GPU 기반 연결 리스트를 이용한 해시 테이블의 생성*

정주현[0], 이성길
성균관대학교 컴퓨터공학과
(jhjeong10, sungkil)@skku.edu

Hash Table Construction using Linked List on GPU

Juhyeon Jeong[0], Sungkil Lee
Dept. of Computer Science and Engineering, Sungkyunkwan University

요약

해시 테이블(hash table)은 키와 값으로 이루어진 연관 배열(associative array)에서 사용되는 자료구조로서 상수 시간 내에 데이터 검색이 가능하다. 본 논문에서는 GPU를 이용하여 해시 테이블을 병렬적으로 생성하는 방식을 제안한다. 해시값의 충돌 처리를 위해 연결 리스트를 이용한 분리연쇄법을 사용한다.

1. 서론

해시 테이블(hash table)은 키와 값이 쌍으로 이루어진 데이터 집합인 연관 배열(associative array)에서 데이터 검색, 추가 및 삭제와 같은 사전적 작업을 지원하는 자료구조이다. 일반적으로 해시 테이블에서 원소를 검색하는데 수행되는 시간은 O(1)이기에, 검색 트리(search tree)를 기반으로 하는 자료구조에 비해 매우 빠른 검색을 수행한다. 이러한 해시 테이블의 효율성으로 인해 캐시(cache), 데이터베이스 인덱싱(database indexing)과 같이 빠른 검색시간을 요구하는 분야에 쓰이고 있다.

해시 테이블은 해시 함수를 이용해 원소가 저장될 버킷(bucket)의 인덱스를 계산한다. 해싱을 하는 경우 서로 다른 키 두 개가 동일한 해시값을 갖는 경우가 있다 이러한 상황을 충돌(collision)이라한다. 이러한 충돌 문제를 해결하기 위한 보편적인 방법으로는 분리연쇄법(separate chaining)이 있다. 분리연쇄법은 충돌이 일어날 경우 원소들을 연결 리스트(linked list)를 이용하여 추가한다 (그림 1). 리스트가 길어질 경우 검색 시간이 선형적으로 증가하나 해시값의 분포가 균등할 경우 최악의 경우(worst case) 수행시간이 나쁘지 않다.

본 논문에서는 이러한 해시 테이블 생성의 성능 향상을 위해 해시 테이블을 GPU상에서 병렬적으로 생성하는 방식을 제안한다. 해싱 과정 중 충돌 문제를 해결하기 위해 연결 리스트를 이용한 분리연쇄법을 사용한다.

* 포스터 발표논문
* 본 논문은 2013년도 정부(미래창조과학부)의 재원으로 한국 연구재단의 중견연구자, <실감교류 인체감응솔루션> 글로벌프론티어사업의 지원으로 수행되었음(2012R1A2A2A01045719, 2012M3A6A3055695).

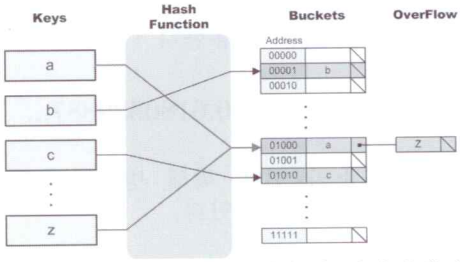

그림 1: 연결 리스트를 이용한 분리연쇄법.

2. 관련연구

2.1. GPU 해시 테이블

충돌 처리 과정은 병렬 생성 시 성능저하를 일으킬 수 있는 원인이 된다. 따라서 해시 테이블의 병렬 생성에 관한 연구는 해시값의 충돌을 최소화 하고 이를 처리하는 방식에 따라 나뉜다. Lefebvre 등[1]은 두 개의 해시 함수 및 해시 테이블과는 별도로 추가적인 오프셋 텍스처를 이용하는 완전 해시 함수를 정의하여 최소 완전 해싱(minimal perfect hashing)을 달성하였다. Alcantara 등[2]은 충돌 처리를 위해 뻐꾸기 해싱(cuckoo hasing)을 사용하였다. 전체 해시 테이블을 세 개의 보조 테이블로 나누어 사용한다. 이 기법은 대용량 해싱에는 적합하지 않은 한계를 지닌다.

2.2. GPU 연결 리스트

대부분의 GPU 알고리즘은 배열 자료구조를 사용하기에 포인터를 이용한 생성방식을 그대로 사용할 수 없다. Yang 등[3]은 이러한 GPU 특성을 고려, 정적인 자료구조에 활용 가능한 병렬 생성방식을 제안하였다. 연결 리스트 생성 시 모든 원소는 한 배열에 저장되며 병렬 처리로 인한 저장 인덱스 충돌을 방지하기 위해 원자적 연산(atomic operation)을 사용한다.

3. 알고리즘

해시 테이블의 생성은 크게 두 단계로 나뉜다. (1) 해시 함수를 사용하여 해시값을 생성, 테이블의 인덱스를 계산한 후, (2) 동일한 해시값을 가지는 원소들을 연결 리스트로 연결하여 해시 테이블을 구성한다.

3.1. 해시값 생성

해시값의 선택은 병렬 처리의 성능 향상을 위해 매우 중요한 요소이다. 연결 리스트 생성 시 저장될 노드 버퍼의 인덱스는 원자적 연산을 통해 얻게 되므로 동일한 해시값을 가지는 원소가 많을 경우 이는 성능 저하의 원인이 된다. 본 논문에서 해시 함수는 곱하기 방법(multiplication method)을 사용하며 식 1과 같다. m은 해시 테이블의 크기이며 A는 $0<A<1$ 범위를 가지는 임의의 상수이다.

$$h(k) = \lfloor m\,(k\,A \bmod 1) \rfloor \qquad (1)$$

m의 값은 임의로 결정할 수 있으나 일반적으로 2의 지수승 값을 선택한다. 해시값의 고른 분포를 위해 A는 Knuth[4]가 제안한 값을 사용한다 (식 2).

$$A \approx (\sqrt{5} - 1) = 0.6180339887... \qquad (2)$$

해시값은 모든 원소의 키에 대해 병렬적으로 계산되기 때문에 성능을 향상시킬 수 있다.

3.2. GPU 연결 리스트의 병렬 생성

본 논문에서는 Yang 등[3]이 제안한 알고리즘을 통해 해시 테이블을 생성한다 (그림 2).

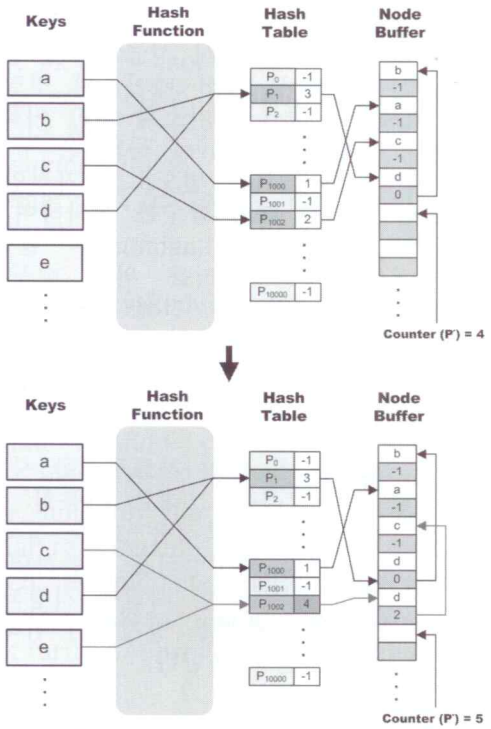

그림 2: 해시 테이블의 원소 추가[3].

해시 테이블은 테이블 버퍼와 노드 버퍼, 두 개로 이루어져 있다. 테이블 버퍼는 원소들의 연결 리스트의 첫 번째 원소를 가리킨다. 노드 버퍼는 연결 리스트를 구성하는 모든 원소들을 저장한다. 각 노드들은 원소를 구성하는 키와 값을 가지며 다음 노드를 가리키는 포인터로 이루어져있다. 원소가 저장될 노드 버퍼의 빈 공간을 가리키기 위해 전역 카운터를 사용한다.

연결 리스트를 생성하기에 앞서 전역 카운터와 해시 테이블을 초기화 한다. 테이블의 값은 리스트의 끝(EOF)을 뜻하는 -1로 초기화 하며 이는 해당 해시값을 지니는 원소가 없음을 뜻한다. 저장될 원소의 포인터를 얻기 위해 전역 카운터의 값 P을 가져오고 카운터를 1 증가시킨다. 이는 원자적 연산을 통해 동시에 이루어지며 이로 인해 각 쓰레드는 서로 다른 노드 버퍼의 포인터를 가지게 된다. 이 값을 해시 함수를 통해 계산된 해시 테이블 $h(k)$ 위치에 저장된 P_i와 교환한다. 이 역시 카운터와 마찬가지로 원자적 연산을 통해 이루어져 포인터 충돌을 방지한다. 교환된 P_i는 다음 노드를 가리키는 포인터로 저장한다. 이에 따라 테이블은 새로 저장될 노드 버퍼의 인덱스 P를 가리키게 되며, 해시 테이블 버퍼로부터 교환 된 이전 인덱스 P_i는 리스트의 다음 노드를 가리키는 포인터가 되어 노드 버퍼에 저장한다. 구성된 연결 리스트는 가장 나중에 삽입된 원소가 리스트의 가장 첫 원소가 된다.

4. 결과 및 토론

제안된 해시 테이블은 CUDA를 이용해 구현되었으며 Intel Core i7 3770, NVIDIA GeForce GTX 780 TI 환경에서 측정 되었다. 성능 측정 결과 GPU를 이용한 생성 성능이 CPU보다 매우 높음을 알 수 있다 (표 1).

표 1: 해시 테이블 생성 성능 측정 결과

원소의 수 (테이블 크기)	디바이스(블록 당 쓰레드)의 생성 시간 (ms)		
	GPU(512)	GPU(1024)	CPU
$2^{23}(2^{22})$	77.9	76.8	2469.1
$2^{23}(2^{21})$	77.4	76.3	2507.8
$2^{24}(2^{23})$	161.2	160.2	4542.6
$2^{24}(2^{22})$	164.8	162.5	4496.1

본 논문에서는 GPU를 이용한 해시 테이블 생성 알고리즘을 소개하였다. 메모리를 정적으로 할당하여 원소의 추가 및 삭제는 다소 제한이 따르기에 생성 이후 원소의 추가 및 삭제가 없는 정적인 해시 테이블에 사용할 시 효과가 좋다. 추후 공유 메모리 등을 이용해 성능을 향상 시킬 수 있으며 메모리 관리를 통해 동적 사용성을 높일 수 있다.

참고문헌

[1] Lefebvre, Sylvain, and Hugues Hoppe. "Perfect spatial hashing." *ACM Transactions on Graphics (TOG).* Vol. 25. No. 3. ACM, 2006.

[2] Alcantara, Dan A., et al. "Real-time parallel hashing on the GPU." *ACM Transactions on Graphics (TOG).* Vol. 28. No. 5. ACM, 2009.

[3] Yang, Jason C., et al. "Real-Time Concurrent Linked List Construction on the GPU." *Computer Graphics Forum.* Vol. 29. No. 4. Blackwell Publishing Ltd, 2010.

[4] Knuth, D. "The Art of Computer Programming 1: Fundamental Algorithms 2: Seminumerical Algorithms 3: Sorting and Searching." (1968).