소괄호 묶기 방법을 이용한 희소 행렬 연쇄 곱셈의 최적화

김정석, 성원용
서울대학교 전기공학부
e-mail : kimjs@dsp.snu.ackr, wysung@snu.ac.kr

Optimization of Sparse Matrix Chain Multiplication by using Parenthesizing method

Jungsuk Kim, Wonyong Sung
School of Electrical Engineering
Seoul National University

Abstract

Matrix chain multiplication is an important computational kernel used in many applications such as scientific computation, signal and image processing. In this paper, sparse matrix chain multiplication is optimized by using dynamic programming method. With the proposed method, total execution time is reduced by 33 %.

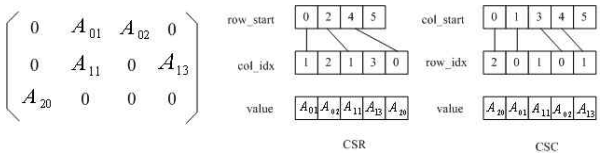
I. 서론

희소 행렬(sparse matrix)은 물리 연산, 신호 처리 등에서 많이 사용되고 있다. 본 논문에서는 희소 행렬 연쇄 곱셈의 최적화 방안을 연구하였다. 동적 프로그래밍(dynamic programming)을 이용하여 곱셈 연산 횟수와 메모리 접근 횟수를 최적화 하였다. 그 결과 5개의 희소 행렬 연쇄 곱셈에 대하여 메모리 접근 횟수는 평균 34.9%, 정수 곱셈 횟수는 15.6%가 감소하였고, 실행 속도는 33% 향상되었다.

II. 희소 행렬 연쇄 곱셈

희소 행렬은 내부에 0인 인자를 많이 가지는 특징을 가지고 있다. 일반적인 행렬 곱셈 방법을 이용하여 행렬 곱셈 연산을 수행할 경우, 0인 인자와의 곱셈은 불필요하다. 따라서, 희소 행렬 곱셈을 효율적으로 수행하기 위한 특별한 표현 방법을 사용한다.

희소 행렬의 표현 방법은 다음 [그림1]과 같은 Compressed Sparse Row (CSR) 방법과 Compressed Sparse Column (CSC) 방법이 있다 [1].



[그림 1] CSR 및 CSC 표현 방법

CSR과 CSC 표현 방법은 각각 3개의 배열을 가지며, start 배열에는 새로운 행, 열이 시작되는 배열번호를 저장한다. idx 배열은 0이 아닌 값을 가지는 인자의 열, 행 번호를 저장한다. value 배열은 0이 아닌 인자를 각각 행, 열 방향으로 저장한다.

CSR 방법은 행 방향, CSC 방법은 열 방향으로 0이 아닌 인자에 접근할 때 효율적이다. AB의 희소 행렬 곱셈에서 행렬 A는 CSR, 행렬 B는 CSC 표현 방법으로 나타내었을 경우, 행렬 곱셈을 효율적으로 수행 할 수 있다.

희소 행렬의 연쇄 곱셈에서 행렬 곱셈의 순서와 표현 방법에 따른 최적화가 가능하다. 행렬 곱셈의 순서와 중간 결과 값의 표현 방법에 따라, 연산 횟수와 메모리 접근 횟수를 최적화 할 수 있다.

III. 최적화 방안

일반적인 연쇄 행렬의 최적화는 행렬의 곱셈에서 결합법칙이 성립하는 점을 이용하여 최소의 곱셈 횟수를 가지는 계산 순서를 구하는 것으로 이루어진다. 연쇄

행렬 곱셈 $A_iA_{i+1}...A_j$에 대한 최적 부분구조를 재귀적 관계로 나타내면 다음과 같다.

$$m[i,j]=\begin{cases}0 & \text{if } i=j \\ \min\limits_{i\le k\le j} m[i,k]+m[k+1,j]+p_{i-1}p_kp_j & \text{if } i<j\end{cases}$$

$m[i,j]$는 희소 연쇄 행렬 곱셈 $A_iA_{i+1}...A_j$의 최소 곱셈 횟수를 의미한다. $p_{i-1}p_kp_j$는 $(A_iA_{i+1}...A_k)\times(A_{k+1}...A_j)$의 곱셈 연산 횟수이다. 일반적인 연쇄 행렬의 곱셈은 위의 최적 부분 구조를 가지고 최적화가 가능하다.

그러나, 희소 행렬의 경우, 0이 아닌 인자 개수에 따라 곱셈 회수가 결정되기 때문에 위의 최적 부분 구조를 이용하여 최적화할 수 없다. 희소 행렬 연쇄 곱셈의 경우, 인자의 행과 열 번호를 비교하여 정확한 곱셈 횟수를 계산 할 수 있으나, 이 경우 행렬 A와 B의 0이 아닌 인자를 모두 비교하여 연산 횟수를 계산한다. 이것은 희소 행렬 곱셈에 준하는 복잡도를 가진다. 따라서, 보다 간단하게 곱셈 횟수를 예측하여야 한다. 희소 행렬 A와 B의 0이 아닌 인자의 수를 각각 z_A, z_B라고 할 때, 희소 행렬 곱셈 AB의 곱셈 횟수는 다음과 같이 예측 할 수 있다.

$$m_An_An_b\times\max\left(\frac{z_A}{m_An_A},\frac{z_B}{m_Bn_B}\right)$$

위의 예측을 이용하여 희소 연쇄 행렬 곱셈 $A_iA_{i+1}...A_j$의 최적 부분 구조의 재귀적 표현은 다음과 같다.

$$m[i,j]=\begin{cases}0 & \text{if } i=j \\ \min\limits_{i\le k\le j} m[i,k]+m[k+1,j] & \\ \qquad +p_{i-1}p_kp_j\times\max(z_{i,k},z_{k+1,j}) & \text{if } i<j\end{cases}$$

최적 부분 구조를 이용하면 최소의 곱셈 횟수를 가지는 희소 행렬 곱셈의 곱셈 순서를 찾을 수 있다. 한편, 행렬 A의 0이 아닌 인자의 개수가 적을수록 메모리에 접근 하는 횟수가 줄어들게 된다. 위에서 설명한 최적화 구조를 사용하여, 행렬 $z_A \ge z_B$ 의 조건에 맞도록 구현 하는 것이 가능하다. 이를 통하여 메모리 접근 횟수를 감소시킬 수 있다.

동적 프로그래밍을 통하여 위의 최적화 구조를 구현한 알고리즘은 다음과 같다.

```
1 n ← length[p]-1
2 for I ← 1 to n
3       do m[i,i]←1
4 for l ← 2 to n
5     do for i←1 to n-l+1
6           do j←i+l-1
7           m[i,j]←∞
8           for k←i to j-1
9               do for w←i to k
10                  avg1←avg1+z_w/m_w n_w
11              for w←k+1 to j
12                  avg2←avg2+z_w/m_w n_w
13                  q←m[i,j]+m[k+1,j]
                        +p_{i-1}p_kp_j×max(avg1,avg2)
14                  if q<m[i,j]
15                  then m[i,j]←q
16                       s[i,j]←k
17 return m and s
```

IV. 실험 결과 및 결론

Intel core2 Quad 2.4GHz 시스템에서 gcc를 사용하였다. 실험에 사용한 희소 행렬 5개의 크기는 500x500 이고, 0이 아닌 인자는 각각 1118, 4322, 2791, 2245, 1482 이다. 행렬을 임의의 순서로 나열한 후 순차적인 곱셈과 최적화된 곱셈을 수행하였다. 실험 결과는 [표 1]과 같다.

	direct	optimized	성능향상
메모리 접근 횟수	35,047,049	22,473,440	34.98%
곱셈 횟수	198,959	167,327	15.56%
수행 시간	0.092777	0.071643	33.22%

[표 1] 실험 결과

메모리 접근 횟수의 경우 순차적으로 희소 행렬 연쇄 곱셈을 수행한 경우보다 최대 61.50% 최소 0%로 20회 평균 34.98% 감소하였다. 그리고 곱셈 횟수는 최고 31.7% 최소 0%로 15.56% 감소하였다. 수행시간의 경우는 약 최고 65% 최소 -1%로 평균 33.22%의 향상을 확인할 수 있었다.

이 논문은 지식경제부 출연금으로 ETRI와 시스템반도체산업진흥센터에서 수행한 ITSoC 핵심설계인력양성사업과 교육과학기술부의 재원으로 한국학술진흥재단에서 수행하는 BK21 프로젝트의 지원을 받아 수행된 연구입니다.

참고문헌

[1] BLAS Forum. Documentation for the Basic Linear Algebra Subprograms(BLAS), Oct. 1999. http://www.nelib.org/blast/blast-forum

[2] Eun-Jin Im, "Optimizing the Performance of Sparse Matrix-Vector Multiplication" Computer Science Division Report No. UCB/CSD-00-1104 University of California, Berkely, June, 2000.