바이토닉 정렬 기반의 GPU 해싱을 이용한 인접 입자의 빠른 접근 기법과 그 응용 사례

이수빈[O], 김종현[*]
[O]강남대학교 소프트웨어응용학부,
[*]강남대학교 소프트웨어응용학부
e-mail: jonghyunkim@kangnam.ac.kr

Fast Access Method of Neighboring Particles Using Bitonic Sort Based GPU Hashing, and Its Applications

SuBin Lee[O], Jong-Hyun Kim[*]
[O]School of Software Application, Kangnam University,
[*]School of Software Application, Kangnam University

● 요 약 ●

본 논문에서는 대용량 데이터에서 빠르게 주변 데이터를 접근하기 위한 자료구조인 최근접 이웃 탐색(Nearest neighbor search, NNS) 문제를 빠르게 풀 수 있는 바이토닉 정렬(Bitonic sort) 기반 해시 테이블을 GPU기반에서 설계하는 방법과 이를 통해 입자 기반 물리 시뮬레이션을 고속화할 수 있는 방법에 대해 살펴본다. 본 논문에서는 CUDA 아키텍처를 이용하여 해시 테이블을 설계하였으며, 계산양이 가장 큰 데이터 정렬부분을 최적화함으로써 NVIDIA에서 제공하는 CUDA 해시 테이블보다 빠른 결과를 얻을 수 있으며, 이 자료구조를 입자 기반 시뮬레이션에 통합함으로써 고성능 시뮬레이션을 쉽게 제작할 수 있다.

키워드: 쿠다(Computed Unified Device Architecture, CUDA),
입자 기반 시뮬레이션(Particle-based simulation), 해싱(Hashing),
바이토닉 정렬(Bitonic sort), 최근접 이웃 탐색(Nearest neighbor search)

I. Introduction

NVIDIA의 CUDA 프로그래밍은 계산양이 큰 문제를 병렬 최적화하여 빠르게 풀 수 있는 대표적인 아키텍처이다. 지오메트리, 시뮬레이션, 수치해석, 인공지능 등 많은 분야에서 활용되고 있으며 특히 처리해야 될 데이터의 수가 많을수록 더욱더 효율적인 성능을 보여준다[1-4]. 대표적으로 입자 시뮬레이션(Particle simulation)은 입자의 수가 많아지면 고속화 자료구조가 필요하며, NVIDIA에서도 트러스트(Thrust) 병렬 알고리즘 라이브러리를 이용한 CUDA기반 해시 테이블을 활용하여 실시간 성능을 보여주었다. 많은 개수의 입자들에서도 빠른 시뮬레이션 성능을 보여주기 때문에 다양한 기법들에서 활용되고 있지만, 데이터 정렬 부분에서 계산 성능이 저하되며, 대용량 입자 기반 시뮬레이션일수록 이 문제는 더욱더 심각하게 나타난다.

Fig. 1은 CUDA에서 제시하는 해시 테이블의 함수별 계산 시간을 보여주는 결과이다. 그림에서 보듯이 대부분의 계산시간이 정렬과정에서 소요되는 것을 알 수 있다. 본 논문에서는 이 문제에 착안하여 정렬을 최적함으로써 GPU 기반 해싱을 고속화 할 수 있는 새로운 프레임워크를 제안한다.

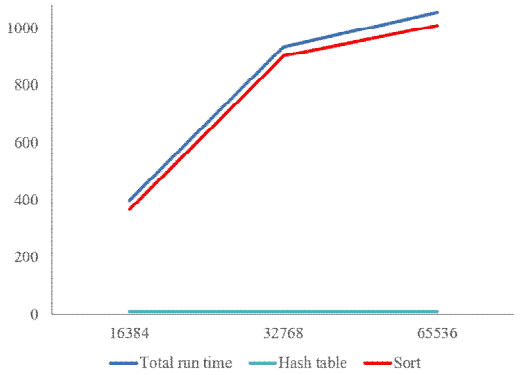


Fig. 1. Calculation time for each function of CUDA Hash table[5].

II. The Proposed Scheme

1. GPU기반 바이토닉 정렬을 위한 커널 디자인

바이토닉 정렬은 병렬적인 환경에서 효율적으로 정렬하기 위한 알고리즘이다. 단일 스레드에서는 일반적인 $O(N\log N)$ 정렬 알고리즘들에 비해 $O(N\log^2 N)$으로 효율성이 떨어지지만, GPU와 같이 프로세서가 무수히 많은 환경에서는 $O(\log^2 N)$에 정렬을 처리할 수 있다. 바이토닉은 어떤 원소를 기준으로 이전에는 증가하고 이후로는 감소하는 두 개의 모노토닉 시퀀스를 갖는 바이토닉 시퀀스를 의미한다. 바이토닉 정렬은 바이토닉 시퀀스로 정리된 원소를 머지하며, 그 결과는 계속해서 바이토닉 시퀀스 형태를 유지한다. 형태를 유지하고, 두 모노토닉 시퀀스 중 어느 하나에 대한 정렬을 먼저 수행한다고 해서 결과에 영향을 주지 않기 때문에 병렬 프로그래밍에 적합하다.

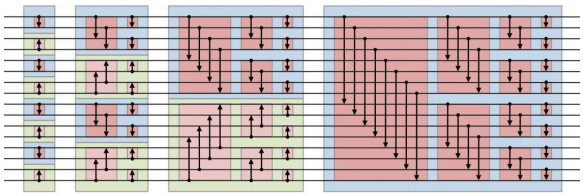


Fig. 2. Sorting process using Bitonic sort.

Fig. 2는 복잡해 보이지만, 원리는 일반적인 병합 정렬과 유사하다. 다만 병합 정렬은 해당 원소 개수 전체를 하나의 스레드가 전부 확인해야 하기 때문에 아무리 많은 프로세서가 있더라도 $O(N)$ 미만으로 단축시킬 수 없는 데에 비해, 바이토닉 정렬에서는 위 그림에서 세로로 나란한 빨간 상자들에 속한 연산들끼리 모두 병렬적으로 처리할 수 있기 때문에 프로세서가 많다면 1부터 $\log N$까지의 합, 즉 $O(\log^2 N)$ 시간에 정렬이 가능하다. 병렬적인 처리를 위해서는 바이토닉 정렬함수(Sort function)와 바이토닉 머지함수(Merge function)의 실행 및 비교할 때의 루프 등이 모두 여러 스레드에서 병렬적으로 실행되어야 한다.

```
for (int i = 2; i <= numParticles; i *= 2) {
    for (int j = i / 2; j >= 1; j /= 2) {
        bitoSortKernel<<<numBlocks, numThreads>>>(hashing, i, j, numParticles);
    }
}
```

Fig. 3. Bitonic sort function.

위 코드는 바이토닉 정렬을 위함 함수이며 (Fig. 3 참조), 이중 루프로 원소를 나누는 간격 i와 원소 비교 오프셋 j를 설정해 Fig. 2의 세로로 나란한 파란 상자와 초록 상자의 묶음으로 바이토닉 정렬함수를 실행한다. 이 함수에서 $bitoSortKernel$은 GPU 커널 함수이며, $numBlocks$와 $numThreads$는 GPU 블록과 스레드 개수를 나타내고, 자세한 GPU 커널은 다음과 같다 (Fig. 4 참조).

```
__global__ void bitoSortKernel(hashTable, int i, int j, int numParticles)
{
    uint tid = threadIdx.x + blockIdx.x * blockDim.x;
    if (tid < numParticles && tid % (j << 1) < j) {
        bool descending = (tid / i) % 2;
        if (descending && hashTable.particleHash[tid] < hashTable.particleHash[tid + j]) {
            swap(hashTable.particleHash[tid], hashTable.particleHash[tid + j]);
            swap(hashTable.particleIndex[tid], hashTable.ParticleIndex[tid + j]);
        } else if (!descending && hashTable.particleHash[tid] > hashTable.particleHash[tid + j]) {
            swap(hashTable.particleHash[tid], hashTable.particleHash[tid + j]);
            swap(hashTable.particleIndex[tid], hashTable.particleIndex[tid + j]);
        }
    }
}
```

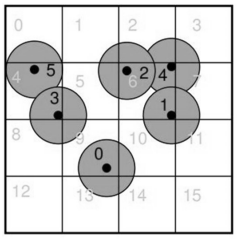
Fig. 4. GPU kernel function in Bitonic sort.

위 그림에서 $hashTable$는 해싱 자료구조를 가지고 있는 객체이며, Fig. 2의 파란 상자 또는 초록 상자와 같이 스레드 인덱스가 해당되는 모노토닉 시퀀스에 따라 해시 인덱스와 입자 인덱스가 교차로 정렬되는 빨간 상자에 속한 연산인 바이토닉 머지 함수의 과정을 보여주고 있다.

2. GPU 기반 해시 테이블 구성

앞에서 언급한 정렬 커널을 이용하여 GPU 기반 해시 테이블을 구성하는 방법에 대해서 설명한다. Fig. 1에서 보여주듯이, 해시 테이블을 구성하는 과정에서 정렬이 가장 계산양이 크기 때문에 이 부분만을 바이토닉 정렬 과정으로 변경하고, 나머지는 NVIDIA의 CUDA 해시 테이블 자료구조를 활용한다[5].

1. 입자 데이터를 해시 테이블에 저장
2. GPU 기반 바이토닉 정렬을 이용한 데이터 정렬
3. 격자 인덱스를 통한 데이터 재정렬



Index	Unsorted list (cell id, particle id)	List sorted by cell id	Cell start
0	(9, 0)	(4, 3)	
1	(6, 1)	(4, 5)	
2	(6, 2)	(6, 1)	
3	(4, 3)	(6, 2)	
4	(6, 4)	(6, 4)	0
5	(4, 5)	(9, 0)	
6			2
7			
8			
9			5
10			
11			
12			
13			
14			
15			

Fig. 5. Uniform grid using sorting.

Fig. 5는 6개의 입자들이 해시 테이블에 저장되는 과정을 보여주고 있는 그림이다. 또한, 추가적인 최적화를 위해 격자 인덱스를 재정렬함으로써 해시 테이블을 사용하는 과정에서 데이터 일관성(Coherence)을 개선한다.

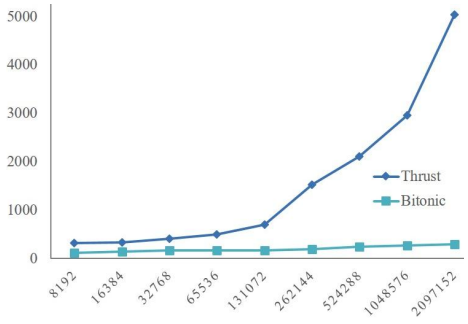


Fig. 6. Comparison results of sorting elements with Thrust and Bitonic algorithm.

Fig. 6은 GPU기반 트러스트 라이브러리와 바이토닉 정렬을 통해 비교한 정렬 시간이다. 두 기법 모두 GPU기반에서 실행되었음에도 불구하고 트러스트 기법은 입자의 개수가 증가할수록 계산양이 증가되는 결과를 보여주었지만, 바이토닉 정렬은 상대적으로 계산 시간의 큰 변화가 없는 결과를 보여준다.

3. 바이토닉 정렬 기반 해싱을 이용한 입자 기반 유체 시뮬레이션

본 논문에서 제시한 자료구조의 효율성을 보여주기 위해, 해시 테이블을 활용하는 대표적인 애플리케이션인 입자 기반 유체 시뮬레이션에 우리의 기법을 적용/비교를 하였다. 입자의 위치에 변경됨에 따라 매 프레임 해시 테이블을 갱신해야 되며, 주변 입자의 값을 활용하는 밀도 값을 계산하는 과정을 비교하였다. 입자 기반 유체 시뮬레이션에서 입자를 계산하는 방법은 아래와 같은 수식을 이용한다 (수식 1 참조).

$$p_i = \sum_j m_j W_{ij} \quad (1)$$

여기서 p는 밀도이고, m은 질량이고, W는 Poly6 스무딩 커널 (Smoothing kernel)이다 (수식 2 참조).

$$W_{ij} = \frac{315}{64\pi h^9}(h^2 - r^2)^3 \quad (2)$$

여기서 h는 스무딩 반지름이고, r은 입자들 사이의 거리이다. 반지름이 이미 수식에서 제곱되기 때문에 입자 사이의 거리에 대한 제곱근을 계산하지 않는다는 특징이 있다.

Fig. 7은 입자 데이터와 해시 테이블을 이용하여 밀도 계산을 비교한 결과이다. 삽입된 그림은 입자의 밀도를 시각화한 결과로써, 바깥쪽에 비해 안쪽의 밀도가 크게 계산된 것을 잘 보여준다. 같은 결과를 만들어 냄에 있어서 계산 시간을 비교했을 때, 우리의 방법인 GPU 바이토닉 정렬 기반의 해싱 알고리즘이 월등하게 빠른 결과를 보여주고 있다.

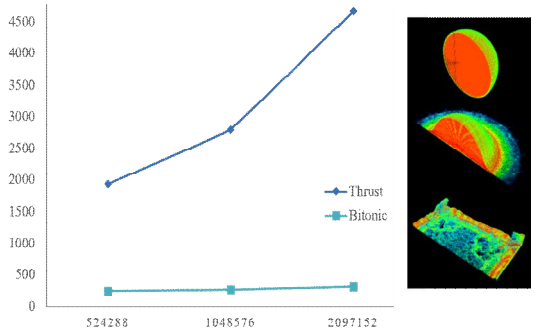
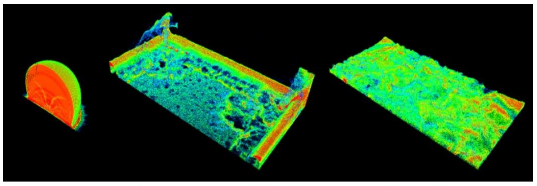
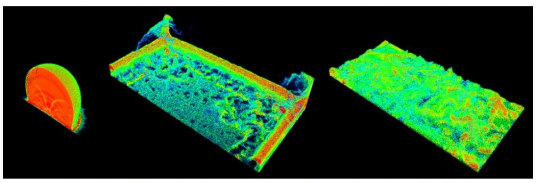


Fig. 7. Comparison results of density calculations (inset image : density of water particle, red : high density, blue : low density).

우리는 제안한 알고리즘을 물 시뮬레이션에 적용하여, 제대로 동작하는지 실험을 진행하였다. 여기서 사용한 유체 시뮬레이션 해법은 격자-입자 하이브리드 기법인 FLIP(Fluid-Implicit Particle)을 이용하였으며[6], 압력을 계산하기 위한 행렬 해법은 Preconditioned Conjugate Gradient 방법을 이용하였다[7].



(a) Visualizing density with Bitonic sort based hash table



(b) Visualizing density with Thrust library based hash table

Fig. 8. Comparison results of density on animating particle data.

Fig. 8은 동일한 프레임에서 측정한 밀도 값을 비교한 결과이며, 트러스트 라이브러리와 바이토닉 정렬 기반 해싱 모두 동일한 결과를 만들어 냈으며, 바이토닉 정렬 기반 해싱이 더 빠른 결과를 만들어냈다.

Fig. 9는 본 논문에서 제안한 방법을 통해 구현된 유체 시뮬레이션 결과이다. 입자의 색은 속도를 나타내며, 어색한 움직임 없이 안정적으로 유체의 움직임을 잘 표현하였다. 본 논문에서 제안한 방법을 NNS문제를 빠르게 풀 수 있는 자료구조이기 때문에 유체 시뮬레이션뿐만 아니라, 헤어, 강제, 변형체, 등 다양한 수치해석 시뮬레이션에 활용될 수 있다.

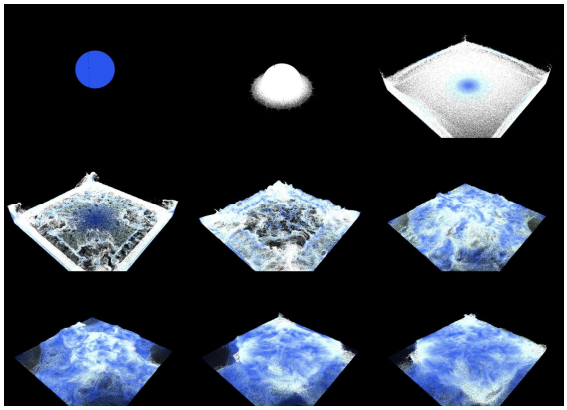


Fig. 9. FLIP based fluid simulations with our method (number of particles : 2,097,152).

IV. Conclusions

본 논문에서는 빠르고 안정적으로 해시 테이블을 계산할 수 있는 새로운 GPU 프레임워크를 제안하였다. 해시 테이블을 구성하는데 있어서 계산 시간이 가장 오래 소요되는 정렬 부분을 GPU기반 바이토닉 정렬 알고리즘을 수정함으로써 병렬화를 개선하였다. 본 논문에서는 이 자료구조를 입자 기반 유체 시뮬레이션에 적용하여 그 성능을 입증하였다.

현재 모델링된 기법은 바이토닉 정렬의 한계점인 2의 거듭제곱에만 동작하기 때문에 다양한 개수의 입자에 대해서는 제대로 동작하지 않는다. 향후, 2의 거듭제곱이 아닌 형태에서도 GPU 바이토닉 정렬이 가능하도록 확장하여 다목적성에 맞는 자료구조로 디자인할 계획이다.

REFERENCES

[1] Bogomjakov, Alexander, and Craig Gotsman. "GPU-Assisted Geometry Processing for Novel View Synthesis from Depth Video." PhD diss., Computer Science Department, Technion, 2008.

[2] Amada, Takashi, Masataka Imura, Yoshihiro Yasumuro, Yoshitsugu Manabe, and Kunihiro Chihara. "Particle-based fluid simulation on GPU." In ACM workshop on general-purpose computing on graphics processors, vol. 41, pp. 42. 2004.

[3] Alimirzazadeh, S., Kumashiro, T., Leguizamón, S., Jahanbakhsh, E., Maertens, A., Vessaz, C., Tani, K. and Avellan, F., "GPU-accelerated numerical analysis of jet interference in a six-jet Pelton turbine using Finite Volume Particle Method", Renewable Energy, vol. 148, pp.234-246, 2020.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mane, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viegas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[5] Green, Simon. "Particle simulation using cuda." NVIDIA whitepaper vol. 6, pp. 121-128, 2010.

[6] Zhu, Y., & Bridson, R., "Animating sand as a fluid". ACM Transactions on Graphics (TOG), vol. 24, no. 3, pp. 965-972, 2005.

[7] Helfenstein, Rudi, and Jonas Koko. "Parallel preconditioned conjugate gradient algorithm on GPU." Journal of Computational and Applied Mathematics, vol. 236, no. 15, pp. 3584-3590, 2012.