Cache Friendly Sparse Matrix-vector Multiplication

[Extended Abstract]

Sardar Anisul Haque
University of Western Ontario
ON N6A 5B7, Canada
shaque4@csd.uwo.ca

Shahadat Hossain
University of Lethbridge
AB T1K 3M4, Canada
shahadat.hossain@uleth.ca

Marc Moreno Maza
University of Western Ontario
ON N6A 5B7, Canada
moreno@csd.uwo.ca

1. INTRODUCTION

Sparse matrix-vector multiplication or *SpMXV* is an important kernel in scientific computing. For example, the conjugate gradient method (CG) is an iterative linear system solving process where multiplication of the coefficient matrix A with a dense vector x is the main computational step accounting for as much as 90% of the overall running time. Though the total number of arithmetic operations (involving nonzero entries only) to compute Ax is fixed, reducing the probability of cache misses per operation is still a challenging area of research. This preprocessing is done once and its cost is amortized by repeated multiplications. Computers that employ cache memory to improve the speed of data access rely on reuse of data that are brought into the cache memory. The challenge is to exploit data locality especially for unstructured problems: modeling data locality in this context is hard.

Pinar and Heath [8] propose column reordering to make the nonzero entries in each row contiguous. However, column reordering for arranging the nonzero entries in contiguous location is NP-hard [8]. In a considerable volume of work [2, 6, 8, 9, 10] on the performance of *SpMXV* on modern processors, researchers propose optimization techniques such as reordering of the columns or rows of A to reduce, for example, indirect access and improving data locality, and blocking for reducing memory load and loop overhead.

In this extended abstract, we present a new column ordering algorithm, based on the binary reflected Gray codes, that runs in linear time with respect to the number of nonzero entries. We analyze the cache complexity of *SpMXV* when the sparse matrix is ordered by our technique. The results from numerical experiments, with very large test matrices, clearly demonstrate the performance gains rendered by our proposed technique.

Categories and Subject Descriptors

F.2.1 [**Numerical Algorithms and Problems**]: Computations on matrices; G.1.3 [**Numerical Linear Algebra**]: Sparse, structured, and very large systems (direct and iterative methods)

General Terms

Algorithms

Keywords

Sparse Matrix, Cache Complexity, Binary reflected Gray Code

PASCO 2010, 21–23 July 2010, Grenoble, France.

2. BINARY REFLECTED GRAY CODE ORDERING

We develop a new column ordering algorithm based on *binary reflected Gray code* (BRGC for short) for sparse matrices. We will call it BRGC ordering. A p-bit *binary reflected Gray code* [3] is a Gray code denoted by G^p and defined by $G^1 = [0, 1]$ and

$$G^p = [0G_0^{p-1}, \ldots, 0G_{2^{p-1}-1}^{p-1}, 1G_{2^{p-1}-1}^{p-1}, \ldots, 1G_0^{p-1}],$$

where G_i^p is the i-th binary string of G^p. We call i the rank of G_i^p in G^p. For example the rank of 011 in G^3 is 2. We consider each column of a $m \times n$ sparse matrix A as a binary string of length m where each nonzero is treated as 1. Hence, we have n binary strings of length m, say $\{b_0, b_1, \ldots, b_{n-1}\}$. Let Π be the permutation of these strings satisfying the following property. For any pair of indices i, j with $i \neq j$, the rank of $b_{\Pi(j)}$ in G^m is less than that of $b_{\Pi(i)}$ if and only if $\Pi(i) < \Pi(j)$ holds. We refer to A_{brgc} as our sparse matrix A after its columns have been permuted by Π. One can check that the BRGC ordering sorts the columns of A according to their ranks in G^m in descending order.

On average, our sorting algorithm proceeds in ρ (the average number of nonzeros in a column) successive phases, which are described below. During the first phase, we sort the columns by increasing position of their first nonzero entries from above, creating equivalence classes where any two columns are uncomparable for this ordering. During the second phase, in each equivalence class, we sort the columns by decreasing position of their second nonzero entries from above, thus, refining the equivalence classes of the first phase into new classes where again any two columns are uncomparable for this second ordering. More generally, during the k-th phase, in each equivalence class obtained at the $(k-1)$-th phase, we sort the columns by increasing position (resp. decreasing position) of their k-th nonzero entry from above, if k is odd, (resp. if k is even) thus, refining again the equivalence classes. Continuing in this manner, we obtain the

desired sorted matrix. Observe that whenver an equivalence class is a singleton, it no longer participates to the next sorting phases.

Based on the above procedure and the counting sort algorithm [4], the matrix A_{brgc} is obtained from A using $O(\tau)$ integer comparisons (on average) and $O(n+\tau)$ data-structure updates, where τ is the total nonzero entries in A [7].

Let C be an equivalence class obtained after the ℓ-th phase and before the $(\ell+1)$-th phase. We call *nonzero stream at level* $(\ell+1)$ in C the set of the $(\ell+1)$-th nonzero entries from above in the columns of C. In the nonzero stream at level $(\ell+1)$ in C, a set of nonzeros having the same row index is called a *step*.

3. CACHE COMPLEXITY

Consider the ideal cache [5] of Z words, with cache line of L words. Assume that n is large enough such that the vector x does not fit into the cache. During *SpMXV*, the total number of accesses in x is τ. These accesses are usually irregular. Note that n of those accesses are cold misses. Each of the other $\tau - n$ accesses creates a cache miss with probability $(n - Z/L)/n$, since no spatial locality should be expected in accessing x. Therefore, the total number of expected cache misses in accessing x is computed as follows.

$$\overline{Q_1} = Z/L + (\tau - Z/L)\frac{n-Z/L}{n}.$$

We claim that A_{brgc} has at least nonzero streams at level 1 and 2. Indeed, each column has at least some nonzeros, which implies that A_{brgc} has nonzero stream at level 1. Observe that each step of the nonzero stream at level 1 is expected to have ρ entries. Moreover, we assume $\rho \geq 2$. This leads to the formation of the nonzero stream at level 2. Therefore, the total number of nonzeros, in all the nonzero streams of level 1 and 2, is $2n$. Due to the LRU replacement policy, one can expect that the n multiplications with the nonzeros in the nonzero stream at level 1 incur the same amount of cache misses as if x was scanned in a regular manner during *SpMXV*. Next, we observe that each of the accesses in x for multipliying with nonzeros in the nonzero streams at level 2 creates cache misses with probability $\frac{n/\rho - Z/L}{n/\rho}$. More generally, each of the other access in x creates cache miss with probability $\frac{cn/\rho - Z/L}{cn/\rho}$, where, c is the average number of nonzero streams under one step of first level nonzero stream and $1 \leq c \leq \rho$. Therefore, the expected cache misses in accessing x is given by:

$$\overline{Q_2} = n/L + Z/L + (n - Z/L)\frac{n/\rho - Z/L}{n/\rho} + (\tau - 2n)\frac{cn/\rho - Z/L}{cn/\rho}.$$

We apply the computer algebra system MAPLE to analyze the difference between $\overline{Q_1}$ and $\overline{Q_2}$. For the large matrices of [1], the equality $n = O(Z^2)$ holds for level 2 cache and our calculations show that we have, $\overline{Q_1}$ - $\overline{Q_2} \approx n$.

4. EXPERIMENTAL RESULTS

We selected 10 matrices from [1] for our experimentation. The basic information for each test matrix is given in Table 4. We run our experiments on an *intel core 2 processor Q6600*. It has L2 cache of 8MB and the CPU frequency is 2.40 GHz [11]. We measure the CPU time (given in seconds) for 1000 *SpMXV*s for three variants reported in Table 4: with BRGC ordering, without any preprocessing and after a random re-ordering of the columns. It shows that

Matrix name	m	n	τ
fome21	67748	216350	465294
lp_ken_18	105127	154699	358171
barrier2-10	115625	115625	3897557
rajat23	110355	110355	556938
hcircuit	105676	105676	513072
GL7d24	21074	105054	593892
GL7d17	1548650	955128	25978098
GL7d19	1911130	1955309	37322725
wikipedia-20051105	1634989	1634989	19753078
wikipedia-20070206	3566907	3566907	45030389

Table 1: Test matrices.

Matrix name	BRGC ordering	no ordering	random ordering
fome21	3.6	3.9	4.8
lp_ken_18	2.7	3.1	3.3
barrier2-10	19.0	19.1	23.2
rajat23	3.0	3.0	3.4
hcircuit	2.6	2.5	2.9
GL7d24	3.0	3.2	3.1
GL7d17	484.6	625.0	580.7
GL7d19	784.6	799.0	899.2
wikipedia-20051105	258.9	321.0	411.5
wikipedia-20070206	731.5	859.0	1046.0

Table 2: CPU time for 1000 *SpMXV*s.

the cost of BRGC ordering is amortized by 1000 *SpMXV*s for all of the matrices. Our experimental results also show that the cost of BRGC ordering algorithm, as a preprocessing step, can be much less than $\sqrt{n}$ *SpMXV*s and thus can improve the performances of CG-type algorithms in practice. Note that other column ordering algorithms reported in [8] and their performances are compared with BRGC ordering algorithm in [6]. As reported in [6], BRGC algorithm outperforms these other column ordering algorithms on three different computer architectures.

5. REFERENCES

[1] T. Davis, Uni. of florida sparse matrix collection. http://www.cise.ufl.edu/research/sparse/

[2] E. Im, Optimizing the performance of sparse matrix-vector multiplication. *PhD Thesis, Uni. of California Berkeley*, 2000.

[3] D. Kreher and D. Stinson, *Combinatorial Algorithms :Gen., Enum., and Search*. CRC Press, 1999.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*. 2nd Edition McGraw-Hill, 2001.

[5] M. Frigo, C. E. Leiserson, Prokop, Harald, Ramachandran, and Sridhar, Cache-Oblivious algorithms. FOCS '99: Proc. of the 40th Annual Symp. on Foundations of Comp. Sc., 1999

[6] S. Haque, A computational study of sparse matrix storage scheme M.Sc. Thesis, Uni. of Lethbridge , 2008.

[7] S. Haque, and M. Moreno Maza, Algorithms for sorting large objects, Tech. Report, Uni. of Western Ontario, 2010.

[8] A. Pinar and M. Heath, Improving performance of sparse matrix-vector multiplication. In *Supercomputing '99: Proc. of the 1999 ACM/IEEE conf. on Supercomputing (CDROM)*, New York, USA, 1999.

[9] S. Toledo, Improving the memory-system performance of sparse-matrix vector multiplication, In *IBM J. Res. Dev.*, vol. 41, num. 6, 1997.

[10] R. Vuduc, Automatic performance tuning of sparse matrix kernels. *PhD Thesis, Uni. of California Berkeley*, 2003.

[11] Intel Webpage, Intel core 2 quad processor q6600. http://ark.intel.com/Product.aspx?id=29765