연결 리스트 트리의 배열 트리 변환에 관한 연구

신동영*, 박준석*
*인하대학교 컴퓨터정보공학과
e-mail : nuclear6@naver.com
joonseok@inha.ac.kr

A Study on Tree Transformation from Linked List Tree to Array Tree

Dongyoung Shin*, Joonseok Park*
*Dept of Computer Information Engineering, In-Ha University

요 약

트리의 검색은 어플리케이션에서 흔하게 사용되는 연산중 하나이다. 하지만 대부분의 트리는 연결 리스트 기반으로 생성되며 연결 리스트 트리 구조는 데이터의 지역성을 가지기 힘들기 때문에 트리구조의 검색을 동반한 응용은 캐시메모리 사용효율의 제약으로 인해 성능상의 문제점이 존재한다. 본 논문에서는 연결 리스트 트리를 배열 기반의 트리로 변형하여 트리 검색 시 성능을 향상시킬 수 있는 방법을 제시한다.

1. 서론

트리의 검색은 컴퓨터 응용 프로그램에서 매우 흔하게 사용되는 연산 중 하나이다. 데이터베이스의 검색이나 [1,2], n-체 문제(n-body problem)[3], 유전자의 염기서열 검색[4], 단백질을 이루는 아미노산 서열 검색[5, 6] 등 그 활용 분야는 다양하다. 트리를 구성하는 대표적 방법으로 연결 리스트를 사용하는 방법이 있다. 하지만 연결 리스트 트리를 사용할 경우 데이터의 지역성을 보장하기 힘들다. 지역성을 보장하지 못하는 응용의 경우, 캐시메모리의 효율적 사용이 어렵게 되고, 그 결과로 성능상의 제약이 뒤따르는 경우가 일반적이다. 연결 리스트로 구성된 데이터를 사용하는 응용 프로그램의 성능을 개선하는 방법에는 FPGA를 사용하여 사용할 데이터 구조를 효율적으로 재구성하는 방법[7] 등이 있다. 하지만 FPGA를 사용하는 방법은 추가적인 하드웨어를 필요로 하므로 트리 검색의 성능을 개선하고자 할 경우 검색할 데이터의 지역성을 향상 시키는 것이 한 방법이 될 수 있다. 본 논문에서는 트리를 DFS(Depth First Search)로 검색할 경우 데이터 지역성을 향상 시킬 수 있는 배열 기반 트리의 구조와 생성 방법에 대하여 기술한다. 본 논문의 구성은 다음과 같다. 2장에서 연결 리스트 트리를 배열트리로 변환하기 위한 데이터 타입의 변경과 배열 트리를 생성하는 방법을 서술하며 3장에서는 연결 리스트 트리를 DFS로 검색하는 경우와 배열 트리를 DFS, BFS(Breath First Search)로 검색하는 경우를 각각 비교한다. 4장에서는 결론과 향후 연구방향에 대해 기술한다.

2. 연결 리스트 트리의 배열 트리 변환

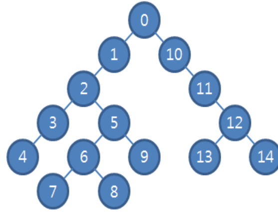
본 단락에서는 연결 리스트 트리를 배열 트리로 바꾸는데 필요한 데이터 타입의 변경에 대하여 서술하고, 변경된 데이터 타입에 저장 될 내용들을 계산하면서 연결 리스트 트리를 배열 트리로 변환하는 방법을 설명한다.

2.1. 데이터 타입 변경

연결 리스트 트리를 배열 트리로 변환하기 위해서는 연결 리스트 트리의 노드들을 연결하는 링크에 대한 정보를 배열 안에 저장해야 한다. 본 논문에서는 연결 리스트 트리의 노드 데이터 타입 중 다른 노드를 가리키는 포인터 변수를 배열의 인덱스를 가리키는 정수형의 변수로 대체하는 방법을 사용한다. 연결 리스트 트리를 구성하는 노드의 크기는 36바이트이다. 하나의 노드는 노드의 종류를 나타내기 위한 정수형 변수와 노드가 포함하는 사각형의 범위를 나타내기 위한 네 개의 정수형 변수를 가지며, 다른 노드를 가리키는 포인터 변수 4개를 추가로 포함한다. 배열 트리의 노드는 60바이트 크기를 가진다. 연결 리스트 트리를 구성하는 노드에 자식 노드의 인덱스를 나타내기 위한 정수형 변수 네 개를 추가하며, 노드 자신의 인덱스와 부모 노드의 인덱스를 저장하는 정수형 변수 두 개를 추가한다.

배열 상에서 다른 노드를 가리키는 인덱스는 절대 주소를 사용한다. 트리의 루트 노드는 배열의 첫 번째 노드에 저장되고 인덱스는 0이다. 임의의 A 노드가 B 노드의 루트 노드이고 배열 상에서 두 노드가 10개의 노드만큼 떨어져 있다고 가정하자. 이때 상대적 주소를 사용하면 A는 B를 가리키는 인덱스로 10을 저장한다. 절대적 주소를 사용할 경우 A 노드는 자신의 인덱스에 10을 더하여 B를

가리키는 인덱스로 저장한다. 그림 1은 절대적 주소 지정 방법과 상대적 주소 지정 방법을 이용하여 배열 상에서 다른 노드를 가리키는 인덱스를 계산한 예이다. 그림 1에 나타낸 트리를 배열 트리로 변환할 경우 각 트리의 노드는 노드에 적혀있는 숫자를 인덱스로 하여 배열 트리에



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
절대적 주소 지정 방법	1, 10	2	3, 5	4	Null	6, 9	7, 8	Null	Null	Null	11	12	13, 14	Null	null

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
상대적 주소 지정 방법	1, 10	1	1, 3	1	Null	1, 4	1, 2	Null	Null	Null	1	1	1, 2	Null	null

(그림 1) 절대 주소 지정 방법과 상대 주소 지정 방법

사상된다. 절대적 주소 지정 방법은 자식 노드를 가리키는 인덱스로 배열 트리의 루트 노드로부터 자식노드까지의 거리를 사용한다. 배열트리로 사상된 8번 노드는 루트와의 거리가 8이고 6번 노드는 자식 노드의 인덱스로 8을 저장한다. 상대적 주소 지정 방법은 배열 트리 상에서 자신과 자식 노드의 거리를 인덱스로 저장한다. 예를 들어 6번 노드는 8번 노드와 2만큼 떨어져 있다. 따라서 자식노드를 가리키는 인덱스로 2를 저장한다.

2.2. 배열 트리 생성

배열 트리는 연결 리스트 트리를 DFS 방식으로 검색하면서 구성한다. 하위 노드로 진행할 때마다 새로운 노드를 추가하고 인덱스를 하나씩 증가 시킨다. 증가시킨 인덱스는 새로 생성한 노드의 인덱스를 나타낸다. 증가시킨 인덱스와 노드의 데이터는 새로 생성된 노드에 저장한다. 트리 노드가 자식 노드를 가지고 있을 경우 현재 노드의 인덱스에 1을 더하여 자식 노드를 가리키는 인덱스로 저장하고, 자식 노드에서 같은 과정을 반복한다.

그림 2의 의사코드을 사용하여 NODELIST를 헤드 노드로 하는 1차원 연결 리스트를 생성하고 연결 리스트의 노드들을 역순으로 저장하면 배열 구조의 트리가 생성된다. 저장할 노드의 개수는 idxCnt 변수의 값으로 알 수 있다.

배열 트리를 생성한 경우 임의의 노드로부터 배열 트리를 순차적으로 검색하는 연산은 트리를 DFS 방법으로 검색하는 것과 동일한 효과를 가진다.

3. 성능 측정 및 평가

```
void arrayTree(tree_node* tn, int parent)
{
 tree_node* res;
 idxCnt++;

 res = new_tree_node();
 res->Parent = parent;

 copyOfData(res, tn);

 if(NODELIST == NULL)
  NODELIST = res;
 else
  {
   res->nodePtr = NODELIST;
   NODELIST = res;
  }

 for(int i=0; i<NODE_SIZE; i++)
  if(tn->ptr[i] != NULL)
   {
    res->arrIdx[i] = idxCnt+1;
    arrayTree(tn->ptr[i], res->nodeNum);
   }
}
```

(그림 2) 배열 트리 생성 의사코드

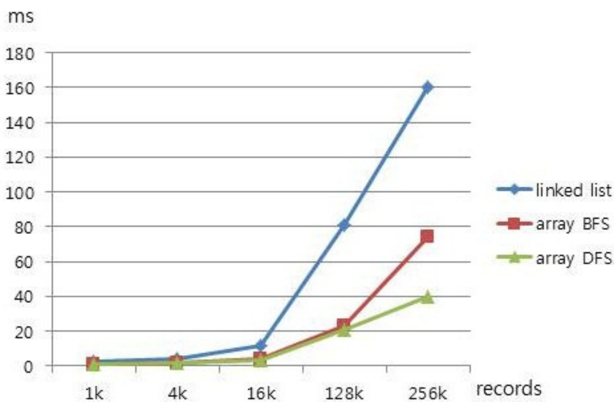
트리의 생성이나 수정이 빈번한 경우 배열 트리는 많은 수의 데이터 삭제나 복사, 이동 연산을 수행해야 한다. 이러한 연산들에 소요되는 시간이 트리 검색을 효율적으로 수행함으로써 얻을 수 있는 성능 개선 효과보다 크게 되면 전체적인 프로그램의 성능을 저하시킬 수 있다. 따라서 본 논문은 트리의 생성이나 수정보다 검색 연산이 빈번한 경우에 초점을 둔다.

3.1. 응용프로그램 소개

수정이나 변경이 적고 검색이 빈번한 연산의 대표적인 예로 DNA 염기 서열 검색, 단백질의 아미노산 서열 검색 등을 들 수 있다. 이러한 검색의 특징은 대용량의 데이터베이스를 질의에 따라 검색 하는 것이다. 대용량 데이터베이스의 크기는 보통 CPU의 캐시 크기보다 크므로 한번 사용 되었던 데이터가 다시 사용될 확률이 적다. 데이터 검색 초기에 캐시 미스가 발생하는 상황을 반영하기 위하여 질의의 시작 전에 캐시에 불필요한 값들을 채우고 데이터베이스를 읽어 오도록 하여 성능을 측정 하였다. 본 논문에서 사용하는 응용프로그램은 질의가 가진 공간 데이터 범위 내의 노드들을 검색하는 작업을 수행한다. 공간 데이터를 표현하는 방법에는 비조밀망(Sparse Mesh), 쿼드트리(Quad Tree)[8]등이 있다. 본 논문에서는 포인터 기반의 쿼드트리를 사용한다. 쿼드트리는 내부 노드(internal node)와 잎사귀 노드(leaf node)로 나뉜다. 내부 노드는 사각형의 공간 범위를 가지며 다른 내부 노드나 잎사귀 노드를 자식 노드로 가질 수 있다. 내부 노드는 자신의 범위를 다시 네 개의 공간 범위로 나눌 수 있

으며 각각의 범위는 자식 내부 노드의 범위가 되거나 잎사귀 노드를 포함하는 범위가 된다. 검색은 루트노드부터 시작하며 트리의 노드가 포함하고 있는 공간 범위 데이터를 검사한다. 질의가 포함하는 범위와 내부 노드가 가진 범위가 겹치는 부분이 있을 경우 하부 트리를 재귀적으로 검사하며 질의의 범위 내에 있는 잎사귀 노드에 도달하면 해당 노드의 번호를 출력한다. 만약, 질의가 포함하는 범위와 내부 노드가 가진 범위가 겹치는 부분이 없다면 그 하부 트리의 모든 노드들은 질의가 포함하는 범위와 겹치지 않는다. 따라서 더 이상 범위가 겹치지 않는 노드의 하부 트리로 검색을 진행하지 않는다.

3.2 성능 측정



(그림 3) 연결 리스트 트리와 배열 트리의 검색 성능 비교

그림 3의 가로축은 실험을 하는데 사용한 데이터베이스의 크기를 나타낸다. 각각의 숫자는 해당 트리를 구성하는데 사용한 레코드의 개수를 나타낸다. 트리의 잎사귀 노드들은 각 레코드에 대한 포인터나 정보를 저장한다. 트리를 구성하는 전체 노드의 개수는 레코드의 개수와 범위에 따라 달라지며 실험에 사용된 데이터는 표1에 나타낸 개수만큼의 노드를 생성한다. 연결 리스트 트리를 구성하는 노드와 배열 트리를 구성하는 노드의 개수는 같다. 세로축은 트리 검색을 완료할 때까지의 경과 시간을 나타내며 단위는 밀리 초(ms) 이다. 검색할 트리의 크기가 커짐에 따라 연결 리스트 트리의 DFS 검색 시간은 점점 증가한다. 배열 트리를 BFS 방식이나 DFS 방식으로

<표 1> 트리를 구성하는 레코드 크기에 따른 트리 노드의 개수

	1k	4k	16k	128k	256k
노드 수	1,780	7,066	28,196	225,666	450,716
내부 노드	756	5,018	11,812	94,594	188,572
잎사귀 노드	1,024	2,048	16,384	131,072	262,144
데이터 크기 (kb)	104	414	1.652	13,222	26,409

검색할 경우에도 검색 시간이 점점 증가하지만 트리의 크기가 커질수록 연결 리스트 트리와의 검색 시간 차이가 커짐을 볼 수 있다. 배열 트리는 DFS 검색에 적합하도록 구성되어 있다. 하지만 BFS 방식으로 트리의 검색을 진행하더라도 연결 리스트로 구성된 트리보다 더 나은 성능을 나타낸다. BFS 방식으로 트리를 검색할 경우 128K개의 레코드를 사용하여 구성한 배열 트리의 검색 시 까지 DFS와 유사한 성능을 나타낸다. 이는 메모리상의 트리 노드를 접근할 경우 캐시 라인 크기만큼 데이터를 캐시 메모리로 복사하고, 데이터가 캐시 메모리에서 쫓겨나기 전에 재사용되기 때문에 128K크기까지의 배열 트리에서 DFS와 BFS가 유사 한 성능을 보인다. 256K 이상의 레코드를 사용하여 트리를 구성하는 경우 배열 트리 상에서 DFS는 BFS의 두 배에 가까운 성능을 나타내며 같은 크기의 연결 리스트 트리를 DFS로 검색할 경우와 비교하여 배열트리의 DFS 검색은 4배에 가까운 성능 개선 효과를 나타낸다.

표 2의 실험 결과에서 L1 데이터 캐시와 L2 캐시의 미스가 발생 횟수가 배열 트리를 사용한 DFS 검색일 때 가장 적은 것을 확인할 수 있다. 또한, 표 2와 그림 3으로부터 L1 캐시의 미스 발생은 배열 트리의 BFS 검색이 연결 리스트 트리보다 더 많지만 L1 캐시 미스는 L2 캐시의 미스보다 페널티가 적기 때문에 배열 트리의 BFS 검색이 더 좋은 성능을 나타낼 수 있음을 알 수 있다. 실험은 Intel Core2Quad Q9400 2.66GHz, 4기가바이트 메인 메모리, Windows vista 환경에서 수행하였다.

<표 2> 연결 리스트 트리와 배열 트리의 검색 시 발생하는 캐시 미스 횟수 비교 (단위 : 백만)

	Array DFS	Array BFS	Linked list DFS
L1 cache data miss	249	401	356
L2 cache miss	74.8	104	204.8

4. 결론 및 향후 연구 방향

트리의 검색 연산은 활용도가 높은 연산 중에 하나이다. 트리 구조의 생성이나 수정이 적고 한번 구성한 데이터를 반복적으로 검색하는 경우라면 트리 생성이나 수정 시간이 조금 증가하더라도 효율적으로 검색할 수 있는 데이터의 구조를 작성하는 것이 바람직하다. 본 연구에서는 데이터 지역성을 저해하는 연결 리스트 트리를 배열 트리로 변환함으로써 데이터의 지역성을 충분히 활용할 수 있도록 하였다. 배열 트리 상에서 작은 크기의 데이터베이스를 검색할 경우에 BFS와 DFS검색 모두 성능상의 차이가 크지 않지만, 데이터의 크기가 커짐에 따라 2배에서 4배까지의 성능개선 효과를 보였다.

배열 트리는 DFS 검색 시 높은 데이터 지역성을 가지며 이는 하나의 트리를 여러 개의 하부 트리(sub tree)로

나누어 검색할 경우 효과적인 성능 향상을 가져올 수 있음을 시사한다. 연결리스트로 구성된 트리의 경우 병렬프로세싱에 의한 성능효과를 기대하기 어려우나, 본 연구와 같이 배열기반 트리로 변형할 경우 병렬 처리의 가능성 역시 열리게 된다. 따라서 향후 GPGPU[9] 및 멀티코어 시스템에서의 병렬 쓰레드를 활용하여 검색 성능을 향상시키는 연구를 진행하고자 한다.

사사

이 논문은 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단의 기초연구사업 지원을 받아 수행된 것임 (2011-0024909)"

참고문헌

[1] K. Kailing, H. -P. Kriegel, S. Schonauer, T. Seidl, "Efficient similarity search in large databases of tree structured objects", Data Engineering, Proceedings, 20th international Conference, 2004

[2] S. E. Rouwhorst, A. P. Engelbrecht, "Searching the forest: using decision trees as building blocks for evolutionary search in classification databases ", Evolutionary Computation, Proceedings of the 2000 Congress, 2000, vol. 1, pp. 633-638

[3] Lars Nyland, Mark Harris, Jan Prins "GPU Gems3, Fast N-Body Simulation with CUDA", NVIDIA, pp. 677-695.

[4] L. Duret, S. penel, M. Gouy, F. Rechenmann, G. Perriere, "Tree pattern matching in phylogenetic trees: automatic search for orthologs or paralogs in homologous gene sequence databases ", Bioinformatics, Jun. 2005, 21(11):2596-2603

[5] Sung W. Shin, Sam M. Kim, "A new algorithm for detecting low-complexity regions in protein sequences", Bioinformatics, vol. 21, Issue 2, Jan. 2005.

[6] S. B. Needleman, C. D. Wunsch "A general method applicable to the search for similarities in the amino acid sequence of two proteins", J Mol Biol, vol. 48, 1970, pp. 443-453.

[7] Pedro C. Diniz, Joonseok Park "Data Search and Reorganization Using FPGAs : Application to Spatial Pointer-based Data Structures", In Proc. of IEEE Symp. on Field-Programmable Custom Computing Machines (FCCM'03), Apr. 2003.

[8] J. Smith and S. Chang, "Quad-Tree Segmentation for Texture-Based Image Query", In Proc. of the 2nd Annual ACM Multimedia Conference, San Francisco, Ca., Oct. 2001.

[9] "NVIDIA CUDA C Programming Guide", version 3.1.1, NVIDIA, 2010.