A Gray Code Mediated Data-Oblivious $(0,1)$-Matrix-Vector Product Algorithm

Andrew A. Anda
Computer Science Department
St. Cloud State University
St. Cloud, MN, U.S.A.

Abstract – *We begin by describing a data-sensitive differencing method which can reduce the number of arithmetic operations within a* $(0,1)$*-matrix vector product. We find that a Gray code has an optimal ordering for exploiting that method. Using one or more Gray code matrix-vector products, we demonstrate how to reduce the number of arithmetic operations for a general* $(0,1)$*-matrix vector product by first performing a set of Gray code matrix-vector products which precompute all of the possible elements of the conformally partitioned inner-products of a general* $(0,1)$*-matrix-vector product. The Gray code bit strings then serve as indices for one or more arrays of those partial inner-products. It then remains only to load the Gray code indexed elements from the Gray code products into the one or more general* $(0,1)$*-matrix partial inner-product vectors and then sum them to yield the final product. This algorithm is data-oblivious – the same number of operations will be required regardless of the pattern of the* $(0,1)$*-matrix. We describe an algorithm for computing the optimal set of Gray code sizes for each possible matrix size and shape, and we present a set of results for a broad range of dimensions. We show how* $(0,1)$*-matrix vector products may be applied to the performance of certain restricted classes of more general matrix vector products.*

Keywords: Gray code, $(0,1)$**-matrix-vector product, data-oblivious.**

1 Introduction

Each element of a rectangular $(0,1)$-matrix (also identified as zero-one or Boolean) has the value of either one or zero.
$(0,1)$-matrices arise from problems in a variety of application areas such as graph theory [3], information retrieval [2], and matrix calculus [4].
Any general matrix may be decomposed into a linear combination of conformal (0,1)-matrices. For a general real matrix $A \in \mathbb{R}^{m \times n}$,

$$A = \sum_{i=1}^{m} \sum_{j=1}^{n} \alpha_{ij} E_{ij}, \left\{ E_{ij} = \mathbf{e}_i \mathbf{e}_j^T, 1 \leq i \leq m; 1 \leq j \leq m \right\} \tag{1}$$

The matrix-vector product operation,

$$Ax = y, \tag{2}$$

represents the product of a matrix, $A \in \{0,1\}^{m \times n}$, by the vector, $x \in \mathbb{R}^n$, to yield the vector, $y \in \mathbb{R}^m$. More generally, the vectors may actually be over *any* algebraic ring for which addition and multiplication is closed and well defined.
The general matrix-vector product is an example of *level 2* BLAS (Basic Linear Algebra Subroutines) operation, and as such, it exhibits a quadratic complexity, i.e. its evaluation requires a doubly nested loop. There is little that can be done to improve the asymptotic efficiency of the general matrix-vector product. Blocking may yield modest performance gains for large matrices on a hierarchical memory architecture. But, we know of no general transformation to reduce the number of scalar additions and multiplications. For certain classes of structured matrices, however, we can attain $O(n \ln n)$ by applying the FFT to the calculation. A structured matrix is one which can be fully characterized by $O(n)$ parameters. In fact the product of a rank one matrix and a vector can be performed in $O(n)$ if we know the two vectors that formed the rank-one matrix, $Ax = yz^T x$ for some vectors y and z. As well, certain types of patterned matrices, e.g. banded matrices, can form a product with reduced complexity – linear in the dimension of the matrix, $O(n)$, if the bandwidth is held constant. We will be considering general $(0,1)$-matrices though.
A Gray code is a circular ordering of all 2^n binary strings of length n in which adjacent strings in sequence differ in exactly one bit, i.e. all adjacent strings have a *Hamming distance* of unity. One can label the vertices of a *hypercube* with bit-strings such that all *adjacent* neighbors have a bit-string differing in exactly one bit. Each Gray code corresponds to a *Hamiltonian cycle* about the vertices of the hypercube.

2 A Differencing Method

The general Matrix-vector product $Ax = y$, $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, is computed with the following equation,

$$y_i = \sum_{j=1}^{n} \alpha_{ij} x_j, 1 \leq i \leq m. \tag{3}$$

Computing this requires an algorithm having a doubly nested loop, for which there are two possible orderings. We'll select the ordering which performs an inner product in the inner loop, so that each outer loop completely calculates one element of the resultant vector y. With either ordering, the number of additions and multiplications each is mn. For a $(0,1)$-matrix though, $\alpha_{ij} \in \{0,1\}, \forall i,j$. So, either the product $\alpha_{ij} x_j$ contributes to the sum as x_j, in the case of $\alpha_{ij} = 1$, or it is skipped, in the case of $\alpha_{ij} = 0$. This implies that the maximum possible amount of work is mn additions which would occur if the matrix were all ones.
Consider computing the difference between two elements of y, y_i and y_k:

$$y_k - y_i = \sum_{j=1}^{n} \alpha_{kj} x_j - \sum_{j=1}^{n} \alpha_{ij} x_j = \sum_{j=1}^{n} \left(\alpha_{kj} x_j - \alpha_{ij} x_j\right) = \sum_{j=1}^{n} x_j \left(\alpha_{kj} - \alpha_{ij}\right) \tag{4}$$

Now consider computing y_k if y_i has already been computed.

$$y_k = y_i + \sum_{j=1}^{n} x_j \left(\alpha_{kj} - \alpha_{ij}\right) \tag{5}$$

$$= y_i + \sum_{j=1}^{n} x_j \left(d_j\right), d_j = \alpha_{kj} - \alpha_{ij} \tag{6}$$

This implies that apart from the first element of y to be computed, each subsequent element of y can be computed as the sum of a previously computed element with the inner product of x with

the difference vector of the two rows of A, d. Let A_k be the kth row of A. Then we have saved $\|A_k\|_1 - \|d\|_1 - 1$ operations in computing y_k. If $\|d\|_1 + 1 < \|A_k\|_1$, the savings is positive. $\|d\|_1$ is also the *Hamming distance* between two rows of A. E.g. consider a *triangular* $(0,1)$-matrix-vector product,

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{pmatrix} = \begin{pmatrix} a \\ a+b \\ a+b+c \\ a+b+c+d \end{pmatrix} = \begin{pmatrix} y_0 = a \\ y_1 = y_0 + b \\ y_2 = y_1 + c \\ y_3 = y_2 + d \end{pmatrix}.$$

We can see that in the above example we have halved the number of additions. And, more generally, for a triangular $(0,1)$-matrix-vector product of size n, the number of additions is reduced from $n(n-1)/2$ to $(n-1)$ via the differencing method.

We can deal with any duplicated rows in A by loading its previously computed value at the expense of no additions.

If the remaining rows are unique, we can determine the minimum possible number of required additions. Because there are no duplicated rows, there must be a Hamming distance of at least one between each row of A and each other row. The optimum in this case would be that, for every row of A, there is another row of A of no more than unit Hamming distance. We have already defined a sequence of Boolean vectors which satisfies this condition, a Gray code.

2.1 Minimizing the Number of Additions

A Gray code q bits wide consists of a sequence of $r = 2^q$ bit-strings. Using the differencing method above, a $r \times q$ Gray code matrix vector product can be computed in a maximum of r additions. In fact, if one substitutes a load for an addition when there are at most 1 nonzero bits in the row, the maximum becomes $r - q - 1$ required additions.

If we have a general $(0,1)$-matrix matrix having q columns and any number of rows, we can first precompute a scratch product vector y computing the Gray code Matrix-vector product with x. Then, for each row of the general matrix A, we load into y the entry in the scratch y corresponding to the bit-pattern for that row. The only additions are from the precomputation of the scratch y from the Gray code matrix. If the number of general matrix rows exceeds the number of Gray code matrix rows, $m > r$, a net reduction in the maximum number of additions will have been effected. If $n > q$, one of two conditions can hold:

- $n \bmod q = 0$ in which case we will use n/q Gray code matrix vertical partitioning stripes across A with a subsequent sum across the stripes for all m.
- $n \bmod q \neq 0$ in which case we will use $\lfloor n/q \rfloor$ Gray code matrix stripes of width q, and one additional Gray code matrix stripe of width $n \bmod q$ with a subsequent sum across the stripes for all m.

Let us define $w(q,m,n)$ as the number of additions required for a general $(0,1)$-matrix. Then for an arbitrary m and n,

$$w = \begin{cases} 2^n - n - 1 & \text{if } n < q \\ 2^q - q - 1 & \text{if } n = q \\ \frac{n}{q}\left(m + 2^q - q - 1\right) - m & \text{if } n > q \text{ and } 0 = n \bmod q \\ \left\lfloor \frac{n}{q} \right\rfloor \left(m + 2^q - q - 1\right) + 2^{n \bmod q} - n \bmod q - 1 & \text{if } n > q \text{ and } 0 \neq n \bmod q \end{cases} \tag{7}$$

The q which minimizes $w(q,m,n)$, $\min_q w(q,m,n)$, can be computed by searching for the smallest w for a reasonably small set of q's bounded by $\lceil \lg m \rceil$. (w is discontinuous. Be sure to test all valid q's lest you find a local minimum.)

Using C++, we computed the optimum Gray code width, number of addition operations, percentage ratio to that of a conformal general dense matrix-vector product, and percent reduction in addition operations for an exponential range of rows and columns, from 5 to 5^{10}, by powers of five.

Table 1: Optimum Gray Code Widths

Matrix Dimensions										
ROWS	COLUMNS									
	5^1	5^2	5^3	5^4	5^5	5^6	5^7	5^8	5^9	5^{10}
5^1	3	3	3	3	3	3	3	3	3	3
5^2	5	4	4	4	4	4	4	4	4	4
5^3	5	5	5	5	5	5	5	5	5	5
5^4	5	7	7	7	7	7	7	7	7	7
5^5	5	9	9	9	9	9	9	9	9	9
5^6	5	13	11	11	11	11	11	11	11	11
5^7	5	13	14	13	13	13	13	13	13	13
5^8	5	13	16	15	15	15	15	15	15	15
5^9	5	13	18	17	17	17	17	17	17	17
5^{10}	5	13	18	19	20	20	20	20	20	20

Table 2: Optimum Addition Operation Counts

Matrix Dimensions										
ROWS	COLUMNS									
	5^1	5^2	5^3	5^4	5^5	5^6	5^7	5^8	5^9	5^{10}
5^1	1.10e+1	8.00e+1	4.11e+2	2.08e+3	1.04e+4	5.21e+4	2.60e+5	1.30e+6	6.51e+6	3.26e+7
5^2	2.80e+1	2.22e+2	1.15e+3	5.77e+3	2.89e+4	1.45e+5	7.23e+5	3.61e+6	1.81e+7	9.03e+7
5^3	2.80e+1	6.35e+2	3.68e+3	1.89e+4	9.49e+4	4.75e+5	2.37e+6	1.19e+7	5.94e+7	2.97e+8
5^4	2.80e+1	2.25e+3	1.27e+4	6.64e+4	3.33e+5	1.67e+6	8.33e+6	4.16e+7	2.08e+8	1.04e+9
5^5	2.80e+1	7.38e+3	4.74e+4	2.50e+5	1.26e+6	6.30e+6	3.15e+7	1.57e+8	7.87e+8	3.94e+9
5^6	2.80e+1	2.79e+4	1.94e+5	9.90e+5	5.02e+6	2.51e+7	1.25e+8	6.27e+8	3.14e+9	1.57e+10
5^7	2.80e+1	9.04e+4	7.64e+5	4.14e+6	2.07e+7	1.04e+8	5.19e+8	2.59e+9	1.30e+10	6.48e+10
5^8	2.80e+1	4.03e+5	3.20e+6	1.74e+7	8.81e+7	4.41e+8	2.20e+9	1.10e+10	5.51e+10	2.76e+11
5^9	2.80e+1	1.97e+6	1.34e+7	7.50e+7	3.81e+8	1.92e+9	9.58e+9	4.79e+10	2.39e+11	1.20e+12
5^{10}	2.80e+1	9.78e+6	6.03e+7	3.29e+8	1.69e+9	8.45e+9	4.22e+10	2.11e+11	1.06e+12	5.28e+12

2.1.1 Results Discussion

The tabular data shows that:

- For a fixed number of rows, as the number of columns increases, the ratio compared to dense approaches a constant.
- For a fixed number of columns, as the number of rows increases, the ratio compared to dense continues to improve. The number of rows is ultimately the limiting factor.
- As the size of square matrices increases, the ratio compared to dense continues to improve.

If one were to parallelize this algorithm, I would recommend that each processor retain a complete set of rows, letting only the columns be distributed. Once each processor had completed computing their respective vectors, it would remain only to perform a gather-sum.
For multiple right hand sides, e.g. matrix-matrix products, the complexity would be a strict linear factor of the number right hand sides. Because of the preference for tall matrices, the Strassen-Winnograd or other recursive decomposition algorithms will probably not be advisable for this Gray code algorithm.

Table 3: Optimum Addition Ratio Percentages Compared to Dense

Matrix Dimensions										
ROWS	COLUMNS									
	5^1	5^2	5^3	5^4	5^5	5^6	5^7	5^8	5^9	5^{10}
5^1	55.00	66.67	66.29	66.67	66.65	66.67	66.67	66.67	66.67	66.67
5^2	28.00	37.00	37.00	37.00	37.00	37.00	37.00	37.00	37.00	37.00
5^3	5.60	21.17	23.71	24.20	24.30	24.32	24.32	24.32	24.32	24.32
5^4	1.12	15.01	16.44	17.02	17.04	17.05	17.05	17.05	17.05	17.05
5^5	0.22	9.84	12.24	12.84	12.90	12.90	12.90	12.90	12.90	12.90
5^6	4.48e-2	7.44	10.03	10.15	10.28	10.27	10.28	10.28	10.28	10.28
5^7	8.96e-3	4.82	7.89	8.50	8.49	8.49	8.50	8.50	8.50	8.50
5^8	1.79e-3	4.30	6.61	7.12	7.22	7.22	7.23	7.23	7.23	7.23
5^9	3.58e-4	4.19	5.54	6.16	6.25	6.28	6.28	6.28	6.28	6.28
5^{10}	7.17e-5	4.17	4.98	5.41	5.53	5.54	5.54	5.54	5.54	5.54

Table 4: Optimum Percentage Reduction of Additions Compared to Dense

Matrix Dimensions										
ROWS	COLUMNS									
	5^1	5^2	5^3	5^4	5^5	5^6	5^7	5^8	5^9	5^{10}
5^1	45.00	33.33	33.71	33.33	33.35	33.33	33.33	33.33	33.33	33.33
5^2	72.00	63.00	63.00	63.00	63.00	63.00	63.00	63.00	63.00	63.00
5^3	94.40	78.83	76.29	75.80	75.70	75.68	75.68	75.68	75.68	75.68
5^4	98.88	84.99	83.56	82.98	82.96	82.95	82.95	82.95	82.95	82.95
5^5	99.78	90.16	87.76	87.16	87.10	87.10	87.10	87.10	87.10	87.10
5^6	99.96	92.56	89.97	89.85	89.72	89.73	89.72	89.72	89.72	89.72
5^7	99.99	95.18	92.11	91.50	91.51	91.51	91.50	91.50	91.50	91.50
5^8	100.00	95.70	93.39	92.88	92.78	92.78	92.77	92.77	92.77	92.77
5^9	100.00	95.81	94.46	93.84	93.75	93.72	93.72	93.72	93.72	93.72
5^{10}	100.00	95.83	95.02	94.59	94.47	94.46	94.46	94.46	94.46	94.46

The data-oblivious characteristic of this algorithm could prove beneficial in certain contexts, e.g. cryptography.

3 Exploiting the $(0,1)$-Matrix Vector Product

Equation 1 has little practical value unless one considers the special case where the number of distinct α_{ij} terms is smaller than the smallest of the two indices. Then all of the E_{ij} matrices will be added together forming denser $(0,1)$-matrices for each set of identical α_{ij} terms. We can then re-factor equation 1 as:

$$A = \sum_{i=1}^{k} \beta_i B_i, \tag{8}$$

Where there are k distinct entries, β_i in A and each B_i represents a distinct $(0,1)$-matrix.
For a single β, the matrix vector product can be written as:

$$Ax = y = \beta Bx = B(\beta x), \tag{9}$$

In equation 9, we see the scalar product moved from being $O(mn)$ to $O(n)$. Equations 8 and 9 can be combined:

$$Ax = y = \left(\sum_{i=1}^{k} \beta_i B_i \right) x = \sum_{i=1}^{k} B_i(\beta_i x). \tag{10}$$

Lets consider the case where $k = 2$. The elements of the matrix A are either of only two values, $\alpha_{ij} \in \{\beta_1, \beta_2\}$. The cardinality of the set that α_{ij} is drawn from is the same as that of the $(0,1)$-matrix. So, rather than use a pair of $(0,1)$-matrices as shown in equation 10, we can use a single

one if we first affinely transform the equation with an appropriate scaling and translation. The elements of A can be scaled simply by multiplying by a scaling factor: γA. Scaling equation 2 we get:

$$\gamma Ax = \gamma y. \tag{11}$$

However to translate the values of A, we need to create a conformal translation matrix, $T = \{1\}^{m \times n}$, which will serve as a basis for the uniform translation of all the entries of A. A unit translation of A towards $+\infty$ is represented by the sum $A + T$. Translating equation 2 we get:

$$(A+T)x = Ax + Tx = y^{(A)} + y^{(T)}, \text{ where } y_j^{(T)} = \sum_{i=1}^{n} x_i \tag{12}$$

We can translate the entries of A an arbitrary distance δ. We then augment equation 12:

$$(A + \delta T)x = Ax + \delta Tx = y^{(A)} + \delta y^{(T)}, \tag{13}$$

We can now combine equations 11 and 12 to represent the full affine transformation:

$$\gamma(A + \delta T)x = \gamma(Ax + \delta Tx) = \gamma y^{(A)} + \gamma\delta y^{(T)} = \gamma(y^{(A)} + \delta y^{(T)})). \tag{14}$$

Equation 14 can be used to transform any two-valued matrix vector product into a $(0, 1)$-matrix based vector product. For example, a common two valued A is $\{-1, 1\}^{m \times n}$. This can be transformed into a $(0, 1)$-matrix based vector product by setting $\gamma = 2$ and $\delta = -\frac{1}{2}$:

$$\begin{aligned}
\{-1,1\}^{m\times n}x &= \gamma\left(\{0,1\}^{m\times n} + \delta\{1\}^{m\times n}\right)x && (15)\\
&= \gamma(A + \delta T)x && (16)\\
&= \gamma(Ax + \delta Tx) && (17)\\
&= \gamma(y^{(A)} + \delta y^{(T)})) && (18)\\
&= 2(y^{(A)} - \frac{1}{2}y^{(T)})) && (19)\\
&= 2y^{(A)} - y^{(T)})). && (20)
\end{aligned}$$

3.1 Future Work

Our other ideas and projects with respect to the $(0, 1)$-matrix-vector product all involve data-aware algorithms. The following approaches are under development:

- Permute rows and columns to a sparse skyline format
- Precondition the matrix using graph partitioning algorithms such as METIS.
- Apply a MST algorithm to minimize overall Hamming distances – this problem is the same as broadcast routing on a sparse hypercube.
- Identify a hierarchy of common subsequence to form a compressing grammar via SEQUITUR. [5]

4 Conclusion

We demonstrated a differencing method for reducing the number of arithmetic operations within a $(0,1)$-matrix vector product. We then used that method to more efficiently perform Gray code matrix vector products. Then using one or more Gray code matrix vector products, we showed how to reduce the number of arithmetic operations for a general $(0,1)$-matrix vector product. We described an algorithm which would find an optimal application of Gray codes to minimize the number of arithmetic operations. We presented and discussed a set of results from the application of this algorithm to the $(0,1)$-matrix vector product. Finally, we then showed how $(0,1)$-matrix vector products may be applied to the performance of certain restricted classes of more general matrix vector products.

References

[1] Andrew A. Anda. A bound on matrix-vector products for $(0,1)$-matrices via gray codes. In *Proceedings of the 37th Midwest Instruction and Computing Symposium (MICS)*, University of Minnesota, Morris, 2004.

[2] Sándor Dominich. *Mathematical Foundations of Information Retrieval*. Kluwer, Dordrecht, The Netherlands, 2001.

[3] Kenneth H. Rosen, John G. Michaels, Jonathan L. Gross, Jerrold W. Grossman, and Douglas R. Shier, editors. *Handbook of Discrete and Combinatorial Mathematics*. crc, Boca Raton FL, 2000.

[4] Darrell A. TURKINGTON. *Matrix Calculus and Zero-One Matrices*. Cambridge University Press, New York, 2002. Statistical and econometric applications.

[5] Aaron Webb and Andrew A. Anda. $(0,1)$-matrix-vector products via compression by induction of hierarchical grammars. In *Proceedings of the 38th Midwest Instruction and Computing Symposium (MICS)*, University of Wisconsin, Eau Claire, 2005.