Vector Radix FFT Algorithm

- Multidimensional rederivation of the 1-D FFT
- Divide-and-conquer algorithm
  - Operates as a recursive procedure
  - Solves problem by dividing it into smaller problems that are the same and then combining answers
- Examples of divide-and-conquer algorithms
  - 1-D FFT
  - Eklundh’s transposition procedure
  - Vector-Radix
- **Note:**
  - The R-C algorithm divided the problem into column/row 1-D DFTs and not into multidimensional DFTs \(\Rightarrow\) not into similar problems \(\Rightarrow\) not divide-and-conquer algorithm
Vector Radix FFT Algorithm

• **1-D FFT**

\[ X(K) = \sum_{n=0}^{N-1} x(n)W_N^{nK}, \quad K = 0,1,\ldots,N-1 \]

Assume N is even:

\[
X(K) = \sum_{n \text{ even}} x(n)W_N^{nK} + \sum_{n \text{ odd}} x(n)W_N^{nK}
\]

\[ = G(K) + W_N^K H(K) \]

where G(K) and H(K) can be calculated by using N/2-point DFTs.
Vector Radix FFT Algorithm

- **2-D FFT**
  Assume array is square for convenience: \( N_1 = N_2 = N \)
  and that \( N_1, N_2 \) are composite (i.e., not prime; have several factors).
  The most convenient factors are power-of-2.
  Example: If \( N = 2^v \) ⇒ Radix.2 algorithm
  If \( N = R^v \) ⇒ Radix-R algorithm

Let \( N_1 = N_2 = N = 2^v \) ⇒ Radix\((2 \times 2)\)

\[
X(K_1, K_2) = \sum_{n_1=0}^{N-1} \sum_{n_2=0}^{N-1} x(n_1, n_2) W_N^{n_1 K_1} W_N^{n_2 K_2};
\]
\( 0 \leq K_1 \leq N - 1 \)
\( 0 \leq K_2 \leq N - 1 \)
Vector Radix FFT Algorithm

- 2D FFT Decimation-In-Time algorithm

\[
X (K_1, K_2) = \sum_{n_1 \text{ even}, \ n_2 \text{ even}} x(n_1, n_2) W_N^{n_1 K_1} W_N^{n_2 K_2} + \sum_{n_1 \text{ even}, \ n_2 \text{ odd}} x(n_1, n_2) W_N^{n_1 K_1} W_N^{n_2 K_2}
\]

\[
+ \sum_{n_1 \text{ odd}, \ n_2 \text{ even}} x(n_1, n_2) W_N^{n_1 K_1} W_N^{n_2 K_2} + \sum_{n_1 \text{ odd}, \ n_2 \text{ odd}} x(n_1, n_2) W_N^{n_1 K_1} W_N^{n_2 K_2}
\]

Note:
- If \( n_1 \) even \( \Rightarrow \) \( n_1 = 2m_1 \)
- If \( n_1 \) odd \( \Rightarrow \) \( n_1 = 2m_1 + 1 \)
Vector Radix FFT Algorithm

- **2D FFT Decimation-In-Time algorithm**

\[
X(K_1, K_2) = \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1, 2m_2) W_N^{2m_1K_1} W_N^{2m_2K_2}
\]

\[
= \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1, 2m_2 + 1) W_N^{2m_1K_1} W_N^{(2m_2+1)K_2}
\]

\[
= \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1 + 1, 2m_2) W_N^{(2m_1+1)K_1} W_N^{2m_2K_2}
\]

\[
= \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1 + 1, 2m_2 + 1) W_N^{(2m_1+1)K_1} W_N^{(2m_2+1)K_2}
\]

**Note:**

\[
W_N^{2m_1K_1} = e^{-j\frac{2\pi}{N}2m_1K_1} = e^{-j\frac{2\pi}{N/2}m_1K_1} = W_{N/2}^{m_1K_1}
\]
Vector Radix FFT Algorithm

- **2D FFT Decimation-In-Time algorithm**

\[
X(K_1, K_2) = \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1, 2m_2)W_{N/2}^{m_1K_1}W_{N/2}^{m_2K_2} \\
+ W_N^{K_2} \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1, 2m_2 + 1)W_{N/2}^{m_1K_1}W_{N/2}^{m_2K_2} \\
+ W_N^{K_1} \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1 + 1, 2m_2)W_{N/2}^{m_1K_1}W_{N/2}^{m_2K_2} \\
+ W_N^{K_1}W_N^{K_2} \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1 + 1, 2m_2 + 1)W_{N/2}^{m_1K_1}W_{N/2}^{m_2K_2}
\]

- **Define**

\[
S_{ij}(K_1, K_2) = \sum_{m_1=0}^{N/2-1} \sum_{m_2=0}^{N/2-1} x(2m_1 + i, 2m_2 + j)W_{N/2}^{m_1K_1}W_{N/2}^{m_2K_2}
\]
Vector Radix FFT Algorithm

- 2D FFT Decimation-In-Time algorithm

\[ X(K_1, K_2) = S_{00}(K_1, K_2) + W_N^{K_1} S_{01}(K_1, K_2) + W_N^{K_2} S_{10}(K_1, K_2) + W_N^{(K_1+K_2)} S_{11}(K_1, K_2) \]

where

\[ S_{ij}(K_1, K_2) = \sum_{m_1=0}^{\frac{N-1}{2}} \sum_{m_2=0}^{\frac{N-1}{2}} x(2m_1 + i, 2m_2 + j) W_N^{m_1 K_1} W_N^{m_2 K_2}, \quad 0 \leq K_1 \leq \frac{N}{2} - 1 \]

\[ 0 \leq K_2 \leq \frac{N}{2} - 1 \]

**Note:** If \( i = 1 \), multiply \( S_{ij} \) by \( W_N^{K_i} \)

If \( j = 1 \), multiply \( S_{ij} \) by \( W_N^{K_2} \)

- \( S_{00}, S_{01}, S_{10}, S_{11} \) : \( N/2 \)-point DFTs
- Still need to calculate \( X(K_1, K_2) \) for \( K_1 \) and \( K_2 \) between \( N/2 \) and \( N \)
  - This is simplified by the fact that \( S_{ij}(K_1, K_2) \) is periodic with period \( N/2 \) in \( K_1 \) and \( K_2 \):

\[ S_{ij}(K_1, K_2) = S_{ij}(K_1 + N/2, K_2) = S_{ij}(K_1, K_2 + N/2) = S_{ij}(K_1 + N/2, K_2 + N/2) \]

- **Note:** \( W_N^{N/2} = e^{j\pi} = -1 \)

So, when \( N/2 \) is added to a variable (\( K_1 \) or \( K_2 \)), reverse the sign of \( W_N \) having that variable as exponent.
Vector Radix FFT Algorithm

- **2D FFT Decimation-In-Time algorithm**

\[
X(K_1, K_2) = S_{00}(K_1, K_2) + W_N^{K_2} S_{01}(K_1, K_2) + W_N^{K_1} S_{10}(K_1, K_2) + W_N^{(K_1+K_2)} S_{11}(K_1, K_2)
\]

\[
X(K_1 + N/2, K_2) = S_{00}(K_1, K_2) + W_N^{K_2} S_{01}(K_1, K_2) - W_N^{K_1} S_{10}(K_1, K_2) - W_N^{(K_1+K_2)} S_{11}(K_1, K_2)
\]

\[
X(K_1, K_2 + N/2) = S_{00}(K_1, K_2) - W_N^{K_2} S_{01}(K_1, K_2) + W_N^{K_1} S_{10}(K_1, K_2) - W_N^{(K_1+K_2)} S_{11}(K_1, K_2)
\]

\[
X(K_1 + N/2, K_2 + N/2) = S_{00}(K_1, K_2) - W_N^{K_2} S_{01}(K_1, K_2) - W_N^{K_1} S_{10}(K_1, K_2) + W_N^{(K_1+K_2)} S_{11}(K_1, K_2)
\]

- **Note:**

\[
0 \leq K_1 \leq N/2 - 1
\]

\[
0 \leq K_2 \leq N/2 - 1
\]

- **Procedure:**

1. 4 smaller DFTs \( S_{00}, S_{01}, S_{10}, S_{11} \) need to be calculated
2. By doing three different complex multiplications, we obtain 4 samples of the DFT, for a given \((K_1, K_2)\), \(0 \leq K_1, K_2 \leq N/2 - 1\)
3. \( S_{00}, S_{01}, S_{10}, S_{11} \) are calculated by repeating same procedure
Vector Radix FFT Algorithm

- **2D FFT Decimation-In-Time algorithm**
  - **Implementation:**
    Basic unit in structure is called a “Butterfly” (even though it looks more like a Butterfly in 1-D case)
  - **Basic Butterfly:**

\[
\begin{align*}
S_{00}(K_1, K_2) & \quad X(K_1, K_2) \\
S_{01}(K_1, K_2) & \quad W_N^{K_2} X(K_1 + N/2, K_2) \\
S_{10}(K_1, K_2) & \quad W_N^{K_1} X(K_1, K_2 + N/2) \\
S_{11}(K_1, K_2) & \quad W_N^{K_1+K_2} X(K_1 + N/2, K_2 + N/2)
\end{align*}
\]
Vector Radix FFT Algorithm

- **2D FFT Decimation-In-Time algorithm**
  
  ➢ Computations
    - #CMULTs = 3 / Butterfly
    - #CADDs = 12 ( = 3×4) / Butterfly
      since 3 CADDs / DFT sample, and 4 DFT samples / Butterfly
      
      Note: one can reduce #CADDs to 8 (refer to Problem 2.11 in Dudgeon & Mersereau)
    - #stages required for 2-D N×N-point FFT = \( \log_2 N = \nu \), for \( N = 2^\nu \)
    - #Butterfly per stage = \( \frac{N^2}{4} \)
      Why? we have N×N initial points;
      I = # of inputs per stage = \( N^2 \); each Butterfly takes care of 4 inputs
      O = # of outputs per stage = \( \frac{N^2}{4} \)

  ➢ Summary:
    
    \[
    \text{# CMULTS} = (3 \text{ CMULTs/Butt}) \left( \frac{N^2}{4} \text{ Butt/stage} \right) \left( \log_2 N \text{ stages} \right) \\
    = \frac{3N^2}{4} \log_2 N
    \]

    \[
    \text{#CADDs} = (8 \text{ CADDs/Butt}) \left( \frac{N^2}{4} \text{ Butt/stage} \right) \left( \log_2 N \text{ stages} \right) \\
    = 2N^2 \log_2 N
    \]
M-Dimensional Vector-Radix FFT

- Array of size \( N \times N \times N \ldots \times N \)
- Same basic Butterfly but different # of input/output points
- Complexity:
  - # inputs/outputs per stage = \( N^M \)
  - # inputs per Butterfly = \( 2^M \) \( N^M \) \( N^M \)
  - # Butterflies per stage = \( \frac{N^M}{2^M} = \left( \frac{N}{2} \right)^M \)
  - # stages = \( \log_2 N = \nu \), for \( N = 2^\nu \)
  - # CMULTs/Butt = (# inputs/Butt-1) = \( 2^M - 1 \)

Total # CMULTs = \( (2^M - 1).\left( \frac{N}{2} \right)^M.(\log_2 N) = \frac{2^M - 1}{2^M}.N^M \log_2 N \)
M-Dimensional Vector-Radix FFT

• Comparison with R-C algorithm
  ➢ Computation:
    ✓ # CMULTs for R-C $= \frac{N^M}{2} \log_2 N^M = \frac{MN^M}{2} \log_2 N$
    ✓ # CMULTs for M-D FFT $= \frac{2^M - 1}{2^M} N^M \log_2 N$

  ➢ Storage
    ✓ M-D FFT requires more storage for $M>2$: at least $2^{M-1}$ rows required to be stored at a time for efficient computation plus one additional entire data read/write cycle needed first to perform the bit-reversed ordering of the data.

  ➢ I/O Passes
    ✓ Assuming that $2^{M-1}$ rows are stored in memory for R-C, M-D Vector-Radix FFT requires one additional I/O pass (read/write of entire data) at initial step to rearrange samples in bit-reversed order.