실행할 때 <<< >>> 안에 3번째 인자로 메모리 크기를 지정해주어야 한다.

4) 글로벌 메모리

글로벌 메모리(Global Memory)는 그래픽 카드에 장착된 DRAM 메모리를 의미하며, 카드 사양마다 사용할 수 있는 메모리 용량의 차이가 있다. 글로벌 메모리는 칩 외부에 있기 때문에 메모리 액세스 속도가 400 사이클에서 600 사이클 정도로 GPU 칩 내부에 있는 레지스터나 공유 메모리보다 많이 느리다. 하지만 CPU의 메모리와 비교하였을 때는 CUDA의 글로벌 메모리는 상당히 빠른 속도에 속한다.

글로벌 메모리를 할당하고 해제할 때는 CUDA에서 제공하는 다음과 같은 API함수를 사용한다.

```
글로벌 메모리 할당 및 해제
checkCudaErrors( cudaMalloc(void** devptr, size_t count ) ); // 할당
checkCudaErrors( cudaFree(void* devptr) ); // 해제
```

글로벌 메모리를 할당한 후 CPU 메모리의 데이터를 글로벌 메모리로 복사를 하거나 초기화를 해주어야 한다. 복사와 초기화의 문법은 다음과 같다.

```
메모리 복사 및 초기화
checkCudaErrors( cudaMemcpy(void* dst, const void* src, size_t count,
                                    cudaMemcpyKind kind) ); // 복사
checkCudaErrors( cudaMemset(void* devptr, int value, size_t count) );
                                                    // 초기화
```

메모리 복사에서 cudaMemcpyKind는 복사 방향을 지정하는 곳으로 그 종류는 표 4와 같다[6].

표. 4 cudaMemcpyKind enum 변수의 종류

종류	동작
cudaMemcpyHostToHost	PC 메모리에서 PC 메모리로 복사
cudaMemcpyHostToDevice	PC 메모리에서 글로벌 메모리로 복사
cudaMemcpyDeviceToHost	글로벌 메모리에서 PC 메모리로 복사
cudaMemcpyDeviceToDevice	글로벌 메모리에서 글로벌 메모리로 복사

5) 상수 메모리

상수 메모리(Constant Memory)는 DRAM에 있는 데이터를 읽기 전용으로 사용하며 캐시를 지원한다. 상수 메모리로 사용할 수 있는 최대 크기는 64KB이다.

처음으로 데이터를 읽을 때는 DRAM에서 읽기 때문에 400~600사이클이 소요되지만 한번 캐시에 올라온 값을 반복하여 재사용할 때에는 레지스터와 동일한 속도로 사용 할 수 있다. 상수 메모리는 호스트에서 할당하고, 디바이스에서 읽기만 할 수 있다. 글로벌 메모리와 같이 모든 스레드가 공유 할 수 있다. 상수 메모리의 할당 방법은 다음과 같다.

```
상수 메모리 할당
__constant__ float cData[6];
float hData[6] = { 1.0f, 2.0f, 3.0f, 4.0f, 5.0f, 6.0f };
checkCudaErrors ( cudaMemcpyToSymbol( cData, hData, sizeof(hData) );
```

상수 메모리를 사용할려면 __constant__ 지시어를 사용하여 상수 메모리 영역을 할당 해야 한다. 그리고 cudaMemcpyToSymbol() 함수를 사용하여 호스트에서 상수 메모리로 값을 전달해야 한다.

PUKYONG NATIONAL UNIVERSITY 부경대학교

3. 공간 다물체 동역학 프로그래밍

3.1. 운동 방정식의 구성

3.1.1. 오일러 매개변수

본 연구에서는 다물체 동역학의 운동을 표현하기 위해 Cartesian 좌표계를 사용하였으며, 물체의 자세를 결정하기 위해 오일러 매개변수(Euler Parameters)를 사용하였다. 운동방정식을 구성하는데 있어서 Cartesian 좌표계는 다음과 같은 이점들을 지닌다[14]. : (1) 운동 방정식의 계수 행렬의 형태를 간단하게 공식화 할 수 있고, 구속식은 일관된 방법으로 생성할 수 있다. (2) 각 물체의 위치, 속도, 가속도 정보는 직접적으로 얻어질 수 있다. (3) 주어진 조인트에 대해서 구속 방정식을 수립하는데 있어 조인트에 의해 연결된 두 물체의 좌표들만 포함하기 때문에 구속방정식들은 시스템의 복잡성에 대해서 독립적이다. 하지만 물체의 위치, 회전의 모든 좌표를 고려하기 때문에 계수 행렬의 차원이 커지며, 자유도가 높아진다는 단점이 있다.

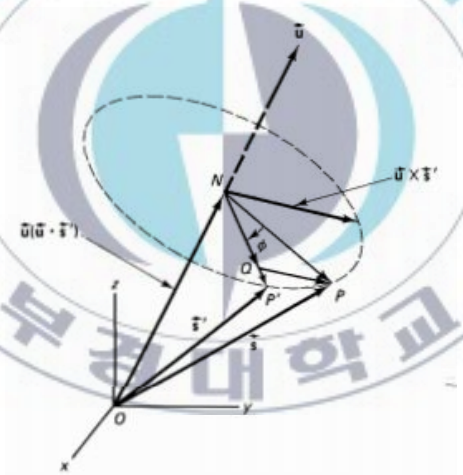


그림. 6　회전 공식 유도를 위한 벡터 다이어그램

오일러 매개변수는 오일러 각(Euler Angle)에서 나타나는 특이(singular) 문제를 해결 할 수 있으며, 내재적 적분의 시스템 자코비안을 구할 때, 편미분 되어지는 변수의 표현이 명확하기 때문에 비교적 쉽게 자코비안을 구성할 수 있다[14][15]. 오일러 매개변수는 하나의 회전각과 회전 지향축의 방향코사인으로 표현되어 진다. 그림 6은 회전 공식의 벡터 다이어그램을 나타내고 있다. 오일러 매개변수의 회전 좌표는 다음과 같이 정의된다.

$$e_0 \equiv \cos\frac{\Phi}{2} \tag{3.1}$$

$$\vec{e} \equiv \begin{Bmatrix} e_1 \\ e_2 \\ e_3 \end{Bmatrix} \equiv \vec{u} sin\frac{\Phi}{2} \tag{3.2}$$

벡터 $\vec{e}$는 $\sin\frac{\Phi}{2}$의 크기를 가지는 회전 지향축을 따른다. 그래서 오일러 매개변수는 총 4개의 회전 파라미터가 정의되며 위치 좌표와 함께 하나의 물체가 7개의 자유도를 가지게 된다. 오일러 매개변수로 표현되는 좌표 변환은 다음과 같다.

$$A = 2\begin{bmatrix} e_0^2 + e_1^2 - \frac{1}{2} & e_1e_2 - e_0e_2 & e_1e_3 + e_0e_2 \\ e_1e_2 + e_0e_3 & e_0^2 + e_2^2 - \frac{1}{2} & e_2e_3 - e_0e_1 \\ e_1e_3 + e_0e_2 & e_2e_3 + e_0e_1 & e_0^2 + e_3^2 - \frac{1}{2} \end{bmatrix} \tag{3.3}$$

오일러 매개변수로 운동방정식을 표현하는데 자주 사용되어지는 두 가지 행렬을 다음에 표기하였다.

$$G = \begin{bmatrix} -e_1 & e_0 & -e_3 & e_2 \\ -e_2 & e_3 & e_0 & -e_1 \\ -e_3 & -e_2 & e_1 & e_0 \end{bmatrix} \tag{3.4}$$

$$= \left[-\boldsymbol{e}, \tilde{\boldsymbol{e}} + e_0\boldsymbol{I}\right]$$

$$L = \begin{bmatrix} -e_1 & e_0 & e_3 & -e_2 \\ -e_2 & -e_3 & e_0 & e_1 \\ -e_3 & e_2 & -e_1 & e_0 \end{bmatrix} \tag{3.5}$$

$$= \left[-\boldsymbol{e}, -\tilde{\boldsymbol{e}} + e_0\boldsymbol{I}\right]$$

PUKYONG NATIONAL UNIVERSITY 부경대학교

3.1.2. 운동방정식

다물체 시스템의 경우, 각 물체 i의 위치는 $r_i = [x_i, y_i, z_i]^T$ 로 표현되며, 자세는 오일러 매개변수 변수 $e_i = [e_0, e_1, e_2, e_3]^T$ 4개의 파라미터로 표현한다. n_b의 물체로 구성된 기계시스템에 대해서 일반화 좌표는 $q = \left[r_1^T e_1^T \cdots r_{n_b}^T e_{n_b}^T\right]^T \in R^p, p = 7n_b$ 이다 [15][16]. 구속이 존재하는 어떤 기계 시스템에서 구속 방정식은 일반화 좌표를 포함하는 대수 방정식으로 공식화 되고 그 표현은 다음과 같다.

$$\Phi(q,t) = \left[\Phi_1(q,t) \cdots \Phi_m(q,t)\right]^T = 0 \tag{3.6}$$

식(3.6)을 시간에 대해서 미분하면 속도 기구 구속방정식을 얻을 수 있다. 그 표현은 다음 식과 같다.

$$\Phi_q(q,t)\dot{q} + \Phi_t(q,t) = 0 \tag{3.7}$$

$\Phi_q = \left[\frac{\partial \Phi_i}{\partial q_i}\right]$ 로 구속식의 편미분을 나타낸다. 가속도 기구 구속 방정식은 식(3.7)를 시간에 대해 미분 하여 얻을 수 있다. 그 표현은 다음 식과 같다.

$$\Phi_q(q,t)\ddot{q} + \left(\Phi_q(q,t)\dot{q}\right)_q \dot{q} + 2\Phi_{qt}(q,t)\dot{q} + \Phi_{tt}(q,t) = 0 \tag{3.8}$$

시스템의 시간 변화에 대한 상태는 구속 운동 방정식의 라그랑지 곱수 형태로 지배되어진다.

$$M(q)\ddot{q} + \Phi_q^T(q)\lambda = Q(\dot{q},q,t) \tag{3.9}$$

여기서 $M(q) = R^{p\times p}$ 는 일반화된 질량이고, $Q(\dot{q},q,t) \in R^p$ 는 일반화 좌표에 작용하는 힘이다. 우리는 시스템의 각 물체의 가속도와 라그랑지 곱수를 정의하기 위해서 식 (3.6)와 (3.8)를 모두 만족하는 일반화된 운동 방정식을 구성할 수 있다. Euler Parameters를 사용하는 일반화된 운동 방정식은 다음과 식과 같다.

$$\begin{bmatrix} M^* & B^T & P^T \\ B^T & 0 & 0 \\ P^T & 0 & 0 \end{bmatrix} \begin{bmatrix} \ddot{q} \\ -\lambda \\ \sigma \end{bmatrix} + \begin{bmatrix} b^* \\ 0 \\ c \end{bmatrix} = \begin{bmatrix} g^* \\ \gamma \\ 0 \end{bmatrix} \tag{3.10}$$

식(3.10)에서의 각 미지수 들은 다음과 같다.

$$M^* = \begin{bmatrix} M & 0 \\ 0 & 4L^T J' L \end{bmatrix} \tag{3.11}$$

$$g^* = \begin{bmatrix} f_1 \\ 2L_1^T n_1' \\ \vdots \\ f_b \\ 2L_b^T n_b' \end{bmatrix} \tag{3.12}$$

$$b^* = \begin{bmatrix} 0 \\ 8\dot{L}_1^T J_1' L_1 \dot{p}_1 \\ \vdots \\ 0 \\ 8\dot{L}_b^T J_b' L_b \dot{p}_b \end{bmatrix} \tag{3.13}$$

$$B = \Phi_q = \left[\Phi_{r1}, \Phi_{p1}, \cdots, \Phi_{rb}, \Phi_{pb}\right], \lambda = \left[\lambda_1, \lambda_2, \cdots, \lambda_m\right]^T \tag{3.14}$$

여기서 아래첨자 b는 시스템을 구성하는 물체의 수를 나타내고 M은 물체의 질량, J'는 물체의 관성모멘트를 나타낸다. f는 축 방향에 작용하는 힘을 나타내고, n'은 물체바디 작용하는 모멘트이다.

PUKYONG NATIONAL UNIVERSITY 부경대학교

3.1.3. HHT-I3 내재적 수치적분

부싱과 조인트 그리고 다른 많은 힘 요소들의 사용은 시스템을 stiff하게 만든다. stiff한 시스템에 명시적 적분법을 사용하면 적분 간격을 좁혀 해석을 할 수 있지만 해석시간의 증가를 감수해야 한다. 그래서 더 안정적이고 강인한 특성을 가지고 있는 암시적 적분법을 고려할 수 있다. 본 연구에 사용된 암시적 적분법인 Hilbert-Hughes-Taylor(HHT) 방법은 α-method에 기반을 두고 있으며, H.M.Hilbert, T.J.R Hughes, 그리고 R.L. Taylor에 의해 제안되어졌다. α-method는 2차 상미분 방정식을 수치적으로 적분하기 위해 구조 동역학에서 널리 사용되어 왔다. 만약 유한 요소 접근이 선형이라면, 그 식은 다음과 같은 형태로 가정된다[17][18].

$$M\ddot{q} + C\dot{q} + Kq - F(t) = 0 \tag{3.15}$$

여기서 파라미터 M, C, K는 각각 질량, 감쇠, 강성 행렬로 상수이다. 그리고 힘 F는 시간에 종속된다. α-method는 N. M. Newmark에 의해 제안된 Newmar 법을 기초로 두고 있으며, 적분공식은 두 파라미터 β와 γ에 의해 결정되며 다음 식과 같다.

$$q_{n+1} = q_n + h\dot{q}_n + \frac{h^2}{2}\left[(1-2\beta)\ddot{q}_n + 2\beta\ddot{q}_{n+1}\right] \tag{3.16}$$

$$\dot{q}_{n+1} = \dot{q}_n + h\left[(1-\gamma)\ddot{q}_n + \gamma\ddot{q}_{n+1}\right] \tag{3.17}$$

α-method에서 파라미터 β와 γ는 파라미터 α에 의해서 결정되어진다. 파라미터 α에 의해 결정되어지는 식은 다음과 같다.

$$\gamma = \frac{1-2\alpha}{2}, \quad \beta = \frac{(1-\alpha)^2}{4} \tag{3.18}$$

Newmark 공식에서 식(3.16)와 (3.17)은 운동방정식을 이산하기 위해 사용되어진다.

$$Ma_{n+1} + Cv_{n+1} + Kq_{n+1} = F_{n+1} \tag{3.19}$$

여기서 $a_{n+1}, v_{n+1}, q_{n+1}$ 은 $\ddot{q}(t_{n+1}), \dot{q}(t_{n+1}), q(t_{n+1})$에 대한 수치적인 근사값이다. Newmark 법은 암시적 적분법이고 A-stability을 보장 한다. α-method는 Newmark 공식의 A-stability를 보존하고 2차 상미분 방정식에서 2차수 해의 정확

성과 수치적인 감쇄 효과를 얻기 위해 보완하면서 제안되어졌다. 식(3.16)와 (3.17)의 적분 공식에 대한 새로운 식은 다음과 같이 대체되어진다.

$$Ma_{n+1} + (1+\alpha)Cv_{n+1} - \alpha Cv_n + (1+\alpha)Kq_{n+1} - \alpha Kq_n = F(\tilde{t}_{n+1}) \tag{3.20}$$

여기서 $\tilde{t}_{n+1} = t_n + (1+\alpha)h$ 이다. 기계시스템의 다물체동역학과 관련된 운동방정식은 식(3.15)와 같다. 그리고 해 $q(t)$는 식(3.6)의 기구 구속 방정식을 만족해야 한다. 다물체 동역학의 운동방정식에 대해서 힘은 $\Phi_q^T\lambda - F(\dot{q}, q, t)$로 계산 되어진다. 그러므로 본래의 HHT 공식과 관련하여 다물체 동역학 운동방정식은 다음과 같이 이산 된다.

$$(M\ddot{q})_{n+1} + (1+\alpha)(\Phi_q^T\lambda - F)_{n+1} - \alpha(\Phi_q^T\lambda - F)_n = 0 \tag{3.21}$$

식 (3.21)은 기구 구속 방정식과 함께 t_{n+1}에서 만족해야 한다.

$$e_1(\ddot{q}, \dot{q}, \lambda) \equiv \frac{1}{1+\alpha}(M\ddot{q})_{n+1} + (\Phi_q^T\lambda - F)_{n+1} - \frac{\alpha}{1+\alpha}(\Phi_n^T\lambda - F)_n \tag{3.22}$$

$$e_2(q,t) \equiv \Phi(q_{n+1}, t_{n+1}) \tag{3.23}$$

HHT법은 암시적 적분법이기 때문에 비선형 자코비안 시스템 매트릭스를 구성하여야 한다. 식 (3.22)와 (3.23)에 대해서 각각 변수에 대해서 편미분을 계산하면 다음과 같다.

$$\frac{\partial e_1}{\partial \ddot{q}} = \hat{M} = \frac{1}{1+\alpha}M - \frac{\partial F}{\partial \dot{q}}h\gamma + \left[\frac{1}{1+\alpha}(M\ddot{q})_q + (\Phi_q^T\lambda)_q - \frac{\partial F}{\partial q}\right]\beta h^2 \tag{3.24}$$

$$\frac{\partial e_1}{\partial \lambda} = \Phi_q^T\lambda \ , \quad \frac{\partial e_2}{\partial \ddot{q}} = \Phi_q \bullet \beta h^2 \ , \quad \frac{\partial e_2}{\partial \lambda} = 0 \tag{3.25}$$

앞에서 정의한 식들을 이용하여 최종적으로 HHT 법의 자코비안 시스템 방정식을 정의 할 수 있다. 적당한 대수적인 조작을 통해서 다음과 같은 식으로 표현할 수 있다.

$$\begin{bmatrix} \hat{M} & \Phi_q^T \\ \Phi_q & 0 \end{bmatrix}\begin{bmatrix} \partial\ddot{q} \\ \partial\gamma \end{bmatrix} = \begin{bmatrix} -e_1 \\ -\dfrac{e_2}{\beta h^2} \end{bmatrix} \tag{3.26}$$

식 (3.26)에서 계산되어지는 $\partial\ddot{q}, \partial\lambda$로 이전 단계에서 계산된 가속도와 라그랑지 곱수를 다음과 같이 수정한다.

$$\ddot{q}^{(k+1)} = \ddot{q}^{(k)} + \partial\ddot{q}^{(k)}, \qquad \lambda^{(k+1)} = \lambda^{(k)} + \partial\lambda^{(k)} \tag{3.27}$$

계속적으로 값을 수정시켜가면서 값의 변화량이 사용자가 정한 허용오차 범위내에 포함되면 최종적으로 수정된 값을 정해로 결정한다.

암시적 적분법은 크게 예측자와 수정자 두 개의 계산 부분으로 나누어 질 수 있다. 예측자는 지난 시간 단계에서의 값들을 이용하여 다음 시간 단계의 값들을 예측하기 위해서 사용된다. 수정자에서는 예측된 값을 initial guess로 사용하여 Newton-Raphson 반복법을 이용하여 정확한 해를 계산하는 역할을 한다. 예측자를 사용하지 않고 식 3.16, 3.17의 $t_{n+1} = t_n$으로 하여 계산을 하기도 한다. 하지만 예측자를 고려했을 때, Newton-Raphson 반복 계산에서 해가 빨리 수렴하였다. 그림 7은 HHT 적분법의 알고리즘을 나타내고 있다.

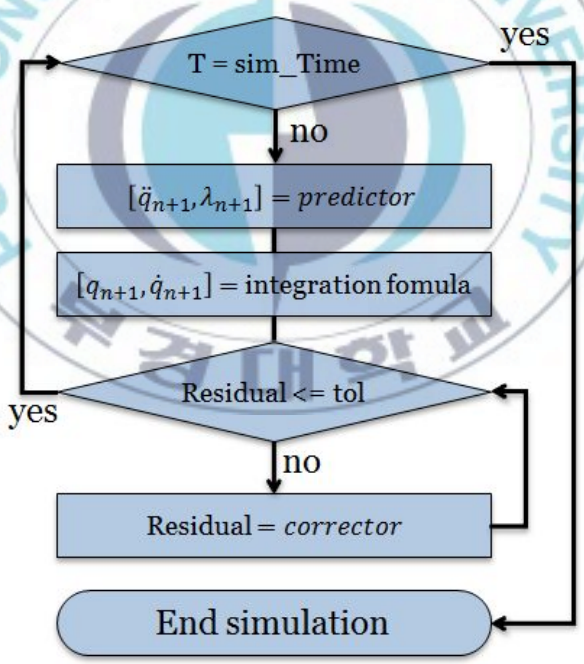


그림. 7 HHT 암시적 적분 알고리즘

3.2. 프로그램 검증

3.2.1. 검증 모델 정의

순차 해석 프로그램과 병렬 해석 프로그램간의 효율성을 비교하기에 앞서 각각의 해석 프로그램의 정확성을 확인할 필요가 있다. 다물체 동역학 해석 프로그램의 검증을 위해 상용 다물체 동역학 해석 프로그램인 ADAMS를 사용하여 해의 정확성을 검증하였다.

해를 검증하기 위한 다물체 동역학 모델은 작은 강체 입자들을 스프링-댐퍼 힘 요소로 서로 연결하여 정사각형 형태의 그물 형상을 모델링 하여 검증을 하였다. 그물 형상의 각 4개의 꼭지점에는 Spherical 조인트를 사용하여 병진 방향의 운동을 모두 구속하였으며, 회전은 자유롭게 하여 그물이 꼭지점 부분에 고정되어 자유롭게 운동 할 수 있도록 모델링하였다. 검증을 위한 강체 입자와 스프링-댐퍼의 물성치와 시뮬레이션 조건은 표 5에 나타내었다.

표. 5 다물체동역학 모델의 물성치와 시뮬레이션 조건

강체 입자의 물성치	
질량	$0.05(kg)$
반지름	$0.015625(m)$
관성 모멘트	$[1,1,1]^T(kg \cdot m^2)$
강체 입자의 수	40(n/a)
스프링-댐퍼 수	48(n/a)
회전 조인트 수	4(n/a)
스프링-댐퍼 물성치	
강성	$1000(N/m)$
감쇠	$200(N \cdot s/m)$
자유길이	0.03125(m)
시뮬레이션 조건	
해석 시간	1.0(sec)
적분 시간 간격	0.001(sec)

그물 형상의 시뮬레이션에서 강체입자는 회전운동을 전혀 하지 않기 때문에 관성 각 축에 대한 관성 모멘트는 동일하게 하였다. 그림 8은 그물 모델링 형상을 보여주고 있다.

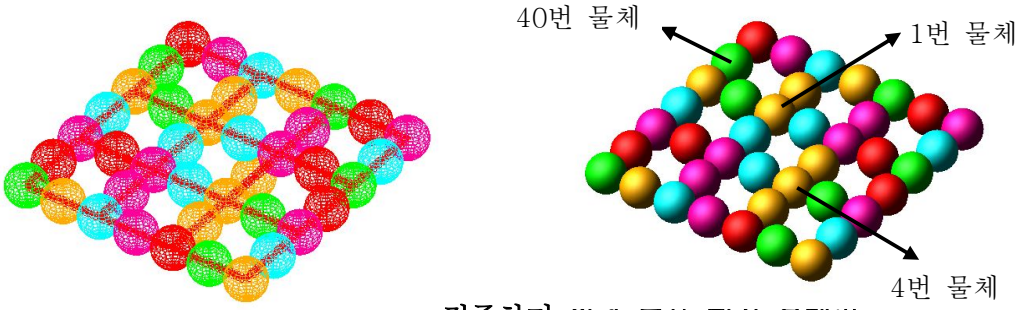


그림 8 해석 정확성을 검증하기 위한 그물 형상 모델링

그림 8의 왼쪽 그림은 각 강체입자가 스프링 댐퍼로 연결된 모습을 보여주고 있으며, 오른쪽 그림은 선명한 구 형상의 강체 입자를 표현하고 있다. 정확성 검증에 사용되어진 강체 입자는 그림 8의 오른쪽 그림에 나타내었다. 공간 해석의 정확성 검토를 위해 1번 입자는 x축의 위치를 4번 입자는 y축의 위치를 그리고 40번 입자는 z축의 위치 값을 검토하였다.

PUKYONG NATIONAL UNIVERSITY
부경대학교

3.2.2. 해의 정확성 검증

해석 프로그램의 정확성을 검증하기 위해 임의의 강체 입자 3개를 선정하여 각 입자의 병진 방향의 위치, 속도, 가속도에 대해서 서로의 해를 비교해 보았다.

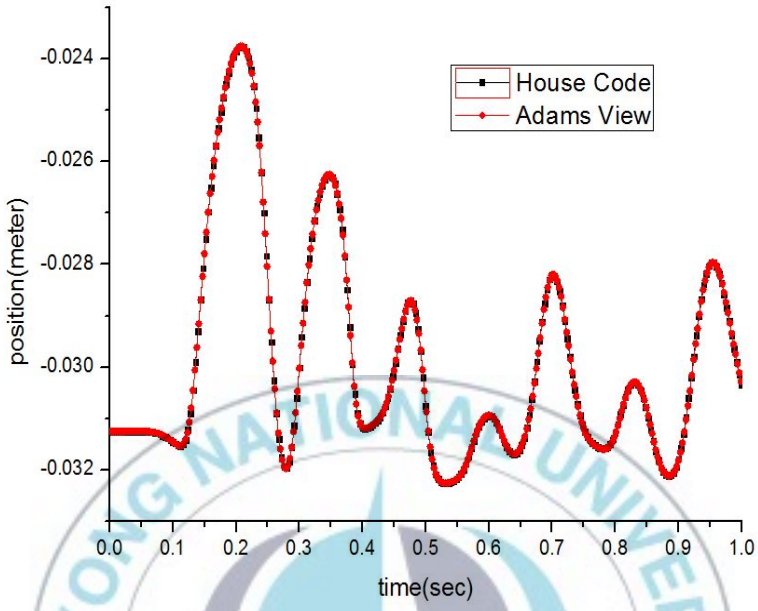


그림. 9 1번 물체의 x축 방향 위치

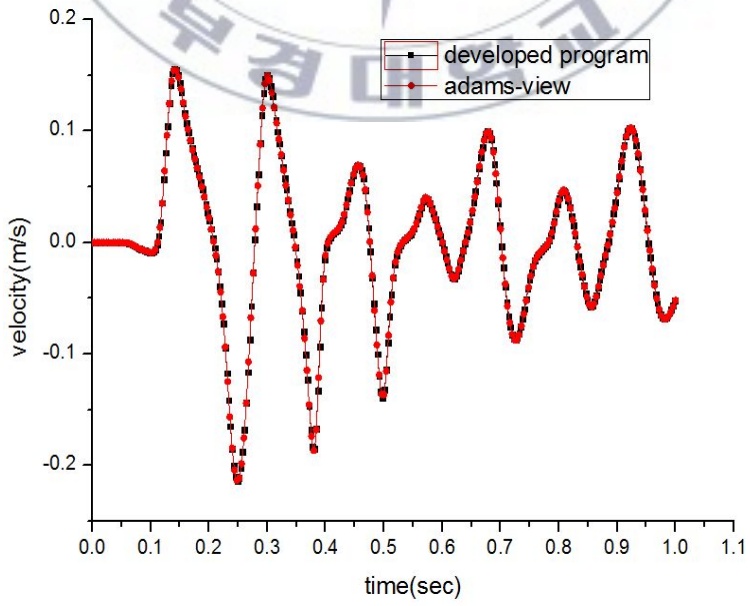


그림. 10 1번 물체의 x축 방향 속도

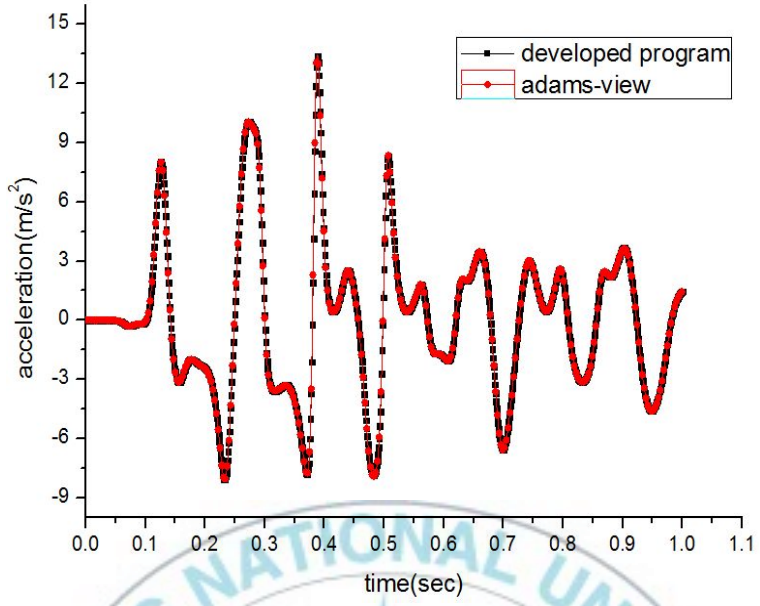


그림. 11 1번 물체의 x축 방향 가속도

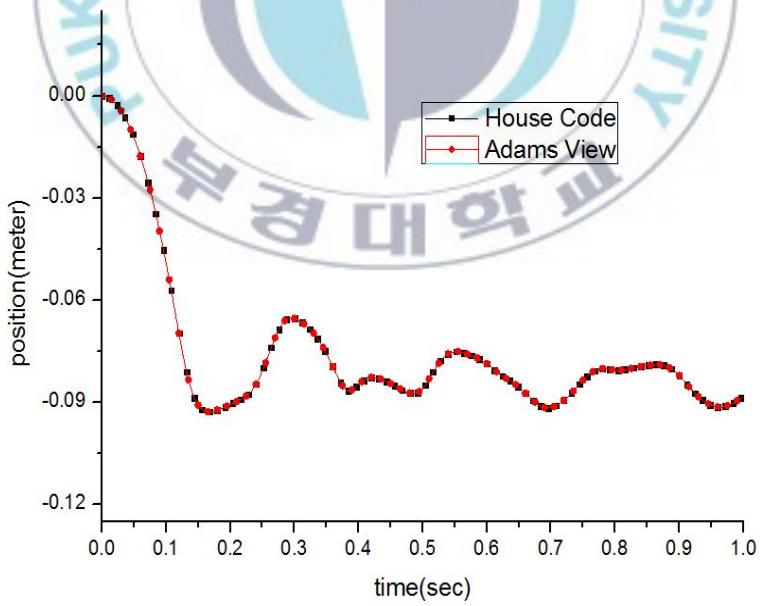


그림. 12 4번 물체의 y축 방향 위치

PUKYONG NATIONAL UNIVERSITY 부경대학교

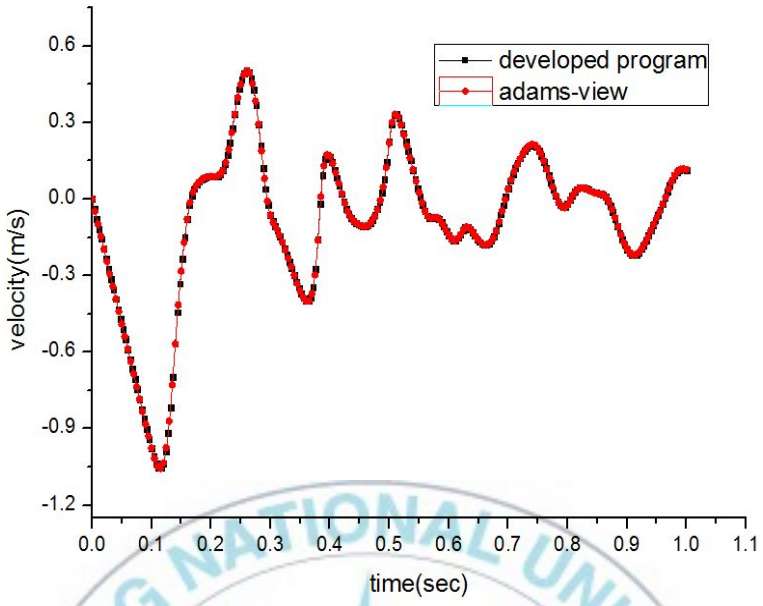


그림. 13 4번 물체의 y축 방향 속도

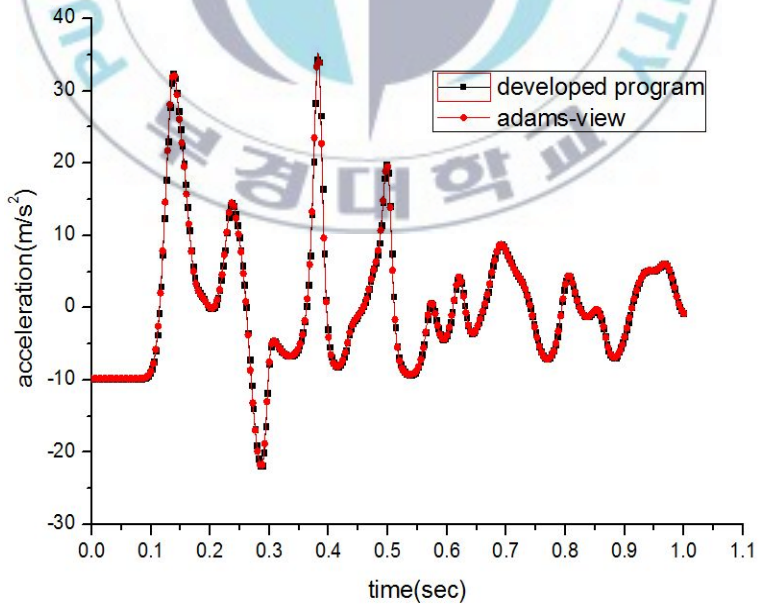


그림. 14 4번 물체의 y축 방향 가속도

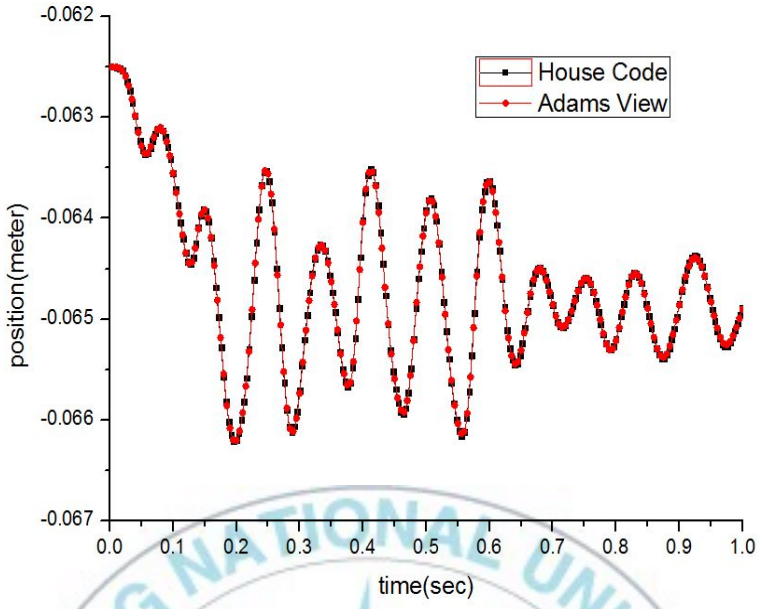


그림. 15 40번 물체의 z축 방향 위치

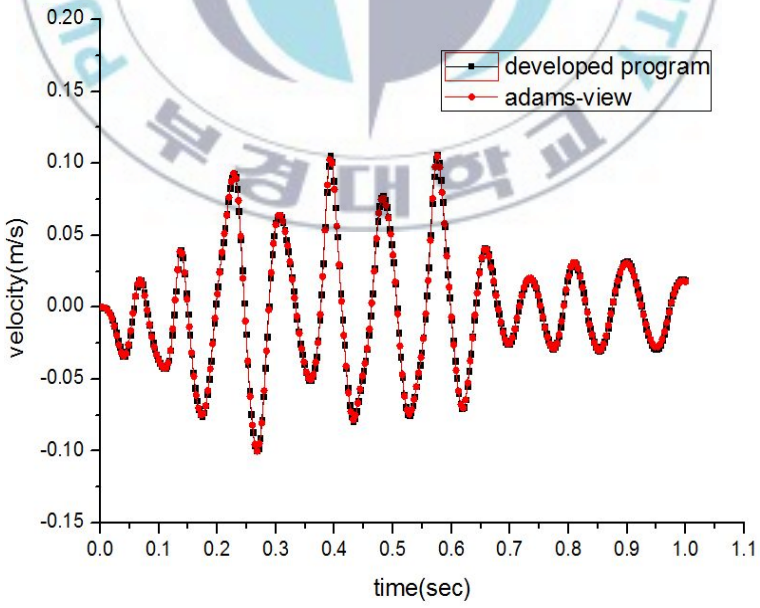


그림. 16 40번 물체의 z축 방향 속도

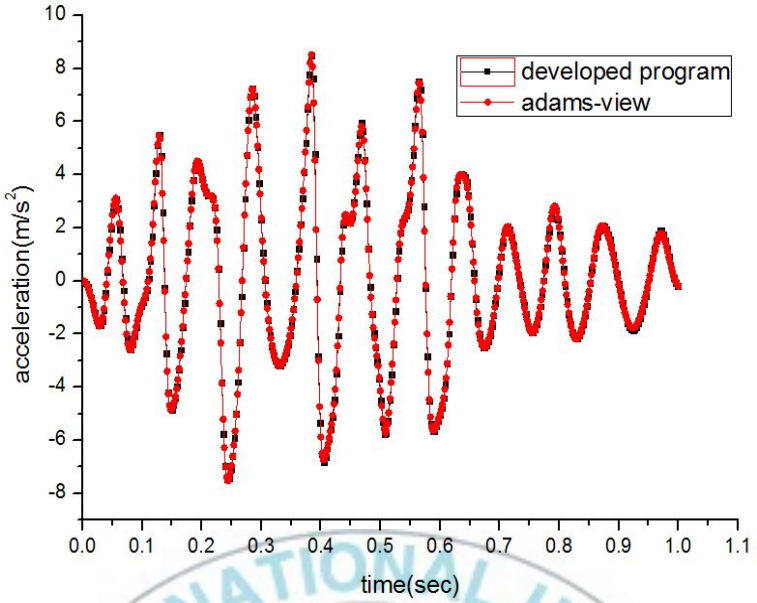


그림. 17　40번 물체의 z축 방향 가속도

그림 9~17에서 상용 프로그램과 비교하였을 때 거의 일치하는 것을 확인 할 수 있다. 표 6은 개발된 해석 프로그램과 상용 프로그램 결과의 차이의 절대 값에 대한 평균을 계산한 값으로 ADMAS 프로그램의 GSTIFF와 HHT 적분기의 1번 물체 x 방향의 가속도의 계산 값의 차이가 0.3035 인 것을 비교하면 개발된 해석 프로그램은 신뢰할 만한 정확성을 가진다.

표. 6　해석 결과 비교

	1번 물체 x 방향	4번 물체 y 방향	40번 물체 z 방향
position	1.2388e-005	2.5845e-005	1.2567e-005
velocity	7.1459e-004	0.0013	8.8938e-004
acceleration	0.0435	0.0914	0.0397

4. 대량 입자간의 접촉 모델

4.1. 입자 모델링

4.1.1. 입자의 공간 모델링

본 연구에서는 CUDA 병렬 프로그래밍의 효율성을 연구하기 위해서 명확한 입자의 물성치를 결정하기 보다 임의의 값을 선택하여 연구를 진행하였다. 입자의 크기는 다물체동역학 모델에서 스프링 댐퍼로 연결된 그물 형상을 이루는 입자들의 크기와 같도록 모델링 하였다. 접촉 판별을 위한 cell의 사이즈 또한 입자의 크기와 같도록 하여 효율적으로 색인화 작업을 할 수 있도록 구성하였다. 표 7은 시뮬레이션에 사용된 각 부분들에 대한 물성치를 나타내었다.

표. 7 대량 입자 거동 시뮬레이션 모델 물성치

입자 물성치	
반지름	0.015625(m)
질량	0.02(kg)
그리드-셀 물성치	
그리드 크기	$[2,2,2]^T$ (m)
셀 크기	$[0.03125, 0.03125, 0.03125]^T$(m)
경계 조건 물성치	
형태	큐브 형상의 정육면체
크기	$[2,2,2]^T$ (m)

입자들은 가로, 세로, 높이가 2미터로 일정한 큐브 형상의 정육면체 내부에서 운동을 한다. 입자의 지름은 0.03125(m)가 되고 셀 크기 또한 이와 같음을 알 수 있다. 그리드의 크기 또한 큐브의 크기와 같다. 그리드 한 변의 길이에 총 64개의 셀들이 위치하고 있으며 큐브 공간 내에 총 262,144(64x64x64)개의 셀들이 존재한다.

4.1.2. 입자 동역학 시뮬레이션

본 연구에서 입자동역학 시뮬레이션은 이산 요소법을 사용하여 수행되어진다. 뉴턴의 운동 방정식은 다음과 같다.

$$m_i\dot{v_i} = f_i, \dot{r_i} = v_i \tag{4.1}$$

$$I_i\dot{w_i} = t_i \tag{4.2}$$

여기서 아래첨자 i=1.....N으로 N개의 입자들을 나타낸다. m_i는 입자의 질량, r_i는 반지름, v_i는 속도, I_i는 관성 모멘트, w_i는 각속도, t_i는 토크를 나타낸다. 주어진 시간 간격에 대해서 다음 시간의 위치와 속도를 결정하기 위해 명시적인 적분법이며 입자동역학 해석에 자주 사용되는 velocity verlet 적분법을 사용하였다[19][20].

$$r_i(t+\triangle t) = r_i(t) + v_i(t)\triangle t + \frac{f_i(t)\triangle t^2}{2m_i} \tag{4.3}$$

$$v_i(t+\triangle t) = v_i(t) + \frac{(f_i(t) + f_i(t+\triangle t))\triangle t}{2m_i} \tag{4.4}$$

$$w_i(t+\triangle t) = w_i(t) + \frac{I^{-1}(t_i(t) + t_i(t+\triangle t))\triangle t}{2} \tag{4.5}$$

velocity verlet 적분기는 먼저 위치 데이터를 적분하고 힘과 토크를 계산하여 $f_i(t+\triangle t)$와 $t_i(t+\triangle t)$를 구하고 속도와 각속도를 적분한다. 본 연구에서 힘과 토크를 구하는데 사용되어진 방법들은 다음과 같다[21][22].

- gravity force
- Hertzian spring
- viscous damping
- tangential force model

그림 18은 velocity-verlet 적분기를 사용한 DEM 해석의 알고리즘을 나타내고 있다.

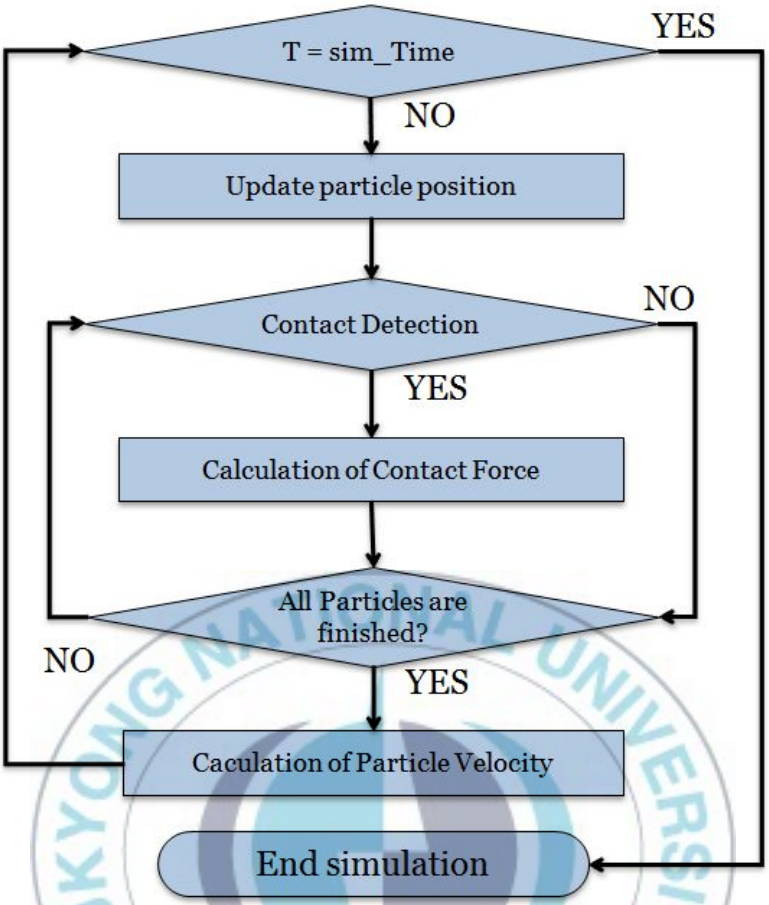
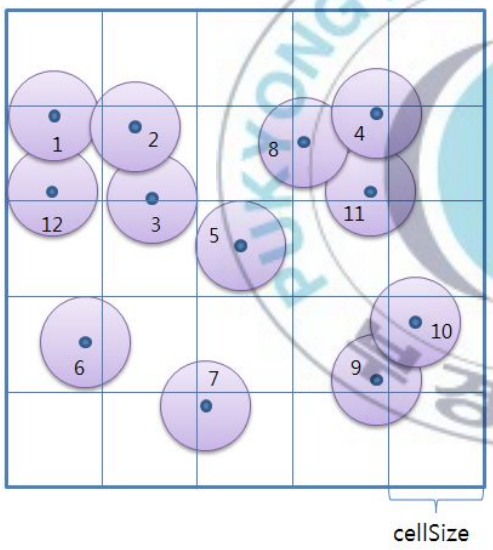


그림. 18 Velocity-Verlet 기반 DEM 해석 알고리즘

PUKYONG NATIONAL UNIVERSITY 부경대학교

4.2. 접촉 판별 알고리즘

대량 입자간의 상호 작용 관계를 정의하기 위해서는 먼저 서로간의 접촉을 판별하는 과정이 우선시 되어야 한다. 접촉을 판별하는 계산은 전체 계산시간의 가장 큰 시간 비용을 차지 하기 때문에 얼마나 효율적인 방법으로 접촉을 정의하냐에 따라 시뮬레이션 시간을 단축 시킬 수 있을 것이다. 본 연구에서는 효율적인 접촉 판별법으로 알려져 있는 neighboring-cell 접촉 판별법을 사용하였다[23]. neighboring-cell 접촉 판별법은 공간을 일정한 그리드로 나누어서 입자의 지름 길이의 cell이라는 공간을 만들어 입자가 어느 cell에 존재하는지를 정의한다. 그리고 바로 이웃한 cell에 존재하는 입자들과 접촉 여부를 확인하여 서로의 상호 관계를 정의 할 수 있다. 3차원 문제에서 하나의 임의의 셀은 3개의 인덱스를 가지고 구별되어 진다. 그림 19는 각 cell에 존재하는 입자들을 구분하는 형식을 보여주고 있다.



particle ID	cell ID
1	5
2	6
3	6
4	8
5	12
6	15
7	22
8	8
9	18
10	19
11	8
12	5

그림. 19 일정한 그리드에서 셀의 입자 구분

평면상의 입자들에 대해서 고려하면, 그림 19의 오른쪽 표와 같이 각 입자가 속한 cell아이디를 나타낼 수 있으며, cell ID를 기준으로 입자를 정렬하면 편리하다.

4.3. 접촉력 계산 모델

본 연구에서는 접촉력으로 normal force와 tangential friction force를 고려하였다.

4.3.1. Normal Force

본 연구에서는 Hertzian spring과 viscous damping을 사용하였다. Hertzian force와 viscous damping의 공식은 다음과 같다[19].

$$\boldsymbol{f}_{ij}^{e} = (\frac{2}{3}\tilde{E}\sqrt{R_{eff}}h_{ij}^{3/2})\hat{\boldsymbol{r}}_{ij} \tag{4.6}$$

$$\boldsymbol{f}_{ij}^{v} = -(\gamma_n\sqrt{R_{eff}}h_{ij}(\boldsymbol{v}_i-\boldsymbol{v}_j)\cdot\hat{\boldsymbol{r}}_{ij})\hat{\boldsymbol{r}}_{ij} \tag{4.7}$$

식 (4.6)에서 $\tilde{E}=E/(1-\nu^2)$이며, E는 Young's modulus 그리고 ν는 Poisson's ratio이다. 입자간의 침투량 h_{ij}는 $R_i+R_j-|\boldsymbol{r}_{ij}|, \boldsymbol{r}_{ij}=\boldsymbol{r}_i-\boldsymbol{r}_j$으로 표현 된다. $R_{eff}=R_iR_j/(R_i+R_j)$이고 $\hat{\boldsymbol{r}}_{ij}=\boldsymbol{r}_{ij}/|\boldsymbol{r}_{ij}|$로 접촉 평면에서의 normal 벡터이다. 식 (4.7)에서 γ_n은 damping parameter이다.

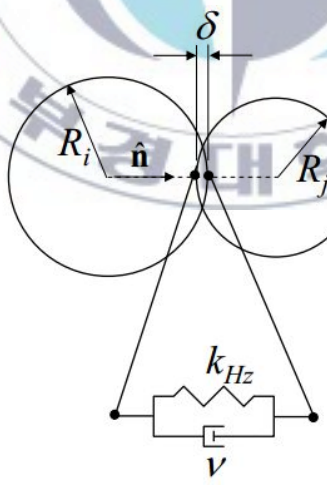


그림. 20 Normal 방향 Hertzian force 모델

4.3.2. Tangential Force

정지 마찰과 미끄럼 마찰은 tangential 스프링에서 설명되어진다. 입자들이 부딪쳤을 때 입자들 사이에는 접촉 점에 가상의 스프링이 부착되었다고 가정한다. 접촉이 이루어 지는 시간동안에 스프링 변위가 생기게 되고 그에 따른 힘이 스프링 방향으로 작용하게 된다. 입자 i에 작용하는 tangential force는 다음과 같이 표현된다[19][21].

$$\boldsymbol{f}_{ij}^{t} = \min\left\{k_s\delta_s + \nu_s(\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \hat{\boldsymbol{s}}_{ij}, \mu_s\left|\boldsymbol{f}_{ij}^{e} + \boldsymbol{f}_{ij}^{v}\right|\right\} \cdot \hat{\boldsymbol{s}}_{ij} \tag{4.8}$$

$$\hat{\boldsymbol{s}}_{ij} = \frac{\Delta\boldsymbol{v}_t}{\left|\Delta\boldsymbol{v}_t\right|},\ \Delta\boldsymbol{v}_t = (\boldsymbol{v}_i - \boldsymbol{v}_j) - ((\boldsymbol{v}_i - \boldsymbol{v}_j) \cdot \hat{\boldsymbol{n}}_{ij})\hat{\boldsymbol{n}}_{ij} \tag{4.9}$$

여기서 δ_s는 tangential 방향의 스프링 변위로써 $\delta_s = \left|\Delta\boldsymbol{v}_t\right| \times \Delta t$ 로 계산된다. 최종적으로 tangential force는 입자 i에 토크가 생기게 한다.

$$t_i = (\boldsymbol{r}_i \cdot \hat{\boldsymbol{n}}) \times \boldsymbol{f}_{ij}^{t} \tag{4.10}$$

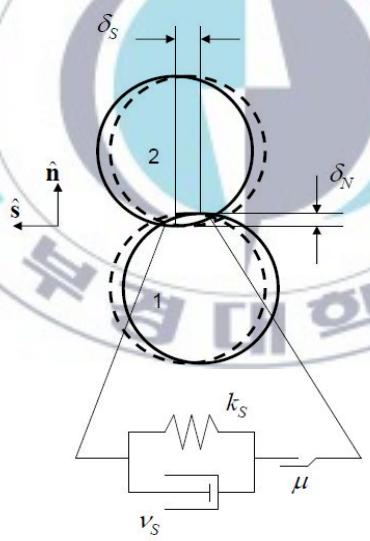


그림. 21 Tangential 방향 Friction force 모델

PUKYONG NATIONAL UNIVERSITY 부경대학교

4.4. 접촉 모델의 검증

4.4.1. 접촉 검증 모델링

입자들 간의 접촉 판별 여부와 접촉력의 검증은 간단한 모델링을 통해서 상용 프로그램인 adams-view와 상호 비교하였다. 입자 동역학 해석은 adams 프로그램과 상이한 적분기와 접촉력 계산법을 사용하기 때문에 적당한 계수 값을 결정하여 위치 데이터에 대해서 비교해 보았다. 본 검증에서는 다음 세 가지 경우에 대해서 위치 데이터에 대한 결과 값을 비교해 보았다.

- CASE 1 : 두 입자의 x축 방향 위치를 같게 하여 y축 방향의 위치 결과 비교

- CASE 2 : 두 입자의 x축 방향 위치를 다르게 하여 y축 방향의 위치 결과 비교

- CASE 3 : 정육면체 경계조건인 벽과 입자들 간의 접촉을 고려한 결과 비교

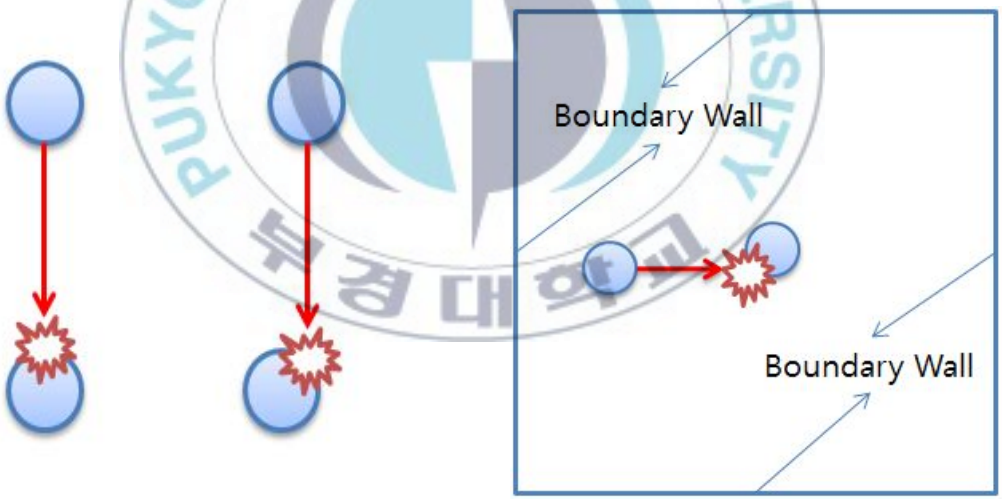


그림. 22 CASE 1, 2, 3에 결과 비교 예제 도식도

4.4.2. 접촉력 검증

본 연구에서 개발된 프로그램과 adams-view 프로그램의 접촉력에 따른 CASE 1, 2의 경우 y축 방향의 위치를 비교하였으며, CASE 3의 경우 i, j, 입자의 x, z 축 방향의 위치를 서로 비교함으로써 경계벽 사이의 접촉력에 따른 위치 변화를 비교 하였다.

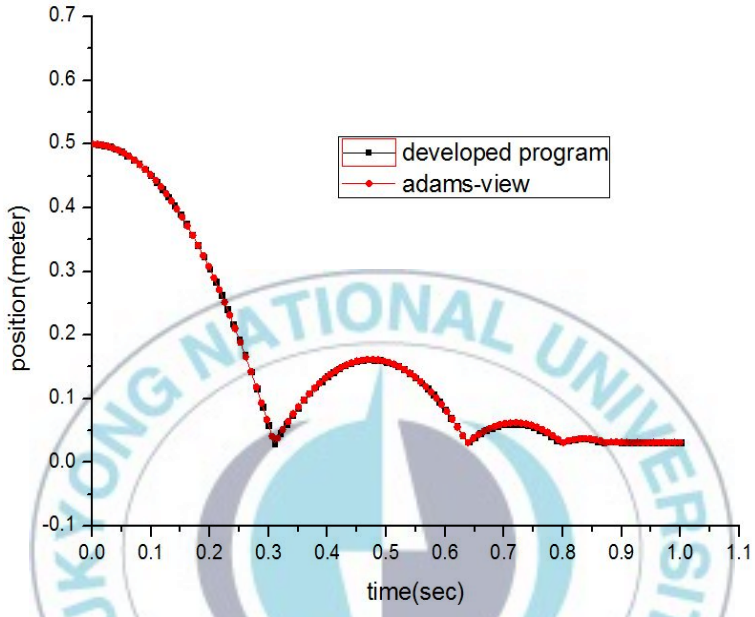


그림. 23 CASE 1에 대한 y축 방향 위치 결과

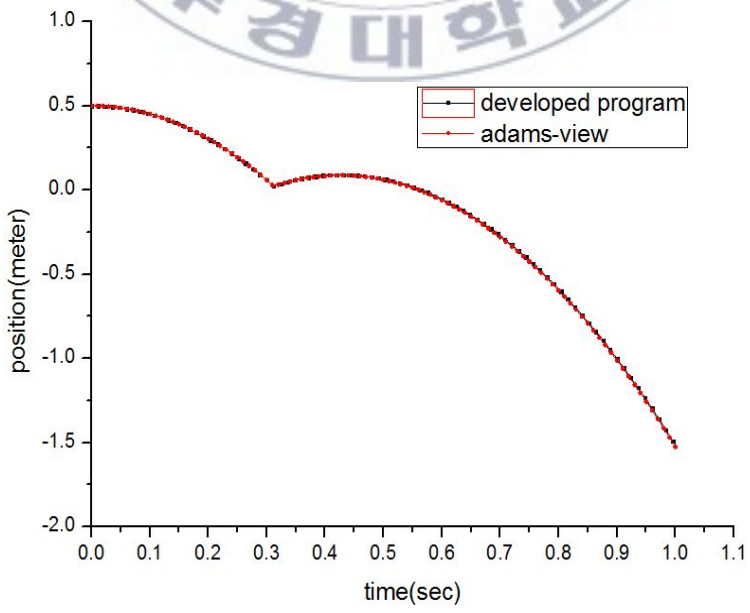


그림. 24 CASE 2에 대한 y축 방향 위치 결과

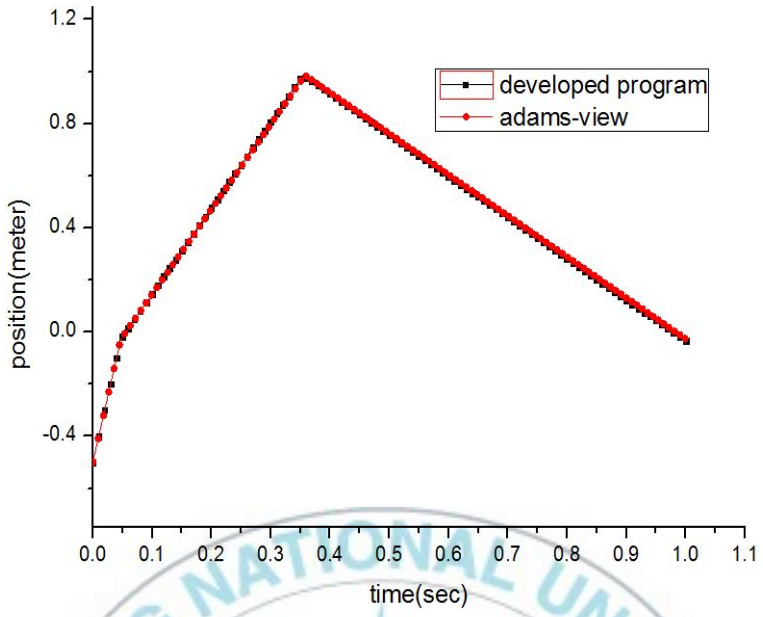


그림. 25 CASE 3에 대한 i 입자의 x축 방향 위치 비교

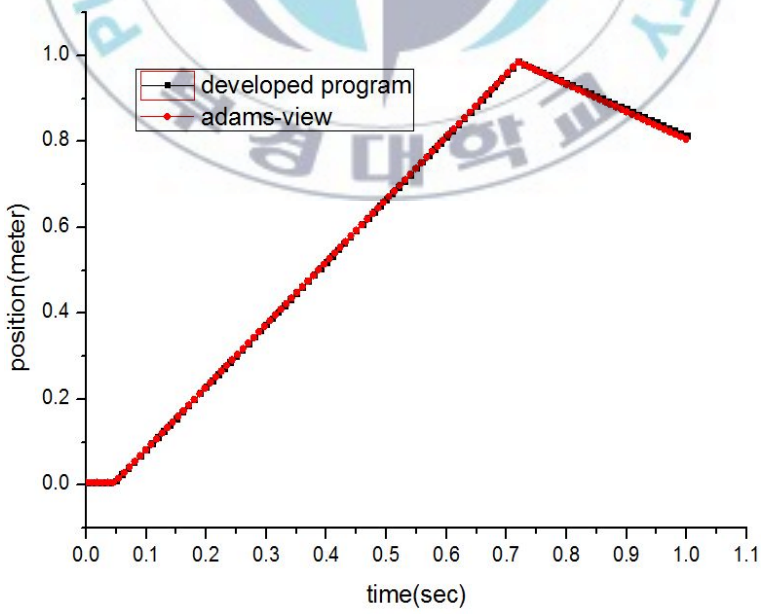


그림. 26 CASE 3에 대한 i 입자의 z축 방향 위치 비교

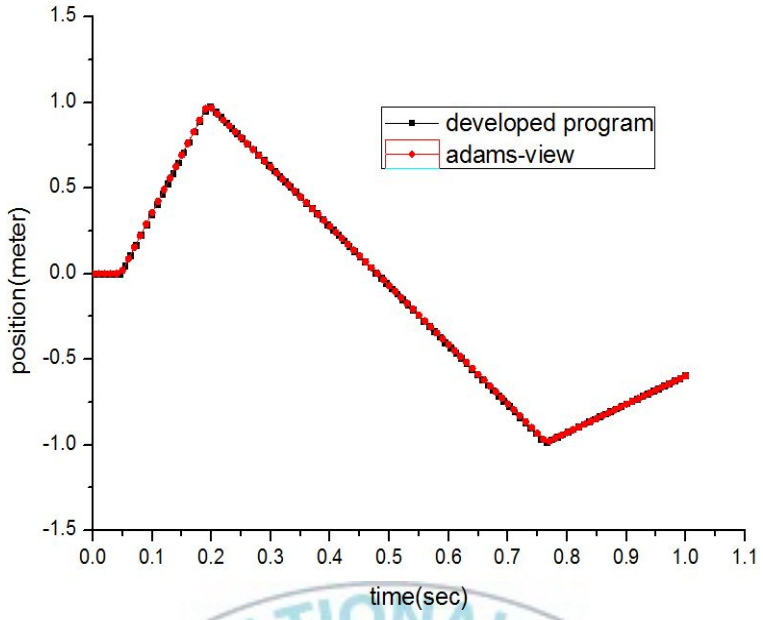


그림. 27 CASE 3에 대한 j 입자의 x축 방향 위치 비교

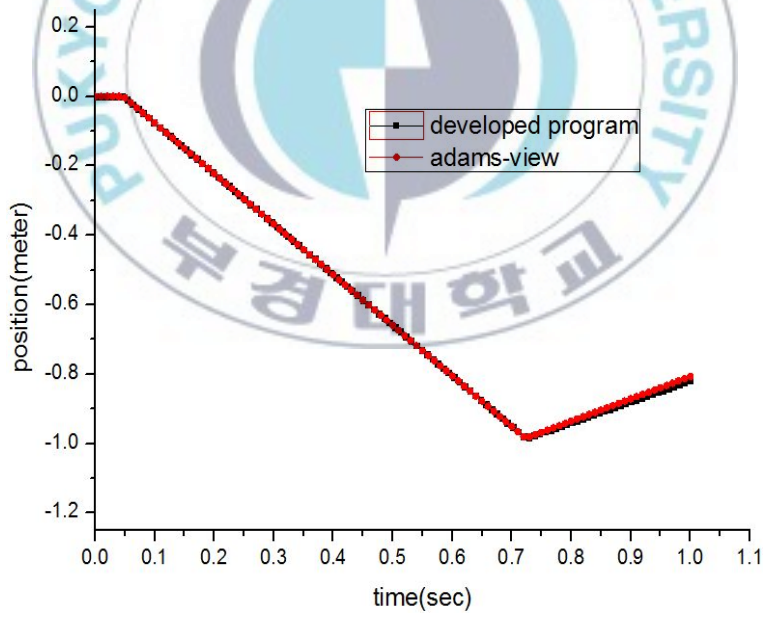


그림. 28 CASE 3에 대한 j 입자의 z축 방향 위치 비교

그림 23~28에서 확인할 수 있듯이 각 CASE에 대한 상용 프로그램과 개발된 프로그램의 상호 접촉에 의한 위치의 변화가 거의 일치함을 확인하였다.

5. CUDA기반 해석 프로그램

5.1. CUDA기반 다물체 동역학 해석

5.1.1. 병렬 다물체 동역학 해석법

순차적인 다물체 동역학 해석은 시스템을 이루는 물체의 수, 조인트의 수, 그리고 힘 요소의 수에 따라 반복 연산이 불가피 하게 요구되어진다. 하지만 시스템의 계수 행렬을 구성하고, 힘 벡터를 구성하는데 있어 반복연산은 서로 독립적인 연산 과정이기 때문에 병렬화 할 수 있는 조건을 만족한다. 다음 그림 29는 순차 프로그램과 병렬 프로그램의 계수 행렬의 일부를 구성하는 방법의 차이를 보여주고 있다.

```
01 : // sequential multibody dynamics programming
02 : for(bodyObj.begin(); bodyObj.end(); bodyObj++)
03 :     get Mass = bodyObj.mass;
04 :     get Iner = bodyObj.iner;
05 :     set coefficientMatrix.massPart = Mass;
06 :     set coefficientMatrix.inerPart = Iner;
07 : end for
```

```
01 : // parallel multibody dynamics programming
02 : __global__ void func(bodyObj)
03 :     get index = threadIdx.x; // unique index
03 :     get Mass = bodyObj.mass[index];
04 :     get Iner = bodyObj.iner[index];
05 :     set coefficientMatrix.massPart[index] = Mass;
06 :     set coefficientMatrix.inerPart[index] = Iner;
07 : end
```

그림. 29　순차 프로그램과 병렬 프로그램의 계수 행렬 구성법의 차이

순차 프로그램의 알고리즘은 매 반복문마다 각 각 물체의 정보를 받아 계수 행렬을 구성하고 있는 모습을 볼 수 있다. 이에 반해 병렬 프로그램 알고리즘은 CUDA의 내장변수인 threadIdx에 의해 각 스레드를 구분하는 고유한 인덱스를 얻고 이 인덱스에 따라 각각의 스레드는 각 물체의 질량과 관성모멘트를 얻는다. 그리하여 계수행렬에는 각 물체의 질량과 관성모멘트가 위치하는 고유한 위치가 존재하기 때문에 모든 스레드가 반복 없이 동시에 계수 행렬을 구성할 수 있다. 다른 구성요소에 대한 알고리즘도 이와 비슷한 과정을 통해 병렬화 된다.

5.1.2. 효율성 비교

순차 다물체동역학 프로그램과 CUDA기반의 병렬 다물체동역학 프로그램의 효율성을 비교하기 위해 그물 형상의 모델을 사용하였다. 그물 형상은 구형의 입자들과 상호 입자들에 연결된 스프링-댐퍼 힘 요소로 연결 되어 있기 때문에, 효율성 비교를 위해서 모델의 입자의 수와 스프링-댐퍼의 힘 요소의 수를 함께 증가 시켜 가면서 서로의 해석시간을 비교해 보았다. 효율성 비교를 위한 시뮬레이션 조건은 1초간 적분 시간간격 0.001초로 동일하게 하여 수행하였다. 해석 모델의 종류는 다음 표 8과 같다.

표. 8 TSDA와 BODY 수에 따른 효율성 비교 CASE

CASE	TSDA 개수	BODY 개수
1	2	3
2	24	21
3	48	40
4	84	49
5	120	96
6	224	176
7	360	280

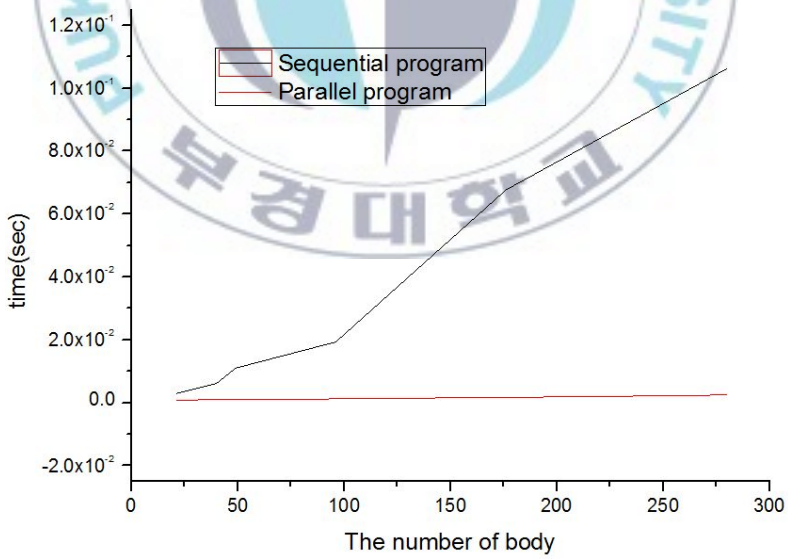


그림. 30 순차, 병렬 프로그램 간의 시스템 방정식 구성 시간 비교

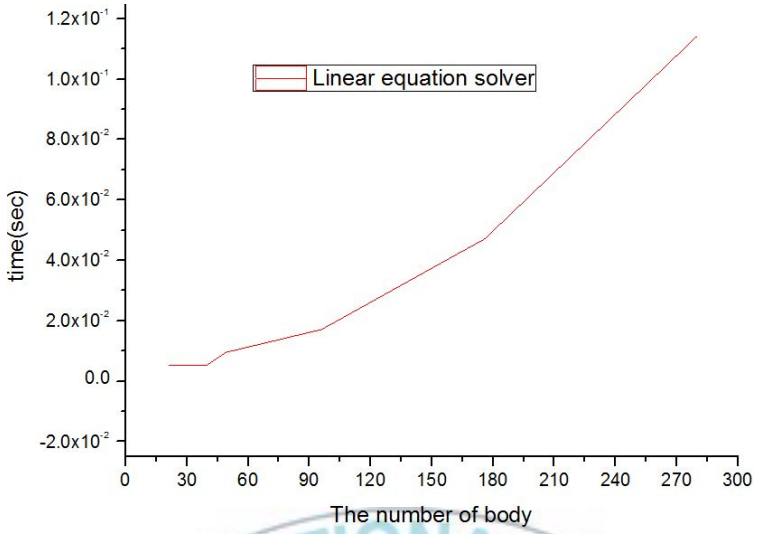


그림. 31 선형 방정식 풀이 함수의 계산 시간

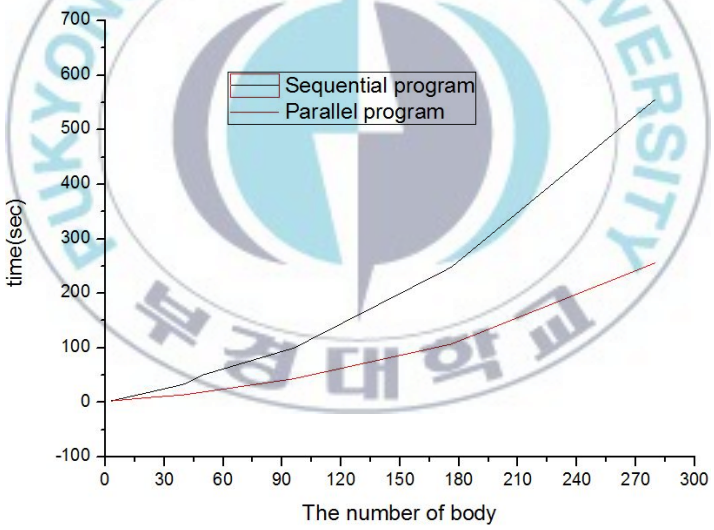


그림. 32 그물 형상 순차, 병렬 프로그램간의 1초 해석 시간 비교

그림 30는 시스템 방정식을 구성하는 과정을 순차 해석 프로그램과 CUDA 기반 병렬 해석 프로그램 간의 시간 비교를 표 8의 각 경우에 대해서 수행한 결과이며, 그림 32는 1초 동안 시뮬레이션 하는데 걸리는 해석시간을 비교한 결과이다. 그림 30의 경우 순차 해석 프로그램은 힘 요소와 강체 입자의 수가 증가 할수록 시간이 증가하는 폭이 큰 것을 확인 할 수 있다. 하지만 병렬 프로그램의 경우 계산 시간의 폭이 거의 일정함을 확인 할 수 있다. 이에 반해 그림 32의 경우 순차 해석 프로그램과 병렬 프로그램의 시간 차이가 점차 커지는 것을 확인 할 수 있지만 그림

30의 시간 증가 폭에 비해 병렬 프로그램의 시간 증가 폭이 어느 정도 큰 것을 확인 할 수 있다. 이는 그림 31에 나타난 선형 방정식의 해를 구하는 함수가 각 경우에 따라 시간 증가 폭이 크기 때문으로 확인 된다. 결과적으로 CASE 7에 경우 최대 약 2.17배의 시간 효율성을 보였다.

PUKYONG NATIONAL UNIVERSITY
부경대학교

5.2. CUDA기반 대량 입자 해석

5.2.1. 병렬 대량 입자 해석법

입자동역학에서 입자들 간의 모든 계산은 입자들 마다 각각 계산이 이루어진다. 각 입자들의 필요한 계산은 서로 독립된 계산 과정이기 때문에 모든 병렬 계산 과정이 모두 입자들 수만큼의 스레드를 할당 하여 계산을 할 수 있다. 입자들 수만큼 할당된 스레드들은 각 입자들의 정보들을 각각 담을 수 있다. 그렇게 하여 병렬적으로 각 스레드들이 각 입자에 필요한 정보를 얻기 위한 계산을 할 수 있다.

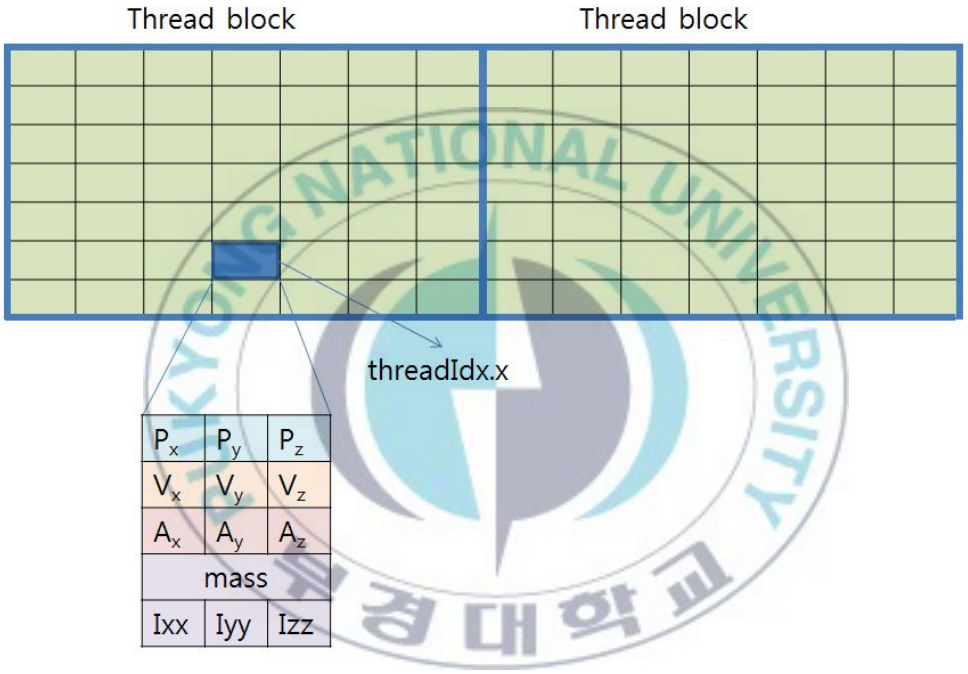


그림. 33 유일한 스레드 내 개별 입자의 정보 저장

그림 33은 커널 함수 실행시에 명시한 수만큼 생성된 블록과 스래드에서 하나의 블록의 각 스래드에 입자의 위치, 속도, 가속도, 질량, 관성 모멘트 등의 정보들이 할당된 상태를 나타내고 있다. CUDA 프로그래밍에서 스레드는 코어에 의해 동시에 실행되기 때문에, 그림 33과 같이 모든 스레드가 하나의 입자 정보를 가진다면 독립적인 계산에 대해서 병렬적으로 계산이 이루어진다.

5.2.2. 효율성 비교

입자동역학 해석에 대한 효율성 비교는 입자의 수를 증가 시켜가면서 10개의 CASE에 대해서 최소 125개에서 최대 10648개를 시뮬레이션 시간 1초에 대해서 적분간격 0.001초로 순차 해석 프로그램과 CUDA 기반 병렬 해석 프로그램의 해석 속도를 비교해 보았다. 효율성 비교에 사용되어진 CASE는 다음 표 9과 같다.

표. 9 입자의 수에 따른 효율성 비교 CASE

CASE	PARTICLES
1	125
2	343
3	729
4	1000
5	1728
6	2744
7	4096
8	5832
9	8000
10	10648

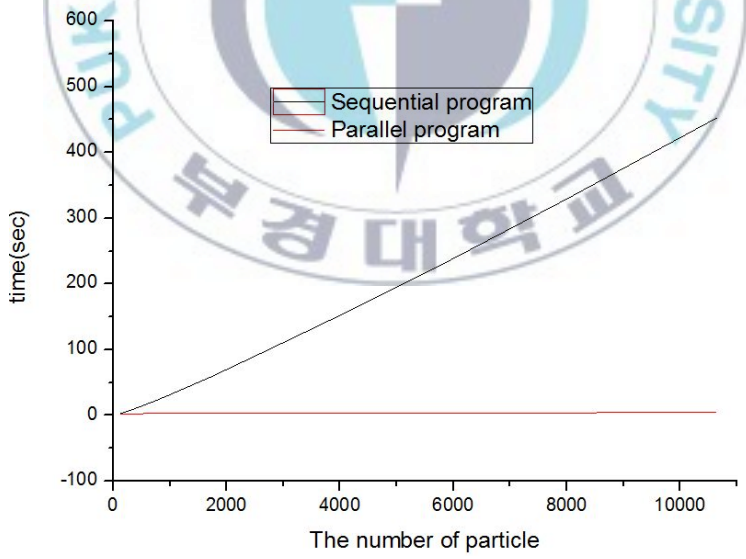


그림. 34 DEM 순차, 병렬 프로그램의 1초 해석 시간 비교

그림 34에서 구형 입자의 수에 따른 해석 속도의 차이가 매우 차이가 크게 나는 것을 확인할 수 있다. 효율성 비교 CASE로 최대 10648개의 입자를 비교한 결과를 보였지만, 실제 216,000개의 입자를 구성하여 시뮬레이션 하였을 때 1초동안 시뮬레이션 하는데 약 29초의 시간이 소요되었다. 이는 순차 해석 프로그램으로 1000개의 입자를 해석한 결과보다 2초가량 더 적은 해석시간이다.

5.3. 통합 해석

5.3.1. 통합 해석법

지금까지 다물체 동역학 해석과 입자동역학 해석에 대해서 각각 프로그램을 구성하고 해의 검증 및 효율성을 비교하였다. 다른 두 시스템의 통합 해석을 위해서는 먼저 각각의 시스템을 이루는 강체 물체들에 대한 접촉을 판별하고 접촉력을 계산하기 위해 서로의 위치와 속도가 공유 되어야 한다. 또한 계산된 힘들을 어느 시스템에 어느 위치에 가할 것인지를 알 수 있어야 한다. 이런 서로 공유되어야 할 부분들 이외에 두 시스템은 사용하는 적분기의 방식과 종류가 다를 수 있다. 본 연구에서는 다물체 동역학 해석은 암시적 적분법을 따르는 반면에 입자동역학 해석은 명시적 적분법을 따른다. 그러므로 두 적분기의 특성을 제대로 파악하고 알맞은 계산 알고리즘을 구성해야 한다.

그림. 35 통합 시스템 구성의 개략도

그림 35에서 각각의 시스템이 하는 역할들과 데이터의 전달 방향들을 도식화 하였다. 통합 시스템은 두 시스템의 정보를 통합하고, 접촉의 판별과 힘을 계산하여 각각의 시스템에 전달한다. 그림 36은 설명한 과정들을 나타내고 있다.

01 : while(!simulation_is_done)

02 : For particle, $r_i(t+\triangle t) = r_i(t) + v_i(t)\triangle t + \frac{f_i(t)\triangle t^2}{2m_i}$

03 : Integration both position and velocity for other system

04 : Excute contact searching and contact force functions

05 : For particle, $v_i(t+\triangle t) = v_i(t) + \frac{(f_i(t) + f_i(t+\triangle t))\triangle t}{2m_i}$

$$w_i(t+\triangle t) = w_i(t) + \frac{I^{-1}(t_i(t) + t_i(t+\triangle t))\triangle t}{2}$$

06 : Excute multi-body system predictor(t_{n+1})

07 : while(1)

08 : Evaluate $\hat{M}, \begin{bmatrix} -e1 \\ -e2 \end{bmatrix}$

09 : Excute contact searching and contact force functions

10 : Get multibody system force from integration force vector

11 : Calculate $\begin{bmatrix} \triangle \ddot{q}^{(k)} \\ \triangle \lambda^{(k)} \end{bmatrix} = \hat{M}^{-1} \begin{bmatrix} -e1 \\ -e2 \end{bmatrix}$

12 : Set $\ddot{q}_{n+1}^{(k+1)} = \ddot{q}_{n+1}^{(k)} + \triangle \ddot{q}^{(k)}$, $\lambda_{n+1}^{(k+1)} = \lambda_{n+1}^{(k)} + \triangle \lambda^{(k)}$

13 : Calculate integration formulas >> get $q_{n+1}, \dot{q}_{n+1}$

14 : if($norm(\begin{bmatrix} \triangle \ddot{q}^{(k)} \\ \triangle \lambda^{(k)} \end{bmatrix})$ < user_defined_tolerance) break;

15 : end while

16 : end while

그림. 36 통합 시스템 해석 알고리즘

입자 거동 해석은 명시적 적분법으로 이루어지기 때문에 Newton-Raphson의 반복 계산이 필요하지 않다. 하지만 다물체동역학 해석은 암시적 적분법이므로 정해를 찾기 위해 Newton-Raphson의 반복 계산이 요구 된다. 먼저 계산된 입자동역학에서 입자의 위치, 속도는 t_{n+1}의 값으로 생각할 수 있다. 그러므로 반복 계산에서 그물 형상을 이루는 입자들의 위치와 속도의 수정이 이루어질 때마다 접촉상태를 확인하고 힘을 계산하여 그물 형상을 이루는 입자들에 힘을 적용시켰다.

5.3.2. 통합 해석 시뮬레이션 결과

통합 해석의 시뮬레이션 결과로는 입자동역학의 입자를 고려하지 않은 그물 형상의 다물체 동역학 모델의 거동만을 해석한 결과와 자유입자를 고려한 통합 모델의 거동을 해석한 결과를 서로 비교하였다. 이는 통합해석에 있어 그물 형상을 이루는 물체와 자유입자와의 상호 접촉 여부와 그 결과를 알아보기 위해서이다. 통합 해석의 시뮬레이션은 물체의 수가 433개, 스프링 댐퍼의 수가 628개 그리고 자유입자의 수가 2744개의 조건으로 결과를 비교해 보았다.

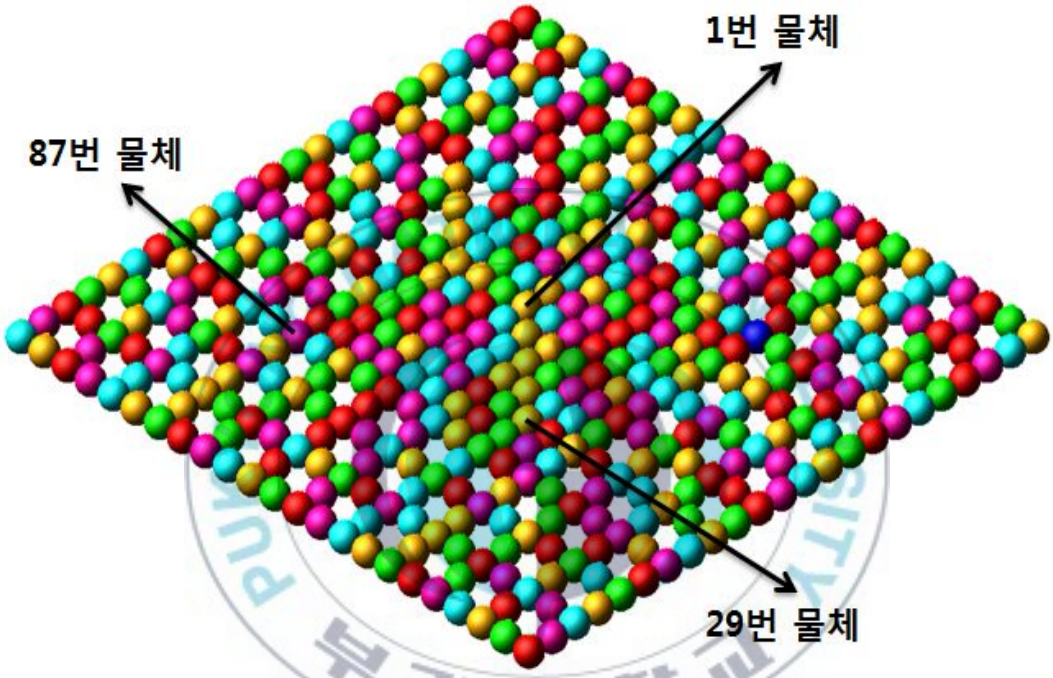


그림. 37 통합 해석 시뮬레이션에 사용하기 위한 그물 형상 모델

그림 37은 통합 해석 시뮬레이션 결과를 얻기 위해 사용된 그물 형상 모델이며, 각각 1번, 29번, 87번 입자의 거동 결과를 확인해 보았다.

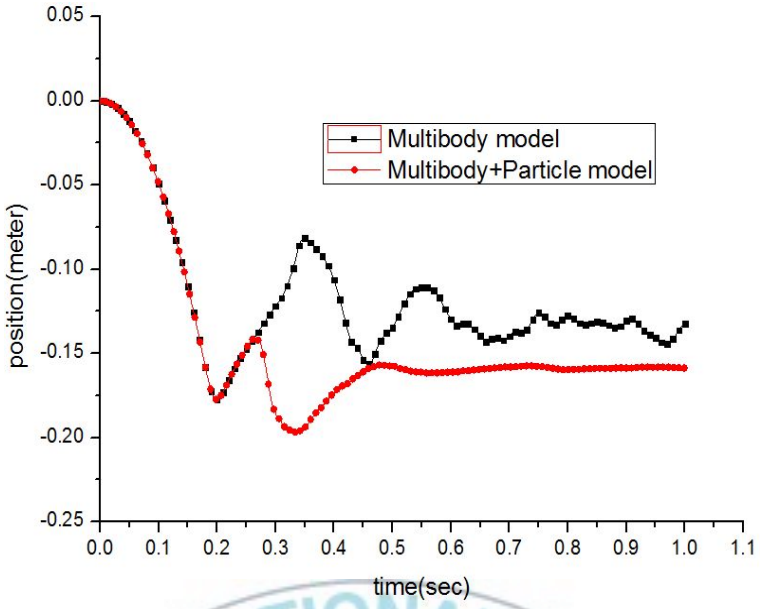


그림. 38 1번 물체의 y축 방향 위치

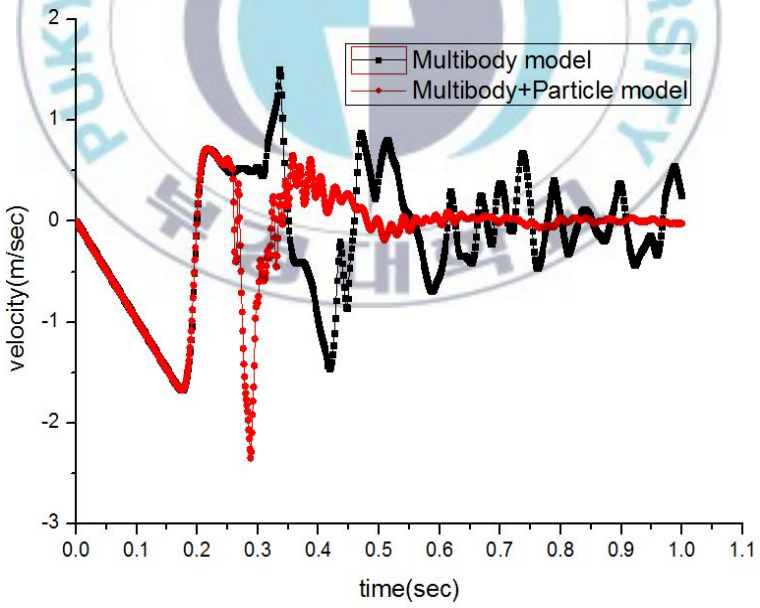


그림. 39 1번 물체의 y축 방향 속도

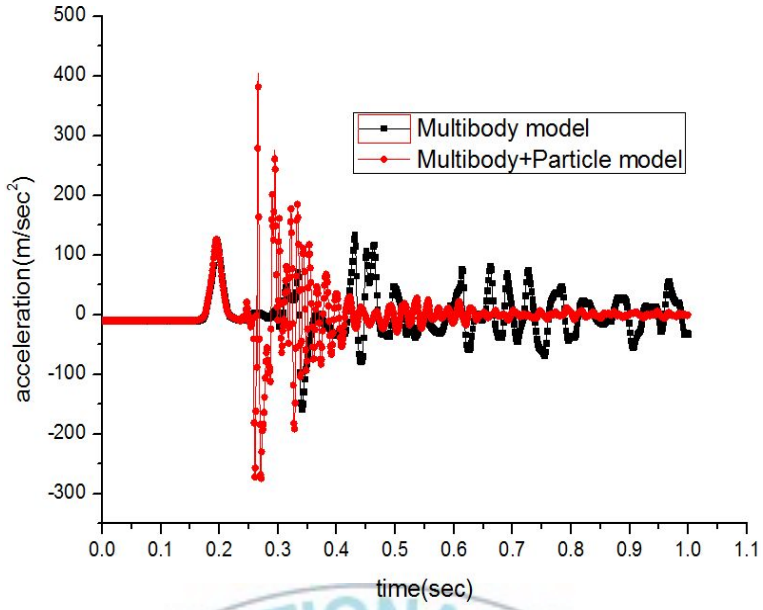


그림. 40 1번 물체의 y축 방향 가속도

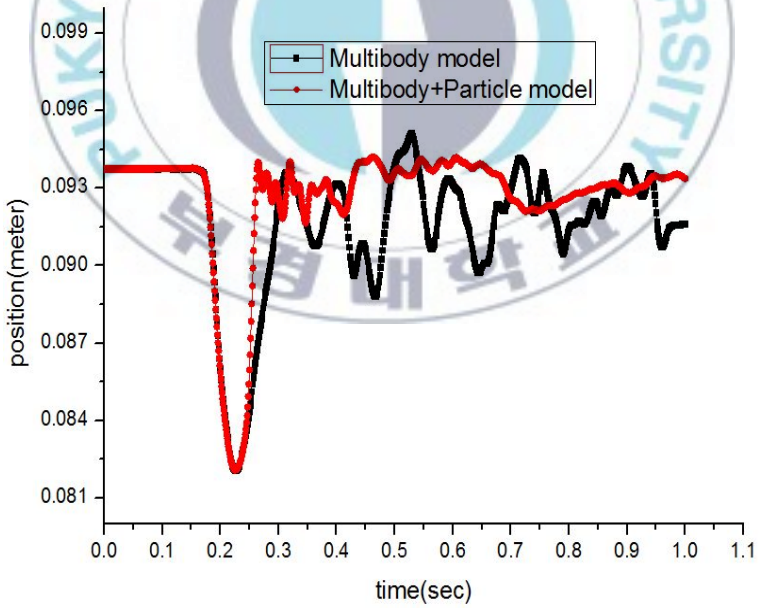


그림. 41 29번 물체의 x축 방향 위치

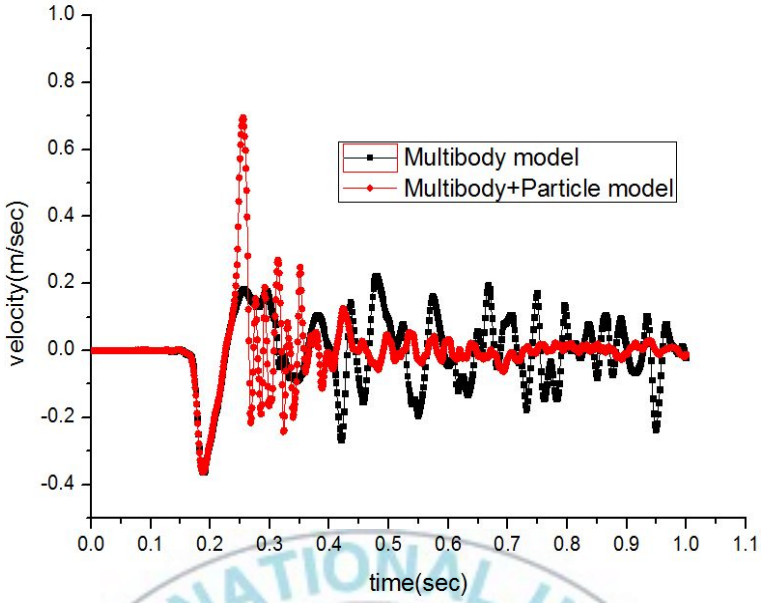


그림. 42 29번 물체의 x축 방향 속도

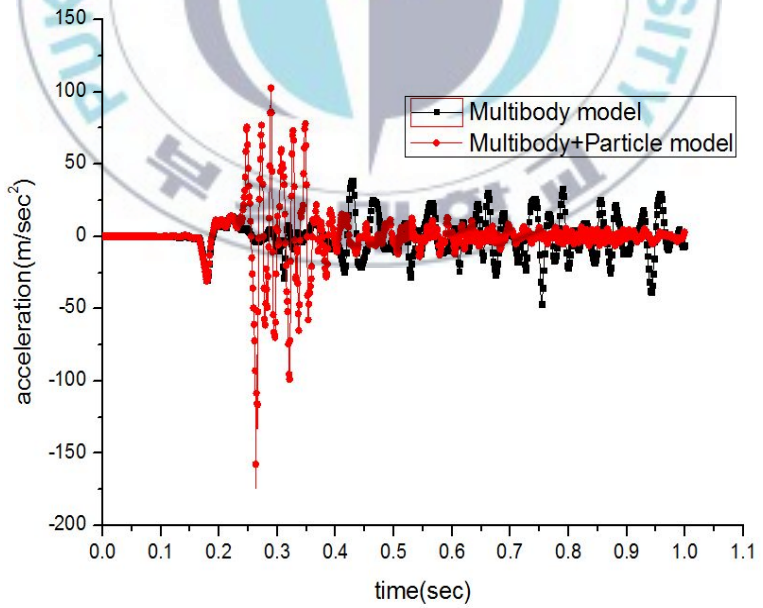


그림. 43 29번 물체의 x축 방향 가속도

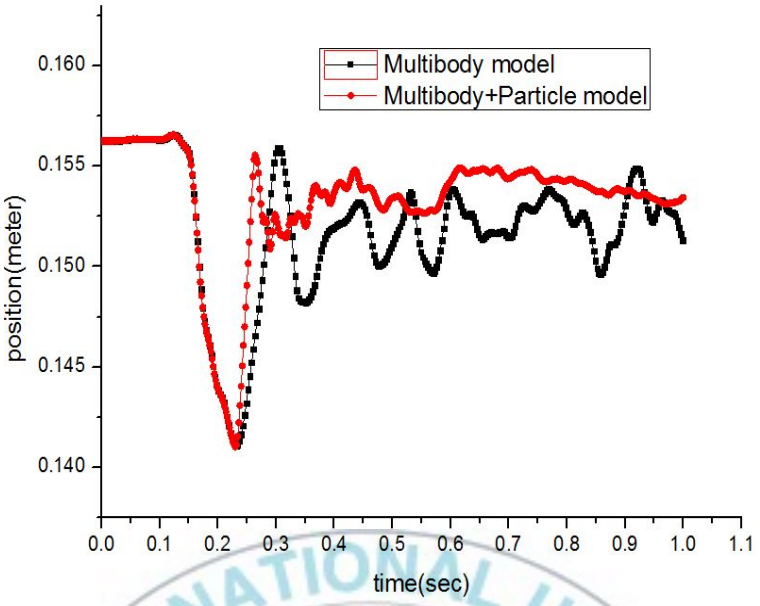


그림. 44 87번 물체의 z축 방향 위치

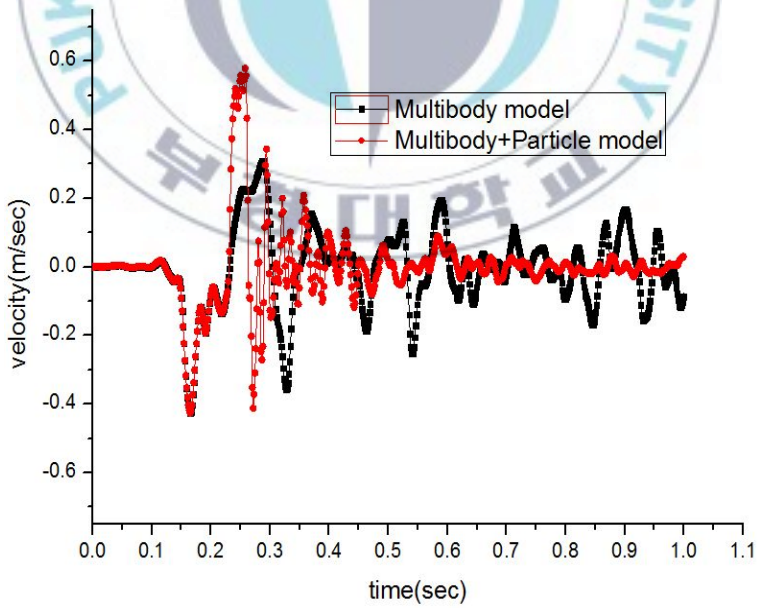


그림. 45 87번 물체의 z축 방향 속도

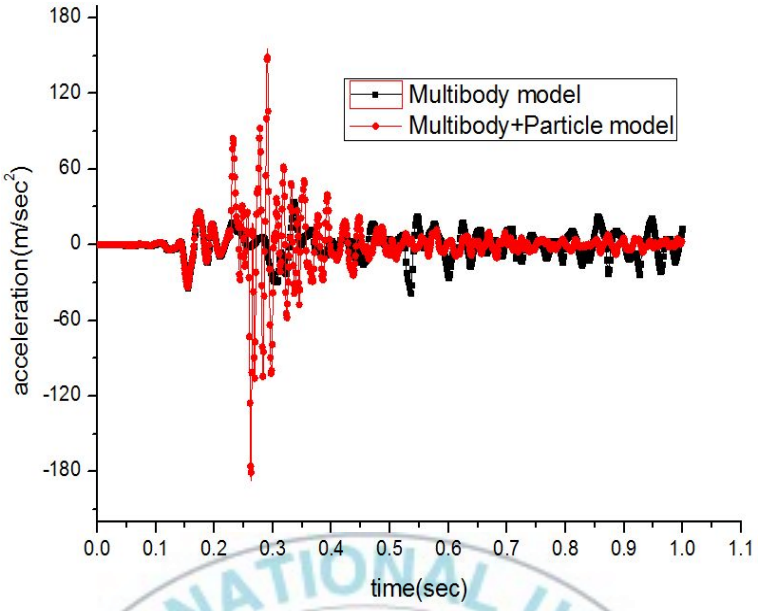


그림. 46　87번 물체의 z축 방향 가속도

그림 38~46에서 1, 29, 87번의 각 y, x, z 축 방향의 위치, 속도, 가속도를 나타내었다. 그림 38번의 경우 1번 그물 물체가 입자들의 무게로 인하여 일정 시간대에서 처진 상태를 유지하는 것을 확인 할 수 있다. 29, 87번 그물 물체 또한 충돌 후의 변화에서 유사한 거동을 보였다. 속도, 가속도 측면에서는 모든 결과에서 충돌 직후 많은 접촉들로 인해 급격한 속도, 가속도 변화를 보이고 있으며 일정 시간대에서 다량의 입자들의 무게로 인하여 그물 형상 모델의 진동 폭이 줄어드는 것을 확인 할 수 있다. 이 결과들로 그물 형상 모델과 다량 입자들이 서로 접촉함으로써 서로 영향을 주고받는 것을 확인 할 수 있다.

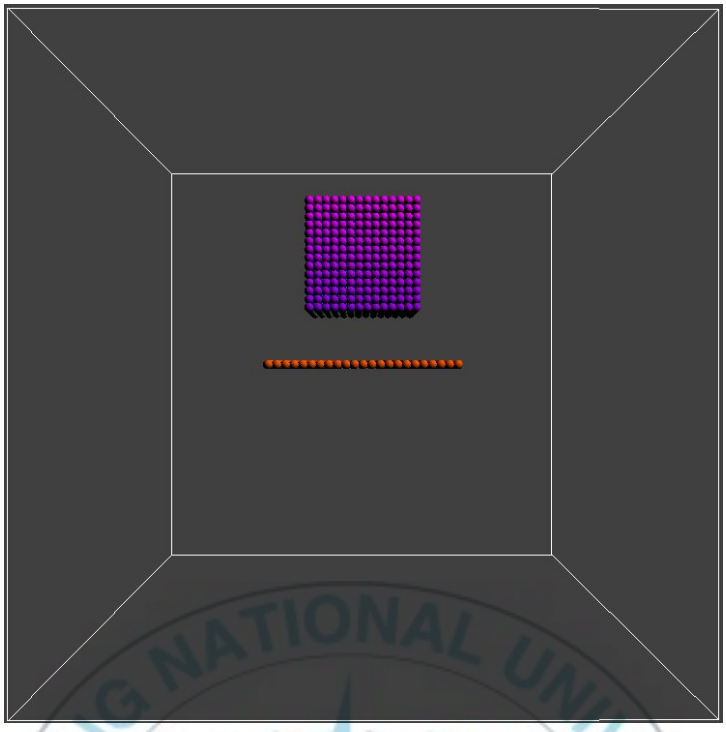


그림. 47　통합 해석 시뮬레이션 예제 모델의 애니메이션 (0.0초)

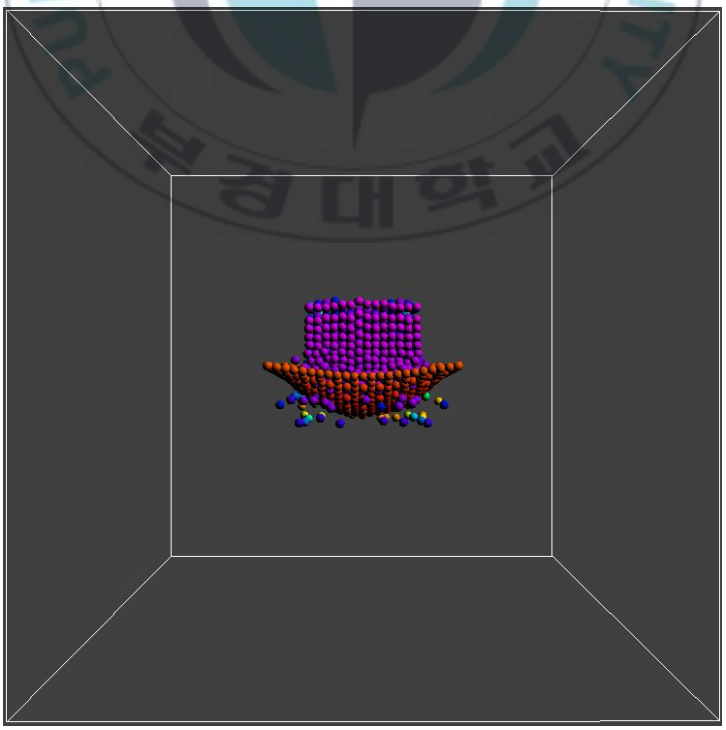


그림. 48　통합 해석 시뮬레이션 예제 모델의 애니메이션 (0.2초)

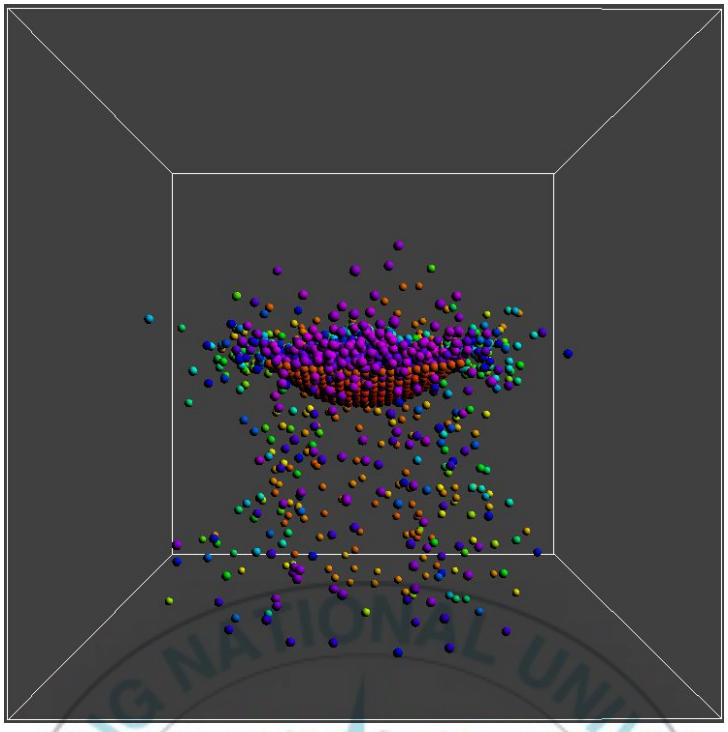


그림. 49　통합 해석 시뮬레이션 예제 모델의 애니메이션 (0.5초)

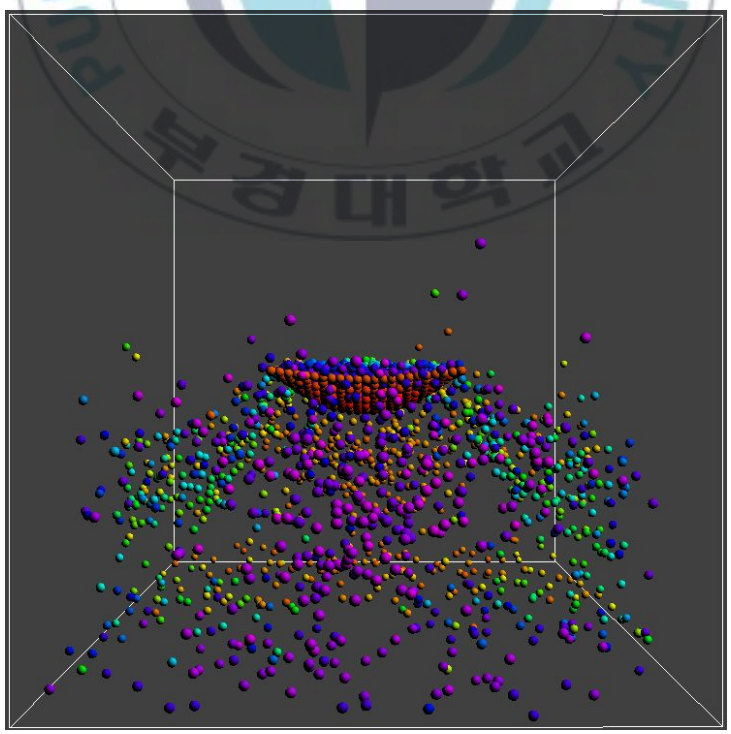


그림. 50　통합 해석 시뮬레이션 예제 모델의 애니메이션 (0.8초)

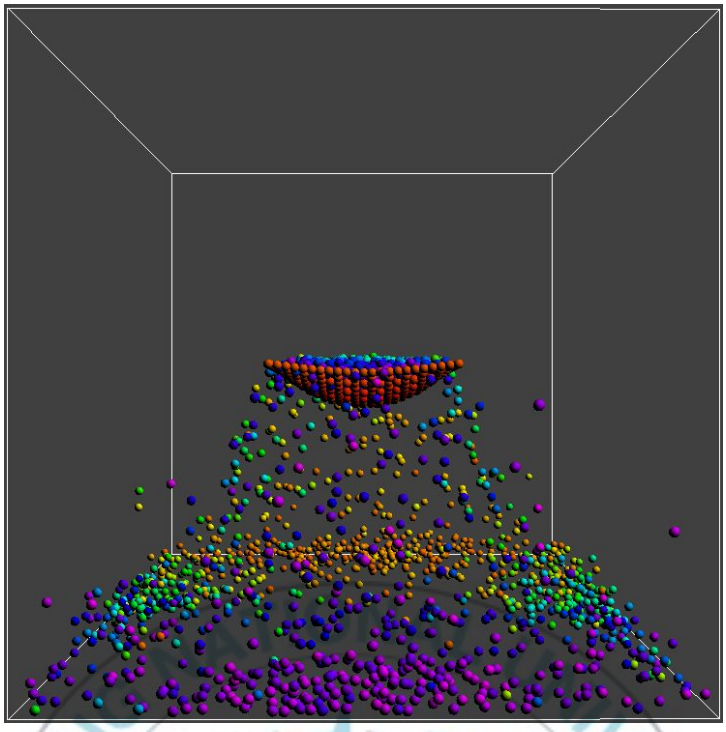


그림. 51　통합 해석 시뮬레이션 예제 모델의 애니메이션 (1.0초)

그림 47~51은 통합 해석 시뮬레이션 예제 모델에 대한 애니메이션 거동을 보이고 있다. 그림 48의 0.2초에서 다량의 입자들이 그물과 접촉하기 시작하면서 그림 37의 그물 형상 모델의 바깥쪽의 구멍으로 빠지기도 하고 충돌하여 바깥쪽으로 떨어지는 입자들을 확인할 수 있다. 그림 51의 1.0초 일 때 그물 위에 입자들이 머무르고 있는 것을 확인할 수 있으며, 이는 그림 38~46에서의 그물 형상 모델의 거동을 나타내고 있다.

5.3.3. 통합 해석 효율성 비교

통합 해석에서의 순차 해석 프로그램과, CUDA를 이용한 병렬 해석 프로그램의 해석 시간의 비교를 위해 자유입자의 수를 고정시키고, 그물 형상을 이루는 입자의 수를 증가시켜가면서 두 프로그램에 대한 해석 시간을 비교해 보았다. 자유입자의 수를 고정시킨 이유는 통합해석에서의 수정자 반복 연산에서 매 반복마다 위치, 속도, 가속도 값이 수정될 때 마다 그물 입자와 자유입자와의 상호 접촉 여부와 힘을 계산하기 때문에 너무 많은 해석시간을 필요로 하고, 해석시간의 차이는 자유 입자의 거동 해석에서 크게 나타나기 때문에, 통합해석의 효율성 비교에 있어서 자유입자를 고정시키고 그물 입자의 수를 변경시켜도 충분한 효율성 비교가 가능하기 때문이다. 다음 표 10에 통합해석 효율성 비교를 위해 사용된 조건들을 나타내었다.

표. 10 통합 해석 효율성 비교를 위한 해석 모델 CASE

자유입자 수	그물입자 수	스프링-댐퍼 수
1000	21	24
1000	40	48
1000	96	120
1000	176	224
1000	280	360

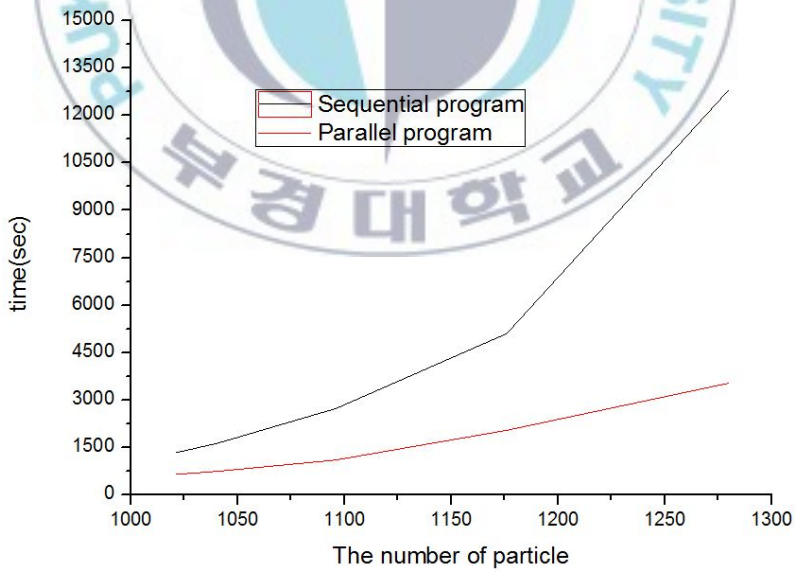


그림. 52 순차, 병렬 프로그램의 통합 해석 시간의 비교

그림 52에서 볼 수 있듯이 자유입자를 고정하고, 그물입자의 수 증가에 따른 순차 해석 프로그램과 병렬 해석 프로그램의 해석 시간의 차이가 점차 커지는 것을 확인 할 수 있다. 자유입자의 수를 고정 시켰고, 그물 입자의 수를 증가 시켰기 때문에 그물 입자의 수에 따라서 상호 접촉하는 입자의 수가 달라지는 것이 당연하다. 앞서 자유입자만을 고려한 효율성 비교에서 입자의 수가 많아질수록 그리고 입자들 간의 접촉 판별 계산이 많아질수록 해석 시간의 차이가 매우 크게 나타남을 확인 하였다. 이와 마찬가지로 그물입자가 늘어남에 따른 자유입자와의 접촉 판별 및 접촉력 계산량이 많아질수록 해석 시간의 차이가 점점 커지는 것을 확인 할 수 있었다. 통합 해석에서 그물입자가 280개, 스프링-댐퍼 힘 요소가 360개, 자유입자의 수가 1000개 일 때, 병렬 해석 프로그램이 순차 해석 프로그램에 비해 약 3.6배 가량 효율성이 좋아졌다. 그물입자가 176개, 스프링-댐퍼 힘 요소가 224개 일때의 효율성 차이가 약 2.5배에 못 미치는 것을 확인 할 수 있으며, 이는 더 많은 입자들이 서로 상호작용을 할수록 그 효율성의 차이는 점점 커질 것이다.

PUKYONG NATIONAL UNIVERSITY
부경대학교

6. 결 론

본 연구에서는 공간 구속 다물체 동역학 시스템과 대규모 입자들과의 상호 접촉 판별과 접촉력 계산에 CUDA를 이용한 병렬 프로그래밍이 사용되었을 때, 순차 해석 프로그램과의 해석 시간을 비교를 통한 효율성을 알아보기 위해, CPU를 이용한 순차 해석 프로그램과 CUDA를 이용한 병렬 해석 프로그램을 각각 개발하였다.

공간 구속 다물체 동역학 시스템의 해석을 위해서 stiff한 시스템에 안정적으로 계산을 하기 위해 암시적 적분법인 HHT-I3알고리즘을 사용하였으며, 대규모 입자의 거동을 표현하기 위해서 명시적 적분법인 Velocity-Verlet 알고리즘을 사용하였다. 입자들과의 상호 접촉을 효율적이고, 병렬화를 용이하게 할 수 있도록 neighboring-cell 접촉 판별법을 사용하였으며, 접촉력 계산에는 Hertzian spring, Viscous damping, Tangential friction 접촉력 모델을 적용하였다.

프로그램의 정확성과 효율성을 비교하기 위해서 작은 입자와 스프링-댐퍼로 이루어진 그물 형상의 다물체 동역학 모델을 고려하였고, 대규모 입자 거동 문제는 구속이 없이 공간상에서 자유로이 움직일 수 있는 입자들을 고려하였다. 다물체 동역학 시스템의 정확성은 Adams-view 프로그램과 비교하였으며, 개발한 해석 프로그램이 정확한 결과를 도출함을 확인하였다. 효율성 비교는 그물 형상에 경우 입자의 수와, 스프링-댐퍼의 수를 증가시켜가며 비교하였고, 대규모 입자 거동 문제 또한 입자들의 수를 증가시켜가면서 비교하였으며, 통합해석의 경우 자유입자의 수를 1000개로 고정하고 그물 입자의 수를 증가시켜가면서 효율성을 비교하였다. 그물 형상에 경우 입자 280개 스프링 댐퍼 360개 일 때 약 2.17배의 효율성을 나타내었으며, 자유입자의 경우 10648개의 입자가 구성되었을 때 약 100배에 가까운 효과를 나타내었으며, 통합 해석에서는 자유입자 1000개, 그물입자 280개, 스프링-댐퍼가 360개 일 때 약 3.6배에 가까운 효과를 보였다. 다물체 동역학 시스템이 포함된 경우에 효율성의 차이가 자유입자 운동에 비해 크지 않았는데, 다물체 동역학 시스템에서 선형방정식의 해를 구하는 함수를 희소행렬을 고려한 알고리즘을 적용한다면 더 높은 효율성을 기대할 수 있다.

참고문헌

(1) S. L. Grand, “Broad-Phase Collision Detection with CUDA” GPU Gems 3, Addison Wesley, 2007

(2) J. M. DANIEL, “On the Validation and Applications of a Parallel Flexible Multi-body Dynamics Implementation”, degree of Master of Science, 2007

(3) 이연희, “CUDA를 이용한 병렬형 충돌검사 및 동역학 시뮬레이션 연구”, 컴퓨터 정보통신 공학과, 이화여자대학교 대학원 석사 학위 청구논문, 2009

(4) 정혜영, “CUDA기반 평면 구속 다물체와 대량 입자들과의 충돌해석”, 메카트로닉스 공학과, 부경대학교 대학원 석사 학위 청구논문, 2012

(5) 박지수, 윤준식, 최진환, 임성수, “NVIDIA의 GPGPU를 이용한 수 많은 구형 접촉 입자가 포함된 다물체 동역학 해석, 대한기계학회논문집 A권, 제36권 제4호, pp. 465~474, 2012

(6) 정영훈, “CUDA 병렬 프로그램밍 : 고성능 GPGPU를 이용한 NVIDIA 병렬 컴퓨팅 아키텍처”, 프리렉, 2010

(7) 김역, “열혈강의 C++언어본색”, 프리렉, 2011

(8) NVIDIA,“CUBLAS LIBRARY v5.0 User Guide”, NVIDIA Corporation, 2012

(9) EM Photonics, “CULA Programmer’s Guide Release R16(CUDA 5.0)”, EM Photonics, Inc, 2012

(10) EM Photonics, “CULA Reference Manual Release R16(CUDA 5.0)”, EM Photonics, Inc, 2012

(11) NVIDIA CUDA C Programming Guide v5.0, NVIDIA Corporation, 2012

(12) NVIDIA CUDA C Best Practices Guide v5.0, NVIDIA Corporation, 2012

(13) Jason Sanders, Edward Kandrot, “CUDA by Example : An Instruction to General-Purpose GPU Programming ”, Addison-Wesley Professional, 2010

(14) R. Serban, E. J. Haug, “Analytical Derivatives for Multibody System Analysis”, Department of Mechanical Engineering, The University of Iowa, 1988

(15) P. E. NIKRAVESH, “Computer-Aided Analysis of Mechanical Systems”, Prentice-Hall International, Inc, 1998

(16) E. J. Haug, “Computer-Aided Kinematics and Dynamics of Mechanical Systems Volume 1 : Basic Methods”, ALLYN AND BACON, 1989

(17) D. Negrut, R. Rampalli, G. Ottarsson, A. Sajdak, “On the Use of the HHT Method in the Context of Index 3 Differential Algebraic Equations of Multibody Dynamics”, INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING, 2002

(18) MSC Software,“ADMAS User Manual”,http://www.mscsoftware.com

(19) C. Bierwisch, T. Kraft, H. Riedel, M. Moseler,“Three-dimensional discrete element models for the granular statics and dynamics of powders in cavity filling”, Journal of the Mechanics and Physics of Solids, Volume 57, Issue 1, pp 10-31, 2009

(20) N. S. Martys, R. D. Mountain, “Velocity Verlet algorithm for dissipative-particle-dynamics-based models of suspensions ”, The Americal Physical Society, Volume 59, Issue 3, pp 3733-3736, 1999

(21) N. H. DUONG, “Modeling the dynamics of toothbrush using discrete element method”, 금오공과대학교 대학원 공학석사학위논문, 2011

(22) R. Y. Yang, R. P. Zou, and A. B. Yu, “Computer simulation of the packing of fine particles, The American Physical Society, Volume 62, Number 3, September 2000

(23) S. Green, “Particle Simulation using CUDA”, NVIDIA Corporation, 2012

(24) W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery, “NUMERICAL RECIPES The Art of Scientific computing THIRD EDITION”, CAMBRIDGE, 2007

(25) D. Negrut, A. Tasora, M. Anitescu, H. Mazhar, T. Heyn, A. Pazouki, “Solving Large Multibody Dynamics Problems on the GPU”, Math. and Comp. Science Division, ANL/MCS-P1777-0710, 2010

(26) C. W. Gear, “Numerical Initial Value Problems of Ordinary Differential Equations”, Prentice-Hall, Englewood Cliffs, NJ, 1971

(27) N. Schafer, D. Negrut, “On the Potential of Implicit Integration Methods for Molecular Dynamics Simulation”, Journal of Computational Physics, 2009

(28) Lars Nyland, Mark Harris, Jan Prins, “Fast N-Body Simulation with CUDA”, GPU Gems 3, Addison Wesley, 2007

(29) C. ericson, “Real-Time collision detection”,MORGAN KAUFMANN Publishers, 2005

(30) F. V. Donze, V. Richefeu, S. A. Magnier, “Advances in Discrete Element Method Applied to Soil, Rock and Concrete Mechanics”, EJGE, 2008

(31) J. G. de Jalon, E. Bayo, “Kinematic and Dynamic Simulation of Multibody Systems-The Real-Time Challenge -”, Springer-Verlag, New-York, 1994

(32) A. Munjiza, “The Combined Finite-Discrete Element Method”, John Wiley & Sons, Ltd, 2004

감사의 글

2년동안 대학원생으로서 생활하면서 많은 분들의 도움을 많이 받았습니다. 아직 부족하고 배울것이 많지만 주변에서 힘이 되어준 분들로 인해서 졸업을 앞두게 되었습니다. 먼저 2년동안 저를 지도해 주시고 멘토가 되어주신 손정현 교수님께 깊은 감사를 드립니다. 또한 저의 논문을 심사해 주신 백운경 교수님과 정영석 교수님께 감사의 인사를 드립니다.

대학원 생활 동안에 졸업한 성준 선배, 진석이 그리고 혜영이 2년동안 계속 함께 했던 규석이, 성필이, 상욱이, 현경이, 뒤늦게 실험실에 들어와 많은 시간을 보내지 못했던 대우, 실험실의 선배로써 좀 더 잘 챙겨 주지 못해서 미안하고 힘들 때 힘이 되어주고 함께 있어줘서 정말 고맙다.

나의 어린 시절부터 함께 해온 나의 자랑스런 친구들아 너희들에게도 고맙다는 마음을 전한다. 바쁜 학교 생활에 많이 연락도 하지 못했고, 모임 자리에도 자주 함께 하지 못하였고, 가까이 있지는 않지만 친구라는 존재만으로도 나에게는 큰 힘이 되어준 친구들아 고맙다.

저를 낳아주시고 지금까지 물신양면으로 저를 보살펴주신 어머니, 아버지, 그리고 나보다 먼저 아버지가 된 나의 하나뿐인 동생 준호, 그리고 제수씨, 사랑하는 조카 예솔이 모두모두 정말 사랑하고 항상 나의 옆에서 힘이 되어줘서 고맙습니다.

약 1년 넘게 나와 함께 해온 가족들 다음으로 가장 사랑하는 혜련이 나의 일이 바빠 신경 써주지 못했던 적도 많았는데 나를 이해해주고 나를 항상 웃게 하고 사랑해줘서 너무 고마워 시간이 지날수록 익숙해져 서로에게 소홀해지기 보다 더 가까워지는 그런 너와 내가 되었으면 좋겠다.