JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 18, No. 6, pp. 1151-1156, Oct. 2017

일반적인 GPU 트리 탐색과의 비교실험을 통한 GPU 기반 병렬 Shifted Sort 알고리즘 분석

김 희 수 · 박 태 정*
덕성여자대학교 디지털미디어학과

Analysis of GPU-based Parallel Shifted Sort Algorithm by comparing with General GPU-based Tree Traversal

Heesu Kim · Taejung Park*
Department of Digital Media, Duksung Women's University, Samyangro 144gil 33, Dobong-gu Seoul 01369, Korea

[요 약]

일반적으로 GPU 기반 트리 탐색을 수행할 경우 병렬 처리 속도가 생각보다 크게 향상되지 않는 경우가 대부분이다. 본 논문에서는 이러한 원인을 분석하고 그 분석 결과로 GPU 병렬 처리 하드웨어 아키텍처 내 최소 물리적 스레드 실행 단위인 warp 내에서 분기문(if문)으로 인한 warp divergence가 일어나기 때문임을 제시한다. 또한 이러한 warp divergence를 최소화할 수 있는 병렬 shifted sort 알고리즘과의 비교를 통해 shifted sort 알고리즘이 일반적인 GPU 내 트리 탐색에 비해 우수한 성능을 보이는 구조임을 제시하였다. 분석 결과 GPU 기반 *k*d-tree 탐색에 비해 warp divergence가 발생하지 않은 shifted sort 탐색은 3차원 공간에서 데이터나 쿼리의 수가 2^{23}개 일 때 16배 이상의 빠른 처리 속도를 보였으며 이 성능 차이는 데이터나 쿼리의 개수가 증가함에 따라 더 커지는 경향을 보였다.

[Abstract]

It is common to achieve lower performance in traversing tree data structures in GPU than one expects. In this paper, we analyze the reason of lower-than-expected performance in GPU tree traversal and present that the warp divergences is caused by the branch instructions ("if... else") which appear commonly in tree traversal CUDA codes. Also, we compare the parallel shifted sort algorithm which can reduce the number of warp divergences with a *k*d-tree CUDA implementation to show that the shifted sort algorithm can work faster than the *k*d-tree CUDA implementation thanks to less warp divergences. As the analysis result, the shifted sort algorithm worked about 16-fold faster than the *k*d-tree CUDA implementation for 2^{23} query points and 2^{23} data points in R^3 space. The performance gaps tend to increase in proportion to the number of query points and data points.

색인어 : CUDA, GPGPU, 이동 정렬, 대략적 k개 최근접 이웃 검색, Warp Divergence, *k*d-tree

Key word : CUDA, GPGPU, Shifted Sort, *k* Approximate Nearest Neighbor Search, Warp Divergence, *k*d-tree

http://dx.doi.org/10.9728/dcs.2017.18.6.1151

Received 01 September 2017; **Revised** 11 October 2017
Accepted 25 October 2017

***Corresponding Author; Tae-Jung Park**

Tel: +82-02-901-8339
E-mail: tjpark@duksung.ac.kr

서 론

최근 화두가 되고 있는 딥러닝은 물론 다양한 분야에서 *k*NN (*k*-Nearest Neighbor Classifier)이 사용되고 있으며 약간의 정밀도를 희생하고 성능을 높이는 *k*ANN (k-Approximate Nearest Neighbor)이 사용된다. 일반적으로 이러한 목적을 달성하기 위해서 *k*d-tree와 같은 탐색 트리를 사용하는데 GPU 내에서 탐색 트리를 만들 경우 분기문(if문)이 많이 쓰이기 때문에 GPU 병렬 처리 속도가 생각보다 빠르게 향상되지 않는다. 하지만 대안으로 제시하는 병렬 shifted sort 알고리즘은 기존의 최근접 이웃 검색 기법으로 널리 사용되는 트리 탐색에 비해 우수한 성능을 얻을 수 있다.

현재까지 발표된 연구는 GPU 기반 트리 탐색보다 shifted sort 탐색이 더 우수한지에 대한 연구가 없었다. 따라서 이 논문에서는 그 구조적인 원인을 분석하고자 한다.

본 논문의 분석 결과, 일반적인 트리 탐색을 NVIDIA CUDA 아키텍처에서 수행하면 필수적으로 if 문이 필요하고 그 결과로 warp divergence가 발생하는 반면 shifted sort는 그렇지 않음을 알 수 있었다. 자세한 내용은 대표적인 탐색 알고리즘인 *k*d-tree의 GPU 버전을 shifted sort와 비교함으로써 본 논문에서 논의하도록 한다.

II. 기존 연구

2-1 Approximate Nearest Neighbor(ANN) 탐색

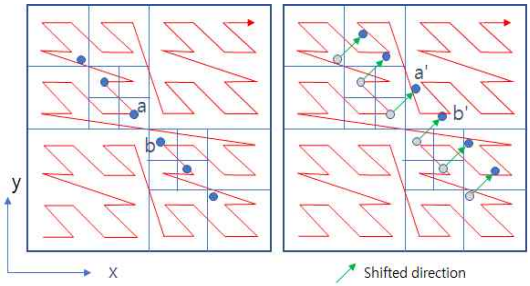
ANN은 다차원공간에서 가장 가까운 점을 찾기 위한 최적화 문제이다. 최근접 이웃 탐색 (NN; nearest neighbor) 문제에서 데이터 점들은 다차원의 공간 속에 주어지고, 그 점들은 데이터 구조에 따라 미리 계산한 후 주어진 쿼리에서 가장 가까운 점들을 찾게 된다. 이 두 점 사이의 거리를 구하기 위해서는 다양한 방법으로 계산이 되는데 Euclidean distance[1], Manhattan distance[2], 그리고 max distance[3] 등의 방법을 사용하여 거리를 계산한다.

ANN은 수천에서 수백만의 데이터가 있는 다차원의 공간에서 효율적으로 작동한다. 이 라이브러리는 *k*d-trees나 box-decomposition trees 방식 등 다양한 공간탐색기법을 기본으로 하며, 기하학 정보 구조 또는 시각적인 데이터 구조를 효과적으로 측정하는 데 사용한다[4].

2-2 GPU에서 *k*d-tree 탐색

*k*d-tree는 공간 분할 자료구조로 *k*-dimensional tree를 의미한다[5]. *k*d-tree는 Binary Space Partitioning tree의 특수형태에 해당하며, *k* 차원의 공간에서 데이터를 분류하는 *k*d-tree는 공간탐색이나 최근접 이웃 데이터를 찾는 응용 분야에 이용된다.

기본적으로 *k*d-tree는 이진 트리(Binary tree)를 다차원



1. Morton code 이동 방향
Fig. 1. Morton code shifting direction

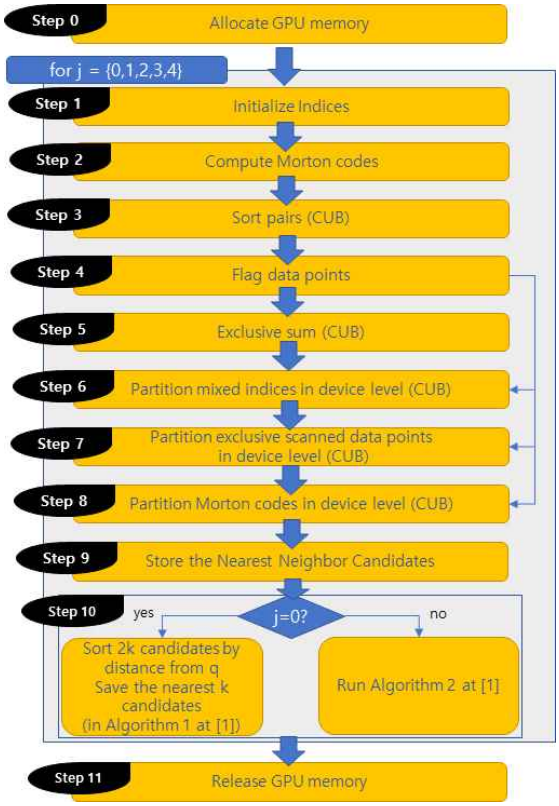


그림 2. Warp 기반 shifted sort의 세부 단계 [6]
Fig. 2. Detailed steps in warp-based shifted sort [6]

공간으로 확장한 것으로, 자식 노드가 있는 경우 그 공간을 두 부분으로 나누고 초평면(hyper plane)을 생성한다. 이 초평면은 차원의 각 축에 수직으로 공간을 반으로 분할한다. *k*d-tree는 일반적인 축의 순서에 따라 x, y, z 축 순서로 나뉘기도 하며, SAH(Surface Area Heuristic)[7] 방식을 통해 다음 순서로 나눠질 축을 탐색하기 위한 우선순위를 정해주는 계산방식을 사용한다. 이 경우 현재 노드가 가지고 있는 프리미티브들의 개수 또는 나눠질 서브트리가 가지고 있는 최소 메쉬 단위인 삼각형의 개수 혹은 서브 트리의 부피 등을 가지고 비용이 더 적은 축을 찾아 우선순위를 정해주는 계산을 한다.

따라서 많은 수의 데이터와 쿼리를 받아서 CPU의 heap 공간

```
kd-tree searching pseudo code
search(q:query, n:node, p:ref point, d:ref distance)
{
    if n.left =n.right = null then {leaf case}
       d2=||q-n.point||;
       if d2 < d then d = d2; p = n.point;
    else
        if q(n.axis)<=n.value then
           search_first = left;
        else
           search_first = right;

        if (search_first == left)
              if q(n.axis) -d <= n.value then search(q, n.left, p, d);
              if q(n.axis) +d > n.value then search(q, n.right, p, d);

        else //search_first == left
              if q(n.axis) +d > n.value then search(q, n.right, p, d);
              if q(n.axis) -d <= n.value then search(q, n.left, p, d);
}
```

3. *k*d-tree 코드 안의 분기문

Fig. 3. Branching statements in *k*d-tree search code

에 저장해주고 *k*d-tree 구조를 만들어준 다음, 쿼리와 데이터들을 GPU로 복사해준 뒤 탐색하는 방식을 통해 공간탐색을 한다.

2-3 GPU에서 Shifted Sort 탐색

GPU는 CPU보다 분기문 처리에 필요한 제어 기능이 구조적으로 취약하기 때문에 GPU에 맞는 효율적인 소프트웨어 구현이 필요하다. 이러한 문제를 해결한 Shifted sort는 그림 2와 같은 세부 단계를 가지고 있으며 데이터와 쿼리들을 Morton code를 이용하여 표현한다[8]. Morton code는 z-curve로 다차원 데이터를 일차원으로 나열할 수 있고, GPU 글로벌 메모리 접근이나 사용에 더 효율적으로 접근이 가능하다. 하지만 Morton code의 한계점으로는 다차원 공간에서 보면 가까운 위치에 있는 데이터들이 z-curve로 표현했을 경우 멀리 떨어져 있는 것처럼 계산되는 문제가 발생한다 (그림 1). Li et al.[8]에서는 이러한 문제를 해결하기 위해 전체 표본에 대해 Scaling과 Shifting 방법을 이용하여 그림 1의 문제를 해결하는 방식을 사용하였다. 3차원 데이터의 경우 전체 표본의 크기를 $[0,0.75]^3$로 scaling을 적용하고 z축에 따라 0.05씩 총 다섯 번의 위치를 이동을 시켜 계산함으로써 다차원 공간상의 데이터를 일차원으로 나열했을 경우 가까운 위치에 있는 데이터들이 멀어지는 오류를 줄일 수 있다. 이렇게 Morton code로 나열한 데이터들은 21비트씩 x, y, z축으로 나타내어 총 63비트로 나타내고, 각 축의 마지막 비트(Least Significant Bit)는 쿼리(1), 데이터(0)으로 나타낼 수 있다. 자세한 내용은 [9]을 참고한다.

Ⅲ. 실험결과 및 분석

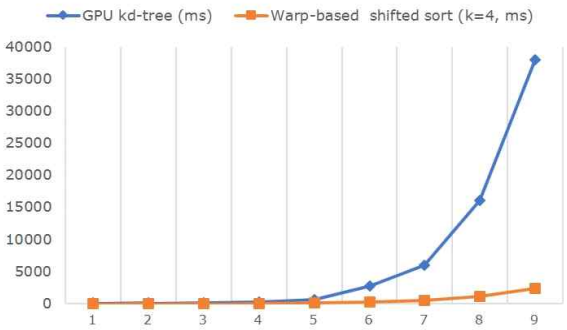
3-1 실험 개요 및 결과

이 실험에서는 NVIDIA사의 GeForce GTX 1080을 장착한 Intel i5 CPU(3.40GHz) 시스템에서 Windows 7(64bit), CUDA 8.0을 적용하여 구현 및 테스트를 수행하였다. 또한, *k*d-tree와

표 1. GPU 기반 *k*d-tree와 warp 기반 shifted sort 결과 비교

Table. 1. Performance comparison between GPU-based kd-tree, and warp-based shifted sort

data size	query size	GPU *k*d-tree (ms)	Warp-based shifted sort (ms)
2^{10}	2^{10}	115	**2**
2^{12}	2^{12}	118	**6**
2^{14}	2^{14}	145	**8**
2^{16}	2^{16}	213	**21**
2^{18}	2^{18}	552	**71**
2^{20}	2^{20}	2661	**270**
2^{21}	2^{21}	5962	**522**
2^{22}	2^{22}	16053	**1059**
2^{23}	2^{23}	38003	**2329**



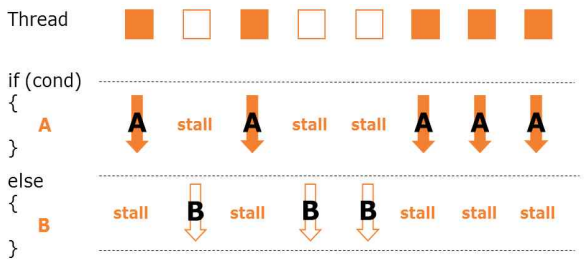
shifted sort의 실행 코드는 Visual Studio 2012 버전 개발 환경에서 실행하였다.

GPU 상에서 데이터 및 쿼리는 각각 2^{23}개를 기준으로 CUDA로 구현한 *k*d-tree 코드의 실행 결과 트리를 만들고 검색하는데 걸리는 시간은 38003ms로 측정되는 반면 같은 조건을 가지고 shifted sort 코드를 실행한 결과 2329ms로 시간이 측정돼 *k*d-tree보다 shifted sort가 약 16배의 속도 차이를 보였으며, 데이터와 쿼리의 크기가 커질수록 이 차이는 더 벌어질 것으로 예상한다 (표 1).

3-2 기본 CUDA 아키텍처 구조

1) Warp

GPU는 CPU와 달리 core의 스레드별로 독립적인 레지스터가 각각 할당이 되어있기 때문에 CPU보다 컨텍스트 스위칭 오버헤드가 적다. 하드웨어적으로 보면 하나의 Streaming Multiprocessor (SM)당 32개의 코어들을 가지고 있고 이 각각의 코어들은 각자 스레드를 물리적으로 실행시킴으로써 연산을 처리하게 된다. 이때 32개의 코어를 묶어 하나의 Warp이라고 하고, 이는 SM에서 처리할 수 있는 기본 단위라고 정의한다.



4. warp divergence 시 스레드 명령 흐름
Fig. 4. Thread instruction flow in warp divergence

2) Warp Divergence

기본적으로 CPU는 제어에 최적화된 회로를 가지고 있는 반면, GPU는 병렬처리를 위해 대규모 연산처리에 특화되어 있기 때문에 분기 예측을 하는 메커니즘이 상대적으로 부족하다. 따라서 분기문 같은 복잡한 연산 처리 시 그 성능이 현저하게 떨어지는 것을 볼 수 있다. 그 이유는 하나의 warp 단위에 포함된 32개의 스레드들은 한 사이클 연산 당 모두 같은 명령어를 받고 서로 다른 데이터를 처리하기 때문에 (SIMD; Single Instruction Multiple Data[10]) 분기문을 사용할 경우 32개의 스레드를 효율적으로 사용하지 못하게 된다. 따라서 한 warp 안에서 모든 스레드들이 같은 명령을 실행하지 못하고 연산 분기가 나누어지는 경우 그림 4와 같이 각각의 스레드는 if 문 안의 명령인 A와 else 문의 명령인 B 두 가지 모두를 실행하게 된다. 첫 번째 스레드의 경우 의도한 정답이 if 문의 A이기 때문에 else 문의 B는 불필요하게 실행되는 stall상태가 된다. 두 번째 스레드의 경우에도 if 문의 A와 else 문의 B가 모두 실행이 되지만 else 문의 B를 의도했기에 A 부분은 버려지게 되는 비효율적인 접근이 된다. 다시 말해 이 분기는 스레드 별로 병렬적으로 처리가 되지 않고 순차적으로 실행되기 때문에 warp divergence가 발생했다고 한다.

3-3 *k*d-tree와 Shifted Sort Searching 커널 분석

1) Visual Profiler

코드의 상세한 warp 및 시스템 분석을 위하여 구현한 코드를 NVIDIA사의 Visual Profiler 8.0 버전을 이용해서 코드를 분석하였다. Visual Profiler는 CUDA C/C++의 성능을 검사해 최적화할 수 있는 피드백을 제공한다[11]. 본 논문에서 사용한 shifted sort코드를 프로파일러에서 실행한 결과 최근접 후보 이웃들의 거리를 계산하고 모아주는 storeNNCandidatesEuclidean 커널은 67.3%의 계산비율을 차지하고 있으며, 이웃들을 탐색하는 SearchBatch 커널은 22.2%, Morton code를 계산하는 computeMortoncodes 커널은 0.3%의 계산 비율을 차지하고 있는 것으로 나타난다. 이후 성능에 영향을 미치는 다양한 요소들은 다음 장에서 논의하고자 한다.

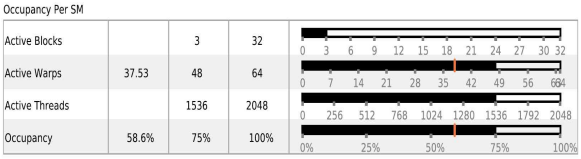


그림 5. SM 당 *k*d-tree 점유율 분석
Fig. 5. *k*d-tree occupancy analysis per SM

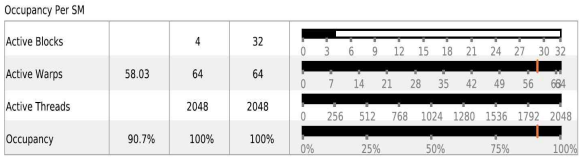


그림 6. SM 당 shifted sort 점유율 분석
Fig. 6. shifted sort occupancy analysis per SM

2) Compute Resources

GPU compute resource는 자원들이 충분하지 못하거나 활용되지 못할 때 커널의 성능을 제한받는다. compute resource가 영향을 받는 요소에는 레지스터, 메모리 대역폭, 점유율 등이 있고 [12], 앞서 제시한 프로파일러의 지표에서 보면 warp execution는 warp 안의 스레드들이 모두 활동하고 있는 비율로 나타낼 수 있으며 GPU의 컴퓨팅 성능을 향상하는데 많은 기여한다.

그림 3에서 볼 수 있듯이 *k*d-tree 검색 알고리즘에는 if 문이 많은 것을 볼 수 있다. 따라서 *k*d-tree의 경우 각각의 warp에서 실행 중인 스레드들이 서로 다른 분기를 가지는 divergence가 발생하였고, 예측할 수 없는 명령어 계산으로 인하여 7.7%의 warp execution efficiency가 나타났다. 또한, 예측 할 수 없는 명령어를 제외한 효율성 측정에서도 0.1% 내외의 오차율로 7.7%의 효율성을 나타내는 것을 알 수 있다.

shifted sort의 경우, 자세한 코드 분석을 위하여 커널을 여러 개로 분할하여 실행하였다. 그중에서도 48%의 가장 큰 비율을 차지하는 storeNNCandidatesEuclidean 커널의 경우, 커널에서의 warp execution 효율성은 100%이며, 예측할 수 없는 명령어를 고려하지 않은 경우 warp 실행 효율성은 99.4%인 것을 알 수 있다. 또 다른 커널 computeMortoncodes의 실행 결과 또한 점유율이 77.4%로 *k*d-tree의 점유율 57.3%보다 20.1% 정도 더 좋은 것으로 나타나 shifted sort는 *k*d-tree와는 달리 warp divergence가 일어나지 않아 더 효율적이라고 볼 수 있다.

3) Memory Latency

메모리 지연 시간은 GPU에 작업량이 충분하지 않을 때 커널의 성능을 제한한다. 이는 다시 말해 실행할 준비가 되어있는 스레드들이 충분하다면 GPU는 각각의 작업들을 병렬적으로 처리하면서 효율적인 파이프라인을 가질 수 있다는 것이다. 예를 들어 A라는 사람이 1, 2, 3 세 가지 일을 처리한다고 할 때, 1이라는 일을 처리하기 위해서 A는 필요한 물품을 택배를 통해

주문한다. 그 택배가 오는 동안 A는 시간을 소비하며 기다리지 않고 2 또는 3의 일을 처리하면서 1의 물품이 오기를 기다리면 된다. 여기서 택배를 보내는 곳은 GPU의 글로벌 메모리가 되고 필요한 물품은 요청할 데이터가 된다. 이렇게 A는 병렬적으로 일을 처리하면서 각각의 일에 대한 파이프라인을 가지게 된다. 만약 2나 3의 일이 없다고 하면 A는 1의 물품이 오기만을 기다리면서 시간, 능력을 낭비하게 되고 이는 GPU의 커널의 성능을 제한하는 일이 된다. 따라서 커널의 성능을 높이려면 메모리 점유율을 높여야 하는데, 점유율은 GPU가 지원하는 최대 warp의 개수에 비례하여 커널에서 활성화한 warp의 개수를 측정한다. 또한, 점유율을 높이기 위해서는 스레드의 개수는 warp 사이즈인 32의 배수로 지정하고, 커널의 성능에 맞게 블록의 사이즈를 조정해서 활동 warp을 더 늘려야 한다.

이 테스트에서 구현한 *k*d-tree의 경우, 각 스레드당 37개의 레지스터를 사용한다. 이는 각 블록 당 18944개의 레지스터를 사용하는 것으로 본 논문에서 사용한 Geforce GTX 1080에 적용된 레지스터가 65539개라는 사실을 고려해 본다면, 약 58.6%의 성능을 가지고 있다 (그림 5).

shifted sort의 storeNNCandidatesEuclidean 커널의 경우, 100%에 가까운 warp 효율성을 가지고 있는 만큼 점유율에서도 90.7%의 효과적인 성능을 보여주고 있다 (그림 6). 이 커널의 구현에 대한 자세한 논의는 [6]를 참고한다.

4) 개선 방향

커널의 warp execution 향상을 위해서 intra-warp divergence와 실행 경로의 분기를 줄여야 한다. 이를 위해서는 같은 warp에 있는 스레드들은 동일한 명령을 실행하도록 해야 한다. 그 이유는 앞서 3-1절에서 논의한 대로 커널 안에서의 물리적 연산은 warp 단위로 이루어지며 한 warp 내부에서 분기문이 존재하고 32개 스레드가 다른 실행 경로를 가지면 각 실행 경로마다 동시에 실행되지 않고 순차적으로 실행되기 때문이다.

점유율을 향상시키기 위해서는 커널에서 사용하는 스레드 블록의 사이즈를 조정함으로써 더 많은 수의 warp을 활동 가능하게 만들거나 레지스터 또는 공유 메모리 같은 compute resource의 이용을 용이하게 조절하는 방법으로 점유율을 향상시킬 수 있다. 만약 스레드 블록의 크기가 작다면 자원 활용도가 낮아져 유휴상태가 된다. 반대로 스레드 블록의 크기가 크다면 자원에 비하여 스레드의 개수가 많아지기 때문에 직접 실험을 통해 최대의 효율을 찾아야 한다.

Ⅳ. 결론 및 논의

기존 방식인 GPU 기반 트리 탐색을 수행할 경우 본질적으로 발생하는 분기문의 처리 때문에 warp divergence가 발생되어 병렬 처리 속도가 생각보다 빠르게 향상되지 않는다는 가정을 Visual Profiler를 통해 확인하였다. 이에 비해 병렬 shifted sort 알고리즘은 기존의 트리 탐색에 비해 우수한 성능을 보였다. 본 논문에서는 이러한 분석을 통해 GPU 기반 트리 탐색보다 shifted sort 탐색이 더 빠른 처리 속도를 나타내는 구조적인 원인을 제시하였다.

분석 결과, shifted sort 탐색 기법은 기존의 최근접 이웃 탐색 기법으로 널리 사용되는 트리 탐색 중에 일반적으로 발생하는 집중적인 분기문이 발생하지 않아 warp 내부에서 스레드들마다 다른 명령을 실행하지 않는다. 따라서 warp divergence가 일어나지 않고 그 결과 GPU 기반 *k*d-tree 탐색에 비해 최대 16배 정도의 처리 속도 차이를 보였다. 이 결과는 입력 데이터나 쿼리의 크기가 더 커지면 그 차이는 더 커지는 것을 확인하였다.

이러한 분석 결과를 토대로 향후 shifted sort 알고리즘을 구현 레벨에서 보다 최적화할 수 있는 연구를 수행하고자 한다.

감사의 글

본 연구는 산업통상자원부(MOTIE)와 한국에너지기술평가원(KETEP)의 지원을 받아 수행한 연구 과제입니다(과제번호 : 20161210200610).

이 논문은 2016년도 정부(교육부)의 재원으로 한국연구재단의 지원을 받아 수행된 기초연구사업임 (NRF-2016R1D1A1B03933660).

참고문헌

[1] Euclidean distance website. Available: https://en.wikipedia.org/wiki/Euclidean_distance

[2] Manhattan distance website. Available: https://en.wikipedia.org/wiki/Taxicab_geometry

[3] Max distance website. Available:https://en.wikipedia.org/wiki/Chebyshev_distance

[4] ANN: A Library for Approximate Nearest Neighbor Searching website. Available: https://www.cs.umd.edu/~mount/ANN/

[5] kd-tree searching website. Available: https://en.wikipedia.org/wiki/K-d_tree

[6] T. Park, “Optimization of Warp-wide CUDA Implementation for Parallel Shifted Sort Algorithm,” Journal of Digital Contents Society, Vol. 18, No. 4, pp. 739-745, July 2017.

[7] Ingo Wald, "On fast Construction of SAH-based Bounding Volume Hierarchies," Proceedings of the 2007 IEEE symposium on Interactive Ray Tracing, Washington, pp. 33-40, 2007.

[8] S.Li, L. Simons, J. B. Pakaravoor, F. Abbasinejad, J. D. Owens, and N. Amenta, "kANN on the GPU with shifted sorting," In

Proceedings of the Fourth ACM SIGGRAPH / Eurographics conference on High-Performance Graphics (EGH-HPG'12), Switzerland, pp. 39-47, 2012.
[9] T. Park, “Analysis of Morton Code Conversion for 32 Bit IEEE 754 Floating Point Variables,” The Journal of Digital Contents Society, Vol. 17, No. 3, pp. 165-172, June 2016.
[10] J. Cheng, M. Grossman, and T. McKercher, Professional CUDA C Programming, 1sted. Wrox, pp. 6-8, 2014.
[11] NVIDIA Visual Profiler website. Available: https://developer.nvidia.com/nvidia-visual-profiler
[12] J. Cheng, M. Grossman, and T. McKercher, Professional CUDA C Programming, 1sted. Wrox, pp. 87-96, 2014.

김희수(Heesu Kim)

2015년 : 덕성여자대학교 디지털미디어 (공학사)
2016년～현재 : 덕성여자대학교 디지털미디어학과 대학원 석사 과정 재학

※관심분야 : 컴퓨터그래픽스, 병렬처리, 3차원 모델링

박태정(Taejung Park)

1997년 : 서울대 전기공학부 (공학사)
1999년 : 서울대 전기공학부 대학원 (공학 석사, 반도체 물리 전공)
2006년 : 서울대 전기컴퓨터공학부 대학원 (공학박사, 컴퓨터 그래픽스 전공)

2006년～2013년: 고려대학교 연구교수
2013년～현재 : 덕성여자대학교 디지털미디어학과 조교수
※관심분야 : 컴퓨터그래픽스, 병렬처리, 게임 물리, 수치해석, 3차원 모델링