LARC automatically does some optimization of storage and of operations, but when implementing any particular algorithm it is wise for the researcher to tailor the algorithm to LARC's strengths. In the following example we discuss a block sparse representation of the discrete Fourier transform, and detail our ideas as to how to perform such an operation efficiently in LARC. In particular, we will see that decisions about how to group sequences of operations can greatly impact the efficiency of the algorithm.

Van Loan's book "Computational Frameworks for the Fast Fourier Transform" gives a block recursive definition for the DFT matrix. In our notation, this equation becomes

$$F_k = C_k(I_1 \otimes F_{k-1})P_k$$

where the subscript k (the level) indicates a matrix of size $n \times n$ where $n = 2^k$. Since in LARC all matrices are power-of-two dimensioned, this convention simplies our notation. (Note that this means that the $2^k \times 2^k$ identity matrix is denoted as I_k, rather than I_n.) In this equation, the Kronecker product of I_1 with the smaller DFT matrix F_{k-1} produces a block diagonal matrix with two copies of F_{k-1} on the diagonal. The two matrices that sandwich this block diagonal matrix are described below.

P_k is the inverse shuffle permutation matrix, defined as

$$P_k(i,j) = \begin{cases} 1 & \text{if} \quad i < 2^{k-1} \text{ and } j = 2i \\ 1 & \text{if} \quad i \geq 2^{k-1} \text{ and } j = 2i + 1 - 2^k \\ 0 & \text{otherwise} \end{cases}$$

or, in a more recursive format, $P_0 = \left[\, 1 \,\right]$; $P_1 = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right]$; and

$$P_{k+1} = \begin{bmatrix} P_{k,0} & P_{k,1} & 0 & 0 \\ 0 & 0 & P_{k,0} & P_{k,1} \\ P_{k,2} & P_{k,3} & 0 & 0 \\ 0 & 0 & P_{k,2} & P_{k,3} \end{bmatrix} \text{ where } P_k = \begin{bmatrix} P_{k,0} & P_{k,1} \\ P_{k,2} & P_{k,3} \end{bmatrix}$$

The sparse matrix C_k is constructed from smaller diagonal matrices, as

$$C_k = \begin{pmatrix} I_{k-1} & D_{k-1} \\ I_{k-1} & -D_{k-1} \end{pmatrix}$$

where $D_{k-1} = \text{diag}(1, w, w^2, \ldots, w^{\frac{n}{2}-1})$ and $w_n = e^{2\pi i/n}$ is the n-th root of unity. Unlike the larger C_k matrices, C_1 is not sparse; it is the 2×2 Hadamard matrix.

Each row and column of the DFT matrix is a permutation of the 2^k powers of the n-th root of unity w. These roots of unity form a multiplicative group, that is, $w^\ell w^m = w^{\ell+m \pmod n}$. LARC can take advantage of this property if you preload the n powers of w; this will ensure (given reasonable LSH parameters) that
`product (MatrixID(`w^ℓ`), MatrixID(`w^m`)) = MatrixID(`$w^{\ell+m \pmod n}$`)`,
reducing the number of nearly-equal scalars and the proliferation of matrices in the MatrixStore. (Similarly, we are not memoizing additional operations using nearly-identical matrices.)

LARC can take advantage of the limited number of unique scalars to achieve some compression, but since there is no other submatrix reuse, the size of the matrix still scales as $O(2^{2k})$. (The exact formula is $(4^{k+1}-1)/3 - 4^k + 2^k$.) However, we are more interested in performing the DFT (i.e., multiplying the DFT matrix and some other vector/matrix) than in constructing the DFT matrix itself, and the matrices C_k and P_k compress well, so we look for a more efficient algorithm using only these matrices.

We expand the expression for F_k in terms of these matrices, using the distributive law for Kronecker products:

$$\begin{aligned} F_k &= C_k(I_1 \otimes F_{k-1})P_k \\ &= C_k(I_1 \otimes [C_{k-1}(I_1 \otimes F_{k-2})P_{k-1}])P_k \\ &= C_k(I_1 \otimes C_{k-1})(I_2 \otimes F_{k-2})(I_1 \otimes P_{k-1})P_k \end{aligned}$$

We continue such expansion until the matrix in the center is $I_k \otimes F_0$. Since $F_0 = [1]$, this central matrix is just the identity matrix. The resulting expression:

$$F_k = C_k(I_1 \otimes C_{k-1}) \dots (I_{k-1} \otimes C_1)(I_{k-1} \otimes P_1)(I_{k-2} \otimes P_2) \dots (I_1 \otimes P_{k-1})P_k$$

Since $P_1 = I_1$, we can simplify the above to

$$\begin{aligned} F_k &= C_k(I_1 \otimes C_{k-1}) \dots (I_{k-1} \otimes C_1)(I_{k-2} \otimes P_2) \dots (I_1 \otimes P_{k-1})P_k \\ &= \left(\prod_{i=0}^{k-1}(I_i \otimes C_{k-i})\right)\left(\prod_{i=2}^{k}(I_{k-i} \otimes P_i)\right) = \overline{C}_k\overline{P}_k \end{aligned}$$

While the expression above implies an algorithm for the DFT which creates two matrices and multiplies them in turn with a vector/matrix, this may

not be the optimal algorithm. Any choice of grouping of the matrices in the fully-expanded expression is a valid algorithm for applying the DFT matrix, and some may be more efficient in LARC than others. In particular, it may be more efficient to perform more multiplies with sparser matrices.

The LARCsize of the P_k matrix is just $5(k-1)+2$ (for $k > 1$); as can be seen from the recursive description of these matrices, there are only two distinct scalars and a great deal of submatrix reuse. The product matrices $\overline{P}_k$ grow as in the table below, with larger increases when going from odd to even k values. The compression for $\overline{P}_k$ is still good, being of order $O(2^k)$, so it may make sense to store $\overline{P}_k$ and use that; this will have to be tested.

In contrast, while the C_k are sparse and LARC compresses them well (the growth is roughly $O(2^k)$, though with LARCsizes larger than $\overline{P}_k$), the product matrices $\overline{C}_k$ are dense and grow as $O(2^{2k})$, so it will probably make more sense to perform k multiplies with the well-compressed $(I_i \otimes C_{k-i})$ than to construct the $\overline{C}_k$ matrix and use it.

We hypothesize that storing the $\overline{P}_k$ and the individual C_k may result in the most efficient implementation of the DFT. However, with any LARC implementation some experiments should be run, as efficiency is impacted by platform, memory, cache sizes and structure, and possibly other factors.

k	P_k	$\overline{P}_k$	C_k	$\overline{C}_k$	F_k
0	1	1	–	–	1
1	3	3	3	3	3
2	7	7	9	8	9
3	12	12	19	23	29
4	17	28	37	74	101
5	22	45	71	261	373
6	27	109	137	976	1429
7	32	174	267	3771	5589
8	37	430	525	14822	22101
9	42	687	1039	58769	87893
10	47	1711	2065	234044	350549
11	52	2736	4115	934119	1400149
12	57	6831	8213	3732370	5596501
13	62	10929	16407	14921277	22377813
14	67	27313	32793	59668712	89494869
k	$5k-3$	$O(2^k)$	$O(2^k)$	$O(4^k)$	$\frac{4^k-1}{3}+2^k$

Table 1: LARCsizes of the matrices discussed in this note. They were calculated using scalarType MPComplex, with regionbitparam set to 200.

k	P_k	$\overline{P}_k$	C_k	$\overline{C}_k$	F_k
		...		...	
2	7	7	9	8	9
3	12	12	19	23	29
4	17	28	37	74	101
5	22	45	71	261	373
		...		...	
12	57	6831	8213	3732370	5596501
13	62	10929	16407	14921277	22377813
14	67	27313	32793	59668712	89494869
k	$5k-3$	$O(2^k)$	$O(2^k)$	$O(4^k)$	$\frac{4^k-1}{3}+2^k$

Table 2: Reduced version of Table 1 for poster.