CUDA를 활용한 병렬 B+-트리 벌크로드 기법

(A Parallel Bulk Loading Method for B+-Tree Using CUDA)

성 주 호 † 이 윤 우 †
(Jooho Sung) (Yoonwoo Lee)

한 아 † 최 원 익 ††
(A Han) (Wonik Choi)

권 동 섭 †††
(Dongseop Kwon)

요 약 대부분의 관계형 데이터베이스 시스템은 대량의 키 값을 효율적으로 검색하고 관리하기 위하여 B+-트리 기반의 인덱스 구조를 사용하며, B+-트리를 효율적으로 생성하기 위해 일반적으로 상향식 벌크로드 기법을 사용한다. 비록 벌크로드 기법이 키를 하나씩 삽입하여 인덱스를 생성하는 방식보다 효율적이긴 하지만, 데이터가 클 경우 전체 데이터를 정렬해야하기 때문에 많은 시간을 필요로 한다. 벌크로드 기법의 성능을 개선하기 위하여, 본 논문에서는 NDIVIA에서 제공하는 병렬 컴퓨팅 아키텍쳐인 CUDA를 활용한 GPU 기반의 효율적인 B+-트리 병렬 벌크로드 기법을 제안한다. 제안하는 병렬 벌크로드 기법의 성능을 증명하기 위하여 실험을 수행한 결과, 기존 CPU 벌크로드 방법보다 약 70% 이상 성능이 향상됨을 확인하였다.

· 이 논문은 2009년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2009-0076144)
· 이 논문은 제36회 추계학술발표회에서 'CUDA를 활용한 병렬 B+-트리 벌크로드 기법'의 제목으로 발표된 논문을 확장한 것임

† 학생회원 : 명지대학교 컴퓨터공학과
joohosung@gmail.com
yoonwoolee812@gmail.com
pinkey83@nate.com
†† 정 회 원 : 인하대학교 정보통신공학과 교수
wichoi@inha.ac.kr
††† 정 회 원 : 명지대학교 컴퓨터공학과 교수
dongseop@gmail.com
(Corresponding author임)

논문접수 : 2009년 12월 23일
심사완료 : 2010년 3월 5일

정보과학회논문지: 컴퓨팅의 실제 및 레터 제16권 제6호(2010.6)

키워드 : GPGPU, CUDA, B+-트리, 벌크로드

Abstract Most relational database systems provide B+-trees as their main index structures, and use bulk-loading techniques for creating new B+-trees on existing data from scratch. Although bulk loadings are more effective than inserting keys one by one, they are still time-consuming because they have to sort all the keys from large data. To improve the performance of bulk loadings, this paper proposes an efficient parallel bulk loading method for B+-trees based on CUDA, which is a parallel computing architecture developed by NVIDIA to utilize computing powers of graphic processor units for general purpose computing. Experimental results show that the proposed method enhance the performance more than 70 percents compared to existing bulk loading methods.

Key words : GPGPU, CUDA, B+-tree, bulk loading

1. 서 론

GPU(Graphic Processing Unit)는 많은 양의 계산을 처리하기 위하여 특화된 멀티 쓰레드, 멀티 코어 기반의 병렬 처리 프로세서로서 CPU에 비하여 저렴하면서도 수십에서 수백개의 코어를 내장하고 있어 데이터 처리 속도나 메모리 대역 등이 탁월한 특성을 지닌다. 최근 이러한 GPU의 성능을 게임이나 그래픽 분야 뿐 아니라 일반적인 컴퓨팅 분야에도 활용하고자 하는 연구들이 시도되고 있다. 이러한 범용 컴퓨팅용 GPU 활용을 가리켜 GPGPU(General Purpose GPU)라고 한다.

GPGPU의 등장에 따라 NVIDIA에서는 CUDA (Compute Unified Device Architecture)[1]라는 범용 병렬 컴퓨팅 아키텍처를 선보였다. CUDA는 CPU가 계산하던 복잡한 컴퓨팅 문제를 GPU로도 쉽게 처리할 수 있도록 지원하는 프레임워크이다. 이에 따라 비디오 인코더, 이미지 프로세싱 프로그램, 데이터 분석 프로그램 등 다양한 분야에서 CUDA가 활용되고 있다.

대량의 키를 효율적으로 검색, 관리하기 위하여 대부분의 관계형 데이터베이스 시스템에서는 B+-트리[2] 기반의 인덱스 구조를 사용한다. 대량의 데이터에서 새로운 B+-트리를 생성할 때, 데이터를 하나씩 삽입하게 되면 노드의 분할 및 재구성이 반복적으로 일어나게 되어 성능이 저하된다. 이를 방지하고 대량의 데이터로부터 B+-트리를 효율적으로 생성하기 위하여 일반적으로 상향식의 벌크로드(bulk load)[3] 기법이 이용된다. 새로운 인덱스를 추가할 때뿐 아니라 대량의 데이터를 추가로 삽입하거나, 삭제, 변경하는 경우 역시 인덱스의 반복적인 변경에 의한 성능 저하를 방지하고자 전체 인덱스를 새롭게 생성하기도 하며, 조인 질의 등의 처리를 위해서

도 경우에 따라 임시 데이터로부터 인덱스를 생성하기도 한다. 따라서 대량의 데이터로부터 트리를 벌크로드하는 성능은 전체적인 시스템 성능에 큰 영향을 준다. 하지만, 벌크로드를 위해서는 전체 데이터를 정렬하는 등의 많은 작업이 필요하므로, 대규모 데이터를 벌크로드 하는 경우 많은 처리 시간을 필요로 하여 성능 개선의 필요성이 존재한다.

본 논문은 B^+-트리의 벌크로드 기법의 성능을 향상시키기 위하여 CUDA를 활용한 병렬 B^+-트리 벌크로드 기법을 제안한다. 기존 벌크로드 기법의 분석 결과 대량의 데이터의 경우 데이터를 정렬하는 부분이 전체 벌크로드 수행 시간의 대부분을 차지하는 것으로 나타나, 본 논문에서는 데이터 정렬과정을 CUDA를 활용하여 병렬화한 기법을 제안한다. 제안기법의 성능을 증명하기 위하여 CPU기반의 벌크로드 기법과의 비교 실험을 수행하였으며 실험결과 제안 기법이 일반적인 벌크로드 기법보다 약 70%이상 성능이 향상되었으며, 데이터의 크기가 증가할수록 성능차이가 더욱 커짐을 알 수 있었다.

본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로서 B^+-트리의벌크로드 기법과 GPU와 CUDA에 대하여 간략히 소개하고, 3장에서는 CUDA를 활용한 B^+-트리 병렬 벌크로드 기법을 제안 한다. 4장에서는 실험결과를 보이고, 실험분석을 통해 제안하는 방법의 우수성을 증명한다. 마지막으로 5장에서는 결론 및 향후 연구 방향을 제시한다.

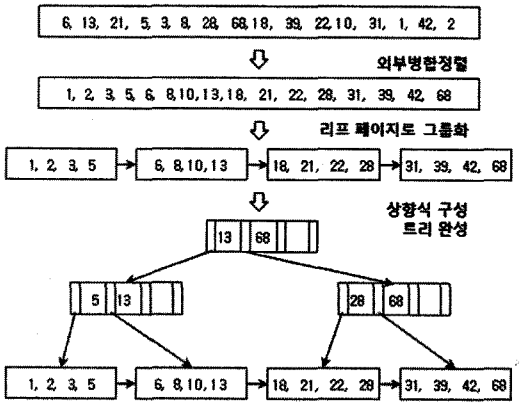

그림 1 B^+-트리 벌크로드 과정

2. 관련연구

2.1 B^+-트리의 벌크로드 기법

대량의 데이터로부터 새롭게 인덱스를 생성하는 경우, 데이터를 하나씩 삽입하는 B^+-트리 생성방법은 반복적인 노드의 분할이나 재구성이 필요하기 때문에, 인덱스를 효율적으로 생성하기 위해 데이터베이스 시스템은 일반적으로 상향식의 벌크로드 기법[3]을 이용한다. 사용자가 명시적으로 새로운 인덱스의 생성을 요청하지 않은 경우에도 전체적인 성능 향상을 위하여 데이터로부터 인덱스를 생성하는 경우가 존재한다. 예를 들어, 대량의 데이터를 추가적으로 삽입, 삭제, 변경하는 경우 성능 향상을 위하여 해당 테이블의 인덱스를 제거한 후 모든 변경이 끝난 후 인덱스를 새롭게 생성하게 된다. 뿐만 아니라 일부 복잡한 질의 처리 등에서는 질의 성능 향상을 위하여 중간 결과 등에 임시로 인덱스를 생성하여 조인 등을 처리하기도 한다. 따라서 현대 데이터베이스 시스템에서 트리의 벌크로드 성능은 전체 시스템 성능에 큰 영향을 미치게 된다.

B^+-트리의 벌크로드는 내부적으로 전체 데이터를 정렬하는 단계와 정렬된 데이터로부터 상향식으로 노드를 구성하여 트리를 완성하는 두 단계로 이루어진다. 그림 1은 벌크로드 과정을 간략히 나타낸 것이다. 우선, 전체 데이터를 정렬하고, 정렬된 데이터들을 차례로 리프 노드로 구성한다. 동시에 리프노드에 대응되는 상위 엔트리를 생성하고, 이를 통해서 차례로 상위 노드를 만들어 나간다. 동일한 방법으로 반복적으로 상위 노드를 생성해 나가면 새로운 B^+-트리를 생성할 수 있다.

2.2 GPU와 CUDA

GPU는 게임이나 그래픽 분야를 위하여 특수화된 병렬 프로세서로써, 멀티 쓰레드 멀티코어 기반으로 CPU에 비해 저렴하면서도 데이터 처리속도나 메모리 대역 등이 탁월하여 대량의 계산을 빠르게 처리할 수 있다. 예를 들어 [1)]NVIDIA의 Geforce GTX 280의 경우 CPU대비 10배 가까운 메모리 인터페이스 속도(141.7GB/sec)와 240개의 코어에서 동시에 데이터를 처리함으로써 최대 200배 이상 계산 속도를 높일 수 있다. GPU의 뛰어난 계산 성능을 그래픽이나 게임 등이 아닌 일반적인 컴퓨팅 작업에 사용하고자 하는 움직임이 계속되고 있으며 이러한 GPU의 활용을 GPGPU라고 일컫는다.

NVIDIA에서는 GPGPU를 이용한 병렬 컴퓨팅 소프트웨어를 쉽게 개발할 수 있도록 CUDA[1]라는 프로그래밍 플랫폼을 개발하였다. CUDA는 C 프로그래밍 언어를 비롯한 산업 표준 언어를 사용하여 GPU에서의 병렬 계산을 처리할 수 있도록 하며, 많은 수의 코어와 다수의 쓰레드를 효율적으로 활용하기 위한 쓰레드 관리 기법, 공용 메모리 처리, 동기화 등 병렬 프로그래밍을 위한 기본 아키텍처를 지원한다.

GPU는 SIMD(Single Instruction, Multiple Data) 형태의 병렬 처리 프로세서이므로 일반적인 CUDA의 처리과정은 4단계로 이루어진다. 1) 메인 메모리에서 처

1) http://www.nvidia.com/docs/IO/55506/GPU_Datasheet.pdf

리할 데이터를 GPU의 전역메모리로 복사한다. 2) CPU에서 GPU로 명령을 지시를 하고 3) GPU안에서 수많은 코어를 이용하여 동시에 병렬 수행한다. 마지막으로 4) GPU 글로벌 메모리에 있는 수행결과를 다시 CPU 메인 메모리로 복사한다.

초기 GPU를 활용한 데이터베이스 연구는 GPU의 복잡한 그래픽 처리 API들을 직접 활용한 연구들[4,5] 대부분이었으나, 최근에는 CUDA를 이용한 병렬화를 통하여 인덱스, 조인, 데이터마이닝 등의 데이터베이스 알고리즘들의 성능을 개선하려는 연구[6-8]가 활발히 이루어지고 있다.

3. CUDA를 활용한 B⁺-트리 벌크로드 기법

이 장에서는 GPGPU의 뛰어난 병렬 처리능력을 이용한 CUDA 기반의 B⁺-트리 병렬 벌크로드 기법을 제안한다. 3.1장에서는 일반적으로 사용되는 CPU기반의 B⁺-트리 벌크로드에서 오버헤드가 발생하는 과정을 설명하고, 개선해야 할 부분을 지적한다. 3.2장에서는 3.1장에서 논의한 문제의 과정을 개선한 GPU기반의 병렬 벌크로드 기법을 설명한다.

3.1 기존 벌크로드 기법의 약점과 개선부분

이전 장에서 논의한 바와 같이 벌크로드과정은 정렬된 페이지를 만드는 과정과, 상향식으로 진행되는 B⁺-트리 구성 과정으로 나눠진다. 그 중, 전체 데이터를 모두 비교하여 정렬하는 과정은 전체 벌크로드 시간 중 대부분의 시간을 소비한다. 특히, 데이터의 크기가 커질수록 낭비가 심해지기 때문에 대용량 데이터를 대상으로 하는 B⁺-트리 벌크로드에 가장 많은 영향을 주여 성능저하의 결정적 원인이 된다.

데이터가 많은 경우 정렬과정에는 일반적으로 외부 병합정렬이 사용된다. 외부 병합정렬은 전체 키들을 런으로 나눈 후 각 런을 정렬하는 내부정렬과, 런들을 병합하는 외부정렬로 이루어진다. 우리는 전체 벌크로드에서 가장 많은 시간을 소비하는 내부정렬을 개선하기 위해 노력하였다(그림 5의 그래프 참고). 외부 정렬과 트리구성의 병렬 처리는 전체 벌크로드 시간에서 큰 비중을 차지 않으므로 현 단계에서는 병렬화를 고려하지 않는다.

3.2 병렬 기수정렬을 이용한 병렬 벌크로드 기법

내부정렬의 병렬화로 인한 효율적인 성능향상을 위하여 CUDA에서 가장 효율적인 정렬방법으로 알려진 병렬 기수정렬(parallel radix sort)[9]을 사용하였다. 기수정렬은 키들의 각 자리 값들을 순차적으로 정렬해나 가는 방식으로써 병렬화가 용이한 특징을 지닌다. 키들의 각 자리단위의 비교과정을 병렬화 함으로써 병렬 처리 시 그 성능이 다른 병렬 정렬 기법들보다 뛰어남이 증명되었다.

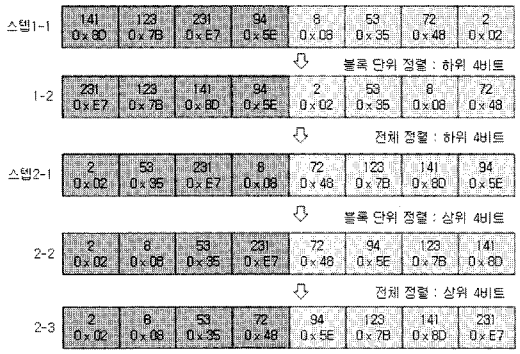

그림 2 병렬 기수 정렬 전체 과정

그림 2는 8개의 정렬되지 않은 데이터에 대한 CUDA를 이용한 병렬 기수정렬 과정을 보여준다. 이때, 4개의 쓰레드를 하나의 블록으로 가정하여, 총 2개의 쓰레드 블록이 생성되었다. 쓰레드 블록이란, 쓰레드들의 집합으로써 GPU에서 동시에 실행되는 병렬처리 단위를 말한다. 즉, 4개의 쓰레드가 한 블록으로 지정되면 그들은 같은 명령을 동시에 수행한다. 또한 한 번에 비교 가능한 데이터의 크기를 4비트로 가정하여, 8비트 데이터의 경우 2 번의 기수정렬을 수행해야만 정렬된 결과를 얻을 수 있다(32비트의 데이터는 8번의 기수정렬이 필요하다).

병렬 기수정렬은 아래의 4가지 처리과정의 반복으로 이루어진다. 각 단계들은 동기화되기 때문에, 이전 단계의 모든 작업이 종료되어야만 다음단계를 처리할 수 있다. 우선 첫 번째로, 쓰레드 블록 단위별로 각각 정렬한다. 그림 2의 스텝 1-2를 보면 스텝 1-1의 데이터들이 블록단위로 각각 하위 4비트를 기준으로 정렬된 것을 볼 수 있다. 이때 정렬된 값은 같은 쓰레드 블록만이 공유하여 접근할 수 있는 공유메모리(shared memory)에 저장되어 있다. 두 번째, 각 블록에서의 해당 값의 위치를 나타내는 블록오프셋(block offsets)과 블록내의 데이터의 개수를 의미하는 카운터(counters)를 계산한다. 이들은 각각의 공유메모리 안에 저장되어 있는 블록단위의 정렬된 데이터들을 통합하여 하나의 리스트로 정렬하기 위해 필요하다. 세 번째, 각 블록의 카운터의 합(counter sum)을 계산한다. 네 번째, 카운터의 합계와 블록 오프셋 그리고 블록 인덱스 값들을 이용하여 각 블록 안의 정렬된 데이터들을 알맞게 저장할 글로벌 메모리의 위치를 계산한다. 이렇게 계산된 위치 값을 바탕으로 각 블록에 나눠져 있는 데이터를 통합하여 하나의 정렬된 리스트를 생성한다(스텝 2-1). 동일한 방법으로 스텝 1의 결과 데이터를 다음 상위 4비트를 기준으로 한 번 더 수행하면 최종적으로 정렬된 결과를 얻을 수 있다(스텝 2-3).

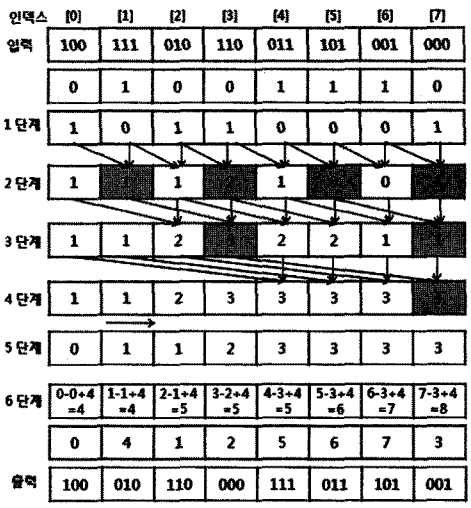

그림 3 1-bit 기반의 병렬 기수정렬 처리 과정

그림 3은 위에서 설명한 병렬 기수정렬의 4가지 처리 과정 중 첫 번째 단계인 블록 내부에서 비트단위로 정렬하는 과정을 자세히 나타낸 것이다. 한 블록에는 8개의 데이터가 있으며, 하나의 데이터는 3비트로 이루어지 값으로 제한하였다. 쉬운 이해를 위해 3비트 중 최하위 비트(가장 오른쪽 자리 비트)만을 타깃(target)으로 기수정렬을 진행한다. 이때, '0'인 비트의 개수를 계산하기 위해 타깃비트를 보수 값으로 변경하고, 다음 계산과정을 병렬로 처리한다.

7번 인덱스의 경우 2단계에서 6번 인덱스와 더해지고(1=0+1), 3단계에서는 5번 인덱스와 더해진다(1=0+1). 이때, 5번 인덱스의 값은 4,5번 인덱스가 더해진 값이다. 마지막으로 4단계에서 3번 인덱스와 더하여(4=3+1) 7번 인덱스는 자신을 포함한 이전의 모든 '0'비트의 횟수를 얻게 되었다.

이후 각 인덱스까지의 '0'비트를 스캔(Scan)한 값들은 오른쪽으로 한 칸 쉬프트(shift)하여 자신을 제외한 스캔 값으로 변경하고, 전체 '0'비트의 개수는(4) 'totalFalses' 변수로 유지한다. 마지막 6단계에서는 [인덱스 번호 - 5단계의 계산된 스캔 값 + totalFalses]를 계산하여, 1단계의 값이 0인 경우 5단계의 값을 유지하고, 1인 경우 6단계의 계산된 값으로 변경하여 최종적으로 데이터들의 정렬순서를 얻을 수 있다. 계산된 순서대로 데이터를 정렬하면 타깃비트를 기준으로 정렬된 것을 볼 수 있다.

위와 같이 인덱스들의 '0'비트 스캔과정을 CUDA로 병렬처리할 경우 1단계에서 4단계까지 총 3회의 계산으로 전체 '0'인 비트의 개수를 얻을 수 있다. 그러나 CPU로 처리하게 되면 모든 인덱스의 값을 순차적으로 계산해야하기 때문에 전체 데이터의 개수인 8회의 계산과정이 필요하다. 즉, 단일 CPU의 O(n)번의 연산을 병렬처리로써 O(log2n)로 절감한 것이다. 또한 6단계의 계산과정도 쓰레드 각각이 병렬로 처리하므로 연산 시간을 줄일 수 있다.

4. 실험결과

4.1 실험환경

제안 기법의 우수성을 증명하기 위하여, 제안 기법과 CPU 기반의 벌크로드 기법을 각각 구현하여 다양한 환경에서 성능을 비교하였다. 본 실험은 4GB 메인메모리를 지닌 Intel Core2 Quad Q8200(2.33GHz) 시스템에서 수행하였으며, GPU는 62개의 CUDA Core를 지니는 Geforce 9600GT 650MHz (512MB)를 사용했다. 병렬컴퓨팅을 위한 프로그램은 CUDA SDK 버전 2.2를 사용하였으며 Microsoft Windows XP professional SP3에서 진행되었다. 실험을 위하여 균등분포로 임의 생성된 10만개, 50만개, 100만개, 500만개, 700만개, 1000만개의 데이터를 사용하였으며, 합병정렬시 런의 크기는 30MB로 제한하였다.

본 논문에서 제안한 CUDA 기반 병렬 벌크로드 기법과 함께 CPU를 사용한 두 가지 벌크로드 기법(퀵소트 기반, 기수정렬 기반)을 함께 구현하여 비교 분석하였다. 각 알고리즘은 병합정렬의 내부정렬에 사용한 알고리즘의 이름을 따서 각각 GPU-Radix, CPU-Quick, CPU-Radix 벌크로드 기법으로 표기한다.

4.2 결과 및 분석

그림 4는 CPU-Quick 벌크로드, CPU-Radix 벌크로드, GPU-Radix 벌크로드 시간을 비교한 결과이다. 데이터의 양이 비교적 적은 10만개의 경우 CPU-Quick 벌크로드(166.5ms)와 CPU-Radix 벌크로드(243.4ms)는 GPU- Radix 벌크로드(410.5ms)보다 빠른 결과를 보였다. 그러나 데이터양이 50만개 이상인 경우 GPU-Radix 벌크로드가 CPU를 사용한 두 벌크로드보다 뛰어난 성능을 보였다. 데이터의 양이 1000만개인 경우 GPU-Radix 벌크로드는 CPU-Quick 벌크로드보다 약 75%, CPU-Radix 벌크로드보다는 약 81% 성능이 향상되었다. 즉, CPU 기반의 두 벌크로드는 데이터양이 늘어나면 벌크로드 시간도 비례하며 증가하지만, GPU-Radix 벌크로드는 보다 낮은 폭으로 증가되었음을 확인할 수 있다. 그 이유는 CUDA의 병렬처리 기술 때문에 정렬에 걸린 시간이 크게 단축되었기 때문이다.

그림 5는 데이터양이 1000만개일 때의 벌크로드 시간을 내부 정렬, 외부 정렬, 트리 구성의 세단계로 나누어 측정한 결과이다. CPU-Quick 벌크로드는 전체 벌크로드 실행시간 중 약 80%를, CPU-Radix 벌크로드는 약 86%를 내부정렬을 처리하는데 소비하였다. 그러나 GPU-

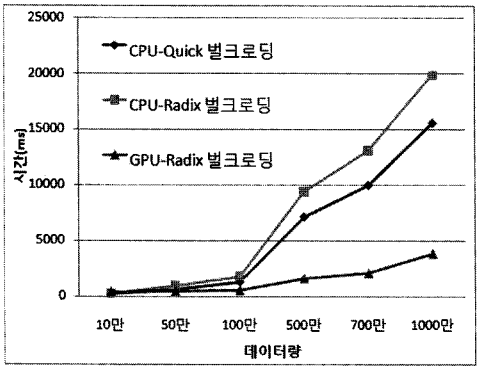

그림 4 B⁺트리 벌크로드 실행시간

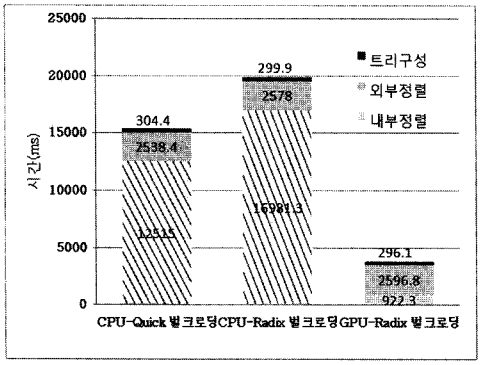

그림 5 벌크로드 단계별 실행시간

Radix 벌크로드의 경우 GPU를 활용한 병렬처리의 도입으로 내부정렬의 시간을 단축시킨 결과, 내부 정렬 단계에서만 CPU-Quick 벌크로드보다 약 93%, CPU-Radix 벌크로드보다 약 95%의 성능이 향상되었음을 알 수 있다. 즉, 결과적으로 내부정령의 성능향상으로 B⁺-트리 벌크로드의 전체적인 성능이 크게 향상되었다.

5. 결 론

대부분의 관계형 데이터베이스 시스템은 B⁺-트리 기반의 인덱스 구조를 사용하며, 이를 효율적으로 생성하기 위해 벌크로드 기법을 사용한다. 비록 벌크로드 기법이 키를 하나씩 입력하는 방법에 비하여 효율적이긴 하지만, 많은 양의 데이터를 처리하는 경우 전체 시스템 성능에 큰 영향을 미치므로 성능 개선의 필요성이 여전히 존재한다. 본 논문에서는 B⁺-트리의 벌크로드 성능을 향상시키기 위하여 GPU를 활용한 병렬 벌크로드 기법을 제안한다. 기존 벌크로드 기법을 분석한 결과 데이터를 정렬하는 부분이 병목구간이 됨을 확인하고, 이 부분을 NVIDIA에서 제공하는 CUDA 플랫폼을 활용하여 병렬화하였으며 실험 결과 데이터가 많은 경우 최대 70% 이상 성능이 향상됨을 확인하였다.

향후 본 논문에서 제안한 병렬 벌크로드 기법을 실제 데이터베이스 시스템에 적용하여 보다 구체적인 성능분석을 시행할 계획이다. 또한, R-트리와 같은 다양한 인덱스의 벌크로드에도 GPU를 활용한 병렬 처리 기법을 활용할 계획이다. 마지막으로, 벌크로드 기법의 추가적인 성능 향상을 위하여 내부 정렬 이후 단계에 대한 추가적인 병렬화 연구를 진행할 계획이다.

참 고 문 헌

[1] NVIDIA CUDA (Compute Unified Device Architecture), http://www.nvidia.com/object/cuda _home.html

[2] Comer, D., "The Ubiquitous B-Trees," *ACM : Computing Surveys*, vol.11, no.2, pp.121-137, 1972.

[3] 김상욱, 황환규, 황규영, "B⁺ 트리를 위한 벌크 로드 알고리즘", *한국정보과학회 학술발표논문집*, 제22권 제2호(A), pp.243-246, 1995.

[4] Sun, C., Agrawal, D. and El-Abbadi, A., "Hardware acceleration for spatial selections and joins," *Proc. of the 2003 ACM SIGMOD International Conference on Management of Data*, pp.455-466, 2003.

[5] Govindaraju, N., Lloyd, B., Wang, W., Lin, M. and Manocha, D., "Fast computation of database operations using graphics processors," *Proc. of the 2004 ACM SIGMOD International Conference on Management of Data*, pp.206-217, 2004.

[6] Yang, K., He, B., Fang, R., Lu, M., Govindaraju, N., Luo, Q., Sander, P. and Shi, J., "In-memory grid files on graphics processors," *Proc. of the 3rd International Workshop on Data Management on New Hardware*, pp.1-7, 2007.

[7] He, B., Yang, K., Fang, R., Lu, M., Govindaraju, N., Luo, Q. and Sander, P., "Relational joins on graphics processors," *Proc. of the 2008 ACM SIGMOD International Conference on Management of Data*, pp.511-524, 2008.

[8] Fang, W., Lu, M., Xiao, X., He, B. and Luo, Q., "Frequent itemset mining on graphics processors," *Proc. of the 5th International Workshop on Data Management on New Hardware*, pp.34-42, 2009.

[9] Satish, N., Harris, M. and Garland, M., "Designing Efficient Sorting Algorithms for Manycore GPUs," *Proc. of the 2009 IEEE International Symposium on Parallel&Distributed Processing*, pp.1-10, 2009.