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

Andrew A. Anda

aanda@stcloudstate.edu

St. Cloud State University, CSCI Dept.

$(0,1)$-Matrix

A $(0,1)$-matrix (also zero-one or Boolean) is a rectangular matrix for which each element of the matrix has the value of either 0 or 1.

$(0, 1)$-Matrix

A $(0, 1)$-matrix (also zero-one or Boolean) is a rectangular matrix for which each element of the matrix has the value of either 0 or 1. E.g.

$$A \left(\in \{0, 1\}^{3\times 4} \right) = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Application Areas

$(0, 1)$-Matrices arise from problems in a variety of application areas including:

Application Areas

$(0, 1)$-Matrices arise from problems in a variety of application areas including:

- √ adjacency matrices for simple graphs, representing the connectivity relationships between vertices;

Application Areas

$(0,1)$-Matrices arise from problems in a variety of application areas including:

- ✓ term-by-document matrices generated by binary vector space model based computational information retrieval (BVIR) algorithms and applications such as search engines; an element w_{ij} in a weight matrix $W \in \{0,1\}^{n \times m}$ is set to 1 if a term t_j appears in document D_i, with 0 otherwise;

Application Areas

$(0, 1)$-Matrices arise from problems in a variety of application areas including:

- ✓ matrix calculus applications in statistics and econometrics which generate special (0,1)-matrices such as selection, permutation, commutation, elimination, duplication, and shifting matrices;

General Matrix Decomposition

In fact, any general matrix may be decomposed into the linear combination of conformal $(0, 1)$-matrices.

General Matrix Decomposition

In fact, any general matrix may be decomposed into the linear combination of conformal $(0,1)$-matrices. For a general real matrix $A \in \mathbb{R}^{m \times n}$,

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

General Matrix Decomposition

In fact, any general matrix may be decomposed into the linear combination of conformal $(0, 1)$-matrices. For a general real matrix $A \in \mathbb{R}^{m \times n}$,

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

E.g.

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} = a \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + b \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + c \begin{pmatrix} 0 & 0 \\ 1 & 0 \end{pmatrix} + d \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Matrix-Vector Product

A matrix-vector product operation may be represented by the equation

$$Ax = y$$

where the components are conformal: $A \in \{0, 1\}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$.

The vectors may actually be over any algebraic ring for which addition and multiplication is closed and well defined.

Matrix-Vector Product

E.g.

$$\begin{pmatrix} 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = \begin{pmatrix} e \\ f \\ g \end{pmatrix}$$

Matrix-Vector Product

The general matrix-vector product

- √ is an example of BLAS *level 2* operation

Matrix-Vector Product

The general matrix-vector product

- √ is an example of BLAS *level 2* operation
- √ exhibits a quadratic complexity, $\Theta(n^2)$

Matrix-Vector Product

The general matrix-vector product

- is an example of BLAS *level 2* operation
- exhibits a quadratic complexity, $\Theta(n^2)$
- may be blocked for modest performance gains

Matrix-Vector Product

The general matrix-vector product

- √ is an example of BLAS *level 2* operation
- √ exhibits a quadratic complexity, $\Theta(n^2)$
- √ may be blocked for modest performance gains
- √ admits no Strassen method type decomposition to improve the complexity

Matrix-Vector Product

The general matrix-vector product

- √ is an example of BLAS *level 2* operation
- √ exhibits a quadratic complexity, $\Theta(n^2)$
- √ may be blocked for modest performance gains
- √ admits no Strassen method type decomposition to improve the complexity
- √ $\exists < \Theta(n^2)$ algorithms for certain patterned, low rank, or low displacement rank matrices

Matrix-Vector Product

The general matrix-vector product

- √ is an example of BLAS *level 2* operation
- √ exhibits a quadratic complexity, $\Theta(n^2)$
- √ may be blocked for modest performance gains
- √ admits no Strassen method type decomposition to improve the complexity
- √ $\exists < \Theta(n^2)$ algorithms for certain patterned, low rank, or low displacement rank matrices

Hamming Distance

The *Hamming distance* between two bit strings is the count of differing corresponding bits.

Hamming Distance

The *Hamming distance* between two bit strings is the count of differing corresponding bits.
E.g., the two strings,

$$
\begin{array}{cccccccc}
1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
\Uparrow & & & & & & \Uparrow &
\end{array}
$$

have a Hamming distance of *two*.

Gray Codes

A Gray code

- ✓ is a circular ordering of all 2^n binary strings of length n in which adjacent strings differ in exactly one bit.

Gray Codes

A Gray code

- is a circular ordering of all 2^n binary strings of length n in which adjacent strings differ in exactly one bit.
- All adjacent Gray code strings have a Hamming distance of unity.

Gray Codes

A Gray code

- √ is a circular ordering of all 2^n binary strings of length n in which adjacent strings differ in exactly one bit.
- √ All adjacent Gray code strings have a Hamming distance of unity.
- √ One can label the vertices of a *hypercube* with bitstrings such that all *adjacent* neighbors have a bitstring differing in exactly one bit.

Gray Codes

A Gray code

- ✓ is a circular ordering of all 2^n binary strings of length n in which adjacent strings differ in exactly one bit.
- ✓ All adjacent Gray code strings have a Hamming distance of unity.
- ✓ One can label the vertices of a *hypercube* with bitstrings such that all *adjacent* neighbors have a bitstring differing in exactly one bit.
- ✓ Each Gray code corresponds to a *Hamiltonian cycle* about the vertices of a hypercube of a dimension corresponding to the string length.

Differencing Method

$Ax = y$, $A \in \mathbb{R}^{m\times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, is computed via

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

$\Theta(mn)$ additions and multiplications.

Differencing Method

$Ax = y$, $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, is computed via

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

$\Theta(mn)$ additions and multiplications.
A doubly nested loop is required having one of two possible orderings.

Differencing Method

$Ax = y$, $A \in \mathbb{R}^{m \times n}$, $x \in \mathbb{R}^n$, and $y \in \mathbb{R}^m$, is computed via

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

$\Theta(mn)$ additions and multiplications.
A doubly nested loop is required having one of two possible orderings.

We'll select that ordering which performs an inner product ($v^T u = \alpha$)

Differencing Method

For a $(0,1)$-matrix though, $\alpha_{ij} \in \{0,1\}, \forall i, j$.

- √ either the product $\alpha_{ij}x_j$ contributes to the sum as x_j when $\alpha_{ij} = 1$

Differencing Method

For a $(0,1)$-matrix though, $\alpha_{ij} \in \{0,1\}, \forall i,j$.

- ✓ either the product $\alpha_{ij}x_j$ contributes to the sum as x_j when $\alpha_{ij} = 1$
- ✓ or it is skipped, when $\alpha_{ij} = 0$.

Differencing Method

For a $(0,1)$-matrix though, $\alpha_{ij} \in \{0,1\}, \forall i, j$.

- ✓ either the product $\alpha_{ij}x_j$ contributes to the sum as x_j when $\alpha_{ij} = 1$
- ✓ or it is skipped, when $\alpha_{ij} = 0$.
- ✓ $\Rightarrow$ All multiplications are obviated leaving only $\leq mn$ additions.
  (**Note:** *For each row, one addition may be replaced by a load.*)

Differencing Method

Consider computing the difference between two elements of y, y_i and y_k:

$$
\begin{aligned}
y_k - y_i &= \sum_{j=1}^{n} \alpha_{kj} x_j - \sum_{j=1}^{n} \alpha_{ij} x_j \\
&= \sum_{j=1}^{n} \left(\alpha_{kj} x_j - \alpha_{ij} x_j \right) \\
&= \sum_{j=1}^{n} x_j \left(\alpha_{kj} - \alpha_{ij} \right)
\end{aligned}
$$

Differencing Method

However, if y_i has already been computed,

$$
\begin{aligned}
y_k &= y_i + \sum_{j=1}^{n} x_j \left(\alpha_{kj} - \alpha_{ij}\right) \\
&= y_i + \sum_{j=1}^{n} x_j \left(d_j\right), d_j = \alpha_{kj} - \alpha_{ij}
\end{aligned}
$$

Differencing Method

However, if y_i has already been computed,

$$
\begin{aligned}
y_k &= y_i + \sum_{j=1}^{n} x_j \left(\alpha_{kj} - \alpha_{ij}\right) \\
&= y_i + \sum_{j=1}^{n} x_j \left(d_j\right), d_j = \alpha_{kj} - \alpha_{ij}
\end{aligned}
$$

$\Rightarrow$ each subsequent element of y can be computed as the sum of a previously computed element with the inner product of x and the difference vector of the two rows of A, d.

Differencing Method

Then we have saved $\|A_k\|_1 - \|d\|_1 - 1$ operations in computing y_k, where $\|d\|_1$ is the *Hamming distance* between two rows of A.

Differencing Method

Then we have saved $\|A_k\|_1 - \|d\|_1 - 1$ operations in computing y_k, where $\|d\|_1$ is the *Hamming distance* between two rows of A.

For a duplicated row in A, load its previously computed value at the expense of no additions.

Differencing Method

If the remaining rows are unique, what's the lowest number of additions that we need?

Differencing Method

If the remaining rows are unique, what's the lowest number of additions that we need?
There must be a Hamming distance of at least one between each row of A and each other row.

Differencing Method

If the remaining rows are unique, what's the lowest number of additions that we need?
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 for there to be, for every row of A, some other row of A of unit Hamming distance. We have already defined a sequence of Boolean vectors which satisfies this condition,

Differencing Method

If the remaining rows are unique, what's the lowest number of additions that we need?
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 for there to be, for every row of A, some other row of A of unit Hamming distance. We have already defined a sequence of Boolean vectors which satisfies this condition,

a Gray code.

Differencing Method

- A Gray code q bits wide consists of a sequence of $r = 2^q$ bitstrings.

Differencing Method

- A Gray code q bits wide consists of a sequence of $r = 2^q$ bitstrings.
- Using the differencing method above, a $r \times q$ Gray code matrix-vector product can be computed in a maximum of r additions.

Differencing Method

- A Gray code q bits wide consists of a sequence of $r = 2^q$ bitstrings.
- Using the differencing method above, a $r \times q$ Gray code matrix-vector product can be computed in a maximum of r additions.
- 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.

Differencing Method

If we have a general $(0,1)$-matrix matrix having q columns and any number of rows,

- ✓ we can first precompute a scratch y using the Gray code Matrix vector product with x.

Differencing Method

If we have a general $(0,1)$-matrix matrix having q columns and any number of rows,

- √ we can first precompute a scratch y using 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 bitpattern for that row.

Differencing Method

If we have a general $(0, 1)$-matrix matrix having q columns and any number of rows,

- ✓ we can first precompute a scratch y using 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 bitpattern for that row.
- ✓ The only additions are from the precomputation of the scratch y from the Gray code matrix. If $m > r$, we have a greater reduction in the maximum number of additions for the Gray code matrix itself.

Differencing Method

If $n > q$, one of two conditions can hold:

- ✓ $n \bmod q = 0$ in which case we will use n/q Gray code matrix stripes across A with a subsequent sum across the stripes for all m.

Differencing Method

If $n > q$, one of two conditions can hold:

- √ $n \bmod q = 0$ in which case we will use n/q Gray code matrix 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.

Differencing Method

If we define $w(q, m, n)$ as the number of additions required for a general $(0, 1)$-matrix. Then for an arbitrary m and n, ignoring the $q + 1$ savings from above,

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

The q which minimizes w may be computed by searching for the smallest w for a reasonably small set of q's bounded by $\lceil \lg m \rceil$.

(**Note:**w *is discontinuous – when searching, beware of local minima.*)

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

Computed Optimum Gray Code 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	1.1e1	8.0e1	4.1e2	2.0e3	1.0e4	5.2e4	2.6e5	1. 3e6	6.5e6	3.2e7
5^2	2.8e1	2.2e2	1.1e3	5.7e3	2.8e4	1.4e5	7.2e5	3. 6e6	1.8e7	9.0e7
5^3	2.8e1	6.3e2	3.6e3	1.8e4	9.4e4	4.7e5	2.3e6	1. 1e7	5.9e7	2.9e8
5^4	2.8e1	2.2e3	1.2e4	6.6e4	3.3e5	1.6e6	8.3e6	4. 1e7	2.0e8	1.0e9
5^5	2.8e1	7.3e3	4.7e4	2.5e5	1.2e6	6.3e6	3.1e7	1. 5e8	7.8e8	3.9e9
5^6	2.8e1	2.7e4	1.9e5	9.9e5	5.0e6	2.5e7	1.2e8	6. 2e8	3.1e9	1.5e10
5^7	2.8e1	9.0e4	7.6e5	4.1e6	2.0e7	1.0e8	5.1e8	2. 5e9	1.3e10	6.4e10
5^8	2.8e1	4.0e5	3.2e6	1.7e7	8.8e7	4.4e8	2.2e9	1. 1e10	5.5e10	2.7e11
5^9	2.8e1	1.9e6	1.3e7	7.5e7	3.8e8	1.9e9	9.5e9	4. 7e10	2.3e11	1.2e12
5^{10}	2.8e1	9.7e6	6.0e7	3.2e8	1.6e9	8.4e9	4.2e10	2.11e11	1.06e12	5.28e12

Results Observations

The tabular data show that:

- √ For a fixed number of rows, as the number of columns increases, the ratio compared to dense approaches a constant.

Results Observations

The tabular data show 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.

Results Observations

The tabular data show 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 a square matrix increases, the ratio compared to dense continues to improve.

Results Observations

The tabular data show 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 a square matrix increases, the ratio compared to dense continues to improve.

Results Discussion

- This algorithm is *data oblivious*, i.e. the number of operations is constant regardless of the distribution of elements of the $(0,1)$-matrix.

Results Discussion

- This algorithm is *data oblivious*, i.e. the number of operations is constant regardless of the distribution of elements of the $(0, 1)$-matrix.
- For this reason, some other algorithm which exploits sparsity may prove superior for highly sparse $(0, 1)$-matrices.

Results Discussion

- ✓ This algorithm is *data oblivious*, i.e. the number of operations is constant regardless of the distribution of elements of the $(0, 1)$-matrix.
- ✓ For this reason, some other algorithm which exploits sparsity may prove superior for highly sparse $(0, 1)$-matrices.
- ✓ However, data-obliviousness is beneficial in such applications as cryptography.

Results Discussion

- This algorithm is *data oblivious*, i.e. the number of operations is constant regardless of the distribution of elements of the $(0, 1)$-matrix.
- For this reason, some other algorithm which exploits sparsity may prove superior for highly sparse $(0, 1)$-matrices.
- However, data-obliviousness is beneficial in such applications as cryptography.
- In the case of multiple right-hand-side vectors, the complexity is a strict linear factor of the number of right-hand-side vectors.

Results Discussion

- This algorithm is *data oblivious*, i.e. the number of operations is constant regardless of the distribution of elements of the $(0, 1)$-matrix.
- For this reason, some other algorithm which exploits sparsity may prove superior for highly sparse $(0, 1)$-matrices.
- However, data-obliviousness is beneficial in such applications as cryptography.
- In the case of multiple right-hand-side vectors, the complexity is a strict linear factor of the number of right-hand-side vectors.

Future Work

The following alternate approaches are under development:

- Permute rows and columns to a sparse skyline format

Future Work

The following alternate approaches are under development:

- ✓ Permute rows and columns to a sparse skyline format
- ✓ Precondition the matrix using graph partitioning algorithms such as METIS.

Future Work

The following alternate 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.

Future Work

The following alternate 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 compression grammar via SEQUITUR.

Future Work

The following alternate 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 compression grammar via SEQUITUR.

Exploiting $(0, 1)$-Mv Products

The initial linear combination equation 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.

Exploiting $(0, 1)$-Mv Products

The initial linear combination equation 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.

Exploiting $(0,1)$-Mv Products

We then get the equation

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

Where there are k distinct entries, β_i in A and each B_i represents a distinct $(0,1)$-matrix.

Exploiting $(0,1)$-Mv Products

For a single β, the matrix vector product can be written as:

$$Ax = y = \beta Bx = B(\beta x),$$

The scalar product has been reduced from $\Theta(mn)$ to $\Theta(n)$.

Exploiting $(0, 1)$-Mv Products

For a single β, the matrix vector product can be written as:

$$Ax = y = \beta Bx = B(\beta x),$$

The scalar product has been reduced from $\Theta(mn)$ to $\Theta(n)$.
The previous two equations may be combined:

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

Exploiting $(0, 1)$-Mv Products

Lets consider the case where $k = 2$.

- ✓ The elements of the matrix A are either of only two values, $\alpha_{ij} \in \{\beta_1, \beta_2\}$.

Exploiting $(0, 1)$-Mv Products

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.

Exploiting $(0, 1)$-Mv Products

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.
- ✓ Rather than use a pair of $(0, 1)$-matrices, 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 getting $\gamma A x = \gamma y$.

Exploiting $(0, 1)$-Mv Products

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 $A + T$.
We get $(A + T)x = Ax + Tx = y^{(A)} + y^{(T)}$,
where $y_j^{(T)} = \sum_{i=1}^{n} x_i$.

Exploiting $(0, 1)$-Mv Products

Translating the entries of A an arbitrary distance δ.
We then augment the above equation getting:
$(A + \delta T)x = Ax + \delta Tx = y^{(A)} + \delta y^{(T)}$.

Exploiting $(0,1)$-Mv Products

Translating the entries of A an arbitrary distance δ. We then augment the above equation getting: $(A + \delta T)x = Ax + \delta Tx = y^{(A)} + \delta y^{(T)}$. The above equations may now be combined to represent the full affine transformation:

$$\gamma(A + \delta T)x = \gamma(Ax + \delta Tx)$$

$$= \gamma y^{(A)} + \gamma\delta y^{(T)} = \gamma(y^{(A)} + \delta y^{(T)}))$$

Exploiting $(0, 1)$-Mv Products

This equation can be used to transform any two-valued matrix vector product into a $(0, 1)$-matrix based vector product.

Exploiting $(0,1)$-Mv Products

This equation can be used to transform any two-valued matrix vector product into a $(0,1)$-matrix based vector product.

E.g., a common two valued A is $\{-1,1\}^{m\times n}$.

Exploiting $(0, 1)$-Mv Products

This equation can be used to transform any two-valued matrix vector product into a $(0, 1)$-matrix based vector product.

E.g., 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 $\delta = -\frac{1}{2}$:

Exploiting $(0,1)$-Mv Products

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

Conclusion

- √ We were able to demonstrate a differencing method for reducing the number of arithmetic operations within a $(0, 1)$-matrix vector product.

Conclusion

- ✓ We were able to demonstrate a differencing method for reducing the number of arithmetic operations within a $(0,1)$-matrix vector product.
- ✓ We used that method to more efficiently perform Gray code matrix vector products.

Conclusion

- √ 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.

Conclusion

- √ 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 then showed how $(0, 1)$-matrix vector products may be applied to the performance of certain restricted classes of more general matrix vector products.