Journal of Korea Multimedia Society Vol. 15, No. 6, June 2012(pp. 794-804)

단백질 분자에 대한 proximity 연산을 위한 복셀 맵과 스피어 트리 구조 비교

김병주[†], 이정은[††], 김영준[†††], 김구진[††††]

요 약

단백질 분자에 대해 공간 상의 한 점으로부터의 최소 거리를 계산하거나, 임의의 점에 대한 충돌을 감지하는 등의 proximity query는 분자에 대한 기하학적 연산을 수행하기 위해 매우 중요한 기본 연산이다. Proximity query의 계산 시간 효율성은 분자가 어떤 자료구조로 표현되는가에 따라 크게 달라질 수 있다. 본 논문에서는 GPU 가속을 이용하여 효율적으로 proximity 연산을 수행하기 위한 기법을 제안하고자 한다. 분자에 대응하는 구의 집합에 대해 복셀 맵 (voxel map)과 스피어 트리 (sphere tree) 를 사용한 자료구조를 제안하며 각 자료구조에 대응되는 알고리즘을 제시한다. 또한, 1,000개~15,000개의 원자를 포함하는 분자에 대한 실험을 통해 두 자료구조의 성능이 기존 자료구조에 비해 최소 3배에서 최대 633배 향상되었음을 보인다.

Comparison of Voxel Map and Sphere Tree Structures for Proximity Computation of Protein Molecules

Byungjoo Kim[†], Jung Eun Lee[††], Young J. Kim[†††], Ku-Jin Kim[††††]

ABSTRACT

For the geometric computations on the protein molecules, the proximity queries, such as computing the minimum distance from an arbitrary point to the molecule or detecting the collision between a point and the molecule, are essential. For the proximity queries, the efficiency of the computation time can be different according to the data structure used for the molecule. In this paper, we present the data structures and algorithms for applying proximity queries to a molecule with GPU acceleration. We present two data structures, a voxel map and a sphere tree, where the molecule is represented as a set of spheres, and corresponding algorithms. Moreover, we show that the performance of presented data structures are improved from 3 to 633 times compared to the previous data structure for the molecules containing 1,000~15,000 atoms.

Key words: Protein Molecule(단백질 분자), Proximity Query, Sphere Tree(스피어 트리), Voxel Map(복셀 맵), Molecular Docking(분자 도킹), Molecular Interface(분자 인터페이스)

※ 교신저자(Corresponding Author) : 김구진, 주소 : 대구시 북구 산격동 1370 경북대학교 컴퓨터학부(702-701), 전화 : 053)950-7573, FAX : 053)957-4846, E-mail : kujinkim@yahoo.com
접수일 : 2012년 1월 11일, 수정일 : 2012년 3월 14일
완료일 : 2012년 4월 19일
[†] 정회원, 경북대학교 전자공학과 대학원
(E-mail : kbj113@hotmail.com)
[††] 준회원, 경북대학교 전자전기컴퓨터학부 대학원
(E-mail : highshia@nate.com)
[†††] 준회원, 이화여자대학교 컴퓨터공학전공
(E-mail : kimy@ewha.ac.kr)
[††††] 정회원, 경북대학교 컴퓨터학부
※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업(No. 2012-0001755)이며 2012학년도 경북대학교 학술 연구비에 의해 연구되었음.

1. 서 론

기존의 약제에 비해 효력이 더욱 증진되고 부작용이 감소된 신약의 개발은 국내외를 불문하고 항시 요구되는 중요한 연구 분야이다. 신약 개발은 고부가 가치를 창출할 수 있는 연구 분야임에도 불구하고, 다양한 분야의 첨단 기술이 필요한 관계로 연구개발이 어렵다는 문제점이 있다. 신약 개발을 위해 매우 중요한 기술 중의 한 가지가 단백질 분자 간의 상호작용을 모델링하는 연구 분야이며, 이때 필수적으로 필요한 연구가 분자의 도킹(docking) 및 바인딩(binding)이다 [1-8]. 단백질 분자 간의 도킹과 바인딩 계산 등에 있어서 분자에 대한 최소 거리를 계산하거나 충돌을 감지 하는 등의 proximity query는 매우 중요한 기본 연산이다.

도킹이나 바인딩 등을 분자에 대한 기하학적인 측면에서 계산할 경우, 단백질 분자를 구성하는 각 원자는 일반적으로 구의 형태로 표현된다. 분자에 포함된 n 개의 원자가 각각 중심점 $\mathbf{c}_i$ $(0 \le i < n)$와 반데르바스 반경 (van der Waals) r_i를 가질 때, 각 원자는 다음과 같이 한 개의 구로 둘러싸인 영역 $B(\mathbf{c}_i, r_i)$에 대응된다.

$$B(\mathbf{c}_i, r_i) = \{\mathbf{p} \in R^3 \mid \|\mathbf{p} - \mathbf{c}_i\| \le r_i\}$$

한 개의 분자는 다음과 같은 집합 M으로 정의된다.

$$M = \{B(\mathbf{c}_i, r_i) \mid 0 \le i < n\}.$$

본 논문에서는 단백질 분자가 집합 M 으로 주어질 때, 3차원 공간의 점의 집합 P에 대해 다음에 제시된 세 가지 종류의 proximity query를 효율적으로 수행하는 알고리즘을 제안한다.

1) 충돌 감지: 각각의 점 $\mathbf{p} \in P$가 원자 내부에 속하는 지를 결정한다.

2) 최소 거리 계산: 각각의 점 $\mathbf{p} \in P$로 부터 단백질 분자까지의 최소 거리를 계산한다.

3) 반경 d 이내의 원자 발견: 각각의 점 $\mathbf{p} \in P$로부터 반경 d인 구 S를 생성할 경우, S와 교차하거나 S에 포함되는 모든 원자를 발견한다.

Proximity query를 수행하거나 flexible molecule을 구현하기 위하여 공간을 균일한 크기의 격자로 나누어 원자 정보를 저장하거나, 원자들을 계층적으로 그룹화 하여 트리(tree) 형태로 나타내는 방법이 현재까지 일반적으로 사용되어 왔으나, 대부분은 자료구조에 대해 명확하게 제시하지 않았다.

최근에 들어서는 변형되는 분자에 대해 빠르게 기하학적 계산을 수행하기 위해 효율적인 자료구조의 연구가 필요해졌다[9-11]. Bajaj 등[9]은 Dynamic Packing Grid (DPG)라는 자료구조를 제시하였다. DPG는 분자 내에 원자가 위치를 변경하면서 flexible하게 변형되는 것을 반영하고, 원자의 삽입, 삭제, 이동이 수행되는 가운데 분자 곡면을 빠르게 계산하기 위하여 제안되었다. DPG는 격자와 트리 형태의 조합으로 만들어진 자료구조이다. DPG는 원자의 중심점을 포함하고 있는 격자들만으로 구성된 자료구조를 생성하고, 생성된 각 격자들을 x, y, z축 각각의 방향에 대해서 *trie*형태의 트리로 구성한다. Zhao 등[10]은 FT (flexibility tree) 구조를 사용하여 단백질 분자를 표현하였다. FT 구조는 분자구조의 고유한 성질에 따라 단백질 분자의 flexibility에 대한 자유도를 개별적으로 표현한다. 자유도를 단계별로 다르게 표현하기 위해 단백질 분자를 계층적인 트리로 나타내었다. 각각의 노드는 자신의 모션 유형(motion type)을 정의하고 있고, 복잡한 모션은 여러 모션 유형의 합성을 통해 표현한다. Kim 등[11]은 분자 곡면(molecular surface)을 빠르게 계산하기 위해 GPU를 기반으로 병렬처리를 수행할 수 있는 분자의 자료구조를 제시하였다. 이들은 분자들을 격자 형태의 복셀 맵에 저장하여 분자에 대한 오프셋 곡면 (offset surface) 상의 표본 점들을 효율적으로 계산하였고, 표본 점들에 대해 kd-tree를 구성하여 표본 점들로부터 일정 거리만큼 떨어진 iso-surface를 분자 곡면으로 근사하였다.

참고문헌 [9,10]에서 제시한 자료구조는 complete tree 형태가 아니고 tree의 height가 상당히 크므로, 트리를 표현하기 위해 포인터를 사용할 수 밖에 없다. 이들 자료구조는 단일코어 CPU에서 수행되기 위해 최적화되어 있으며, GPU를 기반으로 병렬처리를 수행하기에는 부적합하다. GPU는 특성 상 GPU memory 및 수행 코드에서 포인터의 사용을 허용하지 않는 제약이 있기 때문이다. GPU는 병렬처리에 유리한 구조를 가지고 있으나, 트리와 같은 수직적 구조에는 적용하기가 쉽지 않다[12]. 본 논문에서는 GPU 기반으로 병렬 처리를 수행하기에 적합한 자료구조로 복셀 맵과 스피어 트리를 제시한다. 제시된

두 종류의 자료구조는 배열(array)로 표현이 가능하다는 특징에 의하여 GPU 코드로 수행될 수 있다는 장점을 갖는다.

각 자료구조는 특징에 따라 특정한 연산에 대해 더 효율적인 성능을 보일 수 있고, 다른 종류의 연산에 대해서는 비효율적인 성능을 보일 수 있다. 어떤 자료구조를 사용하여 특정한 응용 분야에서 기하학적인 연산을 수행할 경우, 전체 시스템의 성능을 예측하기 위해서는 각 자료구조 별로 proximity query에 대해 어떤 성능을 보이는 지에 대한 기반 연구가 필요하다.

본 논문에서는 격자 형태의 자료구조 및 트리 형태의 자료구조를 대표하여, 복셀 맵 및 스피어 트리 구조를 기반으로 하여 분자를 표현한 뒤 proximity query를 수행한 결과를 비교한다. 복셀 맵은 3차원 배열로 구현되며, 배열의 구조적인 특성 때문에 병렬처리에 효과적이다. 일반적으로 트리는 병렬처리에 부적합하다고 생각할 수 있으나, 제시된 스피어 트리는 complete tree에 가까운 형태를 가지므로 complete tree에 대한 1차원 배열 표현을 사용할 수 있다. 따라서, 스피어 트리의 각 노드는 random access가 가능하며, 배열에 대한 병렬 처리를 활용할 수 있다. 각 자료구조는 GPU를 활용하여 병렬로 구성되고, 각 자료구조에서 제안되는 알고리즘들은 GPU를 활용하여 각각의 점들에 대한 연산을 병렬로 수행하여 계산시간의 속도를 향상시킨다.

본 논문의 구성은 다음과 같다. 2절에서는 분자를 표현하는 구의 집합에 대해 복셀 맵을 구성하는 GPU 알고리즘과 복셀 맵에 대한 proximity query 알고리즘을 제시한다. 3절에서는 구의 집합에 대해 스피어 트리를 구성하는 알고리즘과 스피어 트리에 대한 proximity query 알고리즘을 제시한다. 4절에서는 실험을 통해 복셀 맵과 스피어 트리 구조에 대한 proximity query 알고리즘 수행 성능을 비교한다. 5절에서는 결론을 내린다.

2. 복셀 맵 구조의 분자 표현

분자를 표현하는 집합 M과 이에 대한 바운딩 박스 (bounding box) B가 주어질 때, x, y, z축 방향의 크기가 각각 S_x, S_y, S_z인 B를 $n_x \times n_y \times n_z$ 개의 복셀로 분할하여 복셀의 집합 V를 구성한다. 이때, 각 복셀 $V^{\alpha\beta\gamma}$ ($0 \le \alpha < n_x$, $0 \le \beta < n_y$, $0 \le \gamma < n_z$)는 다음과 같은 성질을 갖는다.

$$V^{\alpha\beta\gamma} = \{ (x, y, z) \mid \alpha (S_x/n_x) \le x \le (\alpha + 1)(S_x/n_x), \beta (S_y/n_y) \le y \le (\beta + 1)(S_y/n_y), \gamma(S_z/n_z) \le z \le (\gamma + 1)(S_z/n_z) \}.$$

$V^{\alpha\beta\gamma}.atoms$는 다음과 같이 복셀과 교차하는 원자의 정보를 가진다.

$$V^{\alpha\beta\gamma}.atoms = \{ B(\mathbf{c}_i, r_i) \mid B(\mathbf{c}_i, r_i) \cap V^{\alpha\beta\gamma} \neq \varnothing \}.$$

또한, $V^{\alpha\beta\gamma}.nn(l)$는 $V^{\alpha\beta\gamma}$ 복셀을 포함하여 l개의 복셀에 해당하는 거리 이내에 이웃하고 있는 복셀들을 나타낸다. l 이 0보다 크거나 같은 값으로 주어질 때, $V^{\alpha\beta\gamma}.nn(l)$는 다음과 같다.

$$V^{\alpha\beta\gamma}.nn(l) = \{ V^{ijk} \mid \alpha - l \le i \le \alpha + l, \beta - l \le j \le \beta + l, \gamma - l \le k \le \gamma + l \}.$$

l 이 0보다 작은 값으로 주어질 때, $V^{\alpha\beta\gamma}.nn(l)$ 는 공집합이다.

그림 1에서는 복셀 맵 기반으로 분자를 2차원으로 표현한 예를 보인다.

그림 1에서 각 복셀은 자신과 교차하고 있는 원자들에 대한 정보를 저장한다. 단백질분자에 속하는 원자들은 고유의 일련번호 순서대로 배열로 나타내고, 그 각각은 중심의 위치, 크기 그리고 포함되는 펩타이드 체인과 아미노산 정보를 가진다. 그림 1을 3차원으로 확장하면, 분자를 표현하는 집합 M에 대한 복셀 맵 자료구조를 구성할 수 있다. n_x ◈ n_y ◈ n_z

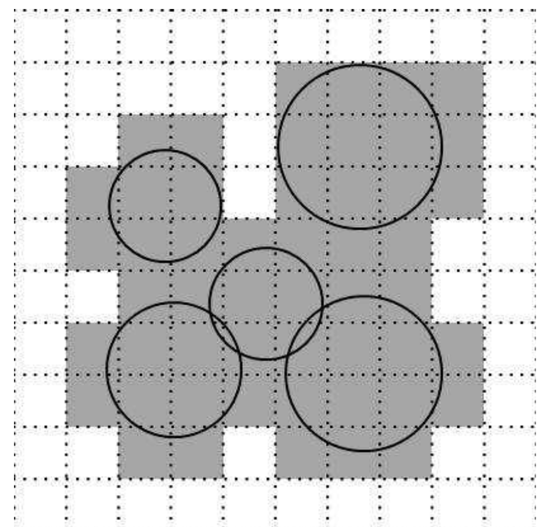

그림 1. 복셀 맵 기반 분자 표현

개의 복셀로 구성된 복셀 맵에서 각 복셀은 자신이 포함하고 있는 원자에 대한 리스트를 저장하고 있다.

복셀 맵을 구성하는 알고리즘은 CUDA 언어를 이용하여 각 원자 별로 thread를 할당하여 병렬로 처리된다. 알고리즘은 Algorithm1과 같다.

```
Algorithm1: 복셀 맵 구성
Function VoxelMapConstruction (M)
/* M: a set hard spheres */
Begin
    For each B(c, r) ∈ M do in parallel begin /*
            GPU code */
            VS := the set of voxels that intersect
            with B(c, r);
            For each voxel V ∈ VS do
                    Add B(c, r) to V.atoms;
    end
End
```

2.1 충돌 감지 알고리즘

Query를 위한 점 p가 주어지면, 이 점이 속한 복셀 V_p 를 발견한다. V_p 와 교차하는 원자들의 정보를 V_p *.atoms* 에서 얻은 뒤, 각 원자에 대해 점 p와 충돌하는지 판단한다.

```
Algorithm2: Collision Detection
Function CollisionDetection (P) /* P: a set of query
        points */
Begin
    For each p ∈ P do in parallel begin /* GPU
            code */
            Vp := the voxel containing p;
            For each B(c, r) ∈ Vp .atoms do
                    if ‖ p - c ‖ - r ≤ 0 then
                            return true;
    end
    return false;
End.
```

2.2 최소 거리 계산

각 점 p로부터 분자까지의 최소 거리를 계산한다. 분자가 $M = \{B(c_i, r_i) \mid 0 \le i < n\}$으로 주어질 때, 점 p 에서 분자까지의 거리는 다음의 식을 사용하여 계산한다.

$$dist = \mathrm{Min}_{0 \le i < n} (\| p - c_i \| - r_i).$$

복셀 맵 구조에서 효율적으로 최소 거리를 계산하기 위해서는 점 p가 속한 복셀 V_p 및 V_p 와 이웃하고 있는 복셀들에 대해 차츰 범위를 넓혀가며 탐색한다. 복셀에 포함된 원자마다 p까지의 거리를 계산하는데, 만약 어떤 원자에 대한 거리 *d*가 이전에 구한 *min_dist*보다 작으면 *d*를 새로운 *min_dist*로 결정한다. 수정된 *min_dist* 이내에 p와 더 가까운 다른 원자가 있는 지 확인하는 추가 작업이 필요하므로, *max_step* 값을 이용하여 거리를 계산할 복셀의 범위를 제한한다. 최소 거리를 계산하기 위한 알고리즘은 Algorithm3에 제시된다.

```
Algorithm3: 복셀 맵에 대한 최소 거리 계산
Function MinDist(P) /* P: a set of query point */
Begin
  For each p ∈ P do in parallel begin/* GPU code */
      max_step := ∞;
      min_dist := ∞;
      update := false;
      For step := 0 to max_step do begin
            Vp := the voxel containing p;
            For each v∈Vp.nn(step) - Vp.nn(step-
                    1) do
                For each B(c, r) ∈ v.atoms do
                    if ‖ p-c ‖ -r ≤ min_dist then
                      begin
                        min_dist :=‖ p - c ‖ - r;
                        update := true;
                      end
      if update == true then
          max_step:=Ceil(min_dist/min(Sx/nx,
          Sy/ny, Sz/nz));
  end
  return min_dist
End.
```

2.3 일정 반경 내의 원자 발견

점 p가 속한 복셀 V_p 로부터 주어진 반경 R 이내 전체 또는 일부가 포함된 복셀들을 먼저 발견한다. 이러한 복셀들에 포함된 원자정보를 이용하여, 점 p로부터 거리 R 이내에 있는 원자들을 추출한다. 일정 반경 내의 원자를 발견하기 위한 알고리즘은 Algorithm4에 제시된다.

```
Algorithm4: 일정 거리 내의 원자 발견
Function R-RadiusAtoms (R, P)
```

```
/* R: user-specified radius */
/* P: a set of query point */
Begin
  For each p ∈ P do in parallel begin /* GPU code */
      AL = ∅;
      V_p := the voxel containing p;
      step := Ceil( R / min(S_x/n_x, S_y/n_y, S_z/n_z) );
      For each v ∈ V_p.nn(step) do
              For each B(c, r) ∈ v.atoms do
                      if ‖ p - c ‖ - r ≤ R then
                      add B(c, r) to AL;
  end
  return AL;
End.
```

3. 스피어 트리 구조의 분자 표현

분자에 대한 스피어 트리 표현은 분자가 가진 계층적인 성질을 반영한다. 단백질 분자는 펩타이드 체인들로 구성된다. 각 펩타이드 체인은 아미노산이 연속적으로 연결된 형태이다. 각각의 아미노산은 메인 체인과 사이드체인 부분으로 구분되는데, 모든 아미노산의 메인체인은 질소-알파카본-카본-산소의 순서로 원자들이 연결된 형태이고, 사이드 체인은 메인 체인의 알파카본에 연결된다.

분자를 구성하는 원자들을 구의 집합으로 표현할 경우, 원자 간에 공유결합이 일어나는 경우에만 해당 구들이 서로 교차한다는 성질을 갖는다. 한 개의 아미노산을 구성하는 원자들의 경우, 메인 체인에 속한 원자들 중 서로 이웃한 것들 간에는 공유결합이 존재한다. 또한, 사이드 체인과 메인 체인이 연결되는 부분에서도 원자 간의 공유결합이 존재한다. 두 개의 서로 다른 아미노산들이 메인 체인에서 연결될 때, 연결되는 두 개의 원자 간에도 공유결합이 존재한다.

Algorithm5에서는 단백질 분자를 스피어 트리로 표현하는 알고리즘을 제시한다. 분자에 포함된 펩타이드체인 각각은 spatial locality를 갖지만, 서로 다른 펩타이드 체인 간에는 spatial locality가 보장되지 않는다. 따라서, 각각의 펩타이드 체인마다 한 개의 스피어 트리가 구성된다. 스피어 트리는 top-down 방식으로 한 개의 펩타이드 체인 전체에 대한 정보가 루트에 저장된다. 루트 노드에 포함된 아미노산들의 sequence 를 중심부에서 이분하여 각각 2개의 child node에 저장하며, 이 과정이 재귀적으로 수행된다. 노드가 한 개의 아미노산을 가지면 재귀적 수행이 끝나며, 해당 노드는 리프노드 (leaf node)가 된다.

Algorithm5에 따라 m개의 아미노산으로 구성된 분자에 대해 스피어 트리를 구하면 $\log_2(m+1)$ 이내의 height를 갖게 된다. 이와 같이 complete한 트리 형태의 스피어 트리로 구성하여 linked list형태가 아닌 배열 형태로 자료구조를 구현할 수 있으며, 각 노드를 인덱스 값으로 탐색할 수 있어서 병렬처리에 효과적이다. 또한 포인터를 사용할 수 없는 현재의 CUDA 아키텍쳐에서, GPU로 생성된 트리 형태의 자료구조에 대한 탐색을 가능하게 한다.

스피어 트리를 구성하는 각 노드 *sNode*는 원자들의 집합 및 원자들에 대한 바운딩 스피어 (bounding sphere) 정보를 포함하며, 이들은 각각 *sNode.atoms*와 *sNode.BS*에 저장된다. 중심점이 **c**이고 반경이 r인 구를 S(**c**, r) 이라고 표시할 때, *sNode*에 포함된 원자들 $\{B(\mathbf{c}_i, r_i) \mid n_1 \le i < n_2\}$에 대해 *sNode.atoms*와 *sNode.BS*는 다음의 식으로 표현된다.

$sNode.atoms = \{ B(\mathbf{c}_i, r_i) \mid n_1 \le i < n_2 \}$, where $1 \le n_1 < n_2 \le n$, $sNode.BS = \{ S(\mathbf{c}, r) \mid \mathbf{c} = \text{average}(\mathbf{c}_i),\ r = \max(\|\mathbf{c} - \mathbf{c}_i\| + r_i) \}$.

스피어 트리를 구성하는 과정은 GPU를 활용하여 병렬로 처리된다. GPU에서 단백질정보와 스피어 트리의 정보는 저장공간의 효율성을 생각해서 분리하여 저장한다. 먼저, 단백질분자에 속하는 원자들은 고유의 일련번호 순서대로 배열로 나타내고, 그 각각은 중심의 위치, 크기 그리고 포함되는 펩타이드 체인과 아미노산 정보를 가진다. 반면에 스피어 트리의 노드는 바운딩 스피어의 정보와 노드가 포함하는 원자의 처음과 마지막 일련번호만 저장한다.

그림 2에서는 2BIW.pdb파일의 126번째 아미노산부터 134번째 아미노산까지의 스피어 트리 생성 과정을 단계별로 제시한다.

```
Algorithm5: 스피어 트리 기반의 분자 표현
Function SphereTree(M, T)
/* M: a molecule consists of hard spheres */
/* T: a set of sphere trees for the molecule */
Begin
For each peptide chain P_i in M do begin
  Construct an empty tree T_i;
  Add a root node R_i to T_i, where R_i.atoms ={every
```

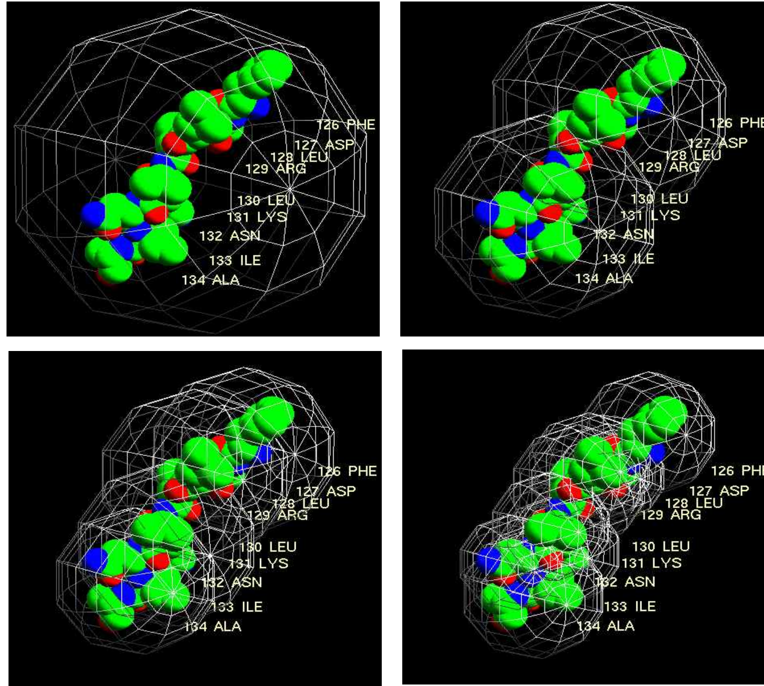

그림 2. 단백질 분자 (pdb id: 2BIW)의 일부에 대한 스피어 트리의 표현

atoms in P_i};
nodelist.push(R_i)
For each node N_i in *nodelist* **do in parallel**/* GPU code */
m_1 := a_a.*amino*, where a_a is the first atom of N_i
m_2 := a_β.*amino*, where a_β is the last atom of N_i
m_3 := a_t.*amino*, where $t = (a + \beta)/2$
if $m_1 < m_2$ **then**
add new children N_L, N_R to the current node N_i where N_L.atoms and N_R.atoms contain two partitioned subsets of N_i.atoms using m_3 as a fiducial amino acid;
nodelist.push(N_L);
nodelist.push(N_R);
For each node N in T_i **do in parallel** /* GPU code */
Compute bounding sphere *N.BS* for the atoms contained in *N.atoms*;
return $T = \{T_i\}$
End.

3.1 충돌 감지

충돌 감지 알고리즘은 Algorithm6과 같다. 임의의 원자 $a = B(c_a, r_a)$와 query point p 간의 거리 dist(a, p)는 다음과 같이 계산된다.

$$\text{dist}(a, \mathbf{p}) = \| c_a - \mathbf{p} \| - r_a$$

스피어트리에 속한 한 개의 노드를 *sNode*라 하고 그 바운딩 스피어를 *sNode.BS* = $S(c_B, r_B)$라 할 때, dist(*sNode.BS*, p)는 다음과 같이 정의된다.

$$\text{dist}(sNode.BS, \mathbf{p}) = \| c_B - \mathbf{p} \| - r_B$$

Algorithm6: 충돌감지
Function CollisionDetection(M, T, p)
/* M: a molecule consists of hard spheres */
/* T: a sphere tree for M */
/* p: a query point */
Begin
For each p $\in P$ **do in parallel begin**
Stack S;
S.push (T.*root*);
while $S \neq \varnothing$ **do begin**
sNode := S.pop( );
if dist(*sNode.BS*, p) ≤ 0 **then begin**
if *sNode* is a leaf node **then**
for each $a \in$ *sNode.atoms* **with** dist(a,p) ≤ 0 **do**
return *true*;
else
for each child of *sNode* that is not null **do**
S.push(*sNode.child*);
end

```
        end
        return false;
    end
End.
```

3.2 최소 거리 계산

점 p에서 스피어 트리가 구성되어 있는 분자까지의 최소 거리를 계산한다. 효율적인 계산을 위해 스피어 트리의 가지 치기 조건을 두어 계산 횟수를 줄인다. 이전에 계산된 최소거리가 *d*일 때, 하나의 노드 *sNode*가 dist (*sNode.BS*, p) > *d* 라면 *sNode*에 속한 모든 원자 *a*에서 dist (*a*, p) > *d* 이므로 *sNode*의 자손(descendant)들에 대해서는 검색할 필요가 없다. 최소 거리 계산의 알고리즘은 Algorithm7과 같다.

lgorithm7: 스피어 트리에 대한 최소 거리 계산

```
Function MinimumDist(M, T, p)
/* M: a molecule consists of hard spheres */
/* T: a sphere tree for M */
/* p: a query point */
Begin
    For each p ∈ P do in parallel begin /* GPU
        code */
        Stack S;
        S.push( T.root);
        d := ∞;
        While S ≠ ∅ do begin
            sNode := S.pop( );
            if dist(sNode.BS, p) ≤ d then begin
                if sNode is a leaf node then
                    for each a∈sNode.atoms with
                        dist(a,p) < d do
                            d:=dist(a, p);
                else
                    for each sNode.child that is not
                        null do
                            S.push(sNode.child);
            end
        end
        return d;
    end
End.
```

3.3 일정 반경 내의 원자 발견

점 p에서 반경 *d* 내의 원자를 발견하는 계산을 효율적으로 하기 위해 최소거리 계산과 같은 방법으로 스피어 트리에 대해 가지 치기를 한다. 임의의 노드 *sNode*에서 dist (*sNode.BS*, p) > *d* 라면 *sNode*에 속한 모든 원자들은 반경 *d*내에 속할 수 없다. 이와 같은 방법으로 스피어 트리를 이용하여 일정 반경 내의 원자들을 발견하는 알고리즘은 Algorithm8에 제시한다.

Algorithm8:일정 반경 내의 원자 발견

```
Function R-radiusAtoms(M, T, p, d)
/* M: a molecule consists of hard spheres */
/* T: a sphere tree for M */
/* p: a query point */
/* d: the radius from p*/
Begin
    For each p ∈ P do in parallel begin
        Stack S;
        S.push( T.root);
        List AL;
        while S ≠ ∅ do begin
            sNode := S.pop();
            if dist(sNode, p) ≤ d then begin
                if sNode is a leaf node then
                    for each a∈sNode.atoms with
                        dist(a,p) ≤ d do
                            AL.add(a);
                else
                    for each sNode.child that is not
                        null do
                            S.push(sNode.child);
                end
        end
    return AL;
    end
End.
```

4. 실험 결과

실험은 Intel core(TM) i5 CPU (2.8GHz)와 4.0GB의 DRAM 및 nVidia GTX590 GPU가 장착된 컴퓨터에서 실행하였다. Window 환경에서 Visual Studio를 이용하여 실험 결과를 얻었고, OpenGL을 이용하여 시각화하였다. 표 1에서는 실험에서 입력으로 사용한 단백질 분자의 pdb id (http://www.pdb.org) 와 분자를 복셀 맵 및 스피어 트리로 구성하는 데 걸린 수행 시간, proximity 연산을 수행하는 데 걸린 시간을 제시한다. 기존 연구와의 비교를 위해 DPG 구조[9]를 구현한 소스코드를 웹사이트(http://cvcweb.ices.utexas.edu/cvcwp/)로부터 다운로드하여 실험

표 1. 질의점 2,000개에 대한 Proximity query 계산 시간 (단위: ms)

단백질 분자		복셀 맵 기반 알고리즘						스피어 트리 기반 알고리즘						Dynamic Packing Grid (DPG) 알고리즘					
PDB id.	원자 개수	복셀 맵 구성	충돌 감지	최소 거리 계산	반경 d A이내의 원자 발견			스피어 트리 구성	충돌 감지	최소 거리 계산	반경 d A이내의 원자 발견			DPG 구성	충돌 감지	최소 거리 계산	반경 d A이내의 원자 발견		
					d=5	d=10	d=15				d=5	d=10	d=15				d=5	d=10	d=15
2LYZ	1,001	0.06	0.12	6.62	4.46	12.89	14.99	0.87	0.35	1.29	1.42	3.03	4.82	8.06	21.75	60.60	20.85	50.38	52.98
1CA2	2,039	0.06	0.14	8.58	4.93	17.51	20.01	1.54	0.47	1.95	1.76	4.05	7.09	10.03	23.01	78.85	22.41	57.22	59.18
1EA1	3,509	0.07	0.13	12.74	4.95	19.22	22.17	1.56	0.45	2.40	1.79	4.20	7.85	16.06	23.32	101.38	22.52	60.44	61.72
1B44	4,190	0.09	0.15	12.14	5.55	20.73	24.19	1.84	0.47	2.82	2.01	4.68	8.80	17.30	25.47	97.81	23.99	66.96	68.45
1DDZ	7,485	0.12	0.16	15.29	6.52	26.67	30.84	2.30	0.66	3.83	2.51	5.98	11.63	24.15	26.29	116.96	24.05	74.07	75.89
1Q0D	11,183	0.17	0.14	21.89	5.56	23.18	26.69	2.89	0.52	4.12	1.94	4.57	8.85	37.04	25.81	138.88	22.52	69.91	71.41
2BIW	15,071	0.21	0.13	37.18	4.92	20.71	24.09	3.59	0.52	5.01	1.93	4.55	8.99	48.69	23.62	189.85	22.97	62.12	63.19

한 결과도 함께 제시한다. 각 자료구조를 구성하는 데 걸린 수행 시간은 100번 반복 수행한 평균값이다. 각 실험에서는 분자에 대한 바운딩 박스 내에서 무작위로 추출된 2,000개의 점이 질의점으로 사용되었다. 표 2에서는 질의점이 10,000개일 때 각 proximity query의 수행시간을 보인다. 그림 3에서는 표 2에 제시된 내용을 각각 그래프로 표시하였다.

사용한 pdb 파일은 1,001개의 원자를 포함하는 2LYZ부터 15,071개의 원자로 구성된 2BIW까지 총 7개의 단백질 분자를 대상으로 하였다. 2LYZ는 하나의 체인으로 구성되어 있고 4개의 helix와 각각 2개와 3개의 strand로 이루어진 2개의 sheet를 가지고 있고, 2BIW는 36개의 helix와 39개의 sheet로 이루어진 단백질로 4개의 펩타이드 체인을 가지고 있다. 가장 많은 체인을 가지는 단백질의 pdb id는 1Q0D로 12개의 체인을 포함하고 있다.

GPU 기반으로 단백질 분자의 자료구조를 생성한 복셀 맵과 스피어 트리는 CPU기반으로 우수한 성능을 보여주는 DPG 자료구조에 비해 최소 3배에서 최대 633배 가량의 개선된 성능을 보인다. 아울러 자료구조의 특성상 충돌 감지 연산에 대해서는 복셀 맵의 성능이 더 좋고, 최소 거리의 계산 및 일정 반경 내의 원자를 추출하는 연산에 대해서는 스피어 트리의 성능이 더 좋다는 점을 확인할 수 있다.

충돌 감지 연산의 성능을 분석하면, 복셀 맵 구조의 경우 주어진 임의의 점의 위치에 의해 점과 충돌 가능성이 있는 원자들을 점이 포함된 복셀 내에서 직접 발견할 수 있는 데 비해, 스피어 트리 구조의 경우에는 루트부터 시작하여 리프 노드 방향으로 주어진 점을 포함하는 바운딩 스피어를 갖는 노드를 발견하기 위해 트리를 탐색해야 하므로 계산시간이 더 소요된다. 최소 거리 계산의 경우에 복셀 맵은 주어진 점을 포함하는 복셀로부터 이웃한 복셀로 반경을 넓혀가며 최소 거리에 있는 원자를 발견하는데,

표 2. 질의점 10,000개에 대한 Proximity query 계산 시간 (단위: ms)

단백질 분자		복셀 맵 기반 알고리즘						스피어 트리 기반 알고리즘						Dynamic Packing Grid (DPG) 알고리즘					
PDB id.	원자 개수	복셀 맵 구성	충돌 감지	최소 거리 계산	반경 d A이내의 원자 발견			스피어 트리 구성	충돌 감지	최소 거리 계산	반경 d A이내의 원자 발견			DPG 구성	충돌 감지	최소 거리 계산	반경 d A이내의 원자 발견		
					d=5	d=10	d=15				d=5	d=10	d=15				d=5	d=10	d=15
2LYZ	1,001	0.06	0.20	9.48	8.41	24.34	28.28	0.87	0.50	1.93	2.03	4.49	7.38	8.06	111.25	310.43	109.87	246.80	255.59
1CA2	2,039	0.06	0.22	17.15	9.90	41.28	45.95	1.54	0.63	4.02	2.55	6.43	12.59	10.03	118.38	394.77	116.13	295.34	302.37
1EA1	3,509	0.07	0.29	17.67	13.16	52.68	58.50	1.56	0.75	4.57	2.93	7.34	15.06	16.06	133.85	577.35	128.89	311.76	317.32
1B44	4,190	0.09	0.29	23.21	14.09	61.05	67.74	1.84	0.89	6.48	3.47	8.62	17.54	17.30	132.91	509.53	125.36	344.62	354.93
1DDZ	7,485	0.12	0.27	27.42	12.28	54.57	61.37	2.30	0.83	7.92	3.17	8.06	17.29	24.15	134.05	604.44	125.09	373.46	386.90
1Q0D	11,183	0.17	0.18	59.61	7.44	33.24	37.87	2.89	0.68	9.90	2.45	5.82	11.58	37.04	133.67	712.44	119.77	357.51	366.55
2BIW	15,071	0.21	0.21	46.16	9.07	40.55	45.90	3.59	0.78	8.73	2.86	6.82	13.46	48.69	116.68	975.69	107.01	311.93	323.42

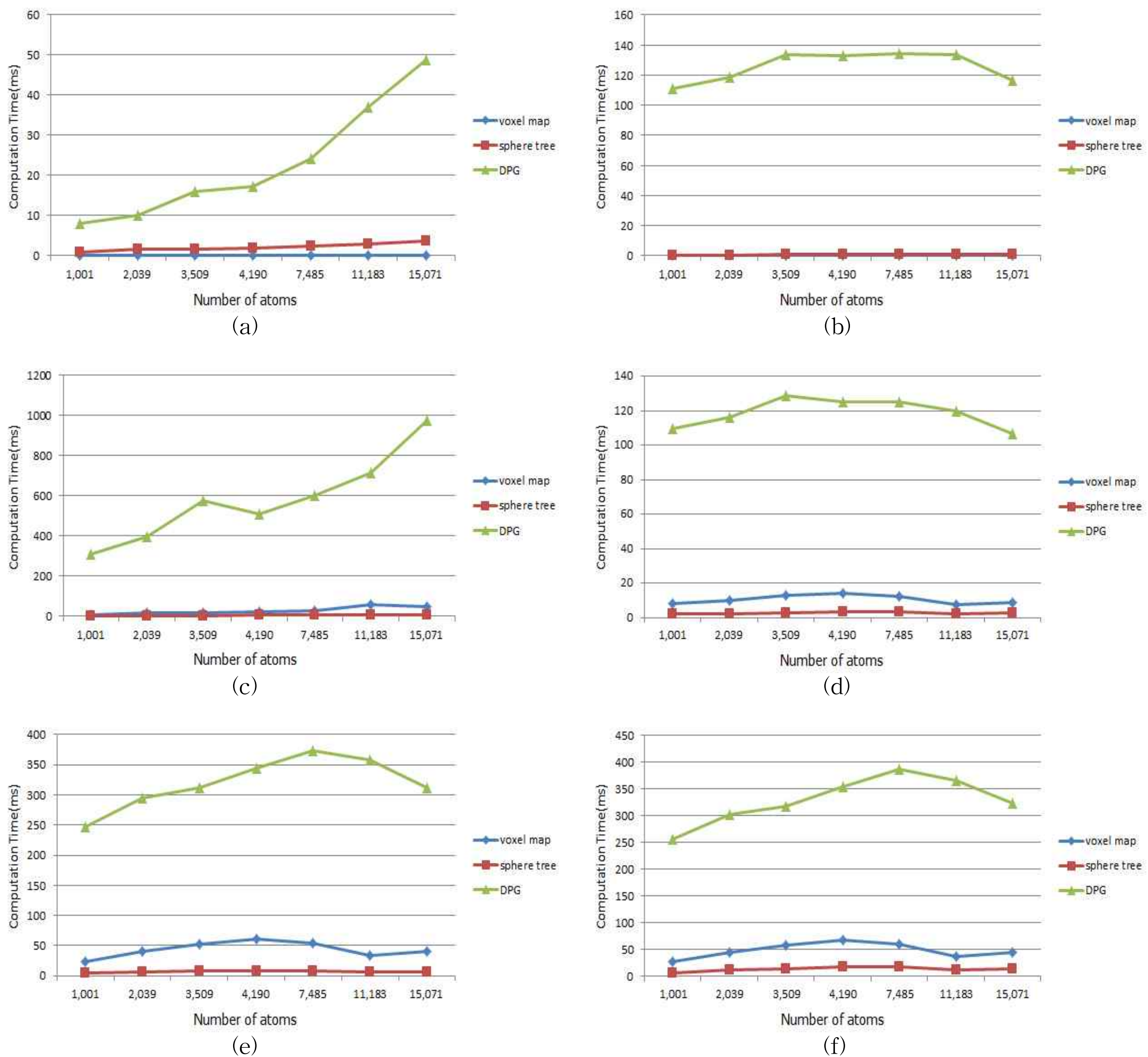

그림 3. 질의점 10,000개에 대한 계산 시간 (a) 자료구조 생성 시간, (b) 충돌 감지, (c) 최소 거리 계산, (d) 반경 5Å 이내의 원자 계산, (e) 반경 10Å 이내의 원자 계산, (f) 반경 15Å이내의 원자 계산

이때 반경을 넓힐 때마다 검사하는 복셀의 개수가 1개에서 (3^3-1)개, (5^3-3^3)개와 같이 급격하게 증가한다. 또한, 각 복셀마다 최대 8개 가량의 원자 정보가 포함되고, 같은 원자가 여러 복셀에 포함되어 있으므로, 전체적인 계산 시간 효율성이 떨어지는 것으로 분석된다. 일정 반경 이내의 원자를 찾는 연산 역시 검사할 복셀 및 원자의 개수가 많다는 이유로 스피어 트리 구조보다 계산 시간이 더 많이 걸리는 것으로 분석된다.

5. 결 론

본 논문에서는 단백질 분자에 대해 복셀 맵과 스피어 트리 기반의 자료구조를 구현하여 proximity query의 성능을 비교하였다. 분자를 구성하는 원자들이 각각 구로 표현될 때, 각 복셀과 교차하는 원자들의 집합을 구하여 복셀 맵을 구성한다. 스피어 트리의 자료구조는 원자들을 아미노산 단위로 취급하여 계층적으로 구성한다. 공간 상에 분포한 대량의 점들로부터 분자에 대한 충돌 감지, 최소 거리 계산, 일정 반경 이내의 원자 발견을 위하여 GPU를 활용할 경우 두 가지 자료구조 모두 실시간에 query를 수행할 수 있었다. 자료구조 생성 및 충돌 감지에는 복셀 맵 자료구조가 스피어 트리보다 좋은 성능을 보였다. 그러나, 최소 거리의 계산 및 일정 반경 내의 원자 추출에 대해서는 스피어 트리가 복셀 맵에 비해

더 좋은 성능을 보였다. 향후, 스피어 트리와 복셀 맵을 결합하여 수행되는 proximity query의 종류에 따라 더 효율적인 자료구조를 선택할 수 있는 방안을 연구하고, 제시된 알고리즘들을 좀 더 병렬화하여 성능을 개선할 계획이다.

참 고 문 헌

[1] T.J.A. Ewing, S. Makino, A.G. Skillman, and I.D. Kuntz, "DOCK4.0: Search Strategies for Automated Molecular Docking of Flexible Molecule Databases," *Journal of Computer-Aided Molecular Design*, Vol.15, No.5, pp. 411-428, 2001.

[2] S.K. Lai-Yuen and Y.S. Lee, "Interactive Computer-Aided Design for Molecular Docking and Assembly," *Computer-Aided Design and Applications*, Vol.3, No.6, pp. 701-709, 2006.

[3] D. Levine, M. Facello, P. Hallstrom, G. Reeder, B. Walenz, and F. Stevens, "Stalk: an Interactive System for Virtual Molecular Docking," *IEEE Computational Science and Engineering*, Vol.4, No.2, pp. 55-65, 1997.

[4] H. Nagata, H. Mizushima, and H. Tanaka, "Concept and Prototype of Protein-Ligand Docking Simulator with Force Feedback Technology," *Bioinformatics*, Vol.18, No.1, pp. 140-146, 2002.

[5] R.D. Taylor, P.J. Jewsbury, and J.W. Essex, "A Review of Protein-Small Molecule Docking Methods," *Journal of Computer-Aided Molecular Design*, Vol.16, No.3, pp. 151-166, 2002.

[6] O. Trott and A.J. Olson, "AutoDock Vina: Improving the Speed and Accuracy of Docking with a New Scoring Function, Efficient Optimization, and Multithreading," *Journal of Computational Chemistry*, Vol.31, No.2, pp. 455-461, 2010.

[7] C.M. Venkatachalam, X. Jiang, T. Oldfield, and M. Waldman, "LigandFit: a Novel Method for the Shape-Directed Rapid Docking of Ligands to Protein Active Sites," *Journal of Molecular Graphics and Modelling*, Vol.21, No.4, pp. 289-307, 2003.

[8] Y. Zhao and M.F. Sanner, "Protein-Ligand Docking with Multiple Flexible Side Chains," *Journal of Computer Aided Molecular Design*, Vol.22, No.9, pp. 673-679, 2008.

[9] C. Bajaj, R.A. Chowdhury, and M. Rasheed, "A Dynamic Data Structure for Flexible Molecular Maintenance and Informatics," *Bioinformatics*, Vol.27, No.1, pp. 55-62, 2011.

[10] Y. Zhao, D. Stoffler, and M. Sanner, "Hierarchical and Multi-Resolution Representation of Protein Flexibility," *Bioinformatics*, Vol.22, No.22, pp. 2768-2774, 2006.

[11] B. Kim, K.-J. Kim, and J.-K. Seong, "GPU Accelerated Molecular Surface Computing," *Applied Mathematics & Information Sciences*, Vol.6, No.1S, pp. 185-194, 2012.

[12] 유보선, 김현덕, 최원익, 권동섭, "GPU를 이용한 R-tree에서의 범위 질의의 병렬 처리," 멀티미디어학회논문지, 제14권, 제5호, pp. 669-680, 2011.

김 병 주

2002년 경북대학교 전자전기공학부 (학사)
2004년 경북대학교 대학원 전자공학과 (공학석사)
2006년 경북대학교 대학원 전자공학과 (공학박사수료)
2006년 (주)휴원 과장
2010년 (주)LG전자MC연구소 선임연구원 (과장)
2011년 경북대학교 대학원 전자공학과 박사과정
관심분야 : 컴퓨터그래픽스, 컴퓨터비전, 기하 모델링, 병렬처리 등

이 정 은

2010년 경북대학교 컴퓨터공학과 (학사)
2010년 경북대학교 대학원 전자전기컴퓨터학부 (석사과정)
관심분야 : 컴퓨터 그래픽스, 계산생물학

김 영 준

1993년 서울대학교 계산통계학과 (학사)
1996년 서울대학교 계산통계학과 (석사)
2000년 미국 Purdue University (박사)
현재 이화여자대학교 컴퓨터공학과 교수
관심분야 : 실시간 컴퓨터 그래픽스, 컴퓨터 게임, 로보틱스, 햅틱스, 기하모델링

김 구 진

1990년 이화여자대학교 전자계산학과(학사)
1992년 한국과학기술원 전자계산학과(석사)
1998년 포항공과대학교 컴퓨터공학과(박사)
1998년 2000년 Dept. of Computer Sciences, Purdue University, PostDoc.
2000년 2002년 아주대학교 정보통신전문대학원 BK21 조교수
2002년 2003년 Dept. of Mathematics and Computer Science, University of Missouri St. Louis, Visiting Assistant Professor
2004년 경북대학교 컴퓨터학부 조교수
2008년 경북대학교 컴퓨터학부 부교수
관심분야 : 컴퓨터 그래픽스, 기하모델링, 계산생물학