희소 행렬 연산의 성능 최적화에 관한 연구

김경훈* 김 병수 임 은진
국민대학교 컴퓨터학부
khkim@cs.kookmin.ac.kr, bskim@cs.kookmin.ac.kr, ejim@kookmin.ac.kr

Performance Optimization of Sparse Matrix Operation

Kyung-Hoon Kim, Byeong-Soo Kim, Eun-Jin Im
School of Computer Science, Kookmin University

요 약

계산 과학을 사용하는 응용 분야는 공학, 물리, 화학, 생명 과학에서 경제학까지 다양하다. 계산 과학에 사용되는 많은 알고리즘들은 행렬 연산을 포함하고 있으며 이 행렬은 크기가 크고 대부분의 원소가 0 값을 갖는 희소 행렬일 경우가 많다. 본 논문에서는 희소 행렬의 연산 중, 희소 행렬 A 와 밀집 벡터 x, y 에 대하여 $y \leftarrow y + Ax$ 와 $y \leftarrow y + A^T Ax$ 의 두 가지 연산에 대한 계산 속도 개선 방법으로서 레지스터 재사용을 높이는 레지스터 블록화와 캐쉬 미스를 줄이기 위한 캐쉬 최적화 방법을 제안하며 또한 희소 행렬의 특성과 target 컴퓨터의 구조에 따라 정해지는 레지스터 블록 크기를 결정하는 방법을 설명한다. Preliminary 결과로 이 방법을 Pentium III system 상에서 실험한 결과를 보이는데 $y \leftarrow y + Ax$ 의 연산에 대하여는 2.5 배, $y \leftarrow y + A^T Ax$ 의 연산에 대하여는 3.5 배까지의 성능 개선을 이룰 수 있다.

1. 서 론

본 논문에서 우리는 희소 행렬 A 와 밀집 벡터 x, y에 대하여 $y \leftarrow y + Ax$ 와 $y \leftarrow y + A^T Ax$ 의 두 가지 종류의 연산에 대하여 성능 튜닝을 하는 방법을 제안하고자 한다. 행렬에 관한 연산은 계산 과학의 많은 알고리즘에서 사용되는데 $y \leftarrow y + Ax$ 형태의 연산은 conjugate gradient 방법 등의 iterative solver [1] 에서 반복적으로 사용되며 $y \leftarrow y + A^T Ax$ 와 같은 연산은 mathematical programming [2], singular value decomposition [1], Kleinberg 의 HITS 알고리즘 [3] 등에서 역시 반복적으로 사용되어서 알고리즘 수행 시간의 대부분은 이 연산을 수행하는데 소비된다. 이러한 알고리즘들의 응용 분야는 공학, 물리, 화학, 생명 과학, 경제학으로부터 컴퓨터 과학의 image processing, 정보 검색, 회로 설계에까지 다양하다.

이 때 사용되는 행렬의 크기는 풀고자 하는 문제의 변수의 개수에 따라 결정되므로 실제적인 문제에서 크기가 매우 커서 모든 원소를 2차원 배열의 형태로 저장하는 밀집 행렬로 표현하였을 때 대부분의 컴퓨터에서 주 메모리의 크기를 넘어선다. 그러나 이를 잘 살펴보면 행렬의 원소는 연관이 있는 변수의 행과 열이 만나는 곳에서만 0이 아닌 값을 가지고 대부분의 경우에 0의 값을 가진다. 이러한 행렬은 0이 아닌 값만을 그 위치 정보와 함께 저장하는 희소 행렬의 형태로 저장된다. 희소 행렬의 형태로 저장을 하면 메모리의 사용량을 줄일 수 있을 뿐 아니라 행렬 연산을 하는데 있어서 0값에 대한 연산을 생략하기 때문에 계산량을 줄일 수 있다.

그러나 희소 행렬의 연산 속도 (0이 아닌 원소에 대한 연산만을 연산 개수에 포함시킬 때) 는 프로세서의 peak 성능보다 훨씬 낮고 밀집 행렬의 같은 연산 보다도 낮은데 그 이유는 첫째로 모든 0이 아닌 원소에 대한 연산에서 그 원소의 행렬 내 위치를 표현하는 index 를 희소 행렬을 표현하는 자료 구조에서 읽어와야 하기 때문에 추가로 메모리를 접근해야 하는 오버헤드 때문이고, 둘째로는 x 벡터의 원소를 접근할 때 희소 행렬의 0이 아닌 원소의 행렬 내 위치 분포에 따라 비순차적인 메모리 접근이 이루어지기 때문이다.

따라서 본 논문에서는 이러한 메모리 접근을 개선하는 방법으로서 레지스터 재사용을 높이는 레지스터 블록화와 캐쉬 미스를 줄이기 위한 캐쉬 최적화 방법을 제안하며 또한 희소 행렬의 특성과 target 컴퓨터의 구조에 따라 정해지는 레지스터 블록 크기를 결정하는 방법을 설명한다.

희소 행렬을 저장하는 자료 구조는 행렬의 특성, 응용 분야의 특성에 따라 다양한데[4], 본 논문에서는 그 중 가장 일반적인 compressed sparse row (CSR) 형태를 가정한다.

2. 최적화 방법

2.1 레지스터 블록화

메모리 사용을 효율적으로 하기 위해서 먼저 메모리 계층

구조의 최상위에 있는 레지스터의 재사용률을 높이는 희소 행렬의 레지스터 블록화는 밀집 행렬의 블록화 [5,6,7]와 유사하게 행렬의 연산을 행별로 계산하는 대신, 행렬을 전체 행렬 크기보다 작은 사각형의 블록들로 나누어 블록 단위로 연산을 행함으로써 메모리 상위 계층에 적재된 값들을 재사용하도록 하는 것이다.

희소 행렬의 레지스터 블록화를 위하여는 희소 행렬의 자료 구조를 CSR 형태에서 blocked compressed sparse row (BCSR) [4,8,9] 형태로 변환하는데 이는 레지스터 블록의 행 개수 r과 열 개수 c에 대하여 $r\times c$ 의 블록 원소들을 인접하여 저장되도록 하는 것이다. 이 때 블록 크기 인자인 r과 c를 결정하는 것이 문제가 되는데 밀집 행렬의 경우는 타겟 시스템의 구조만으로 정적으로 결정할 수 있지만[5,6,7], 희소 행렬의 경우는 컴퓨터 구조뿐 아니라 행렬의 0이 아닌 원소의 분포 형태에도 관련되어 있기 때문에 정적인 결정이 불가능하다. 이는 밀집 행렬의 경우는 컴퓨터 구조에 따라 메모리 상위 계층이 수용할 수 있는 원소만큼의 블록 크기를 정하면 되는 반면, 희소 행렬의 경우는 블록 크기에 따라 연산의 양이 증가할 수 있기 때문에 동시에 이 연산량의 증가를 최소화하는 블록 크기를 정해야 하기 때문이다. 자세히 설명하면, 희소 행렬의 경우 행렬 전체를 정해진 크기를 가진 블록들로 나누어 이 중 0이 아닌 원소가 1개 이상 있는 블록들을 모두 BCSR 형태로 저장하는데 이 때 저장되는 블록들은 블록 내의 0을 포함한 $r\times c$ 개의 모든 원소를 저장하게 되고 이렇게 저장되는 0값을 갖는 원소 (0-fill 이라 칭함)는 행렬 연산 시에 계산에 포함되므로 연산량이 증가하게 되는 것이다.

따라서 블록 크기를 결정하기 위해 두 가지 요소를 반영해야 하는데 첫번째는 컴퓨터 구조에 따라 어떤 블록 크기가 좋은가를 결정하기 위하여 그림 1과 같은 레지스터 프로화일을 이용한다. 이 프로화일은 1000x1000의 행렬에 대하여 모든 원소가 0이 아닌 원소로 보고 1x1 부터 12x12 까지의 블록 크기로 블록화하여 $y \leftarrow y + Ax$ 의 연산을 했을 때 연산 속도를 색으로 표현한 것이다. 왼쪽 아래가 1x1 블록에 대한 것으로 이것은 블록화를 하지 않은 reference 구현에 해당하고 나머지 사각형 중 성능이 높은 것에 대해서는 성능개선 수치를 숫자로 표시하였다. 블록 크기 결정을 위한 두번째 요소는 행렬의 구조를 반영하는 것으로 $r\times c$ 의 블록에 대하여 행렬의 채움정도 (fill ratio: 0이 아닌 원소 개수에 대한 저장된 원소 개수의 비율)를 추정해 보는 것이다.

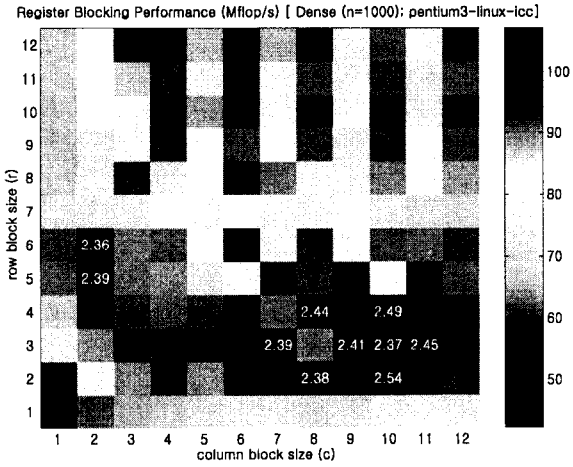

그림 1. 레지스터 프로화일

이 두 가지 요소로부터 $r\times c$ 의 블록에 대한 연산 속도를 다음 식에 의하여 추정하여 최고값을 가지는 블록 크기를 정한다.

$$Estimated\ \ Mflop/s = \frac{Mflop/s\ on\ dense\ matrix\ in\ r\times c}{Estimated\ \ fill\ ratio\ \ for\ r\times c}$$

2.2 캐쉬 최적화

$y \leftarrow y + A^T Ax$ 연산에 대하여 캐쉬 최적화를 다음과 같이 적용할 수 있다. 먼저 reference 구현은 Ax 를 계산하여 z라는 중간 벡터에 저장하고 다시 $A^T z$ 를 계산하는 것이다. 이 경우 행렬의 원소를 두 번 접근해야 하는데 중간 벡터를 저장하지 않는 대신 행렬의 한 행 A_{i*} 에 대하여 scalar 값 $t = A_{i*}x$ 를 계산한 후, 이에 대하여 $y+ = tA_{i*}^T$ 를 계산하면 행렬의 원소를 캐쉬 내에서 한 번만 접근할 수 있다.

3. 성능 튜닝의 적용 결과

앞에서 기술된 성능 개선 방법을 펜티엄 III system 상에서 적용하여 연산 속도를 각 연산에 대하여 그림 2와 3에서 보여주고 있다. 사용된 시스템은 clock speed는 500MHz 이고 double-precision부동소수점 레지스터가 8개, L1 캐쉬가 16K, L2 캐쉬가 512K이며 compiler는 intel C v6.0을 사용하였다. 참고로 이 시스템에서 밀집 행렬과 벡터곱셈 연산의 속도는 최적화된 BLAS 루틴을 사용하였을 때 96Mflop/s 였다. 성능 측정을 위하여 희소 행렬의 모음[8] 중 33개의 행렬에 대하여 각각 메모리 최적화 기법을 적용하여 reference 속도와 함께 보여주고 있다.

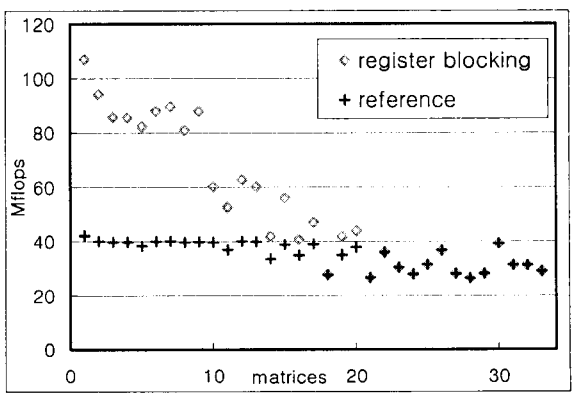

그림 2. $y \leftarrow y + Ax$ 연산의 성능 비교

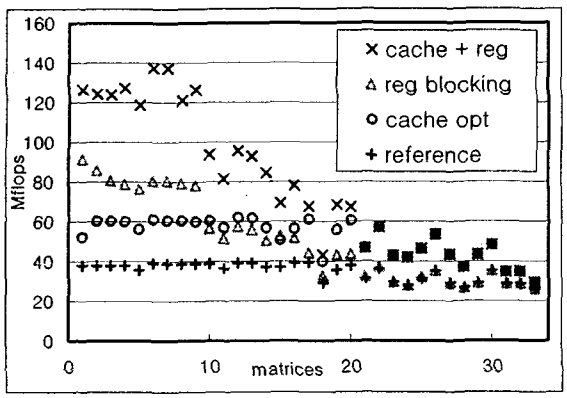

그림 3. $y \leftarrow y + A^T Ax$ 연산의 성능 비교

4. 관련 연구

밀집 행렬에 대한 블록화 혹은 타일화에 관한 연구는 컴파일러가 정적으로 블록화에 관한 결정을 내릴 수 있을 정도로 되어왔으나[5,6,7] 앞에서 기술한 이유로 이를 그대로 희소 행렬에 적용하기는 어렵다. Heras 등[10]과 Fraguela 등[11]은 희소 행렬 연산의 캐쉬 미스의 확률적 모델을 세워 이 연산의 최적화 모델을 제시하였으나, 이들은 0이 아닌 원소가 균일하게 분포되어 있다는 가정하에 모델을 제시하였다. 대부분의 희소 행렬은 0이 아닌 원소가 블록을 이루어 모여있는 등 균일하게 분포되어 있지 않다는 것을 감안하여야 한다. Bik 의 sparse compiler[12]과 Bernoulli compiler[13,14] 는 희소 행렬 연산의 성능 개선을 위하여 희소 행렬과 그에 대한 연산을 컴파일러의 코드 생성 단계에 제시하는 표현 방법에 관한 연구를 하고 있는데 이와 같은 infrastructure는 본 연구 결과를 적용하기 위한 도구로 응용할 가능성이 있다. Pinar[15] 등은 희소 행렬 연산 속도 개선에 행과 열의 reordering을 적용하기도 하였다.

5. 결론 및 향후 연구

희소 행렬에 대한 연산들은 자주 사용되는 연산 커널이면서 메모리 접근이 비효율적이어서 연산 속도의 개선 여지가 많다. 따라서 이들 연산에 대한 속도 개선 방법으로서 레지스터 블록화와 캐쉬 최적화 방법을 제안하였다. 이 방법을 Pentium III system 상에서 실험한 결과 $y \leftarrow y + Ax$ 의 연산에 대하여는 2.5 배, $y \leftarrow y + A^T Ax$ 의 연산에 대하여는 3.5 배까지의 성능 개선을 이룰 수 있었다.

본 연구 결과를 바탕으로 앞으로의 연구 방향은 1) 이와 같은 최적화에 관한 성능 모델링에 관한 연구, 2) 실제 응용에 바탕을 둔 다른 희소 행렬 연산에 관한 최적화, 3) 다양한 컴퓨터 구조 상에서의 최적화 효과에 대한 비교 연구와 분석 4) 희소 행렬 연산의 최적화 컴파일러에 관한 연구 등이 있다.

[참고 문헌]

1. J. W. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997
2. W. Wang and D. P. O'Leary. Adaptive use of iterative methods in interior point methods for linear programming. *Technical Report UMIACS-95-111, University of* Maryland at College Park, College Park, MD, USA, 1995.
3. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM*, 46(5):604-632, 1999.
4. S. Blackford, G. Corliss, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, C. Hu, W. Kahan, L. Kaufman, B. Kearfott, F. Krogh, X. Li, Z. Marrny, A. Petitet, R. Pozo, K. Remington, W. Walster, C. Whaley, and J. W. von Gudenberg. Document for the Basic Linear Algebra Subprograms (BLAS) standard : BLAS Technical Forum, 2001. Chapter 3: www.netlib.org/blast.
5. M. E. Wolf and M. S. Lam. A data locality optimizing algorithm. In *Proceeding of the ACM SIGPLAN '91 Congerence on Programming Language Design and Implementation*, June 1991.
6. K. S. McKinley, S. Carr, and C.-W. Tseng. Improving data locality with loop transformations. *ACM Transactions on Programming Languages and Systems*, 18(4):424-453, July 1996.
7. S. Chatterjee, E. Parker, P. J. Hanlon, and A. R. Lebeck. Exact analysis of the cache behavior of nested loops. In *Proceedings of the ACM SIGPLAN 2001 Conference on Programming Language Design and Implementation*, pp. 286-297, Snowbird, USA, June 2001.
8. E.-J. Im. *Optimizing the performance of sparse matrix-vector multiplication*. PhD thesis, University of California, Berkeley, May 2000.
9. E.-J. Im and K. A. Yelick. Optimizing sparse matrix computations for register reuse in SPARSITY. In *Proceedings of the International Conference on Computational Science*, volume 2073 of LNCS, pages 127-136, San Francisco, CA, May 2001. Springer.
10. D. B. Heras, V. B. Perez, J. C. C. Dominguez, and F. F. Rivera. Modeling and improving locality for irregular problems: sparse matrix-vector product on cache memories as a case study. In *HPCN Europe*, pages 201-210, 1999.
11. B. B. Fraguela, R. Doallo, and E. L. Zapata. Memory hierarchy performance prediction for sparse blocked algorithms. *Parallel Processing Letters*, 9(3), March 1999.
12. A. J. C. Bik and H. A. G. Wijshoff. Automatic nonzero structure analysis. *SIAM Journal on Computing*, 28(5):1576-1587, 1999.
13. P. Stodghill. *A Relational Approach to the Automatic Generation of Sequential Sparse Matrix Codes*. PhD thesis, Cornell University, August 1997.
14. W. Pugh and T. Shpeisman. Generation of efficient code for sparse matrix computations. In *Proceedings of the 11th Workshop on Languages and Compilers for Parallel Computing*, LNCS, August 1998.
15. A. Pinar and M. Heath. Improving performance of sparse matrix-vector multiplication. In *Proceedings of Supercomputing*, 1999.