Matrix compression by common subexpression elimination

Tomas Oppelstrup

November 11, 2009

Abstract

In this report a method for common subexpression elimination in matrices is explored. The method is applied to several types of matrices occurring in numerical simulations. In all cases, the cost of a matrix-vector multiplication is reduced by a significant amount. The amount of storage required for the eliminated matrices is also less than that required for the original matrices. When the proposed method is applied to the Fourier transform matrix, the output is equivalent to the fast Fourier transform. For some matrices used in the fast multipole method for dislocation dynamics, the cost of a matrix-vector multiplication is reduced from $O(p^6)$ to $O(p^{4.5})$, where $p$ is the expansion order. Using an expansion order of 5, one can expect a factor of four speedup of the fast multipole part of a dislocation dynamics code.

1 Introduction

Many matrices used in scientific computing exhibit symmetries and repeated patterns. If these can be realized and exploited, significant savings in storage and computer time can be made. One of the most commonly used and successful examples of this is the fast Fourier transform. An implementation closely following the definition of the discrete Fourier transform requires $O(N^2)$ operations for transforming a vector of length $N$. In contrast, the fast Fourier transform needs only $O(N \log N)$ operations.

The intent of this report is to develop a black box algorithm which analyzes a given matrix and yields a representation that requires less storage for the matrix, and less computer time for a matrix-vector multiplication. The motivation is that there are many cases of matrices with similarities which have not been studied by hand, and thus no efficient multiplication algorithm exists. This work is in fact the result of an attempt to optimize certain linear operators with unknown structure that are used in the fast multipole method for dislocations, developed in [6]. The results for these matrices were quite impressive, and are discussed in the results section.

The method developed in this report should be thought of as a preprocessor. It is computationally costly to perform, and therefore most useful in cases where the
matrix is used (i.e. multiplied with a vector) very many times. Two examples of this are how the fast Fourier transform and its inverse are used in every timestep in a time dependent fast Poisson solver, and how the far field to near field conversion operators are used in the fast multipole algorithm [4, 6].

The main idea in the report is to find repeated patterns in matrices, and represent them in a computationally efficient way. When it comes to evaluating mathematical expressions, repeated patterns are usually called common subexpressions, and the process of optimizing a computation with respect to them is called common subexpression elimination.

Common subexpression elimination is performed by optimizing compilers. It is also a technique used in the design of complex integrated circuits, or more precisely VLSI design, to eliminate the use of multipliers and reduce the number of adders needed for hardware implemented filters and transforms [1, 2]. The algorithm explored in this report is similar to that used in [2], but here arbitrary matrices and scaling factors are used, rather than binary linear operators and scaling factors of ±1.

2 Compression algorithm

The types of patterns that have been considered are rank one submatrices of the input matrix.

By a submatrix $S$ of a matrix $A$ is meant a matrix formed in the following way: Take an ordered set of row indices $P$ and an ordered set of column indices $Q$ of the input matrix $A$. Then $S$ is defined by $S_{ij} = A_{P_iQ_j}$, where $i = 1, \ldots, |P|$ and $j = 1, \ldots, |Q|$.

A rank one matrix can be described by $uv^T$, where $u$ and $v$ are column vectors. One of the elements in $u$ can always be taken to be one. Applying $uv^T$ to a vector costs $|u| + |v| - 1$ multiplications and $|u| + |v| - 2$ additions, where $|x|$ denotes the number of elements in the vector $x$. In comparison, multiplying a full $N \times N$ matrix by a vector requires $N^2$ multiplications and $N(N - 1)$ additions. Thus even for a $2 \times 2$ matrix of rank one, a special representation is computationally more efficient than the standard representation.

In this report, submatrices of a given matrix are changed to a more efficient representation. The concept can be visualized by a simple example:

$$
\begin{pmatrix}
a & b & c \\
d & e & f \\
4a & 4b & g
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z
\end{pmatrix}
= \left\{ t := x + \frac{b}{a}y \right\} =
\begin{pmatrix}
0 & 0 & c & a \\
d & e & f & 0 \\
0 & 0 & g & 4a
\end{pmatrix}
\begin{pmatrix}
x \\
y \\
z \\
t
\end{pmatrix}.
$$

The submatrix $\begin{pmatrix} a & b \\ 4a & 4b \end{pmatrix}$ is of rank one. A temporary variable $t := x + \frac{b}{a}y$ is introduced. It is defined by the action of the first row of the submatrix acting on the vector, and scaled so that the coefficient in front of $x$ is one. In the original matrix the submatrix has been set to zero, and its action is instead represented by a newly
introduced column. It acts on the temporary variable which has been appended to the input vector. To realize the savings the new matrix is stored in a sparse format, so that only the non-zero elements are used in computations. Applying the matrix to a vector consists of the following steps: First, all temporary variables are evaluated. Each such evaluation has a cost of one multiplication and one addition. Then the values of the temporary variables are appended to the input vector. Finally, the sparse matrix is multiplied with the augmented input vector. We turn this idea into an iterative algorithm: In each iteration the largest rank one submatrix is replaced by a temporary variable.

2.1 Implementation

The above algorithm can be implemented in different ways. This section describes briefly the implementation used to produce the results in this report. The implementation looked for $k \times 2$ submatrices of rank one. In order to do that all pairs of columns need to be considered. Note that if there is a $k \times 3$ submatrix of rank one, then part of it will first be found as a $k \times 2$ submatrix. Its new representation together with the unprocessed column of the $k \times 3$ submatrix will lead to another $k \times 2$ submatrix. Therefore $k \times p$ submatrices will be exploited iteratively. The resulting representation has no computational disadvantage compared to discovering the whole submatrix in one step. The algorithm uses a sparse representation for the matrix throughout the search for rank one submatrices. This saves on the number of comparisons, especially for matrices where a large fraction of the elements get replaced by temporaries and matrices that are initially sparse.

The number of operations needed to apply the algorithm described above to an $N \times N$ matrix is in the worst case $O(\rho^2 N^2 (N + N_{\text{temp}})^2)$, where $\rho$ is the fraction of non-zero elements in the input matrix, and $N_{\text{temp}}$ is the number of temporary variables generated. In the worst case, there are $O(N^2)$ temporary variables to be generated, in which case the total running time cost is $O(N^6)$. One immediate way to improve on that is by looking for rank one submatrices of size $k \times p$ rather than $k \times 2$.

3 Results

The compression scheme described above has been applied to a various matrices occurring in scientific computing. It has been applied to test cases with known optimal results, as well as to matrices with unknown optimal multiplication algorithms. The test cases used were the standard complex Fourier transform, and a real symmetric sine transform. Other applications are matrices occurring in the fast multipole method for dislocation dynamics and random binary matrices. The results are presented in the following sections.
Figure 1: Performance after compression of the Fourier transformation matrix defined by (3.1).

3.1 The Fourier transform matrix

When the suggested compression scheme is applied to the $N \times N$ Fourier transform matrix $F$, with

$$F_{kj} = \frac{1}{\sqrt{N}} e^{-i\frac{2\pi}{N}jk},$$

then the resulting output matrix is equivalent to the fast Fourier transform [3], and the application of the compressed $F$ costs only $O \left( \left( \frac{P}{\log P} \right) N \log N \right)$ operations, where $P$ is the largest prime factor of $N$. One can show that the proposed compression scheme will discover the fast Fourier transform (FFT). For a proof, see appendix A.

Figure 1 shows the number of operations required for application of the compressed $F$ for various input sizes $N$. The number of operations is divided by $N \log 2 N$ (which is proportional to the number of operations required for an FFT when $N$ is a power of 2). We can see that the lower envelope is a constant line. On that line we will find $N$ that are powers of 2. Close to that line are $N$ comprised only of small prime factors. The upper envelope is proportional to $N/\log_2 N$. On that curve we find $N$ which are prime numbers. For prime $N$, the algorithm does not offer any compression.

3.2 A real symmetric sine transform

As another test case, the compression algorithm has been applied to the type III sine transform, with

$$F_{kj} = \frac{1}{N} \sin \left( \frac{\pi}{N}(j+1)(k+1) \right).$$
The number of operations after compression becomes proportional to $O(N^{3/2})$ for the most favorable cases, which happen to be $N = 24k - 1$, for positive integers $k$. Compression results for the sine transform are displayed in Figure 2. The number of operations required after compression divided by $N^{3/2}$ is plotted against $N$. This result is not optimal, since the sine transform can be computed in $N \log N$ operations using FFT’s or specialized algorithms.

3.3 Matrices in the fast multipole method

Dislocation dynamics [5, 6] is a method used to study plasticity and defects in materials. The paper [6] describes a fast multipole method [4] for the long range interaction between dislocations. The multipole method is used to evaluate the stress field resulting from the dislocations, and in the following, multipole and Taylor expansions are assumed to be expansions of the stress field in the spatial coordinates. The computationally most expensive part is the transformation of multipole expansions (far field expansions) to Taylor expansions (near field expansions). The transformation is a linear operator and can therefore be described by a matrix that maps multipole coefficients to Taylor expansion coefficients.

A detailed description of the transformation is given in Appendix B of [6]. Here a very brief definition follows. We will use the Einstein summation convention, so that an index occurring more than once in a factor is summed from 1 to 3. We let $x_1, x_2, x_3$ be coordinates in a right-handed Cartesian coordinate system, and denote the corresponding coordinate vector $\mathbf{x} = (x_1, x_2, x_3)$. We also define $R(\mathbf{x}) = |\mathbf{x}| = \sqrt{x_1^2 + x_2^2 + x_3^2}$, and its derivatives $R_i = \frac{\partial}{\partial x_i} R$, $R_{ij} = \frac{\partial}{\partial x_i} \frac{\partial}{\partial x_j} R$, etc. $\mu$ and $\nu$ are
material related constants (the bulk modulus and Poisson’s ratio, respectively).

The multipole transformation used in the dislocation dynamics implementation described in [6], transforms a multipole expansion in $1/R = 1/\sqrt{x_1^2 + x_2^2 + x_3^2}$ to a Taylor expansion in $x_1, x_2, x_3$. Let the multipole coefficients that define the stress field far away from the origin be denoted by $\eta$. The definitions of the near-field Taylor expansions and far-field to near-field transformation are as follows.

First we define the Taylor polynomial that represents the stress tensor $\sigma_{ij}(x')$ at a point $x'$ in a neighborhood of some point $x$ sufficiently far away from the origin for the multipole expansion to be valid there. It is defined as follows:

$$\sigma_{ij}(x') = \sum_{\zeta=0}^{\infty} \frac{(-1)\zeta}{\zeta!} \sum_{\alpha=0}^{\zeta} \sum_{\beta=0}^{\zeta-\alpha} (x'_1 - x_1)^\alpha (x'_2 - x_2)^\beta (x'_3 - x_3)^\gamma \tau_{ij}^{\alpha\beta\gamma}(x),$$

where $\tau$ is the Taylor expansion coefficients, and $\gamma = \zeta - \alpha - \beta$. Next we define the transformation which transforms the $\eta$ to $\tau$. In order to do that we let the intermediate multi-index object $G$ be

$$G_{ijklm}^{\alpha\beta\gamma} = \sum_{\xi=0}^{\infty} \sum_{\alpha=0}^{\xi-a} \sum_{\beta=0}^{\xi-a} \sum_{\gamma=0}^{\xi-a-\beta} M_{abc}^{\alpha\beta\gamma} \eta_{abc}^{\alpha\beta\gamma} \partial_{x_1}^\alpha \partial_{x_2}^\beta \partial_{x_3}^\gamma R_{klm}(x),$$

with $c = \alpha - \beta$, and $M_{abc}^{\alpha\beta\gamma}$ is a combinatorial factor defined by

$$M_{n1n2n3} = \frac{(n_1 + n_2 + n_3)!}{n_1! n_2! n_3!}.$$

Finally we write the transformation as

$$T_{ij}^{\alpha\beta\gamma} = \frac{\mu}{8\pi} \left[ \epsilon_{jmn} G_{nimpp}^{\alpha\beta\gamma} + \epsilon_{imn} G_{njmpp}^{\alpha\beta\gamma} + \frac{2\epsilon_{kmn}}{1-\nu} (G_{nkijm}^{\alpha\beta\gamma} - \delta_{ij} G_{nkmp}^{\alpha\beta\gamma}) \right].$$

Here, $\epsilon_{ijk}$ is the permutation symbol, and $\delta_{ij}$ the Kronecker delta.

If the multipole expansion order is $p$ and the Taylor expansion order is $u$, then the transformation matrix has dimensions $9(p+1)(p+2)(p+3)/6 \times 9(u+1)(u+2)(u+3)/6$. For the case $x = (0,0,1)$, the transformation matrix has about 20% non-zero elements. With $u = p$ or $u = 2p$, the number of non-zero matrix elements is proportional to $p^6$. After compression of the $x = (0,0,1)$ operator, the number of operations for application of the transformation matrix seems to be proportional to $p^{4.5}$. This is shown in the left panel of Figure 3. In that figure, the number of operations required after compression divided by $p^{6.5}$ is plotted against $p$. The two cases of $u = p$ and $u = 2p$ are shown. The dimensions of the input matrices for $p = 8, u = 8$ and $p = 7, u = 14$ are $1485 \times 1485$ and $6120 \times 1080$ respectively.

### 3.3.1 Total gain in a multipole scheme by compression

Here we will briefly analyze the overall speedup of a multipole algorithm resulting from increasing the performance of the far-to-near field conversion operator. We assume that there are $N$ particles (or dislocations).
Figure 3: Compression of fast multipole matrices used in dislocation dynamics simulations. (Left panel) The number of operations after compression divided by the total number of elements in the input matrix raised to the power of 3/4 is plotted against the expansion order. The results are shown for $\alpha = (0, 0, 1)$. (Right panel) Net speedup of a complete far-field to near-field conversion step, involving all 189 conversion operators.

A multipole method consists of five steps. The particles are located in a hierarchy of boxes. If there are $k$ boxes on the finest grid level, the total number of boxes is approximately $\frac{5}{2}k$. In the first step, a multipole expansion is computed at the cost $c_M$ per particle. In the second step, multipole expansions for higher level boxes are created by applying translation operators to expansions of lower level boxes, this has the cost $c_{M2M}$ per box. The third step consists of transforming multipole (far field) expansions into near field local expansions. In this step each box interacts with 189 neighboring boxes. The cost of this step is $189c_{M2L}$ per box. In the fourth step, local expansions are propagated downwards in the box hierarchy by applying local expansion translation operators, this has the cost $c_{L2L}$ per box. In the fifth step particle forces are computed by evaluating a local expansion for each particle and computing direct interaction with the particles in the nearest neighboring boxes, this has the cost $27Nc_D/2k + c_L$ per particle, where $k$ is the number of boxes on the finest level.

The total cost per particle of one application of the multipole algorithm is approximately

$$c(N, k) = N(c_M + c_L) + \frac{27N}{2k}c_D + \frac{8k}{7}(c_{M2M} + c_{L2L} + 189c_{M2L}).$$

The creation and evaluation operators $c_M$ and $c_L$ are in the dislocation case of complexity order $p^3$ where $p$ is the expansion order, and the translation and conversion operators $c_{M2M}$, $c_{M2L}$, and $c_{L2L}$. are of order $p^6$. The majority of the computational cost is therefore carried by the direct evaluation and the conversion of multipole expansions to Taylor expansions. To analyze the total cost we use the simplified cost
function

\[ C(N, k) = \frac{27N}{2k} + 216k\gamma, \]

where \( \gamma = \frac{c_{M2L}}{c_D} \) and \( C \) is the total cost of the multipole scheme relative to the cost of evaluating one direct pair-wise force or energy.

We optimize \( C \) with respect to \( k \), and find the minimum where \( k = \frac{N}{4\sqrt{\gamma}} \). For the optimal \( k \), the total cost is \( 108N\sqrt{\gamma} \).

In the fast multipole method for dislocations, there are 189 far-field to near-field conversion operators. The different operators experience exhibit different levels of compression. The right panel of Figure 3 shows the speedup for a complete transformation step including all 189 operators, as a function of expansion order. For a multipole expansion order of \( p = 5 \) and a Taylor expansion order of \( u = 10 \), the speedup of the multipole to Taylor expansion conversion step is estimated to be about four (see the right panel of Figure 3), which means a theoretical speedup of a factor of two for the entire multipole scheme.

### 3.4 Random binary matrices

One may expect that large matrices containing only small integers contain some rank one submatrices by chance. To illustrate this, the compression algorithm was tested on random binary matrices, i.e. matrices containing only ones and zeros, and with the locations of the ones being uniformly random. Figure 4 shows the compression ratio of 200 \( \times \) 200 random binary matrices as a function of \( \rho \), where \( \rho \) is the fraction of non-zeros. The compression ratio is defined as the number of additions required before the compression divided by the number required after compression.

### 4 Discussion

As we have seen, a reduction of the number of required operations can be achieved with the proposed matrix compression scheme. This has been demonstrated in a number of different cases. For the Fourier transform we obtain an essentially optimal result. For the sine transform, the reduction is substantial, but not optimal (since sine transforms can be done with \( O(N \log N) \) by use of FFT’s or specially developed algorithms. In the multipole case the optimal number of operations is unknown, but the savings are substantial. The main idea of this work is to illustrate that a black box method can achieve vast savings in computational times without special algorithms or analyzing the structure of the transformation matrix.

There may of course be many other ways of exploiting patterns in matrices, for instance singular value decomposition compression\(^1\) \([7, 8]\). The scheme presented here is different in that it searches submatrices, rather than the whole matrix at once. It is also exact. One is of course not bound to look for only submatrices of rank one, but can in principle use any rank, for instance 4 \( \times \) 4 submatrices of rank

\(^1\)These methods are generally not exact, but reproduces the action of the matrix with a controllable maximum relative error.
two. For a brute force search algorithm this is very computationally expensive, but may offer a way of improving the current method.

5 Acknowledgements

I thank professor Olof Runborg for a good discussion, which resulted in a greatly improved quality and readability of this report.

A Proof that the algorithm finds the fast Fourier transform

Let’s consider the discrete Fourier transform of a vector of length \( n \). The transform is a linear operator which can be represented by the matrix \( F_{jk} = e^{-i \frac{2\pi}{n} jk} \), where the indices \( j \) and \( k \) are in the range \( 0, \ldots, n - 1 \). Now, assume that \( n \) can be factorized into \( n = pm \), where \( p \) is the smallest prime factor of \( n \).

In applying the compression algorithm to the matrix, we first need to find the rank one submatrices with the largest number of rows. Those can be identified as follows: Decompose the indices \( j \) and \( k \) into \( j = j_0 + \frac{n}{p} j_1 \) and \( k = k_0 + pk_1 \). Consider

Figure 4: Compression ratio of random binary matrices of size 200 × 200 and various densities of ones (non-zero elements).
the matrix elements

\[
F_{k=k_0+k_1p,j=j_0+j_1p} = e^{-i\frac{2\pi}{n}(j_0+j_1p)k} = e^{-i\frac{2\pi}{p}j_1k}e^{-i\frac{2\pi}{n}j_0k} = (e^{-i\frac{2\pi}{p}k})^{j_1}(e^{-i\frac{2\pi}{n}j_0})^k = \sigma_{j_0}^{j_1}k^0
\]

where \(j_0, k_1 = 0, \ldots, \frac{n}{p} - 1, j_1, k_0 = 0, \ldots, p - 1\) and \(k = 0, \ldots, n - 1\). For each choice of \((j_0, k_0)\), we get a rank one matrix of size \(\frac{n}{p} \times p\).

In order to show that the found submatrices are the largest ones, we proceed as follows. Assume that some elements given by a row index set \(K\) of column \(j_0\) and \(j_1\) form a rank one submatrix. We then have

\[
e^{-i\frac{2\pi}{n}j_0k} = e^{-i\frac{2\pi}{n}j_1k}\alpha \Rightarrow e^{i\frac{2\pi}{n}(j_1-j_0)k} = \alpha \quad \forall k \in K
\]

for some constant \(\alpha\). Now we try to maximize the number of \(k\)-values for which this relation holds. The number of distinct values of \(e^{i\frac{2\pi}{n}(j_1-j_0)k}\) for integral \(k\) is \(n/\gcd(n, j_1-j_0)\). This is minimized by choosing \(j_1-j_0\) to be the largest factor of \(n\). In the rank one submatrices described above, \(p\) was the smallest prime factor of \(n\), and therefore \(n/p\) is the largest factor of \(n\). The rank one submatrices derived above are identical to the maximal ones found here, and therefore the submatrices that the algorithm would replace.

All elements of \(F\) belong to one of the independent rank one submatrices described above. Therefore, all elements will be removed (set to zero), and the complete action of the matrix is represented by the newly introduced columns.

After the replacement of the matrix elements by the temporaries, each row \(i\) will have \(n/p\) non-zeros utilizing the temporaries \(i + kp \mod n\). The matrix will thus have a checkerboard pattern with \(p\) totally independent components (e.g. no rank one sub-matrix has elements in more than one component), each of size \(\frac{n}{p} \times \frac{n}{p}\). A component \(F(q)\) \((q = 0, \ldots, p - 1)\) has the elements 

\[
F_{jk}(q) = \exp[-i\frac{2\pi}{n}(jkp + qk)]
\]

Each of the components is therefore in itself a Fourier transform matrix for vectors of length \(n/p\) multiplied to the right with a diagonal matrix (i.e. column scaled). The column scaling does not change the rank-one submatrix structure, since \(uw^TD = u(v^TD) = uw^T\), with \(w^T = v^TD\) being a row vector. Therefore, by recursion, the algorithm will continue to find rank one submatrices until \(q = n/(p_1p_2 \cdots)\) is prime. The resulting matrix then consists of a set of independent size \(q\) discrete Fourier transform matrices, which acts on the temporaries generated by the compression algorithm. Computing the temporaries is equivalent to applying twiddle factors in conventional FFT.
B A worked example: The $8 \times 8$ Fourier transform matrix

Let $F$ be the Fourier transform matrix for vectors of length 8. We define $w = \exp\left(-i\frac{2\pi}{8}\right)$, and have $w^4 = -1$ and $w^8 = 1$. The size 8 discrete Fourier transform $\bar{y}$ of a vector $\bar{x}$ is $\bar{y} = F\bar{x}$, or explicitly:

$$
\begin{bmatrix}
y_1 \\
y_2 \\
y_3 \\
y_4 \\
y_5 \\
y_6 \\
y_7 \\
y_8 \\
\end{bmatrix} =
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & w & w^2 & w^3 & -1 & -w & -w^2 & -w^3 \\
1 & w^2 & -1 & -w^2 & 1 & w^2 & -1 & -w^2 \\
1 & w^3 & -w^2 & w & -1 & -w^3 & w^2 & -w \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -w & w^2 & -w^3 & -1 & w & -w^2 & w^3 \\
1 & -w^2 & -1 & w^2 & 1 & -w^2 & -1 & w^2 \\
1 & -w^3 & -w^2 & -w & -1 & w^3 & w^2 & w \\
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7 \\
x_8 \\
\end{bmatrix}.
$$

The maximum rank one submatrices found in $F$ are:

$$
\begin{bmatrix}
F_{11} & F_{15} \\
F_{31} & F_{35} \\
F_{51} & F_{55} \\
F_{71} & F_{75} \\
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\begin{bmatrix}
1 & 1 \\
1 & 1 \\
1 & 1 \\
1 & 1 \\
\end{bmatrix},
$$

$$
\begin{bmatrix}
F_{21} & F_{25} \\
F_{41} & F_{45} \\
F_{61} & F_{65} \\
F_{81} & F_{85} \\
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\begin{bmatrix}
1 & -1 \\
1 & -1 \\
1 & -1 \\
1 & -1 \\
\end{bmatrix}.
$$

$$
\begin{bmatrix}
F_{12} & F_{16} \\
F_{32} & F_{36} \\
F_{52} & F_{56} \\
F_{72} & F_{76} \\
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\begin{bmatrix}
w^2 \\
w \\
w^3 \\
w^3 \\
\end{bmatrix},
$$

$$
\begin{bmatrix}
F_{22} & F_{26} \\
F_{42} & F_{46} \\
F_{62} & F_{66} \\
F_{82} & F_{86} \\
\end{bmatrix} =
\begin{bmatrix}
w \\
w \\
w \\
w \\
\end{bmatrix}
\begin{bmatrix}
1 & -1 \\
1 & -1 \\
1 & -1 \\
1 & -1 \\
\end{bmatrix}.
$$

$$
\begin{bmatrix}
F_{13} & F_{17} \\
F_{33} & F_{37} \\
F_{53} & F_{57} \\
F_{73} & F_{77} \\
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\begin{bmatrix}
w^2 \\
w^2 \\
w^2 \\
w^2 \\
\end{bmatrix},
$$

$$
\begin{bmatrix}
F_{23} & F_{27} \\
F_{43} & F_{47} \\
F_{63} & F_{67} \\
F_{83} & F_{87} \\
\end{bmatrix} =
\begin{bmatrix}
w^2 \\
w^2 \\
w^2 \\
w^2 \\
\end{bmatrix}
\begin{bmatrix}
1 & -1 \\
1 & -1 \\
1 & -1 \\
1 & -1 \\
\end{bmatrix}.
$$

$$
\begin{bmatrix}
F_{14} & F_{18} \\
F_{34} & F_{38} \\
F_{54} & F_{58} \\
F_{74} & F_{78} \\
\end{bmatrix} =
\begin{bmatrix}
1 \\
1 \\
1 \\
1 \\
\end{bmatrix}
\begin{bmatrix}
w^3 \\
w \\
w \\
w \\
\end{bmatrix},
$$

$$
\begin{bmatrix}
F_{24} & F_{28} \\
F_{44} & F_{48} \\
F_{64} & F_{68} \\
F_{84} & F_{88} \\
\end{bmatrix} =
\begin{bmatrix}
w^3 \\
w \\
w \\
w \\
\end{bmatrix}
\begin{bmatrix}
1 & -1 \\
1 & -1 \\
1 & -1 \\
1 & -1 \\
\end{bmatrix}.
$$

and we can immediately find the first 8 temporaries $t_1 \ldots t_8$ to be:

$$
t_1 := x_1 + x_5 \\
t_2 := x_1 - x_5 \\
t_3 := x_2 + x_6 \\
t_4 := x_2 - x_6 \\
t_5 := x_3 + x_7 \\
t_6 := x_3 - x_7 \\
t_7 := x_4 + x_8 \\
t_8 := x_4 - x_8.
$$
After these are computed, the first step of the matrix compression turns the operator into:

\[
\begin{bmatrix}
  y_1 \\
  \vdots \\
  y_8
\end{bmatrix} =
\begin{bmatrix}
  1 & 1 & 1 & w^{2} & 1 & w^{3} \\
  1 & w^{2} & w^{3} & -1 & -w^{2} & w \\
  1 & 1 & -w & 1 & 1 & -w \\
  1 & -w^{2} & -w & 1 & -w^{2} & -w
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  \vdots \\
  x_8 \\
  t_1 \\
  \vdots \\
  t_8
\end{bmatrix}.
\]

If we extract the two independent components in the checkerboard pattern in the operator, we have:

\[
\begin{bmatrix}
  y_1 \\
  y_3 \\
  y_5 \\
  y_7
\end{bmatrix} =
\begin{bmatrix}
  1 & 1 & 1 & 1 \\
  1 & 1 & -1 & -w^{2} \\
  1 & -1 & 1 & 1 \\
  1 & -w^{2} & -1 & w^{2}
\end{bmatrix}
\begin{bmatrix}
  t_1 \\
  t_3 \\
  t_5 \\
  t_7
\end{bmatrix},
\]

and:

\[
\begin{bmatrix}
  y_2 \\
  y_4 \\
  y_6 \\
  y_7
\end{bmatrix} =
\begin{bmatrix}
  1 & w & w^{2} & w^{3} \\
  1 & w^{3} & -w^{2} & -w \\
  1 & -w & w^{2} & -w^{3} \\
  1 & -w^{3} & -w^{2} & -w
\end{bmatrix}
\begin{bmatrix}
  t_2 \\
  t_4 \\
  t_6 \\
  t_8
\end{bmatrix} =
\begin{bmatrix}
  1 & 1 & 1 & 1 \\
  1 & 1 & -1 & -w^{2} \\
  1 & -1 & 1 & 1 \\
  1 & -w^{2} & -1 & w^{2}
\end{bmatrix}
\begin{bmatrix}
  1 \\
  w \\
  w^{2} \\
  w^{3}
\end{bmatrix}
\begin{bmatrix}
  t_2 \\
  t_4 \\
  t_6 \\
  t_8
\end{bmatrix}.
\]

Now we have two matrices which are discrete Fourier transforms of lengths 4, where one of them is column-scaled.

References


