Fast Fourier Transform: Practical aspects and Basic Architectures

Lecture 9

Vladimir Stojanović

6.973 Communication System Design – Spring 2006

Massachusetts Institute of Technology

Multiplication complexity per output point

- CTFFT and SRFFT

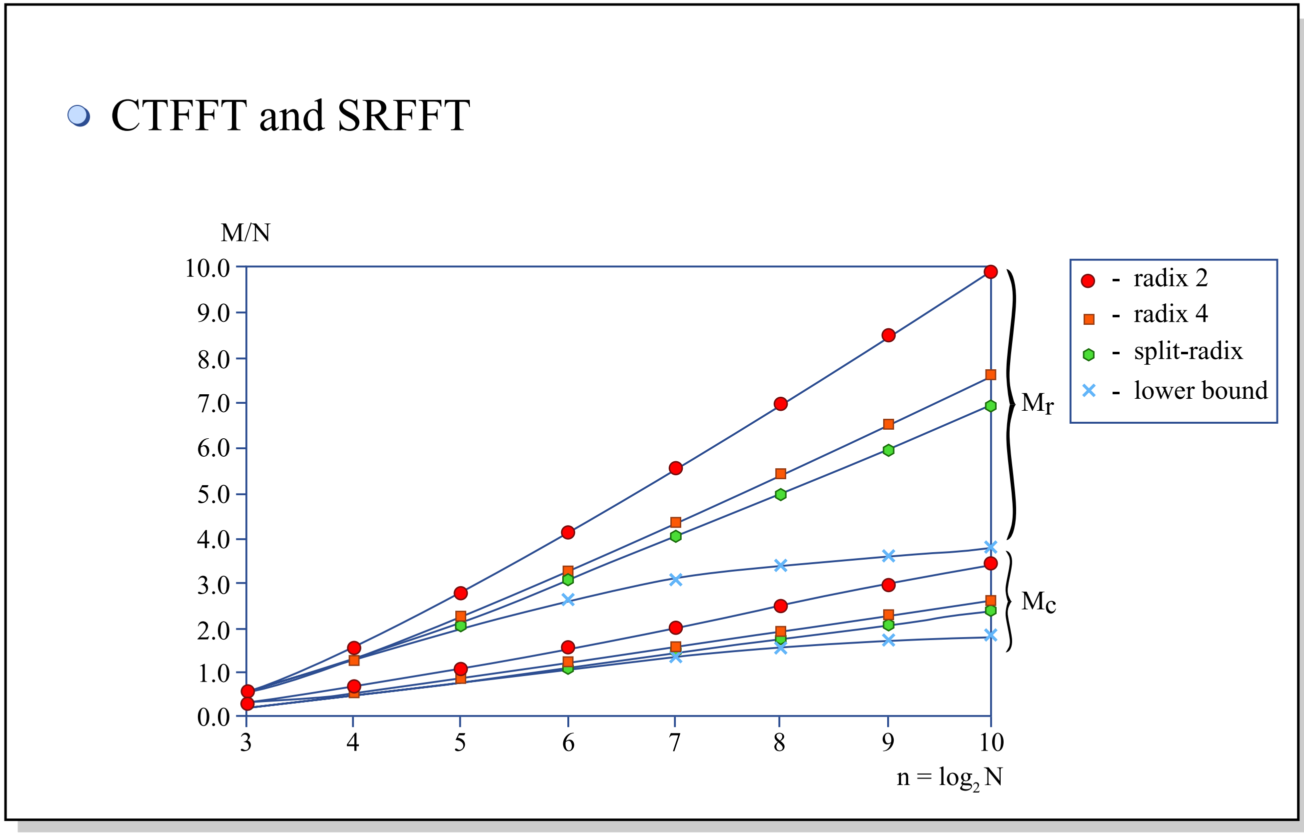


Figure by MIT OpenCoursWare.

Multiplies and adds

Real multiplies

N		Radix 2	Radix 4	SRFFT	PFA	Winograd
16		24	20	20		
	30				100	68
32		88		68		
	60				200	136
64		264	208	196		
	120				460	276
128		712		516		
	240				1100	632
256		1800	1392	1284		
	504				2524	1572
512		4360		3076		
	1008				5804	3548
1024		10248	7856	7172		
2048		23560		16388		
	2520				17660	9492

Real adds

N		Radix 2	Radix 4	SRFFT	PFA	Winograd
16		152	148	148		
	30				384	384
32		408		388		
	60				888	888
64		1032	976	964		
	120				2076	2076
128		2504		2308		
	240				4812	5016
256		5896	5488	5380		
	504				13388	14540
512		13566		12292		
	1008				29548	34668
1024		30728	28336	27652		
2048		68616		61444		
	2520				84076	99628

Figure by MIT OpenCourseWare.

Structural considerations

- How to compare different FFT algorithms?
- Many metrics to choose from

- The ease of obtaining the inverse FFT
- In-place computation
- Regularity
  - Computation
  - Interconnect
- Parallelism and pipelining
- Quantization noise

Inverse FFT

- FFTs often used for computing FIR filtering
  - Fast convolution (FFT + pointwise multiply + IFFT)
- In some applications (like 802.11a)
  - Can reuse FFT block to do the IFFT (half-duplex scheme)
- Simple trick [Duhamel88]
  - Swap the real and imaginary inputs and outputs
    - If FFT(x_R,x_I,N) computes the FFT of sequence $x_R(k)+jx_I(k)$
    - Then FFT(x_I,x_R,N) computes the IFFT of $jx_R(k)+x_I(k)$

$$X_k = \sum_{n=0}^{N-1} x_n W_N^{nk} = \mathrm{DFT}_k\{x_n\}$$

$$x'_n = \sum_{k=0}^{N-1} X_k W_N^{-nk} = \mathrm{IDFT}_n\{X_k\}$$

$$x_n'^* = \sum_{k=0}^{N-1} X_k^* W_N^{nk}$$

$$x_n = a_n + j \cdot b_n \Rightarrow j \cdot x_n^* = b_n + j \cdot a_n$$

$$j \cdot x_n'^* = \sum_{k=0}^{N-1} j \cdot X_k^* W_N^{nk}$$

$$x'_n = j\left[\sum_{k=0}^{N-1} (jX_k^*)\, W_N^{nk}\right]^*$$

Exchange the real and imag part

In-place computation

- Most algorithms allow in-place computation
  - Cooley-Tukey, SRFFT, PFA
  - No auxilary storage of size dependent on N is needed
  - WFTA (Winograd Fourier Transform Algorithm) does not allow in-place computation
    - A drawback for large sequences

- Cooley-Tukey and SRFFT are most compatible with longer size FFTs

Regularity, parallelism

- Regularity
  - Cooley-Tukey FFT very regular
    - Repeat butterflies of same type
      - Sums and twiddle multiplies
  - SRFFT slightly more involved
    - Different butterfly types in parallel
      - e.g. radix-2 and radix-4 used in parallel on even/odd samples
  - PFA even more involved
    - Repetitive use of more complicated modules (like cyclic convolution, for prime length DFTs)
  - WFTA most involved
    - Repetition of parts of the cyclic conv. modules from PFA
- Parallelization
  - Fairly easy for C-TFFT and SRFFT
    - Small modules applied on sets of data that are separable and contiguous
  - More difficult for PFA
    - Data required for each module not in contiguous locations

Quantization noise

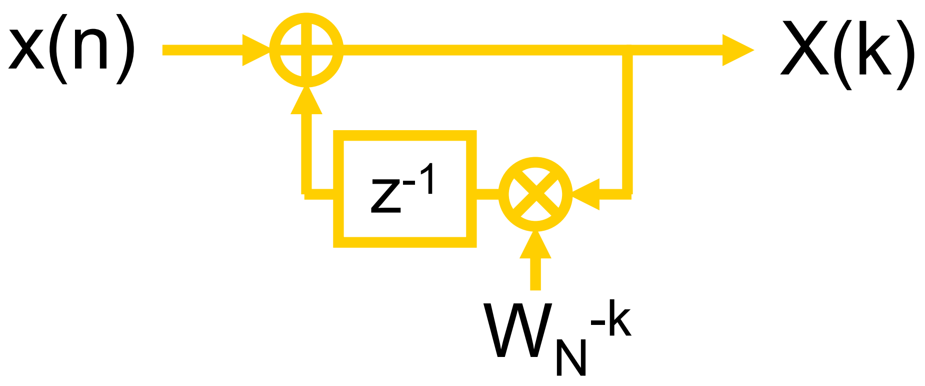
- Roundoff noise generated by finite precision of operations inside FFT (adds, multiplies)
- CTFFT (lengths 2^n)
  - Four error sources per butterfly (variance $2^{-2B}/12$)
    - Total variance per butterfly $2^{-2B}/3$
  - Each output node receives signals from a total of N-1 butterflies in the flow graph (N/2 from the first stage, N/4 in the second, …)
    - Total variance for each output ~ $N/3*2^{-2B}$
  - Assuming input power $1/3N^2$ ($|x(n)|<1/N$ to avoid overflow)
  - Output power is 1/3N
    - Error-to-signal ratio is then $N^2 2^{-2B}$ (needs 1 additional bit per stage to maintain SER)
  - Since a maximum magnitude increases by less than 2x from stage to stage we can prevent the overflow by requiring that $|x(n)|<1$ and scaling by ½ from stage-to-stage
    - The output will be 1/N of the previous case, but the input magnitude can be Nx larger, improving the SER
    - Error-to-signal ratio is then $4N*2^{-2B}$ (needs 1/2 additional bit per stage to maintain SER)
  - Radix-4 and SRFFT generate less roundoff noise than radix-2
- WFTA
  - Fewer multiplications (hence fewer noise sources)
  - More difficult to include proper rescaling in the algorithm
    - Error-to-signal ratio is higher than in CTFFT or SRFFT
  - Two more bits are necessary to represent data in WFTA for same error order as CTFFT

Particular cases

- DFT algorithms for real data sequence x_k
  - X_k has Hermitian symmetry ($X_{N-k}=X_k^*$)
    - X_0 is real, and when N even, $X_{N/2}$ real as well
  - N input values map to
    - 2 real and N/2-1 complex conjugate values when N even
    - 1 real and (N-1)/2 complex conjugate values when N odd
- Can exploit the redundancy
  - Reduce complexity and storage by a factor of 2
    - If take the real DFT of x_R and x_I separately
      - 2N additions are sufficient to obtain complex DFT
      - Goal to obtain real DFT with half multiplies and half adds
  - Example DIF SRFFT
    - X_{2k} requires half-length DFT on real data
    - Then b/c of Hermitian symmetry $X_{4k+1}=X^*_{4(N/4-k-1)+3}$
      - Only need to compute one DFT of size N/4 (not two)

DFT pruning

- In practice, may only need to compute a few tones
  - Or only a few inputs are different from zero
  - Typical cases: spectral analysis, interpolation, fast conv
  - Computing a full FFT can be wasteful
- Goertzel algorithm
  - Can be obtained by simply pruning the FFT flow graph
  - Alternately, looks just like a recursive 1-tap filter for each tone



Related transforms

- Mostly focused on efficient matrix-vector product involving Fourier matrix

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & W_N^3 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & W_N^6 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \end{bmatrix}.$$

- No assumption made on the input/output vector
  - Some assumptions on these leads to related transforms
  - Discrete Hartley Transform (DHT)
  - Discrete Cosine (and Sine) Transform (DCT, DST)

Related transforms: DHT

$$X_k = \sum_{n=0}^{N-1} x_n(\cos(2\pi nk/N) + \sin(2\pi nk/N))$$

- Proposed as an alternative to DFT
  - Initial (false) claims of improved arithmetic complexity
    - Real-valued FFT complexity is equivalent
- Self-inverse
  - Provided that X_0 further weighted by 1/sqrt(2)
  - Inverse real DFT on Hermitian data
    - Same complexity as the real DFT so no significant gain from self-inverse property of DHT

Related transforms: DCT

$$X_k = \sum_{n=0}^{N-1} x_n \cos(2\pi(2k+1)n/4N).$$

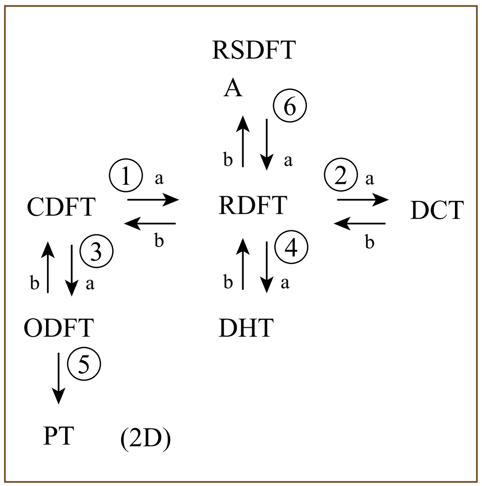
- Lots of applications in image and video processing
- Scale factor of 1/sqrt(2) for X_0 left out
  - Formula above appears as a sub-problem in length-4N real DFT
  - Multiplicative complexity can be related to real DFT

$$\mu(\text{DCT}(N)) = (\mu(\text{real-DFT}(4N)) - \mu(\text{real-DFT}(2N)))/2.$$

  - Practical algorithms depend on the transform length
    - N odd: Permutations and sign changes map to real DFT
    - N even: Map into same length real DFT + N/2 rotations

Relationship with FFT

- DHT, DCT, DST and related transforms can all be mapped to DFT



1	a	Complex DFT 2^n	2 real DFT's 2^n $+2^{n+1}$ - 4 additions
	b	Real DFT 2^n	1 real DFT 2^{n-1} + 1 complex DFT 2^{n-2} + $(3.2^{n-2} - 4)$ multiplications + $(2^n + 3.2^{n-2} - n)$ additions
2	a	Real DFT 2^n	1 real DFT 2^{n-1} + 2 DCTs 2^{n-2} + 3.2^{n-1} -2 additions
	b	DCT 2^n	1 real DFT 2^n + $(3.2^{n-1} -2)$ multiplications + $(3.2^{n-1} -3)$ additions
3	a	Complex DFT 2^n	1 odd DFT 2^{n-1}+ 1 complex DFT 2^{n-1} $+2^{n+1}$ additions
	b	Odd DFT 2^{n-1}	2 complex DFT's 2^{n-2} + $2(3.2^{n-2}-4)$ multiplications + $(2^n+3.2^{n-1} -8)$ additions
4	a	Real DFT 2^n	1 DHT 2^n -2 additions
	b	DHT 2^n	1 real DFT 2^n +2 additions
5		Complex DFT 2^nx2^n	3.2^{n-1} odd DFT 2^{n-1} + 1 complex DFT 2^{n-1}x2^{n-1} + $n.2^n$ additions
6	a	Real DFT 2^n	1 real symmetric DFT 2^n + 1 real antisymmetric DFT 2^n + (6n+10), 4^{n-1} additions
	b	Real symmDFT 2^n	1 real symmetric DFT 2^{n-1} + 1 inverse real DFT + $3(2^{n-3}-1)+1$ multiplications + $(3n-4).2^{n-3}+1$ additions

Figure by MIT OpenCourseWare.

- All transforms use split-radix algorithms
  - For minimum number of operations

Implementation issues

- General purpose computers
- Digital signal processors
- Vector/multi processors
- VLSI ASICs

Implementation on general purpose computers

- FFT algorithms built by repetitive use of basic building blocks
  - CTFFT and SRFFT butterflies are small – easily optimizable
  - PFA/WFTA blocks are larger
- More time is spent on load/store operations
  - Than in actual arithmetic (cache miss and memory access latency problem)
  - Locality is of utmost importance
  - This is the reason why PFA and WFTA do not meet the performance expected from their computation complexity!
  - PFA drawback partially compensated since only a few coefficients have to be stored
- Compilers can optimize the FFT code by loop-unrolling (lots of parallelism) and tailoring to cache size (aspect ratio)

Digital Signal Processors

- Built for multiply/accumulate based algorithms
- Not matched by any of the FFT algorithms
  - Sums of products changed to fewer but less regular computations
- Today's DSPs take into account some FFT requirements
  - Modulo counters (a power of 2 for CTFFT and SRFFT)
  - Bit-reversed addressing

Vector and multi-processors

- Must deal with two interconnected problems
  - The vector size of the data that can be processed at the maximal rate
    - Has to be full as often as possible
  - Loading of the data should be made from data available inside the cache memory to save time
- In multi-processors performance dependent on interconnection network
  - Since FFT deterministic, resource allocation can be solved off-line
  - Arithmetic units specialized for butterfly operations
  - Arrays with attached shuffle networks
  - Pipelines of arithmetic units with intermediate storage and reordering
  - Mostly favor CTFFTs

ASICs

- Area and throughput are important
  - A – area, T- time between two successive DFT computations
  - Asymptotic lower bound for AT^2

$$\Omega_{AT2}(\mathrm{DFT}(N)) = N^2 \log^2(N)$$

  - Achieved by several micro-architectures
    - Shuffle-exchange networks
    - Square grids
  - Outperform the more traditional micro-architectures only for very large N
    - Cascade connection with variable delay
- Dedicated chips often based on traditional micro-architectures efficiently mapped to layout
  - Cost dominated by number of multiplies but also by cost of communication
  - Communication cost very hard to estimate
- Dedicated arithmetic units
  - Butterfly unit
  - CORDIC unit
- Still, many heuristics and local tricks to reduce complexity and improve communication

Architectures

- $1, N, N^2$ cell type – direct transform
- Cascade (pipelined) FFT
- FFT network
- Perfect-shuffle FFT
- CCC network FFT
- The Mesh FFT

The naive approach

$$\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & \cdots & 1 \\ 1 & W_N & W_N^2 & W_N^3 & \cdots & W_N^{N-1} \\ 1 & W_N^2 & W_N^4 & W_N^6 & \cdots & W_N^{2(N-1)} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 1 & W_N^{N-1} & W_N^{2(N-1)} & \cdots & \cdots & W_N^{(N-1)(N-1)} \end{bmatrix} \times \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{N-1} \end{bmatrix}.$$

- Compute all terms in the matrix-vector product
  - N^2 multiplications required
- Three degrees of parallelism
  - Calculate on one multiply-add cell
  - On N multiply-add cells
  - On N^2 multiply-add cells

1 multiply-add cell

- Performance O(N^2logN)

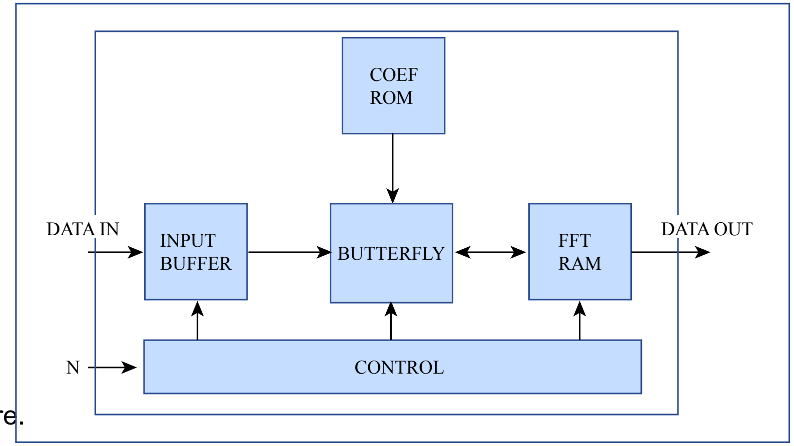


Figure by MIT OpenCourseWare.

- For large FFTs storage of intermediate results is a problem
  - N-long FFT requires
    - N/r*$\log_r$N, radix-r butterfly operations
    - 2N*$\log_r$N read or write RAM accesses
      - E.g. to do the 8K FFT in 1ms, need to access internal RAM every 9ns, using radix-4
- To speed up
  - Either use higher radix (to reduce the overall number of memory accesses at the price of increase in arithmetic complexity)
  - Or partition the memory to r banks accessed simultaneously (more complex addressing and higher area)
- Need a very high rate clock

Cascade FFT

- Cascade of logN multiply-add cells
  - Nicely suited for decimation in frequency FFT

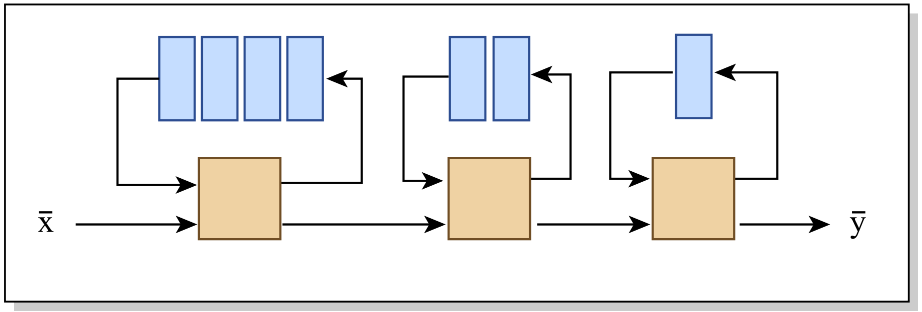


Figure by MIT OpenCourseWare.

  - E.g. 8pt DIF FFT

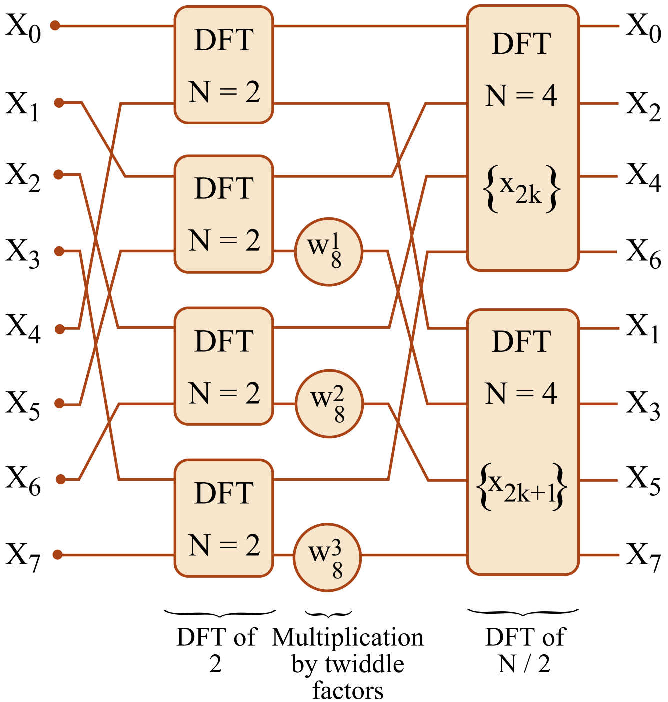


Figure by MIT OpenCourseWare.

  - Produces the output values in bit-reversed order

FFT network

- One of the most obvious implementations
  - Provide a multiply-add cell for each execution statement
  - Each cell also has a register holding a particular value of z^j (twiddle factor)
  - How many such cells do we need for length-N (radix-2 DIT)?
- One possible layout
  - logN rows, N/2 cells each row
- Pipelined performance O(logN)
  - A new problem instance can enter the network as soon as the previous one has left the first row
  - Delay limited by cell's multiply-add and long-wire driver to the next row O(logN)
    - Total network delay is $O(\log^2 N)$

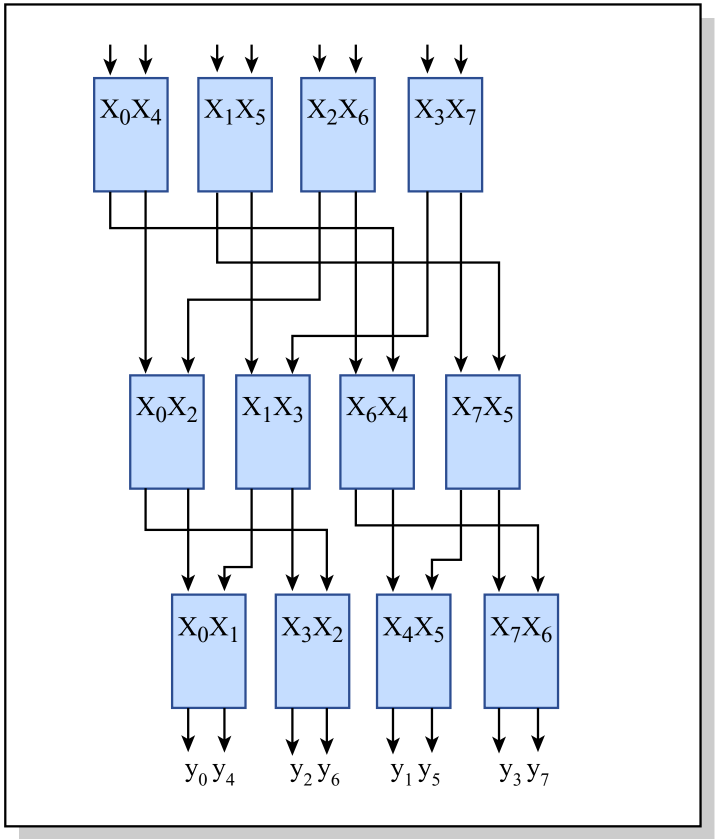


Figure by MIT OpenCourseWare.

FFT network

- Inputs are in “bit-shuffled” order (decimated)
- Outputs are in “bit-reversed” order
  - Minimizes the amount of interconnects

- General scheme for interconnections
  - Number the cells naturally
    - 0 to N/2-1, from left to right
  - Cell i in the first row is connected to two cells in the second row
    - Cell i and (i+N/4) mod N/2
  - Cell i in the second row is connected to cells
    - i and floor(i/(N/4))+((i+N/8) mod N/4) in the third row
  - Cell i in the k-th row (k=1,…logN-1) is connected to (k+1)-th row
    - Cell i and cell floor(i/(N/2^k))+((i+N/2^{k+1}) mod N/2^k)

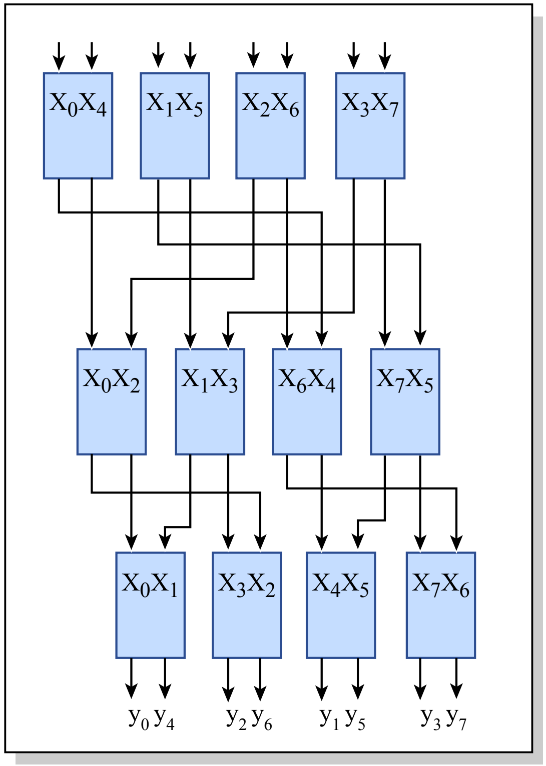


Figure by MIT OpenCourseWare.

The perfect-shuffle network

- N/2 element network perfectly suited for FFT, radix-2 DIT

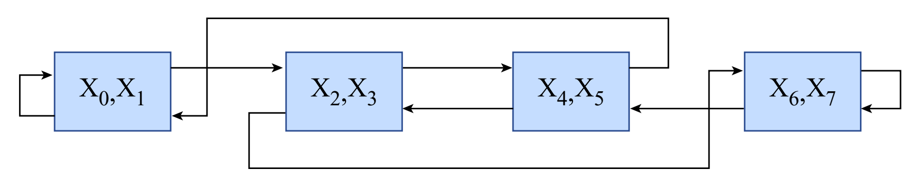


Figure by MIT OpenCourseWare.

- Each multiply-add cell associated with x_k and x_{k+1} (k- even number between 0 and N-1)
- A connection from cell with x_k to cell with x_j when j=2k mod N-1 (this mapping is one-to-one)
  - Represents “circular left shift” of the logN-bit binary representation of k
- First the x_k values are loaded into cells
- In each iteration, output values are shuffled among cells
- At the end of logN steps, final data is in cell registers in bit-reversed order

Cube-Connected-Cycles (CCC) network

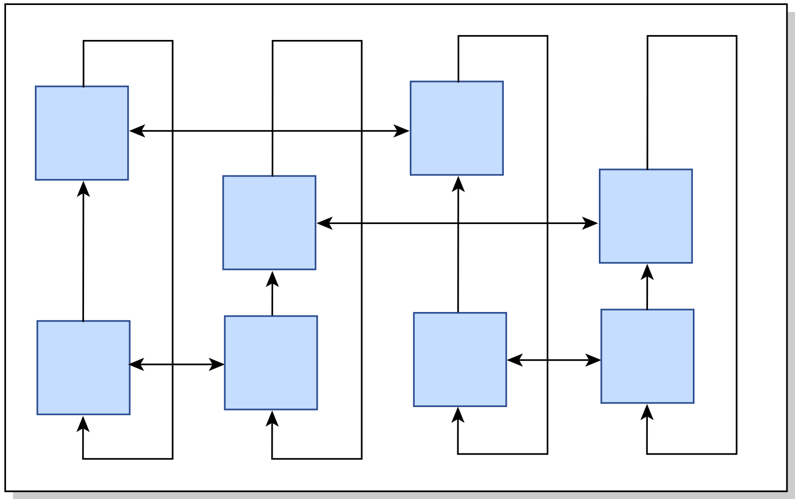


Figure by MIT OpenCourseWare.

- N cells capable of performing N-element FFT in O(logN) steps
- Closely related to the FFT network
  - Just has circular connections between first and last rows (and uses N instead of N/2logN cells)
    - Does not exist for all N (only for N=(K/2)*logK for integer K

The Mesh implementation

- Approximately sqrt(N) rows and columns
- N-long FFT in logN steps

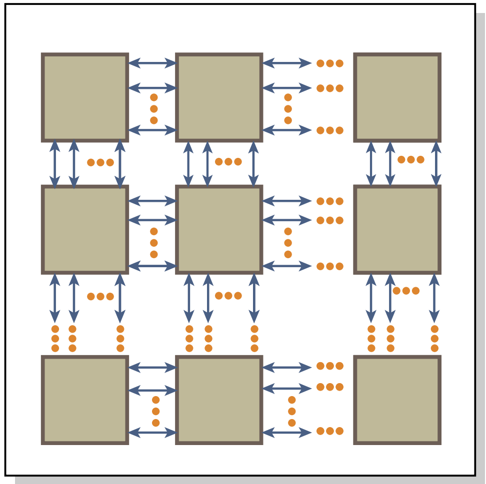


Figure by MIT OpenCourseWare.

- View as time-multiplexed version of the FFT network
  - In each step, N/2 nodes take the role of N/2 cells in FFT network
  - Other half routes the data other nodes

Performance summary

Design	Area	Time	Area*Time^2	Delay
1-cell DFT	$N \log N$	$N^2 \log N$	$N^5 \log^3 N$	$N^2 \log N$
N-cell DFT	$N \log N$	$N \log N$	$N^3 \log^3 N$	$N^2 \log N$
N^2-cell DFT	$N^2 \log N$	$\log N$	$N^2 \log^3 N$	$N^2 \log N$
1-proc FFT	$N \log N$	$N \log^2 N$	$N^3 \log^5 N$	$N \log^2 N$
Cascade	$N \log N$	$N \log N$	$N^3 \log^3 N$	$N \log^2 N$
FFT Network	N^2	$\log N$	$N^2 \log^2 N$	$\log^2 N$
Perfect Shuffle	$N^2 / \log^2 N$	$\log^2 N$	$N^2 \log^2 N$	$\log^2 N$
CCC	$N^2 / \log^2 N$	$\log^2 N$	$N^2 \log^2 N$	$\log^2 N$
Mesh	$N \log^2 N$	$\sqrt{N}$	$N^2 \log^2 N$	$\sqrt{N}$

Figure by MIT OpenCourseWare.

- Cascade FFT has the best trade-off
  - Less complicated wiring and NlogN delay
- FFT network is as fast as N^2 cell FFT but much less area (only N/2logN cells)
- Perfect-Shuffle and CCC use less cells than FFT network, but take a bit more time

Readings

- [1] C.D. Thompson "Fourier Transforms in VLSI," no. UCB/CSD-82-105, 1982.
  - Same, but hard to find publication
  - [2] C.D. Thompson "Fourier Transforms in VLSI," *IEEE Trans. Computers* vol. 32, no. 11, pp. 1047-1057, 1983.
- [3] P. Duhamel and M. Vetterli "Fast fourier transforms: a tutorial review and a state of the art," *Signal Process.* vol. 19, no. 4, pp. 259-299, 1990.