캐쉬 효과를 고려한 확장된 이진 탐색 트리 알고리즘에 관한 연구

김경훈, 정균락
홍 익 대 학 교 전 자 계 산 학 과
{khkim, chong}@cs.hongik.ac.kr

A Study on Exteded Binary Search Tree Algorithms Considering Cache Effect

Kyoung-Hoon Kim Kyun-Rak Chong
Dept. of Computer Science, Hongik University

요 약

VLSI 기술의 발전에 따라 프로세서의 속도는 빠르게 증가하고 있는 반면 메모리의 속도는 이를 뒷받침하지 못하여 속도의 차이를 줄이기 위해 캐쉬(cache) 메모리를 사용하고 있다. 캐쉬가 알고리즘의 실행시간에 미치는 영향이 점점 더 커지고 있으나 이제까지 개발된 대부분의 알고리즘들은 이러한 캐쉬의 중요성을 고려하지 않고 개발되었다. 본 논문에서는 캐쉬 효과를 고려한 확장된 이진 탐색 트리 알고리즘에 관해 연구하였고, 실험을 통하여 기존의 이진 탐색 트리와 제안된 알고리즘의 성능을 비교하였다.

1. 서론

CPU성능의 비약적 발전에 따라 연산자를 처리하는 비용보다는 메모리 접근(access)에 드는 비용이 상대적으로 증가되었다. 대부분의 시스템에서는 메모리 접근에 드는 비용을 줄이기 위하여 캐쉬라는 빠른 메모리를 사용하고 있다.

캐쉬 메모리를 얼마나 효과적으로 사용하여 메모리 접근에 드는 비용을 최소화하는가의 문제는 알고리즘의 성능에 결정적인 영향을 미치지만 이전의 알고리즘들은 캐쉬 효과에 대한 고려 없이 설계되어져 왔다.

최근에 이 문제에 대한 연구가 진행되고 있는데, Lam과 Rothberg 그리고 Wolf는 블록(block)을 이용한 행렬의 곱셈 알고리즘에 대하여 연구하였다 [1]. Lamarca와 Ladner [2]는 heap에서의 캐쉬 효과에 연구하였고 이를 통하여 d-heap을 제안하였다. 또 그들은 정렬 알고리즘 [3]과 탐색과 무작위 접근 알고리즘 [4]에서의 캐쉬 효과에 대해서도 연구하였다.

본 논문에서는 캐쉬를 효율적으로 이용하기 위하여 슈퍼노드(supernode)와 블록노드 개념을 이용한 탐색 트리 알고리즘을 개발하였고, 실험을 통해 이진 탐색 트리 알고리즘과 비교하였는데 실행시간은 두 배에 가까운 향상을 보였다.

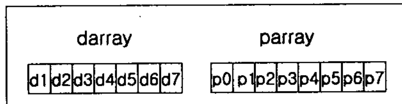
2. 슈퍼노드와 블록노드를 이용한 알고리즘

현재 SUN과 같은 컴퓨터의 메모리 구조를 보면 메인 메모리, 레벨 2 (L2) 캐쉬, 그리고 레벨 1(L1) 캐쉬로 이루어진 계층 구조를 가지고 있다. L1 캐쉬의 경우 두 개의 캐쉬로 구성되어 있다. 하나는 데이터를 위한 캐쉬이고 하나는 인스트럭션을 위한 캐쉬이다. 메모리 접근 요청 시 L1 캐쉬 , L2 캐쉬의 순서로 접근하여 요청된 메모리가 캐쉬에 있는지를 확인하고 없으면 메인 메모리로 접근하게 된다. 메인 메모리에서는 보통 64 바이트만큼의 데이터가 L2 캐쉬로 이동하게 되며 다시 32 바이트만큼이 L1 캐쉬로 이동하게 된다.

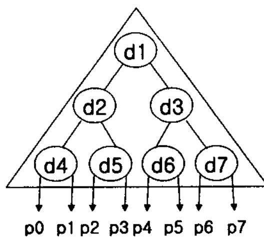
이진 탐색 트리의 경우 한 노드는 데이터와 자식을 가리키는 두 개의 포인터로 구성되어 있는데, 데이터의 크기를 4 바이트로 가정하면 64 바이트 중에서 12 바이트만을 사용하게 되어 캐쉬의 활용도를 떨어뜨리게 된다.

이러한 비효율성을 개선하려면 여러 개의 데이터를 한 노드에 저장하여야 하는데 먼저 N-슈퍼노드를 정의하기로 한다. N-슈퍼노드는 N-1개의 데이터(darray)와 N개의 포인터(parray)로 이루어져 있는데, 데이터와 포인터 모두 일차원 배열에 저장되어 있다. 데이터 배열은 완전이진 트리와 유사한 구조를 갖는데 다른 점은 중간 레

벨에 있는 노드도 자식이 없을 수 있다. 자신이 저장된 곳이 i 번째 원소이면 부모는 i/2, 왼쪽 자식은 2i, 오른쪽 자식은 2i+1 번째 원소에 저장되어 있다. 또 자신이 슈퍼노드의 마지막 레벨에 있는 노드라면 2i-N 번째 포인터가 왼쪽자식이 속해 있는 슈퍼노드를 가리키며 2i+1-N 번째 포인터가 오른쪽 자식이 속해 있는 슈퍼노드를 가리킨다. 그림 1(a)에 N=8일 때 슈퍼노드의 실제 구조가, 그림 1(b)에 트리로 표현된 구조가 나타나 있다.



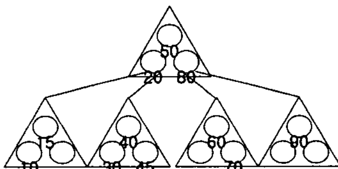
(a) 물리적 구조



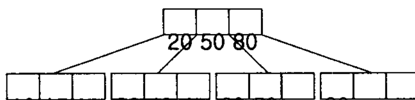
(b) 트리로 표현된 구조

[그림 1] 슈퍼노드의 구조

N-블록노드는 N-슈퍼노드와 물리적 구조는 같은데, 블록노드 안에서 데이터들은 순차적으로 정렬된 상태로 유지된다. 블록노드는 잎(leaf)이 아니면 항상 N-1 개의 데이터를 가지고 있다. 그림 2(a)에 N=4일 때의 슈퍼노드를 사용한 탐색 트리와 그림 2(b)에 블록노드를 사용한 탐색 트리의 예가 나타나 있다



(a) 슈퍼노드를 이용한 탐색 트리



(b) 블록노드를 이용한 탐색 트리

[그림 2] 확장된 이진 탐색 트리의 예

트리에 새로운 데이터 x를 삽입하고자 하는 경우 슈퍼노드를 이용한 탐색 트리에서는 첫 번째 슈퍼노드에서 n=1이라 하고 darray[n] 값과 비교한다. x가 작으면 darray[2n] 값과 비교하고 크면 darray[2n+1]의 값과 비교한다. 이 때 darray[2n]이나 darray[2n+1]이 비어 있으면 x를 그 곳에 저장한다. 만약 2n이나 2n+1이 N보다 크거나 같으면 parray[2n-N] 또는 parray [2n+1-N]이 가리키는 슈퍼노드로 이동하여 같은 방법을 반복하게 된다.

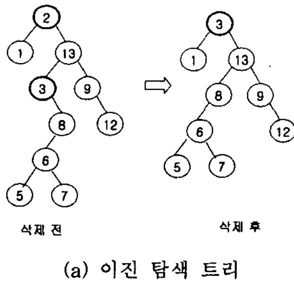
블록노드를 이용한 탐색 트리에서는 먼저 블록노드에 비어 있는 원소가 있으면 x를 이 블록노드에 삽입한다. 이 때 블록노드에 있는 다른 데이터와 정렬된 상태가 되게 삽입한다. 비어 있지 않으면 darray[n] 값과 비교하여 x가 작으면 parray[n-1]이 가리키는 블록노드로 이동하고 크면 darray[n+1] 값과 비교한다. 만약 n+1이 N과 같다면 parray[n+1]이 가리키는 블록노드로 이동하면 된다. 블록노드를 이용한 탐색 트리에서는 메모리 공간을 낭비하지 않기 위하여 블록노드 내에서의 데이터 이동이 발생한다.

탐색 알고리즘은 삽입 알고리즘과 유사하게 위 수행되어진다.

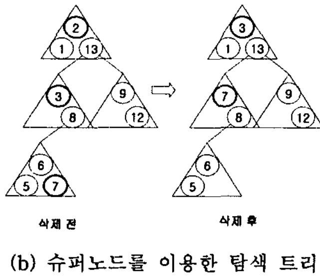
슈퍼노드와 블록노드를 이용한 탐색 트리의 경우 삭제 연산은 이진 탐색 트리의 경우보다 훨씬 복잡한 데이터의 이동이 발생한다. 이진 탐색 트리의 경우에는 삭제할 데이터를 포함하는 노드를 찾아낸 후 오른쪽 자식 노드를 루트로 하는 서브트리에서 최소 값 또는 왼쪽 자식 노드를 루트로 하는 서브 트리에서 최대 값을 갖는 노드를 찾는다. 그 다음 이 노드의 데이터 값을 삭제할 데이터를 포함하는 노드로 이동시키고 이 노드를 제거한 후 삭제된 노드의 부모 노드가 삭제된 노드의 자식 노드를 가리키도록 포인터를 수정하면 된다. 하지만 슈퍼노드를 이용한 탐색 트리의 경우는 자식 노드를 루트로 하는 서브 트리에서 최소 값을 갖는 노드 혹은 최대 값을 갖는 노드를 찾아내어 이를 삭제할 데이터를 포함하는 노드의 데이터 값으로 이동시킬 때 슈퍼노드의 중간에 빈 원소가 생길 수 있다. 이 경우에는 다시 이 노드에서부터 위의 과정을 잎(leaf) 노드에 도달할 때까지 반복하여야 한다.

그림 3은 이진 탐색 트리와 슈퍼노드를 이용한 탐색 트리에서 루트에 있는 데이터 '2'를 삭제할 때의 데이터의 이동 과정을 보여준다. (a)의 경우 '2'를 포함하고 있는 노드의 오른쪽 자식 '13'을 루트로 하는 서브트리에서 최소 값 '3'을 찾아서 이 값을 삭제할 데이터를 포함하는 노드로 이동시키고 '13'의 왼쪽 포인터가 '3'의 오른쪽 자식 '8'을 가리키도록 하면 된다. 하지만 (b)의 경우에는 '3'을 이동시킨 후 '3'의 오른쪽 자식 '8'의 왼쪽 서브트리에서 최대 값인 '7'을 '3'의 자리로 다시 이동시켜야 한다.

자세한 삽입, 탐색, 삭제 알고리즘은 지면 관계상 생략하기로 한다.



(a) 이진 탐색 트리

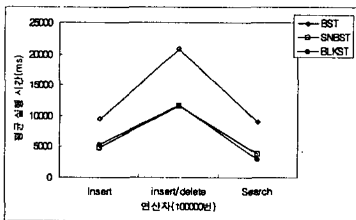


(b) 슈퍼노드를 이용한 탐색 트리

[그림 3] 데이터 삭제

3. 실험 결과

제안된 알고리즘은 C 언어를 사용해서 구현되었고 SUN 서버 엔터프라이즈 3000에서 실험되었다. 데이터 크기가 100만인 경우의 실험 결과가 그림 4에 나타나 있다. 그림 4는 이진 탐색 트리(BST)와 N=8인 슈퍼노드를 이용한 탐색 트리(SNBST) 그리고 N=8인 블록노드를 이용한 탐색 트리(BLKST) 알고리즘에서의 삽입, 삽입/삭제, 그리고 탐색 연산의 평균 실행 시간을 나타낸다. 두 방법 모두 이진 탐색 트리 알고리즘 보다 성능이 월등히 향상되었다.



[그림 4] 평균 실행 시간

슈퍼노드를 이용한 트리의 경우 삽입 연산에서 블록노드를 이용한 트리에서의 삽입 연산 보다 성능이 좋아지는데 이는 블록노드를 이용한 트리에서 삽입 시에 블록 내에서의 데이터의 이동이 발생하기 때문이다. 탐색 연산의 경우에는 블록노드를 이용한 탐색 트리의 성능이 나아지는데 이는 슈퍼노드를 이용한 탐색 트리는 하나의 슈퍼노드에서 탐색되어지는 평균 데이터 수가 $\log_2 N$인데 반하여 블록노드를 이용한 탐색 트리는 하나의 블록노드에서 탐색되어지는 평균 데이터 수는 N/2으로 N>4인 경우 슈퍼노드를 사용하는 경우 보다 블록 활용율이 높기 때문이다.

[표 1] 노드의 수와 메모리 크기

	노드의 크기	노드의 수	메모리 크기
BST	12bytes	1,000,000	12M
SNBST	64bytes	333,139	21M
BLKST	64bytes	280,465	18M

표 1은 각각의 트리에서의 노드의 크기와 백만 개의 데이터를 저장하고자 할 때 필요한 노드의 수를 나타낸다. 노드의 수에 노드의 크기를 곱하면 필요한 메모리 공간의 크기가 된다. 슈퍼 노드를 이용한 트리의 경우 모든 슈퍼노드는 데이터를 포함하지 않는 필드를 가질 수 있다. 블록노드를 이용한 트리의 경우에는 마지막 레벨에 있는 블록노드만이 데이터를 포함하지 않는 필드를 가진다. 그러므로 슈퍼노드나 블록노드를 이용한 트리는 이진 탐색 트리보다 많은 양의 메모리 공간을 사용한다.

4. 결론

메모리 접근에 드는 비용이 커짐에 따라 캐쉬를 얼마나 효율적으로 사용하느냐의 문제는 프로그램의 실행시간에 결정적인 영향을 준다.

본 논문에서는 슈퍼노드와 블록노드를 이용한 확장된 이진 탐색 트리 알고리즘을 개발하였고, 실험을 통하여 이진 탐색 트리보다 성능이 향상됨을 보였다.

5. 참고문헌

[1] M. S. Lam and E. E. Rothberg. The Cache Performance and Optimizations of Blocked Algorithms. Conf on Architectural Support for Programming Languages and Operating Systems (ASPLOS IV), pages 63-74, 1991

[2] A. LaMarca and R. E. Ladner. The Influence of Caches on the Performance of Heaps. Journal of Experimental Algorithmics, Vol 1, Article 4, 1996

[3]. A. Lamarca and R. E. Ladner. The Influence of Caches on the Performance of Sorting . In Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 370-379, 1997

[4] R. E. Ladner , J. D. Fix and A. LaMarca. Cache Performance Analysis of Traversals and Random Accesses. In Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 613-622, 1999