# 병렬 GPU를 이용한 분자 도킹 시스템

## Molecular Docking System using Parallel GPU

**박성준**
호서대학교 게임공학과

Sung-Jun Park(sjpark@hoseo.edu)

**요약**

분자 도킹 실험은 일반적으로 계산 량이 매우 많아 슈퍼 컴퓨팅 파워를 요구하는 실험이다. 따라서 시간이 많이 소요되기 때문에 일반적으로 CPU가 탑재된 컴퓨터를 여러 대 묶어서 사용하는 분산 환경 혹은 그리드 환경에서 실험을 수행하고 있다. 이와 같은 실험 환경은 시간적, 공간적 제약성이 많아 일반적으로 과학자들이 접근하기가 어렵다. 그래서 근래에는 많은 CPU를 사용하기 보다는 월등히 성능이 높은 GPU를 병렬 화하여 과학 분야에 계산하는 연구가 매우 활발히 이루어지고 있는 추세이다. CUDA는 병렬 GPU 프로그래밍을 가능하게 하는 공개 기술이다. 본 논문에서는 이러한 CUDA 기술을 사용하여 분자 도킹 실험을 할 수 있는 시스템을 제안한다. 또한, 분자 도킹 실험에 있어서 중요한 에너지 최소화 계산을 병렬 화하는 알고리즘을 제안한다. 이와 같은 실험을 검증하기 위해 본 논문에서는 일반적인 CPU에서 분자 도킹 실험 시간과 본 논문에서 제안한 병렬 CPU 기반의 분자 도킹 시간을 비교 분석 하였다.

■ 중심어 : | GPGPU | CUDA | 분자 모델링 |

**Abstract**

The molecular docking system needs a large amount of computation and requires super-computing power. Since the experiment requires a large amount of time, the experiment is conducted in the distributed environment or in the grid environment. Recently, researches on using parallel GPU of far higher performance than that of CPU in scientific computing have been very actively conducted. CUDA is an open technique by which a parallel GPU programming is made possible. This study proposes the molecular docking system using CUDA. It also proposes algorithm that parallels energy-minimizing-computation. To verify such experiments, this study conducted a comparative analysis on the time required for experimenting molecular docking in general CPU and the time and performance of the parallel GPU-based molecular docking which is proposed in this study.

■ keyword : | GPGPU | CUDA | Molecular Modelaing |

## I. 서 론

분자 도킹 시뮬레이션은 분자 모델링 분야중의 하나이다. 분자 모델링은 분자의 행동 양식을 묘사하는 것을 의미하는데, 도킹 시뮬레이션은 그 중 두 분자간의 결합을 묘사하는 시뮬레이션을 의미한다. 도킹 시뮬레

* 본 연구는 2006년 호서대학교 재원으로 학술연구비 지원을 받아 수행된 연구입니다(20060019).
접수번호 : #081016-003
접수일자 : 2008년 10월 16일
심사완료일 : 2008년 12월 02일
교신저자 : 박성준, e-mail : sjpark@hoseo.edu

이션은 두 분자가 어떤 상태로 결합하는지를 예측하는 것이다. 그래서 어떤 바이러스에 대한 항체를 개발하는 등의 신약개발에서 사용된다[1][2]. 생화학 분야의 연구자들은 분자 모델을 조작 및 결합하는 과정을 통해, 계산된 수치를 통해, 실험이 성공적인지를 판단하게 된다. 도킹 시뮬레이션은 3차원의 분자의 정보를 기반으로 여러 화학적 수식을 통해 도킹 여부를 계산한다. 또한 분자와 분자간의 정보를 얻기 위해 여러 복잡한 수식을 계산하기 때문에, 분자의 크기, 수식의 개수에 따라 계산량이 기하급수적으로 증가한다. 그래서 분자 도킹 시뮬레이션 시스템은 주로 그리드나 웹서비스와 같은 분산처리 시스템, 병렬 처리시스템[3]에서 실행된다.

분산처리 시스템과 다르게, 최근에는 그래픽카드에서 연산처리를 담당하는 프로세서인 GPU(Graphic Processing Unit)가 새로운 병렬처리 후보로 대두되고 있다. 일반적으로 GPU는 빠르고, 화려한 그래픽 효과를 위해서 많은 수의 내부 프로세서를 탑재하고 있으며, SIMD(Single Instruction Multiple Data)구조로 처리하여 같은 계산을 많은 데이터에 대해서 수행하는 경우 최적의 성능을 낼 수 있다[4]. 이러한 특징을 통해 최근에는 많은 병렬처리 시뮬레이션에서 GPU가 사용되고 있다. GPU를 사용하여 과학 계산 분야에서 시뮬레이션 실험을 하기 위해서는 기존의 그래픽 API와 쉐이더 기술을 사용하였다. 그렇기 때문에 그래픽API의 구조와 그래픽스 지식이 있어야 프로그래밍이 가능했다. 하지만 최근에는 nVIDIA사에서 발표한 CUDA(CUDA : Compute Unified Device Architecture)[5]를 사용하여 복잡한 과학용 계산을 처리할 수 있다. CUDA는 C기반의 GPU 프로그래밍 라이브러리로서 GPU 환경에서 누구나 쉽게 개발이 가능하며, 기존의 그래픽 API나 Shader 언어가 제공하지 않았던 문법들도 지원하고 있기 때문에 쉽고 빠른 개발이 가능하다. 또한 여러 GPU를 병렬적으로 사용할 수 있도록 지원하고 있기 때문에 1대의 PC에 여러 GPU가 있는 경우, 동시에 계산하는데 사용할 수 있다. 본 논문에서는 최근 대두되고 있는 병렬처리 장치인 GPU와 CUDA를 이용하여 많은 계산량을 요구하는 분자 도킹시뮬레이션을 구현 하였다. 이때 여러 GPU를 동시에 사용하여 복잡한 수식을 나누어 계산하는 한 개 이상의 수식을 처리할 수 있는 분자 모델링 시뮬레이션 시스템을 제안한다.

본 논문에서는 제안하는 시스템의 성능을 검증하기 위해서 분자 구조의 에너지 계산 결과를 보고 분자 물질의 안정된 위치를 찾는 분자 도킹 시뮬레이션 실험에 적용하였다. 그리고 기존의 CPU를 이용한 시뮬레이션 실험과 성능을 비교 분석하였다.

## II. 관련 연구

### 1. GPGPU & CUDA

CUDA(Compute Unified Device Architechture)는 GPGPU라는 개념의 일환으로 개발되었다. GPGPU[9]는 General Purpose GPU의 약자로, 일반적인 목적으로 GPU를 사용하는 것을 의미한다. 1970년대 후반부터 1990년대까지 초창기 GPGPU에서는 주로 Ikonas, Pixel Machine, Pixel-Planes 5 등의 그래픽 머신에서 동작하는 응용에 대한 연구가 이어졌다[10-12]. 1990년대 후반에는 프로그래밍 가능한 그래픽 하드웨어의 등장으로 이를 Procedural Texturing과 Shading에 사용하는 연구가 진행되었다[13]. 또한 다른 연구자들은 그래픽 장치를 그래픽 응용이 아닌 다른 분야에 적용하였다. Robot Motion Planning과 보르노이 다이어그램의 계산, Unix 시스템의 비밀번호 해킹, 신경망 계산등 다양한 분야에서 사용되어 왔다. 주로 그래픽 API, Shader 언어, 텍스쳐를 사용하여 응용을 구현하였다. 하지만, 그래픽 API나 Shader 언어는 분기분이나 반복문 같이 자주 쓰이는 문법들을 지원하지 않기 때문에, 이와 유사한 효과를 내는 트릭을 사용하여 구현하였다. 그래서 그래픽스 지식이 없는 사용자들은 개발하기 어려웠다.

2007년 NVIDIA사에서 CUDA 발표하였다. CUDA는 GPU 프로그래밍을 하기 위한 라이브러리이며, C 프로그래밍 언어 기반의 문법을 사용하기 때문에 그래픽스 지식이 없어도 구현이 가능하며, 기존에 지원하지 않던 문법들도 지원하고 있다. 또한 병렬처리를 위한 특별한 문법을 포함하고 있고, 컴파일러도 제공하고 있다. 현재

그래픽스 하드웨어는 빠른 속도로 성능이 향상되고 있으며, CPU 보다 월등한 성능을 보이고 있다. CUDA는 이러한 고성능 그래픽 디바이스를 사용하여, 현재 많은 과학 응용 분야에서 사용되고 있다. 주로 우주의 각 행성 혹은 은하 간의 중력을 계산하여 시뮬레이션하는 N-Body 시뮬레이션의 경우, 많은 계산량을 필요로 하기 때문에 실시간에 처리가 어려웠으나, CUDA와 GPU를 사용하면서 실시간 계산 및 렌더링이 가능한 수준에 이르렀다. 또한, 유체 시뮬레이션이나 분자 동역학을 모사하는 분야에서 사용되고 있으며, 영상 처리등 다양한 분야에서 사용되고 있다[14-16].

## III. 병렬 GPU 기반의 분자 도킹 시스템

### 1. 분자 도킹 알고리즘

분자 도킹 실험은 리셉터(Receptor) 물질과 후보 물질(Ligand) 사이에서 에너지 상태가 안정한 결합위치를 찾은 작업이다. [그림 1]은 인면역결핍바이러스(HIV:Human Immunodeficiency Virus)와 신약후보물질사이의 도킹 작업을 나타낸 그림이다. 리간드는 리셉터 물질과 실시간으로 에너지 계산을 하면서 두 분자 물질의 결합이 안정 혹은 불안정인가를 검사해야 한다. 이때 에너지 계산의 중요한 요소는 두 분자 물질간의 거리 정보와 회전 정보를 기반으로 이루어진다. [그림 2]는 실시간적으로 에너지 계산을 화면상에서 보여준 그림이다. 리간드 물질이 리셉터 물질의 Acitive Site 근처에 위치하고 있을 때 에너지 계산 값이 0 에 가까운 것을 확인 할 수 있다. 이러한 전체적인 과정을 도킹 시뮬레이션 이라고 한다.

본 논문에서는 도킹 시뮬레이션을 위해 실시간으로 분자 구조의 위치에 따라 에너지 최소화 (Energy Minimization)[1] 계산이 이루어진다. 정확한 에너지 최소화 계산을 하기 위해서는 두 분자 물질의 거리 정보, 분자 구조 자체의 에너지 상태 정보, 수소 결합 정보 등과 같은 다양한 상태를 고려해야 한다.

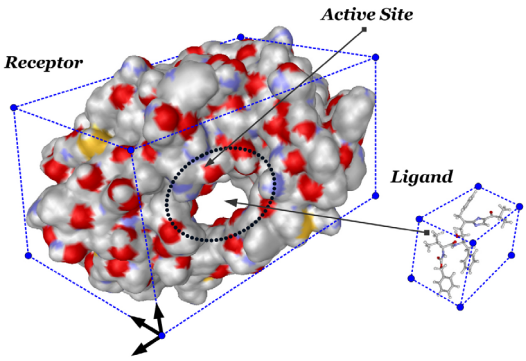



그림 1. 리셉터와 리간드사이의 도킹 작업

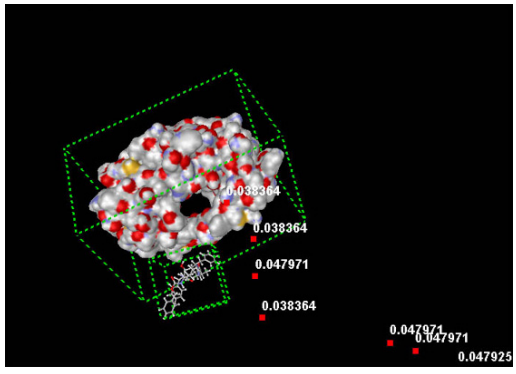



그림 2. 실시간 에너지 계산 과정

본 논문에서는 분자 구조의 다양한 상태 중에서 분자 구조의 안정한 상태를 유지할 수 있는 가장 중요한 요소인 전기적 힘 과 반데르 발스 힘 두 가지 요소만을 고려하여 에너지 안정화 계산을 구현하였다. 이 두 가지 요소에 대한 계산식은 리셉터 물질과 리간드 물질 사이의 두 분자 물질 간에 계산이 이루어지기 때문에 각 물질의 분자량의 크기에 따라서 계산량이 크게 달라진다. 그림3 은 에너지 최소화 계산에서 사용하는 전기적인 힘과 반데르발스 힘을 계산하기 위한 수식이다.

그림 3에서 보는 바와 같이 리셉터와 리간드의 각각의 원자 사이의 전기적인 힘 [수식 (2)]과 반데르발스 힘 [수식 (3)]을 계산하고, 그 합[수식 (1)]이 최종 결과값이기 때문에 리셉터와 리간드의 분자량 크기가 클수록 계산 량은 기하 급수로 늘어나고 두 분자 물질 사이의 거리에 따라 결과에 큰 영향을 미친다. 이러한 문제점을 해결하기 위해서 일반적으로 리셉터의 크기를 분

할하는 분산 및 병렬 처리 알고리즘 계산 방법에 대한 연구가 이루어지고 있다[8]. 하지만 이런 경우에는 보다 정밀한 계산을 하기 위해 확장을 해야 하는 경우, 리셉터의 각 분리된 분자 물질 데이터들 간의 상호계산이 필요하므로 데이터들 간의 통신을 위한 추가적인 비용이 발생한다.

$$E_{total} = E_{elec} + E_{vdw} \quad (1)$$

$$E_{elec} = \sum_{exd(i,j)=1} \frac{q_i q_j}{4\pi r_{ij}} \quad (2)$$

$$E_{vdw} = \sum_{exel(i,j)=0} (\left\{ (\frac{v_r + v_l}{r_{ij}^{12}} - \frac{v_r + v_l}{r_{ij}^{6}}) \right\} \sqrt{e_r e_l} \quad (3)$$

$where$
$i$ : $i$번째 리셉터 원자
$j$ : $j$번째 리간드 원자
$q$ : 전하량
$v$ : 반데르발스 반지름
$r_{ij}$ : $i, j$ 원자 사이의 거리
$e$ : 입실론

그림 3. Energy Minimization 공식

본 논문에서는 위와 같은 문제점을 해결하기 위해서, GPU기반의 논리적 병렬 계산 처리 알고리즘을 제안한다. 제안하는 알고리즘은, 리셉터와 리간드의 데이터 구조를 병렬화 하는 것이 아닌, 각 그래픽 카드당 처리 계산 알고리즘을 병렬화 하여 계산함으로써 처리속도를 향상 시키며, 데이터 구조를 병렬화 하여 처리하는 경우 발생하는 분자 구조 데이터간의 상호 계산의 충돌 문제를 효율적으로 해결할 수 있다.

## 2. 시스템 구조

본 논문에서 제안하는 시스템은 병렬 GPU를 사용하여 [그림 3]에서 제시한 에너지 최소화 계산을 수행하기 위해 전기적인 힘 계산과 반데르발스 힘 계산은 각각 서로 다른 GPU에서 병렬로 처리하게 된다. 각 수식들은 실험에 사용하는 분자의 크기가 클수록 계산 량이 많아지기 때문에, 순차적으로 수식을 계산하는 경우 계산 시간이 상당히 오래 걸리기 때문에 빠른 결과를 얻을 수 없다.

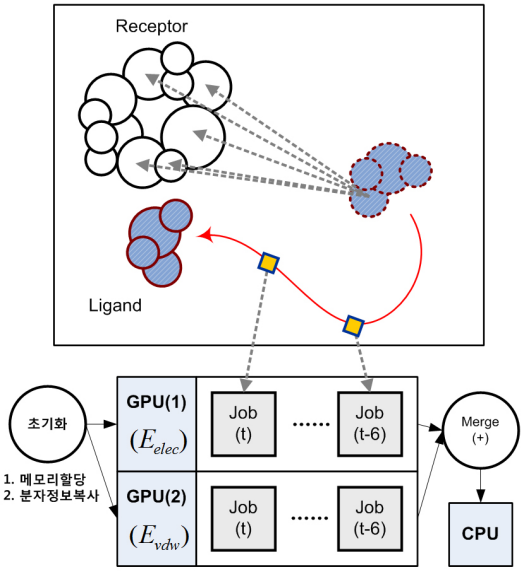



그림 4. 멀티 GPU 계산 과정

[그림 4]는 개략적인 계산 과정을 보여준다. 먼저 GPU에 데이터를 넘기기 위해서 CPU에서는 분자의 위치, 전하량, 반지름 등 기본 정보를 직렬화하여 전송한다. 이런 기본 정보는 실험 내내 변화하기 않기 때문에 대량의 데이터 전송은 계산하기 전 한번 이루어진다. 사용하는 GPU는 두 개 이상이기 때문에 쓰레드를 통해서 동시에 전송 작업을 진행한다. 넘겨진 데이터는 GPU의 메모리에 저장된다. 매 계산의 시작에 있어서 3차원 공간상에서 분자 물질의 위치가 바뀔 때마다 위치 정보를 갖고 있는 행렬만 전송되고, 계산이 종료된 후 계산 결과를 다시 GPU에서 CPU로 전송된다.

## 3. 병렬 GPU

CUDA에서 GPU로 계산을 하기 위해서는 일반적으로 초기화, 계산, 종료로 3단계로 구분된다.

- 초기화 단계는 주로 계산에 필요한 데이터를 위해 메모리 할당과 CPU로부터 데이터 복사를 수행한다.
- 계산은 주어진 데이터를 통해 수식이나, 알고리즘

을 수행하는 단계를 말한다.

- 종료는 할당된 메모리를 해제한다.

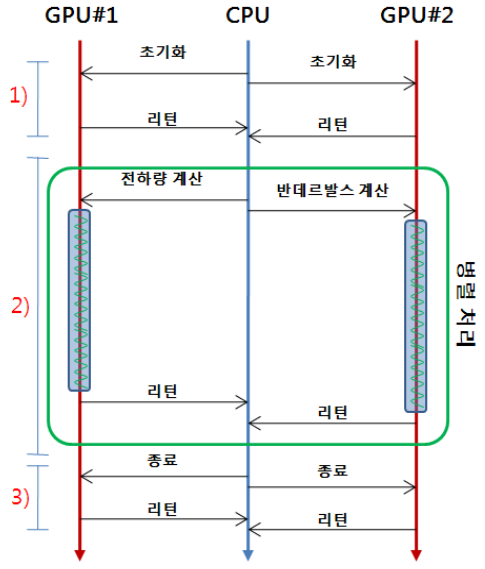



그림 5. CPU 와 GPU 간의 관계

CUDA에서는 이 모든 단계를 하나의 쓰레드에서 수행해야 한다. 즉, 위의 3단계가 같은 쓰레드에서 수행되어야 한다. 하지만, 제안하는 시스템을 구현하기 위해서는 계산이 수행되기 전에 분자의 위치와 회전 정보를 알려주어야 한다. 또한 계산된 결과 값을 표현하기 위해서 계산된 결과를 다시 CPU로 전송해야 한다. 그래서 GPU 계산을 위해 생성되는 쓰레드(이하 GPU쓰레드)는 언제 계산을 시작하는지, 언제 계산이 끝나는 지를 메인 쓰레드에게 알려야 하고, 이를 해결하기 위해 시그널 객체를 사용하여 구현하였다.

## 4. CUDA 알고리즘

본 논문에서 제안한 도킹 시뮬레이션 알고리즘을 구현하기 위해서는 두 가지 방법을 사용할 수 있다. 첫 번째 방법은 모든 분자 쌍에 대해서 계산 쓰레드를 만들어서 계산하는 방법이다. 이 방법의 경우 생성되는 쓰레드의 개수는 (리셉터의 분자 수) * (리간드의 분자 수)이므로 엄청난 많은 쓰레드가 생성된다. 이러한 방법은 구현하기 쉬운 장점은 있지만, 하나의 쓰레드가 처리하는 계산 량이 적어 오버헤드가 생기며, 계산결과를 저장하기 위해서 많은 메모리를 소비한다는 단점이 있다.

두 번 째 방법은 리셉터의 분자 수 만큼 쓰레드를 생성하고, 리간드의 분자 수 만큼 루프를 돌아서 계산하는 방법이다. 본 논문에서는 두 번째 방법을 사용하였으며 [표 1]은 이에 대한 슈도 코드이다.

표 1. CUDA 계산 슈도 코드

```
void Simulation(Molecule Recepter, Molecule* Ligand)
{
    __shared Molecule shLigand[];
    int tile = nLigand / TILE;
    // 현재 쓰레드 블록의 쓰레드 개수
    int threadNum = blockDim.x;
    for(int i=0; i<tile; i++) {
        shLignad[threadIdx] =
                Ligand[threadIdx+tile*threadNum];
        __syncthreads();
        Calculate(Recepter, shLigand);
    }
}
void DockingSimulation(...)
{
    Simulation<<<nReceptor, ...>>>(...);
}
```

먼저 리셉터의 크기만큼 쓰레드를 생성한다(DockingSimulation함수). 그 후에, 빠른 계산을 위해서 리간드의 데이터를 공유 메모리로 복사한다. 공유 메모리로 복사된 데이터는 여러 쓰레드에서 참조하더라도 오버헤드가 생기지 않는다. 이 때, 한 번에 모든 리간드의 정보를 공유 메모리로 복사하는 것이 아니라 TILE 상수 만큼 분할해서 불러오게 된다. 이유는 일반적으로 공유 메모리는 그 크기가 적기 때문에 한 번에 저장할 수 있는 데이터의 양이 한정되어 있기 때문이다. 그 후에 복사된 데이터에 대해서 계산을 수행하게 된다. [그림 6]은 Tile 방법의 개념도이다. 세로 방향으로는 병렬 처리 하는 단위를 의미한다. 각각의 리셉터의 원자마다 쓰레드가 할당되어 병렬로 처리가 되며, 각 쓰레드에서 리간드의 원자 개수만큼 루프를 통해 순차적으로 처리가 되며, 공유 메모리의 한계로 임의의 크기만큼 나누어, 메모리에 복사 후 계산을 수행하게 된다.

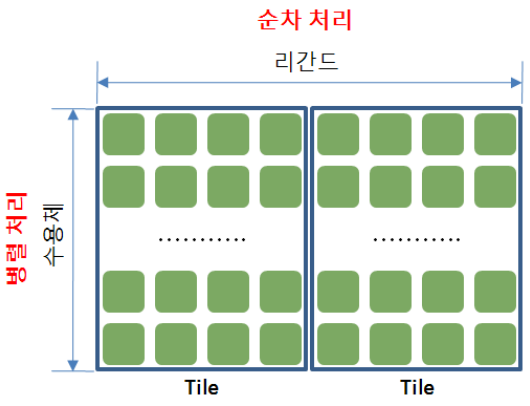



그림 6. Tile 기법의 개념도

위와 같은 방식으로 구현하게 되면 필요한 메모리의 양을 줄일 수 있으며, 오버헤드가 적기 때문에 좀 더 나은 성능을 보인다. 이렇게 구현하는 방식을 Tile기법이라고 하는데, N-Body 시뮬레이션에서 많이 사용하며, 주로 많은 양의 데이터간의 Join 연산을 수행할 때 많이 사용한다.

## Ⅳ. 실험 및 결과

본 논문에서 제안하는 시스템을 성능평가 하기 위해 실제 분자 시뮬레이션에서 사용되는 HIV 바이러스 물질에 대한 도킹 시뮬레이션의 실행 시간을 각각 CPU와 GPU 에 대해서 비교하였다. 실험을 위한 시스템 사양은 [표 2]와 같다.

표 2. 실험 시스템 사양

| 항목 | 내용 |
|---|---|
| CPU | Intel Core2 Quad 2.4GHz Q600 |
| RAM | 2GB |
| VGA | nVIDIA Geforce 8600GT X 2 |
| OS | Windows XP |

실험을 위해서 CPU 기반의 프로그램은 쓰레드를 2개를 생성하여 전하량 계산과 반데르발스 힘 계산을 수행하여 각각 하나의 CPU 코어에서 계산되도록 하였으며, 인텔 SSE2를 이용해서 수식에 CPU 가속이 가능하도록 했다.

실험에 사용한 분자의 크기(분자 개수)는 [표 3]과 같다.

표 3. 실험 분자의 크기

| 실험 | 리셉터(Receptor) | 리간드(Ligand) |
|---|---|---|
| 1 | 3112 | 92 |
| 2 | 6234 | 276 |
| 3 | 12836 | 460 |

### 1. 에너지 계산 실험

#### 1.1 전기적인 힘 계산

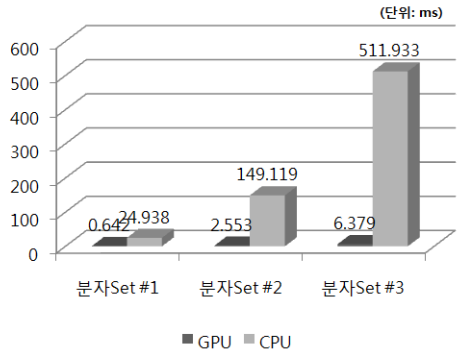



그림 7. 전기적인 힘 계산 시간 (단위 : ms)

전기적인 힘 계산 시간은 [그림 7]과 같다. 계산 량이 많아질수록 계산 시간도 큰 폭으로 증가했다. 1번 실험과 3번 실험의 계산 량은 약 20배 차이가 나는데, 실제 계산 시간도 1번 실험에 비해서 20배 정도 오래 걸렸다. 하지만 GPU는 내부적으로 수식을 병렬 처리하여 계산하기 때문에 계산 량이 증가하더라도, 계산시간이 큰 폭으로 변하지 않았다.

#### 1.2 반데르발스 힘 계산

[그림 8]은 반데르발스 힘을 계산한 결과이다. 반데르발스 힘 계산은 전하량 계산에 비해서 복잡하기 때문에 전하량 계산보다 많은 시간이 걸렸다. 하지만 전하량

계산과 마찬가지로, GPU는 CPU에 비해서 최대 80배까지 성능 향상 효과를 얻을 수 있었으며, 이는 CPU가 순차적인 계산하는 것에 기인한 결과로 분석된다.

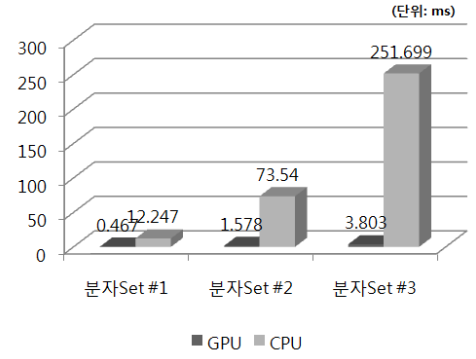



그림 8. 반데르발스 힘 계산 시간 (단위 : ms)

## 2. GPU와 CPU의 성능 비교 실험

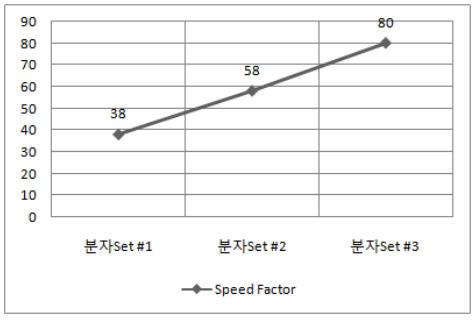



그림 9. CPU와 GPU 성능 비교(스피드 팩터)

이번 실험에서는 전하량 계산 과 반데르발스 계산을 동시에 수행한 결과이다. [그림 9]는 CPU와 GPU의 성능이 몇 배나 차이가 나는지 나타내는 스피드 팩터 그래프이다. 계산 량이 많아질수록 계산 시간에 많은 차이를 보였다. 실험3의 경우 약 80배 이상의 성능을 보였으며, 계산 량이 보다 많아질수록 이보다 더 많은 계산 시간 차를 보였다.

# V. 결론

본 논문에서는 CUDA를 이용하여 많은 계산이 필요한 도킹 시뮬레이션 시스템을 구현하여, CPU보다 더 높은 성능을 낼 수 있는 시스템을 개발하였다. 또한 2개의 GPU를 사용하여 동시에 각각 다른 수식을 처리하여 논리적인 병렬 시스템을 구현하여, 계산 속도를 높였다. 실험 결과, 분자의 크기에 따라서 최소 40배 이상의 성능을 향상시켰다.

향후 연구에서는 GPU개수보다 많은 수식을 계산할 때, 최적의 순서로 수식을 처리 할 수 있는 스케쥴링 알고리즘을 개발할 것이다. 이렇게 함으로써 계산 성능을 높이기 위해서 추가의 고가 CPU를 구입하는 것보다 비교적 저렴한 GPU를 구입할 수 있어, 높은 성능 향상 및 비용 절감을 하는데 유용하게 사용될 수 있다.

## 참 고 문 헌

[1] B. R. Brooks, R. E. Bruccoleri, B. F. Olafson, D. Vid J. States, S. Swaminathan, and M. Karplus, "CHARM M: A Program for macromolecular energy, minization, and dynamics calculation," J.Comp.Chem, Vol.4, pp.187-217, 1983.

[2] Villanueva-García, A. Martínez-Richa, and R. Juvencio, "Assignment of vibrational spectra of labdatriene derivatives and ambers: A combined experimenta l and density functional theoretical study Manuel," ARIKIVOC(EJ-1567C), pp.449-458, 2005.

[3] L. V. Kale, M. Bhandarkar, R. Brunner, N. Krawetz, J. Phillips, and A. Shinozaki, "NAMD: A Case Study in Multilingual Parallel Programming," the 10th International Workshop on Languages and Compilers for Parallel Computing, pp.367-38, 1997.

[4] L. Aaron, K. Joe, O. John, "Implementing Efficient Parallel Data Structures on GPUs," GPU Gems2, Addison Wesley, pp.521-545.

[5] http://www.nvidia.com/object/cuda_home.htm

[6] Ephriam Katchalski-Kater, Isaac Shariv, Miriam Eisenstein, Asher A.Friesem, Claude Aflalo, Ilya A. Vasker, "Molecular surface recognition: Determination of geometric fit between proteins and their ligands by correlation techniques," Proc. Natl. Acad. Sci. USA Vol.89, pp.2195-2199, 1992(3).

[7] D. S. Kim, C. H. Cho, Y. S. Cho, C. I. Won, and D. U. Kim, "Pocket Recognition on a Protein Using Euclidean Voronoi Diagram of Atoms," LNCS 3480, pp.700-715, 2005.

[8] S. J. Park, B. S. Kim, and J. I. Kim, "A Web Service-based Molecular Modeling System using Distributed Processing System," LNCS, Human.Society@international, 2005.

[9] http://www.gpgpu.org

[10] J. N. England, "A system for interactive modeling of physical curved surface objects," In Proceedings of SIGGRAPH 78 1978, pp.336-340, 1978.

[11] M. Potmesil and E. M. Hoffert, "The Pixel Machine: A Parallel Image Computer," In Proceedings of SIGGRAPH 89 1989, ACM, pp.69-78, 1989.

[12] J. Rhoades, G. Turk, A. Bell, A. State, U. Neumann, and Varshney, "A. Real-Time Procedural Textures," In Proceedings of Symposium on Interactive 3D Graphics 1992, ACM / ACM Press, pp.95-100, 1992.

[13] M. Olano and A. Lastra, "A Shading Language on Graphics Hardware: The PixelFlow Shading System," ACM, pp.159-168, 1998.

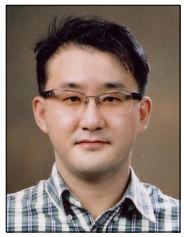
[14] Low Viscosity Flow Simulations for Animation. Jeroen Molemaker, Jonathan M. Cohen, Sanjit Patel, Jun-yong Noh. Symposium on Computer Animation 2008.

[15] S. U. Ivan and J. M. Todd, "Quantum Chemistry on Graphical Processing Units. 1. Strategies for Two-Electron Integral Evaluation," Journal of Chemical Theory and Computation, Vol.4, pp.222-231, 2008.

[16] M. L. Oscar and O. Kazuhiro, Real-time Visual Tracker by Stream Processing, Journal of Signal Processing System, 2008(7).

## 저 자 소 개

**박 성 준(Sung-Jun Park)** 정회원



- 1997년 2월 : 호서대학교 컴퓨터공학과(공학사)
- 1999년 2월 : 건국대학교 컴퓨터공학과(공학석사)
- 2005년 2월 : 건국대학교 컴퓨터공학과(공학박사)
- 2006년 3월 ~ 현재 : 호서대학교 게임공학과 조교수

<관심분야> : 게임, 가상현실, HCI, Bioinfomatics