

# Generating and Searching Families of FFT Algorithms

Steve Haynal<sup>1</sup>   Heidi Haynal<sup>2</sup>

<sup>1</sup>SofterHardware

<sup>2</sup>Department of Mathematics and Computer Science  
Walla Walla University

July 15, 2010

- Proof of the lowest operation count for classes of discrete Fourier transforms
  - Require fixed flowgraph structure of common FFTs
  - Require all complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Technique is exhaustive and supports various search objectives
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)

- Proof of the lowest operation count for classes of discrete Fourier transforms
  - Require fixed flowgraph structure of common FFTs
  - Require all complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Technique is exhaustive and supports various search objectives
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)

- Proof of the lowest operation count for classes of discrete Fourier transforms
  - Require fixed flowgraph structure of common FFTs
  - Require all complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Technique is exhaustive and supports various search objectives
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)

- Proof of the lowest operation count for classes of discrete Fourier transforms
  - Require fixed flowgraph structure of common FFTs
  - Require all complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Technique is exhaustive and supports various search objectives
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms
- 3 Searching a Family of FFT Algorithms
- 4 Results and Conclusions

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms
- 3 Searching a Family of FFT Algorithms
- 4 Results and Conclusions

# Outline

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms
- 3 Searching a Family of FFT Algorithms
- 4 Results and Conclusions

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms
- 3 Searching a Family of FFT Algorithms
- 4 Results and Conclusions

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms
- 3 Searching a Family of FFT Algorithms
- 4 Results and Conclusions

# Fourier Transform

- Fourier Transform is an Integral

$$X(f) = \int_{-\infty}^{\infty} a(t)e^{-i2\pi ft} dt, \quad f \in (-\infty, \infty)$$

- But a discrete sum is used to compute the Fourier Transform

$$\begin{aligned} X(k) &= \sum_{j=0}^{n-1} a_j e^{-\frac{i2\pi}{n}jk} \\ &= \sum_{j=0}^{n-1} a_j \omega_n^{jk \pmod{n}}, \quad k = 0, 1, 2, \dots, n-1 \end{aligned}$$

## Multiplication Example

$$\begin{aligned} \omega_{16}^{13} \omega_{16}^6 &= \omega_{16}^{(13+6 \pmod{16})} \\ &= \omega_{16}^3 \end{aligned}$$

# Fourier Transform

- Fourier Transform is an Integral

$$X(f) = \int_{-\infty}^{\infty} a(t)e^{-i2\pi ft} dt, \quad f \in (-\infty, \infty)$$

- But a discrete sum is used to compute the Fourier Transform

$$\begin{aligned} X(k) &= \sum_{j=0}^{n-1} a_j e^{-\frac{i2\pi}{n}jk} \\ &= \sum_{j=0}^{n-1} a_j \omega_n^{jk \pmod{n}}, \quad k = 0, 1, 2, \dots, n-1 \end{aligned}$$

## Multiplication Example

$$\begin{aligned} \omega_{16}^{13} \omega_{16}^6 &= \omega_{16}^{(13+6 \pmod{16})} \\ &= \omega_{16}^3 \end{aligned}$$

# Fourier Transform

- Fourier Transform is an Integral

$$X(f) = \int_{-\infty}^{\infty} a(t)e^{-i2\pi ft} dt, \quad f \in (-\infty, \infty)$$

- But a discrete sum is used to compute the Fourier Transform

$$\begin{aligned} X(k) &= \sum_{j=0}^{n-1} a_j e^{-\frac{i2\pi}{n}jk} \\ &= \sum_{j=0}^{n-1} a_j \omega_n^{jk \pmod{n}}, \quad k = 0, 1, 2, \dots, n-1 \end{aligned}$$

## Multiplication Example

$$\begin{aligned} \omega_{16}^{13} \omega_{16}^6 &= \omega_{16}^{(13+6 \pmod{16})} \\ &= \omega_{16}^3 \end{aligned}$$

# Expanded 8-Point Discrete Fourier Transform

$$X(k) = \sum_{j=0}^{n-1} a_j \omega_n^{jk \pmod{n}}, \quad k = 0, 1, 2, \dots, n-1$$

$\longleftarrow \hspace{10em} \longrightarrow$   
 $n$

$$\begin{aligned} X(0) &= a_0\omega_8^0 + a_1\omega_8^0 + a_2\omega_8^0 + a_3\omega_8^0 + a_4\omega_8^0 + a_5\omega_8^0 + a_6\omega_8^0 + a_7\omega_8^0 \\ X(1) &= a_0\omega_8^0 + a_1\omega_8^1 + a_2\omega_8^2 + a_3\omega_8^3 + a_4\omega_8^4 + a_5\omega_8^5 + a_6\omega_8^6 + a_7\omega_8^7 \\ X(2) &= a_0\omega_8^0 + a_1\omega_8^2 + a_2\omega_8^4 + a_3\omega_8^6 + a_4\omega_8^0 + a_5\omega_8^2 + a_6\omega_8^4 + a_7\omega_8^6 \\ X(3) &= a_0\omega_8^0 + a_1\omega_8^3 + a_2\omega_8^6 + a_3\omega_8^1 + a_4\omega_8^4 + a_5\omega_8^7 + a_6\omega_8^2 + a_7\omega_8^5 \\ X(4) &= a_0\omega_8^0 + a_1\omega_8^4 + a_2\omega_8^0 + a_3\omega_8^4 + a_4\omega_8^0 + a_5\omega_8^4 + a_6\omega_8^0 + a_7\omega_8^4 \\ X(5) &= a_0\omega_8^0 + a_1\omega_8^5 + a_2\omega_8^2 + a_3\omega_8^7 + a_4\omega_8^4 + a_5\omega_8^1 + a_6\omega_8^6 + a_7\omega_8^3 \\ X(6) &= a_0\omega_8^0 + a_1\omega_8^6 + a_2\omega_8^4 + a_3\omega_8^2 + a_4\omega_8^0 + a_5\omega_8^6 + a_6\omega_8^4 + a_7\omega_8^2 \\ X(7) &= a_0\omega_8^0 + a_1\omega_8^7 + a_2\omega_8^6 + a_3\omega_8^5 + a_4\omega_8^4 + a_5\omega_8^3 + a_6\omega_8^2 + a_7\omega_8^1 \end{aligned}$$

$\updownarrow$   
 $n$





# Expanded 8-Point Discrete Fourier Transform

$$X(k) = \sum_{j=0}^{n-1} a_j \omega_n^{jk \pmod{n}}, \quad k = 0, 1, 2, \dots, n-1$$

←----- n ----->

$X(0) = a_0\omega_8^0 + a_1\omega_8^0 + a_2\omega_8^0 + a_3\omega_8^0 + a_4\omega_8^0 + a_5\omega_8^0 + a_6\omega_8^0 + a_7\omega_8^0$	↑ $n$ ↓
$X(1) = a_0\omega_8^0 + a_1\omega_8^1 + a_2\omega_8^2 + a_3\omega_8^3 + a_4\omega_8^4 + a_5\omega_8^5 + a_6\omega_8^6 + a_7\omega_8^7$	
$X(2) = a_0\omega_8^0 + a_1\omega_8^2 + a_2\omega_8^4 + a_3\omega_8^6 + a_4\omega_8^0 + a_5\omega_8^2 + a_6\omega_8^4 + a_7\omega_8^6$	
$X(3) = a_0\omega_8^0 + a_1\omega_8^3 + a_2\omega_8^6 + a_3\omega_8^1 + a_4\omega_8^4 + a_5\omega_8^7 + a_6\omega_8^2 + a_7\omega_8^5$	
$X(4) = a_0\omega_8^0 + a_1\omega_8^4 + a_2\omega_8^0 + a_3\omega_8^4 + a_4\omega_8^0 + a_5\omega_8^4 + a_6\omega_8^0 + a_7\omega_8^4$	
$X(5) = a_0\omega_8^0 + a_1\omega_8^5 + a_2\omega_8^2 + a_3\omega_8^7 + a_4\omega_8^4 + a_5\omega_8^1 + a_6\omega_8^6 + a_7\omega_8^3$	
$X(6) = a_0\omega_8^0 + a_1\omega_8^6 + a_2\omega_8^4 + a_3\omega_8^2 + a_4\omega_8^0 + a_5\omega_8^6 + a_6\omega_8^4 + a_7\omega_8^2$	
$X(7) = a_0\omega_8^0 + a_1\omega_8^7 + a_2\omega_8^6 + a_3\omega_8^5 + a_4\omega_8^4 + a_5\omega_8^3 + a_6\omega_8^2 + a_7\omega_8^1$	



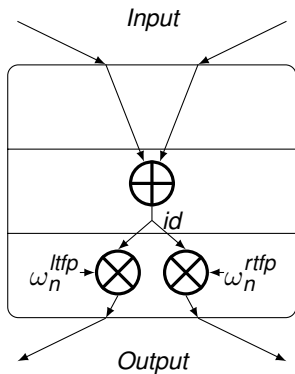
# Expanded 8-Point Discrete Fourier Transform

$$X(k) = \sum_{j=0}^{n-1} a_j \omega_n^{jk \pmod{n}}, \quad k = 0, 1, 2, \dots, n-1$$

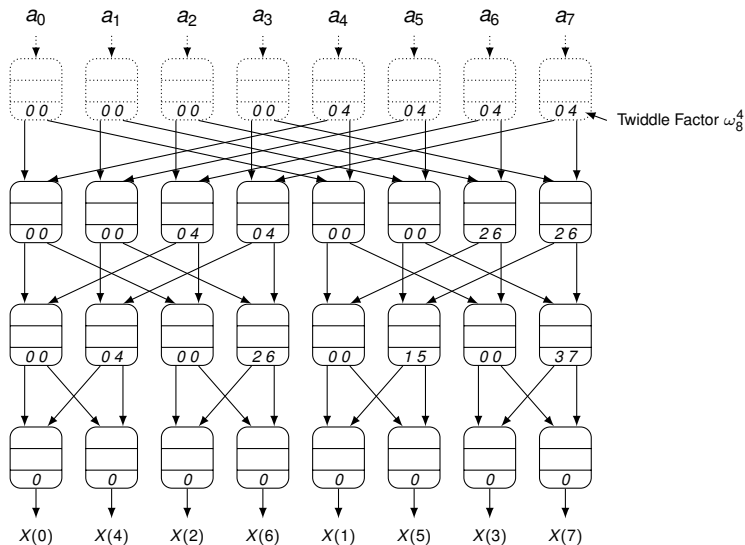
←----- n ----->

$X(0) = a_0\omega_8^0 + a_1\omega_8^0 + a_2\omega_8^0 + a_3\omega_8^0 + a_4\omega_8^0 + a_5\omega_8^0 + a_6\omega_8^0 + a_7\omega_8^0$	↑ $n$ ↓
$X(1) = a_0\omega_8^0 + a_1\omega_8^1 + a_2\omega_8^2 + a_3\omega_8^3 + a_4\omega_8^4 + a_5\omega_8^5 + a_6\omega_8^6 + a_7\omega_8^7$	
$X(2) = a_0\omega_8^0 + a_1\omega_8^2 + a_2\omega_8^4 + a_3\omega_8^6 + a_4\omega_8^0 + a_5\omega_8^2 + a_6\omega_8^4 + a_7\omega_8^6$	
$X(3) = a_0\omega_8^0 + a_1\omega_8^3 + a_2\omega_8^6 + a_3\omega_8^1 + a_4\omega_8^4 + a_5\omega_8^7 + a_6\omega_8^2 + a_7\omega_8^5$	
$X(4) = a_0\omega_8^0 + a_1\omega_8^4 + a_2\omega_8^0 + a_3\omega_8^4 + a_4\omega_8^0 + a_5\omega_8^4 + a_6\omega_8^0 + a_7\omega_8^4$	
$X(5) = a_0\omega_8^0 + a_1\omega_8^5 + a_2\omega_8^2 + a_3\omega_8^7 + a_4\omega_8^4 + a_5\omega_8^1 + a_6\omega_8^6 + a_7\omega_8^3$	
$X(6) = a_0\omega_8^0 + a_1\omega_8^6 + a_2\omega_8^4 + a_3\omega_8^2 + a_4\omega_8^0 + a_5\omega_8^6 + a_6\omega_8^4 + a_7\omega_8^2$	
$X(7) = a_0\omega_8^0 + a_1\omega_8^7 + a_2\omega_8^6 + a_3\omega_8^5 + a_4\omega_8^4 + a_5\omega_8^3 + a_6\omega_8^2 + a_7\omega_8^1$	

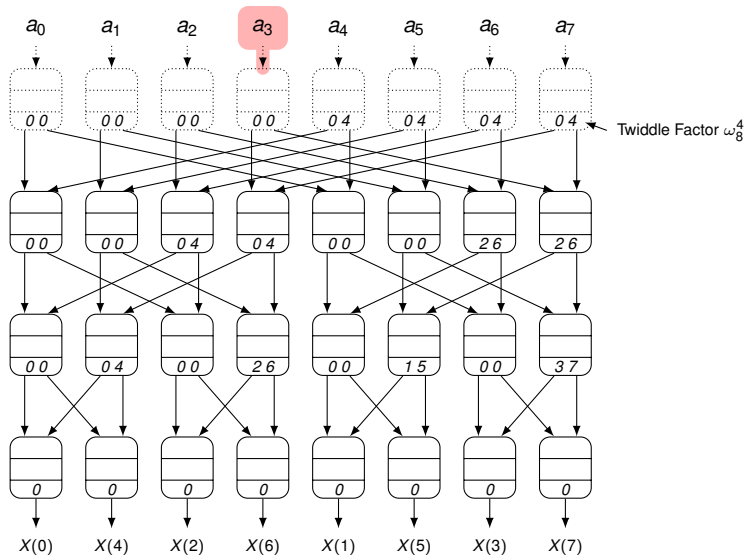
# Graph Vertex Internals



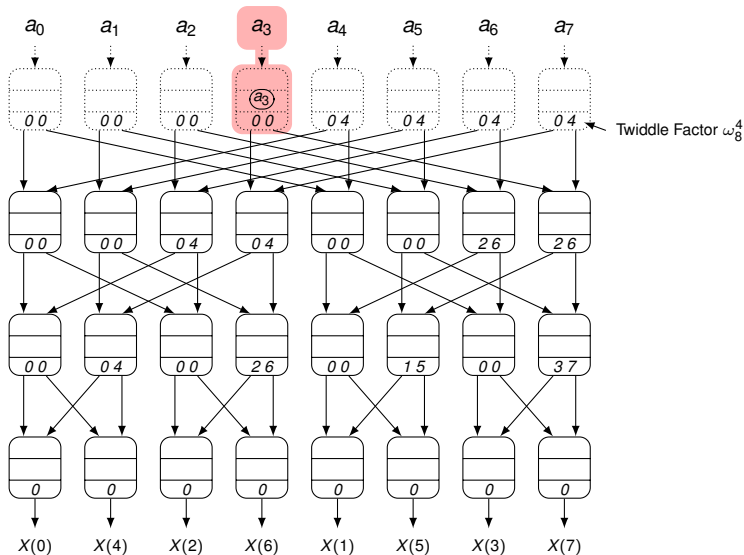
# Classic Fast Fourier Transform



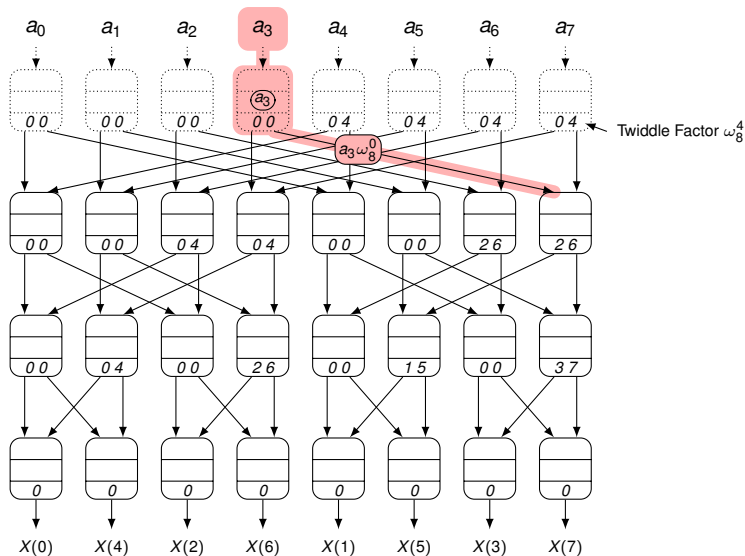
# Classic Fast Fourier Transform



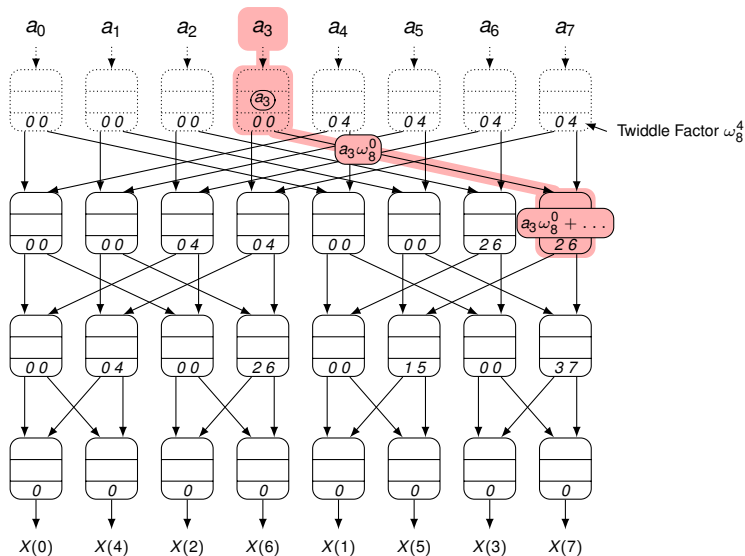
# Classic Fast Fourier Transform



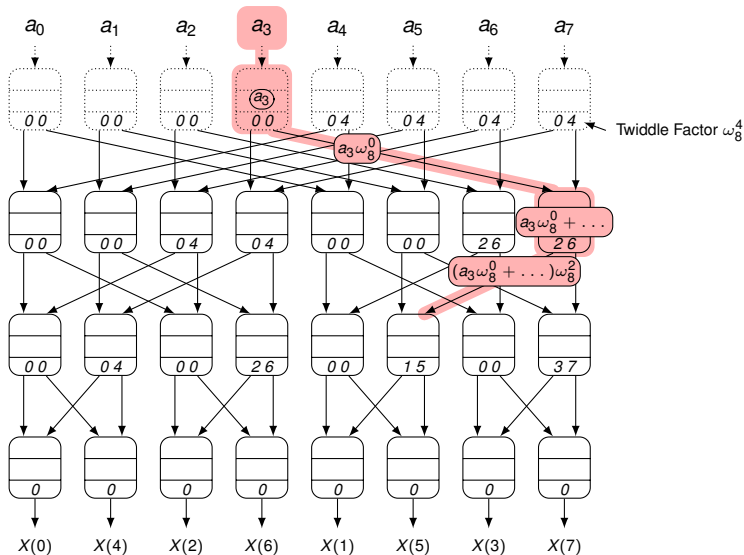
# Classic Fast Fourier Transform



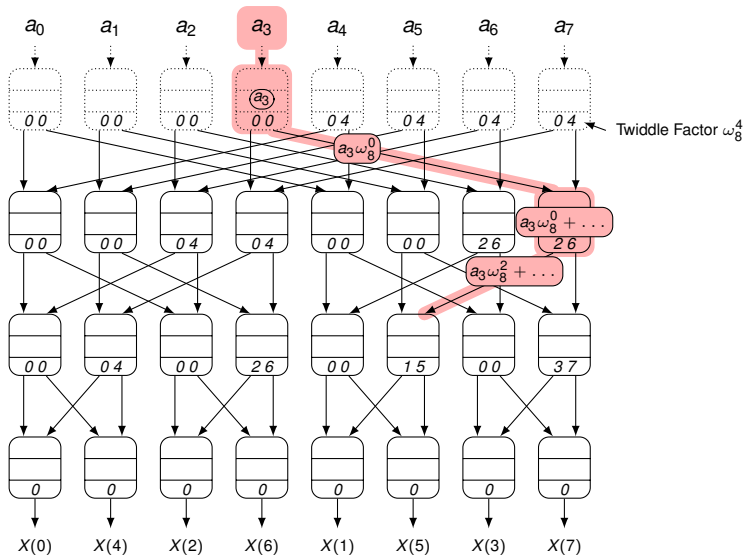
# Classic Fast Fourier Transform



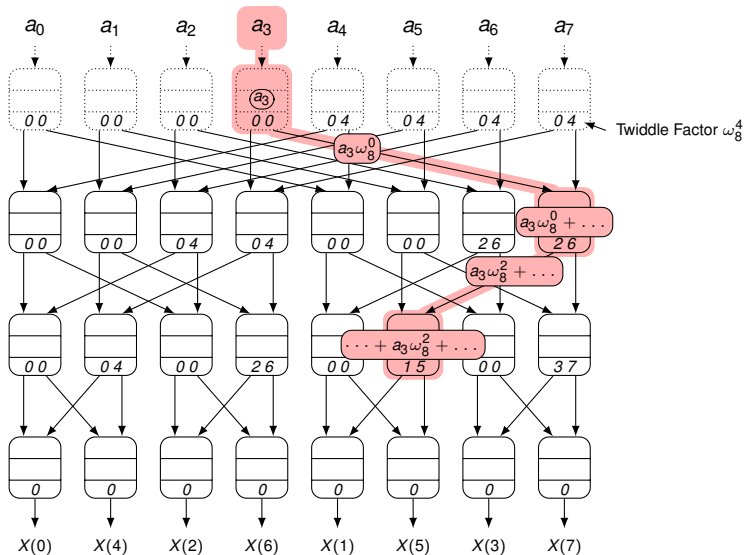
# Classic Fast Fourier Transform



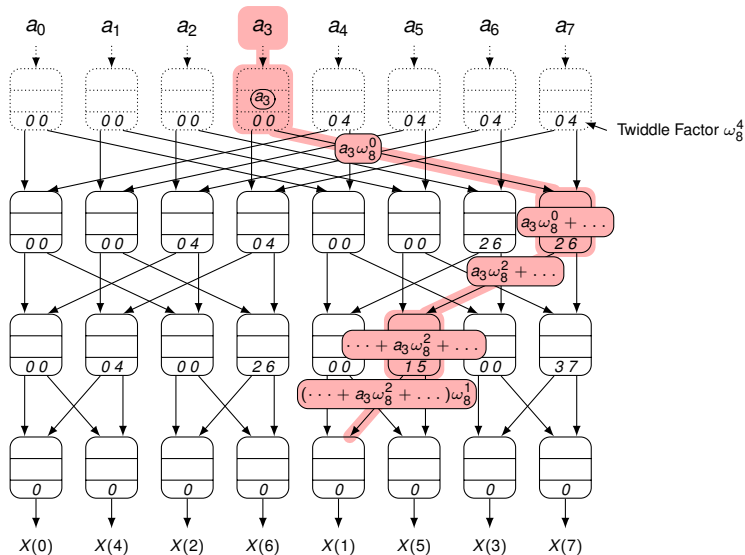
# Classic Fast Fourier Transform



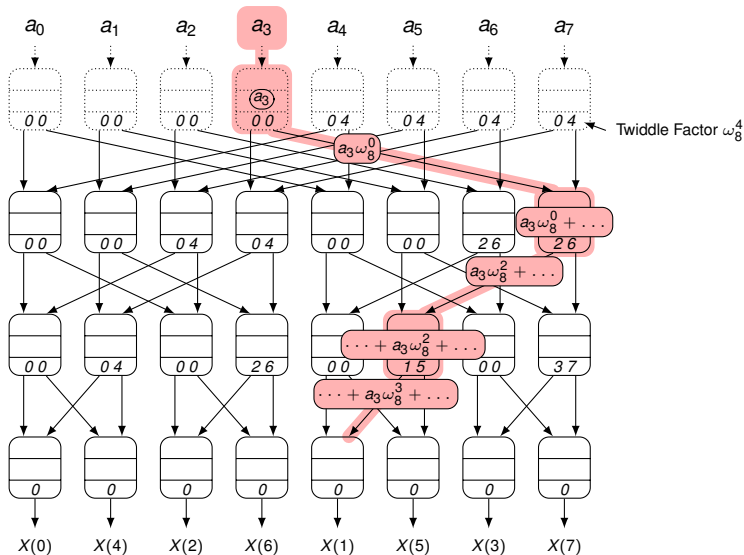
# Classic Fast Fourier Transform



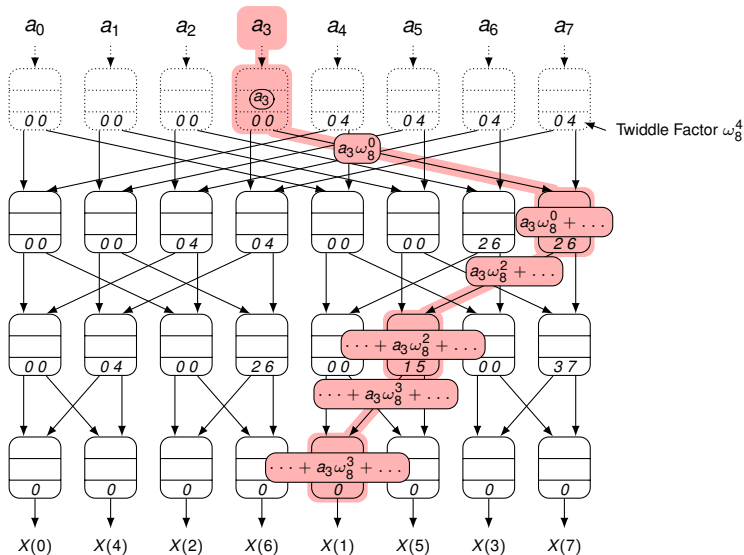
# Classic Fast Fourier Transform



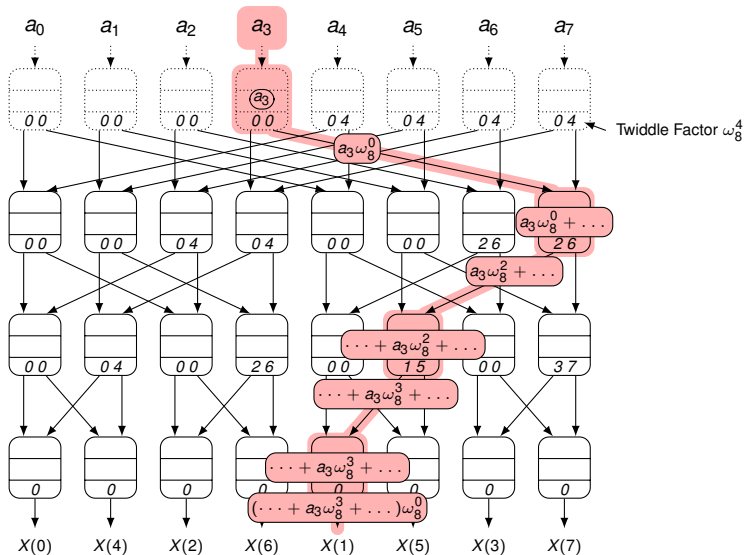
# Classic Fast Fourier Transform



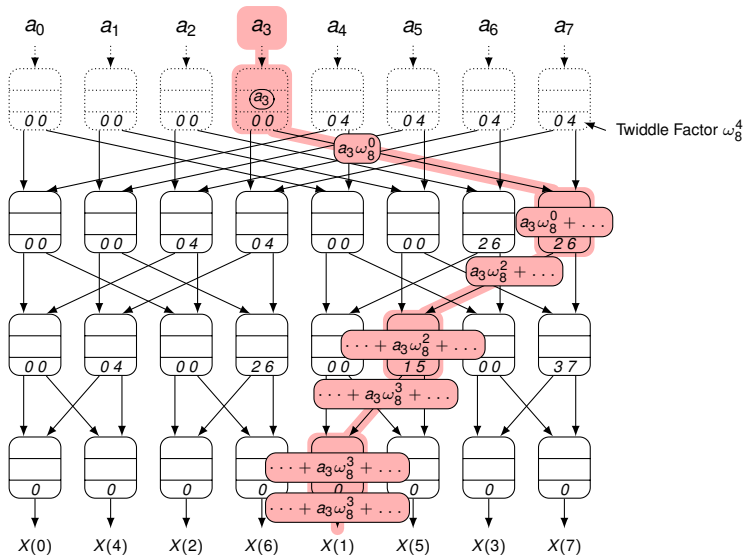
# Classic Fast Fourier Transform



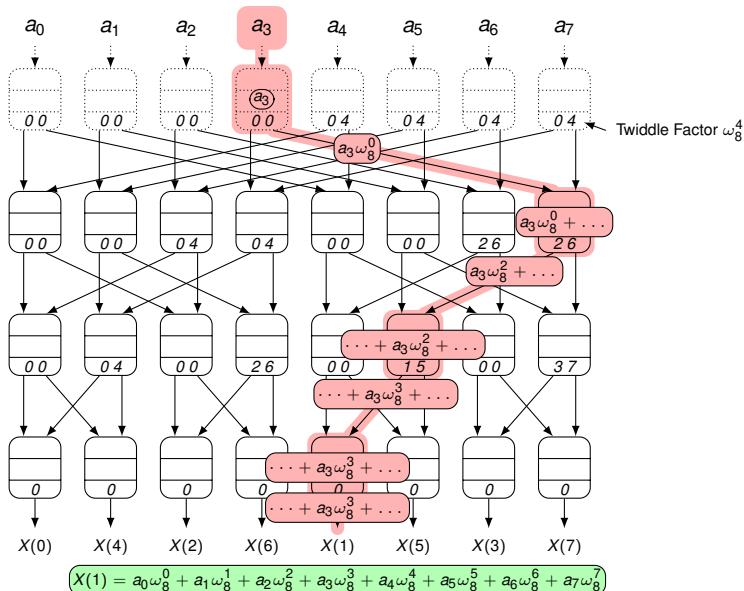
# Classic Fast Fourier Transform



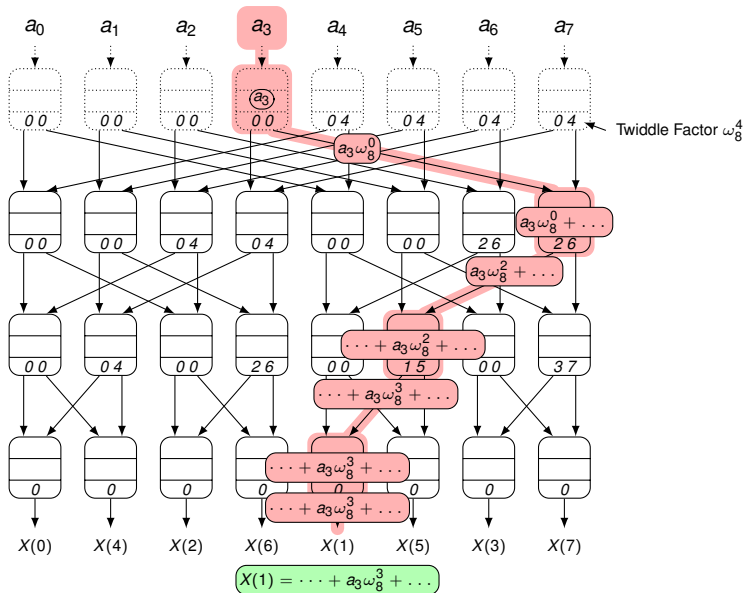
# Classic Fast Fourier Transform



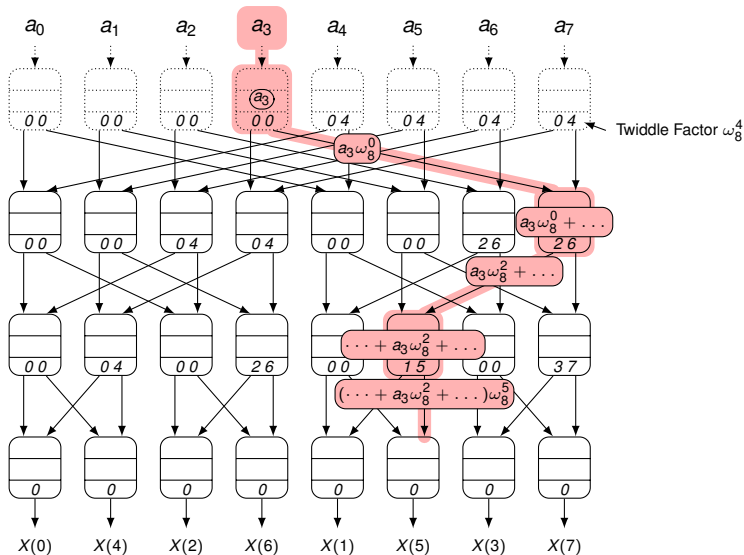
# Classic Fast Fourier Transform



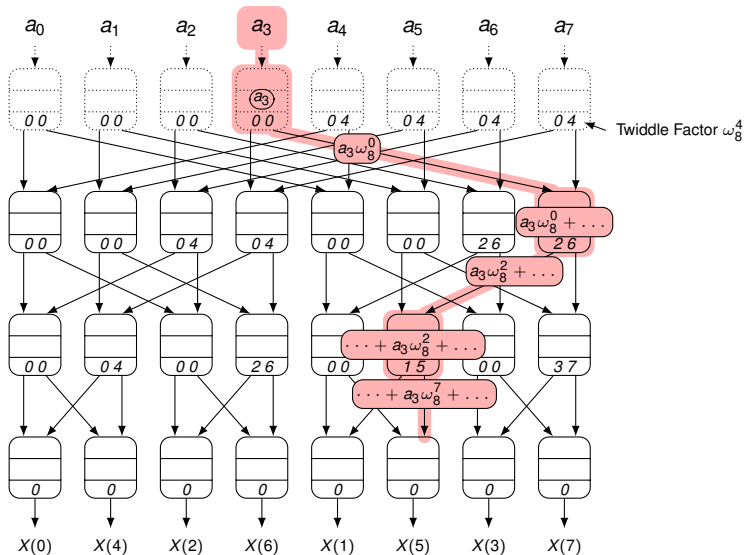
# Classic Fast Fourier Transform



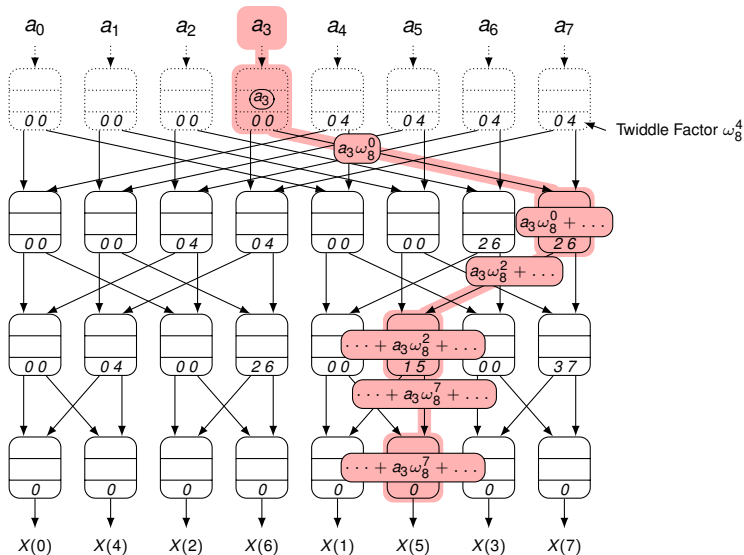
# Classic Fast Fourier Transform



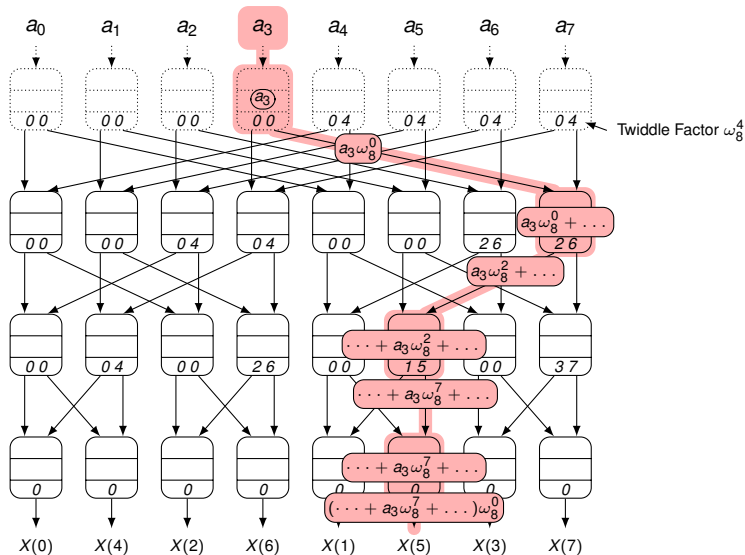
# Classic Fast Fourier Transform



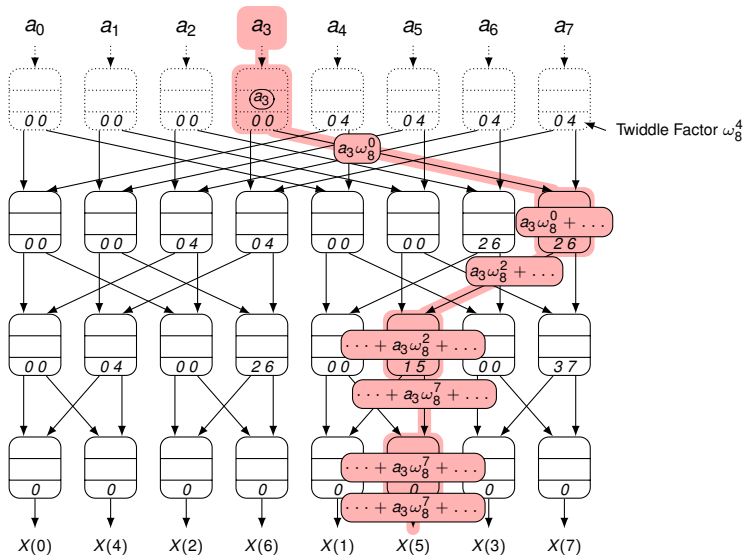
# Classic Fast Fourier Transform



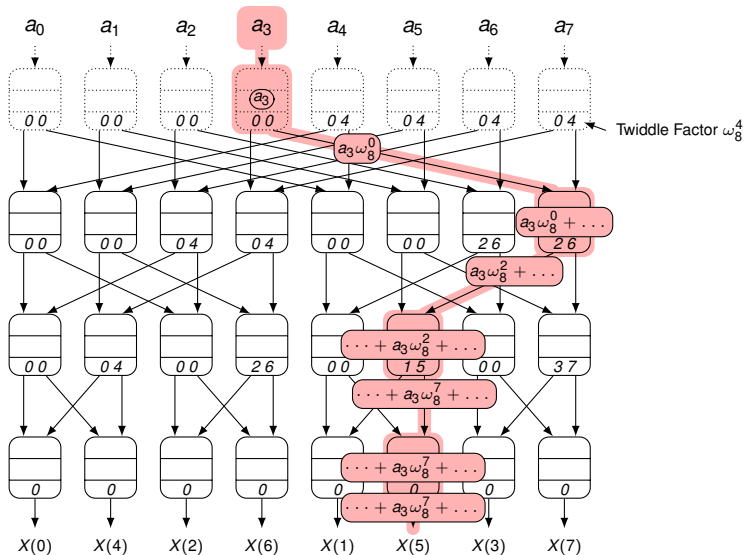
# Classic Fast Fourier Transform



# Classic Fast Fourier Transform

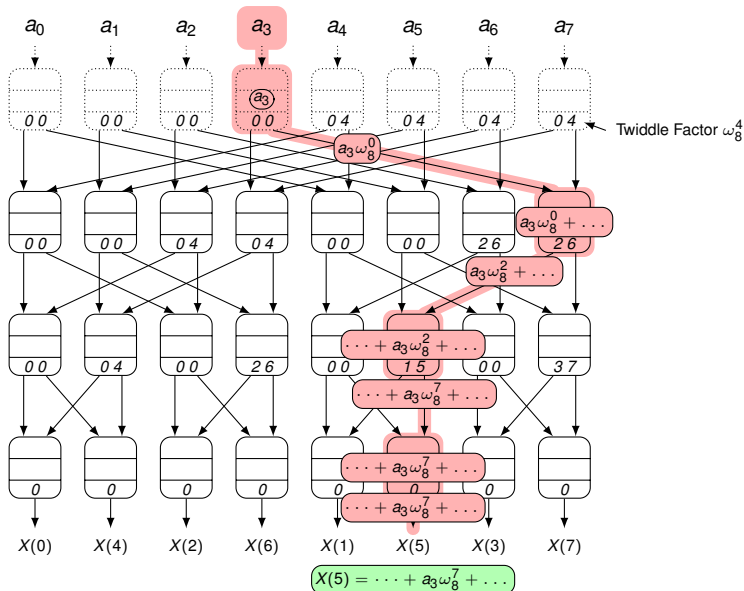


# Classic Fast Fourier Transform

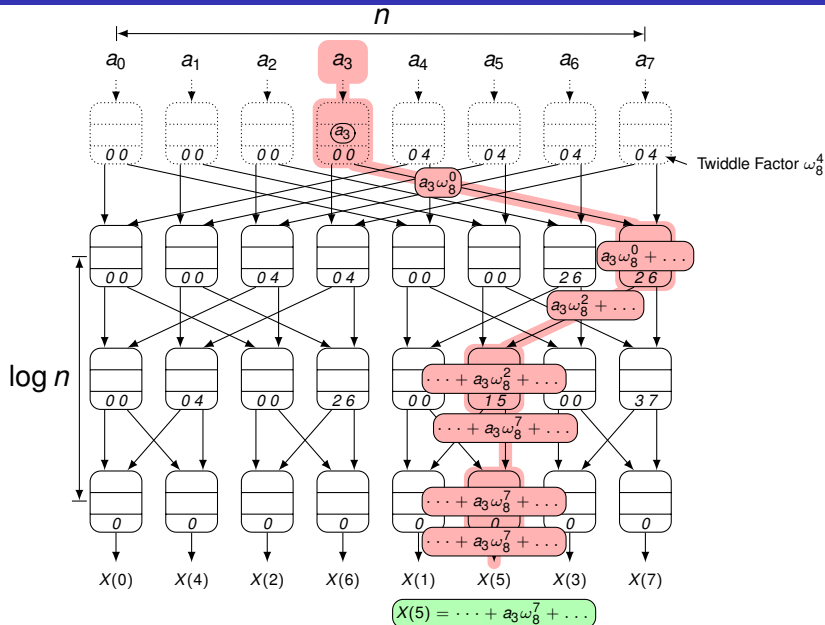


$$X(5) = a_0\omega_8^0 + a_1\omega_8^5 + a_2\omega_8^2 + a_3\omega_8^7 + a_4\omega_8^4 + a_5\omega_8^1 + a_6\omega_8^6 + a_7\omega_8^3$$

# Classic Fast Fourier Transform



# Classic Fast Fourier Transform



# The Costs of Calculating Complex Multiplication

- A computer represents complex numbers as  $a + bi$

$$\begin{aligned}z &= (a_1 + b_1 i)(a_2 + b_2 i) \\ &= a_1 a_2 + a_1 b_2 i + b_1 a_2 + b_1 b_2 i^2 \\ &= (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) i\end{aligned}$$

$$\Re(z) = (a_1 a_2 - b_1 b_2)$$

$$\Im(z) = (a_1 b_2 + b_1 a_2) i$$

- $\Re(z)$  requires 2 real multiplications and 1 real addition
- $\Im(z)$  also requires 2 real multiplications and 1 real addition
- 6 floating point operations (FLOPS) required for a complete complex multiplication

# The Costs of Calculating Complex Multiplication

- A computer represents complex numbers as  $a + bi$

$$\begin{aligned}z &= (a_1 + b_1 i)(a_2 + b_2 i) \\ &= a_1 a_2 + a_1 b_2 i + b_1 a_2 + b_1 b_2 i^2 \\ &= (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) i \\ \Re(z) &= (a_1 a_2 - b_1 b_2) \\ \Im(z) &= (a_1 b_2 + b_1 a_2) i\end{aligned}$$

- $\Re(z)$  requires 2 real multiplications and 1 real addition
- $\Im(z)$  also requires 2 real multiplications and 1 real addition
- 6 floating point operations (FLOPS) required for a complete complex multiplication

# The Costs of Calculating Complex Multiplication

- A computer represents complex numbers as  $a + bi$

$$\begin{aligned}z &= (a_1 + b_1 i)(a_2 + b_2 i) \\ &= a_1 a_2 + a_1 b_2 i + b_1 a_2 + b_1 b_2 i^2 \\ &= (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) i \\ \Re(z) &= (a_1 a_2 - b_1 b_2) \\ \Im(z) &= (a_1 b_2 + b_1 a_2) i\end{aligned}$$

- $\Re(z)$  requires 2 real multiplications and 1 real addition
- $\Im(z)$  also requires 2 real multiplications and 1 real addition
- 6 floating point operations (FLOPS) required for a complete complex multiplication

# The Costs of Calculating Complex Multiplication

- A computer represents complex numbers as  $a + bi$

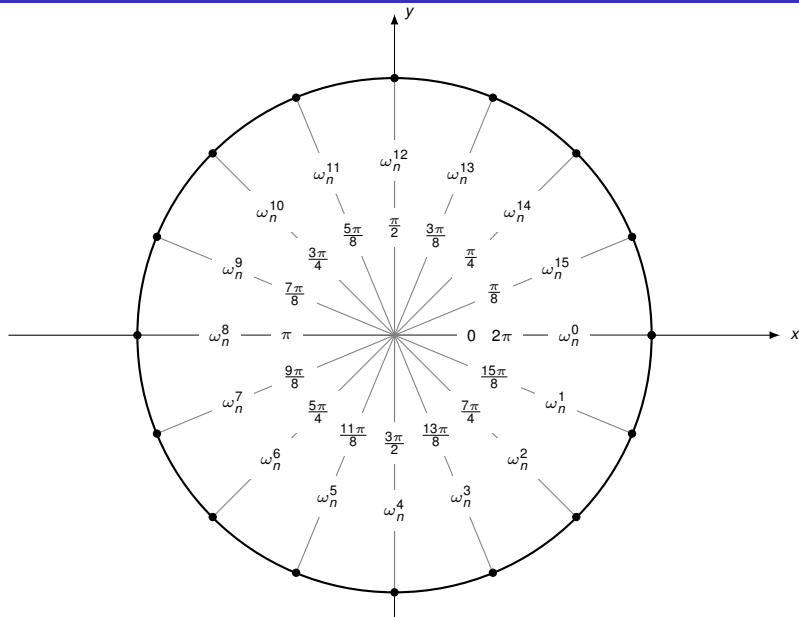
$$\begin{aligned}z &= (a_1 + b_1 i)(a_2 + b_2 i) \\ &= a_1 a_2 + a_1 b_2 i + b_1 a_2 + b_1 b_2 i^2 \\ &= (a_1 a_2 - b_1 b_2) + (a_1 b_2 + b_1 a_2) i\end{aligned}$$

$$\Re(z) = (a_1 a_2 - b_1 b_2)$$

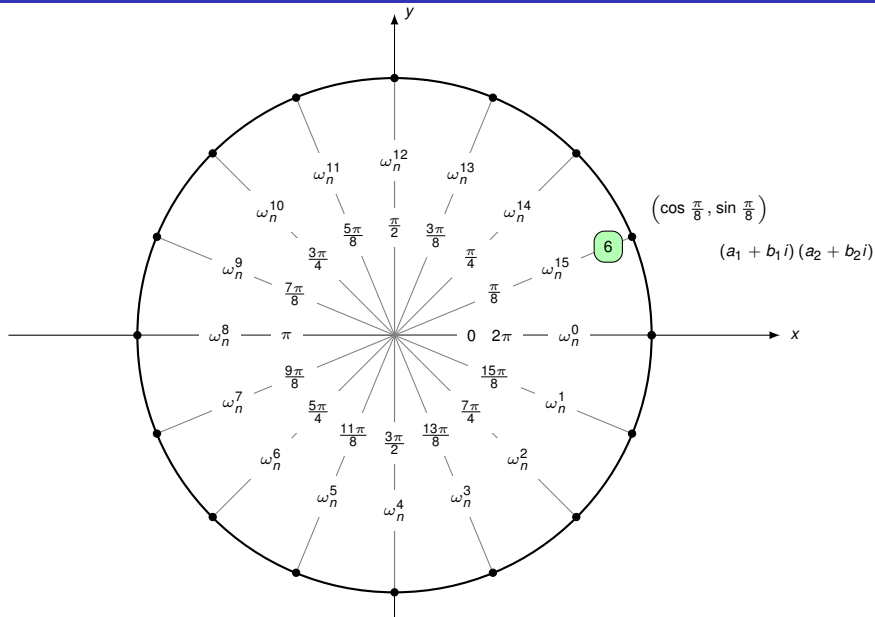
$$\Im(z) = (a_1 b_2 + b_1 a_2) i$$

- $\Re(z)$  requires 2 real multiplications and 1 real addition
- $\Im(z)$  also requires 2 real multiplications and 1 real addition
- 6 floating point operations (FLOPS) required for a complete complex multiplication

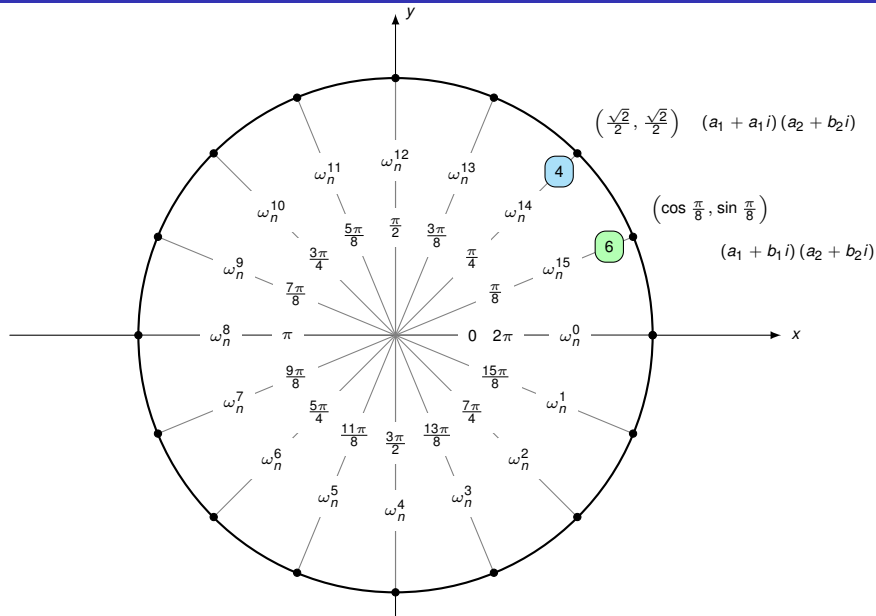
# Not All Complex Multiplications Cost the Same



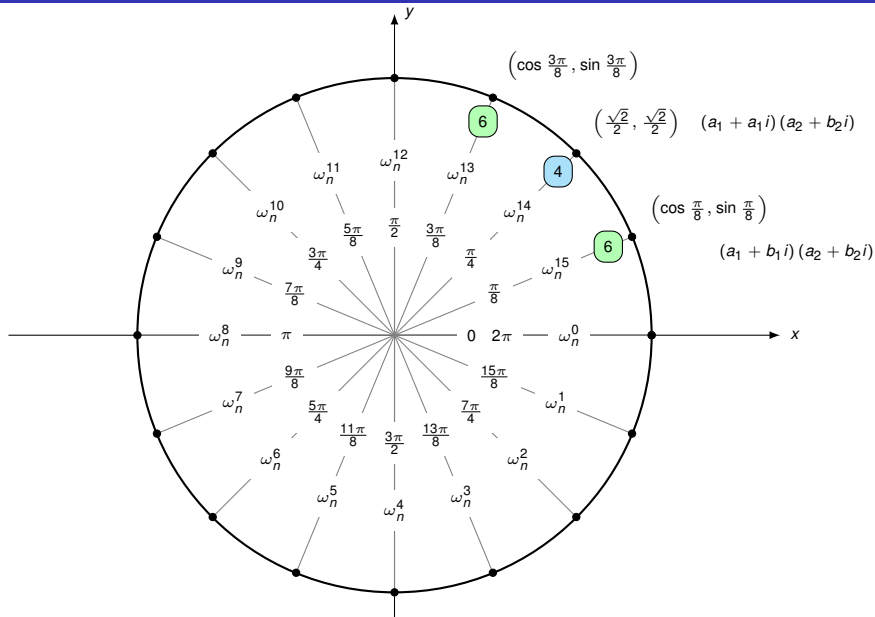
# Not All Complex Multiplications Cost the Same



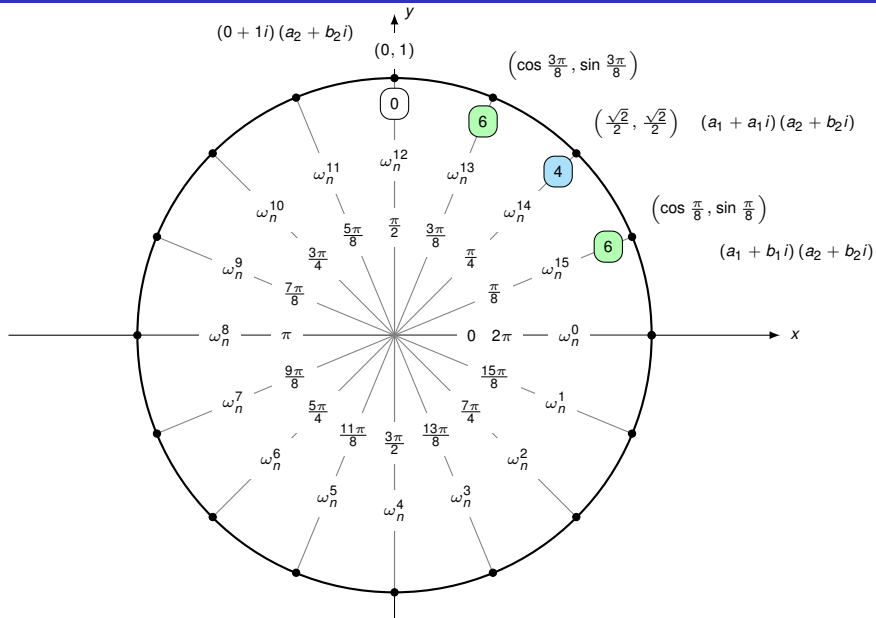
# Not All Complex Multiplications Cost the Same



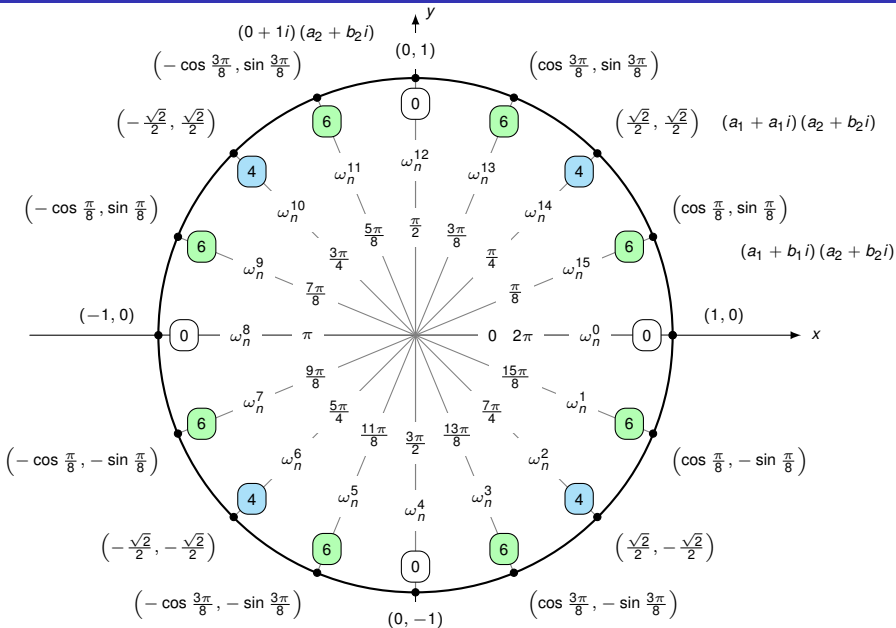
# Not All Complex Multiplications Cost the Same



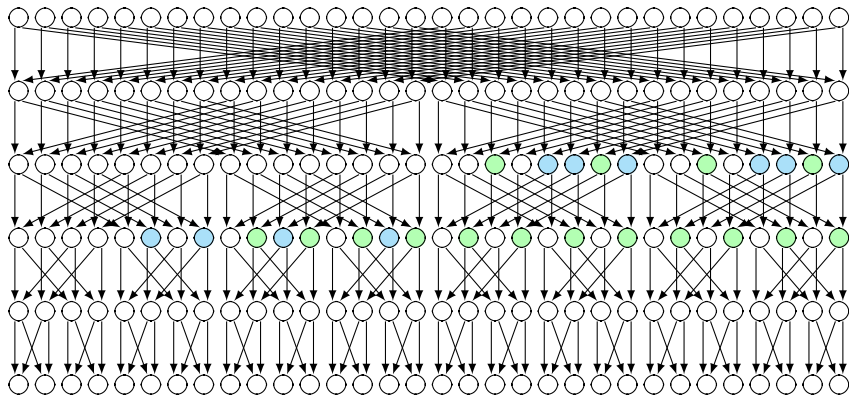
# Not All Complex Multiplications Cost the Same



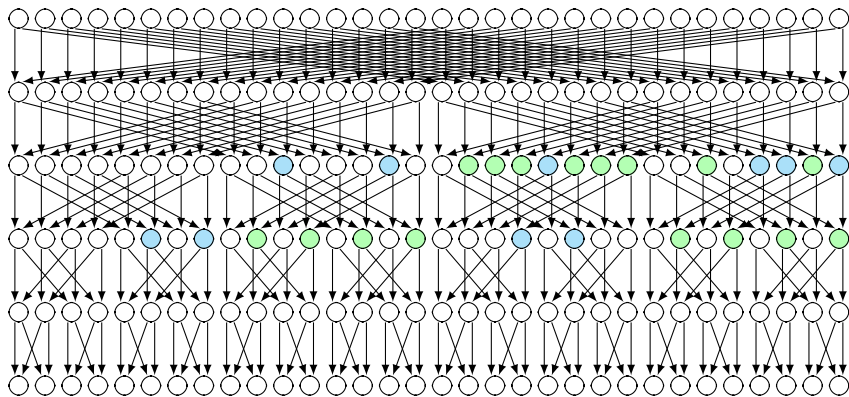
# Not All Complex Multiplications Cost the Same



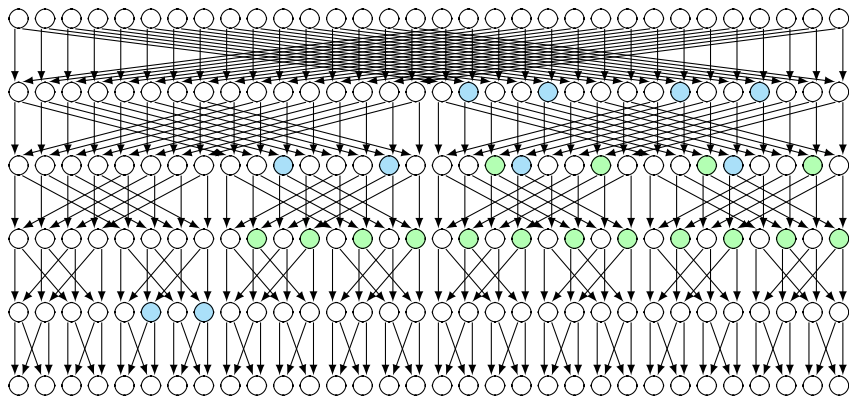
# 32-Point FFT requiring 456 FLOPs



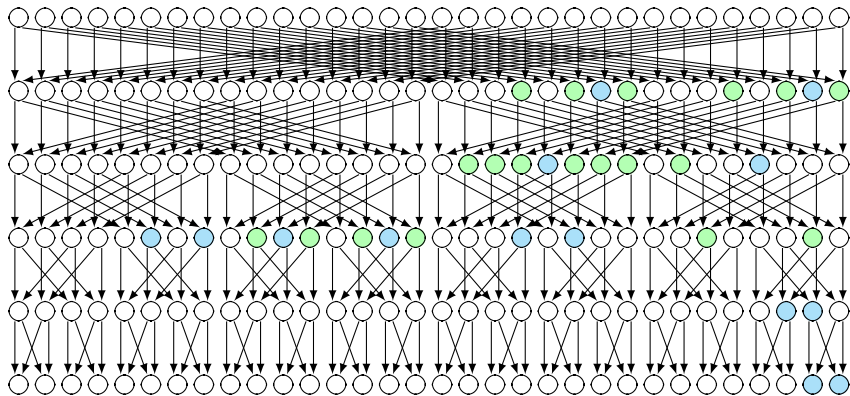
# Another 32-Point FFT requiring 456 FLOPs



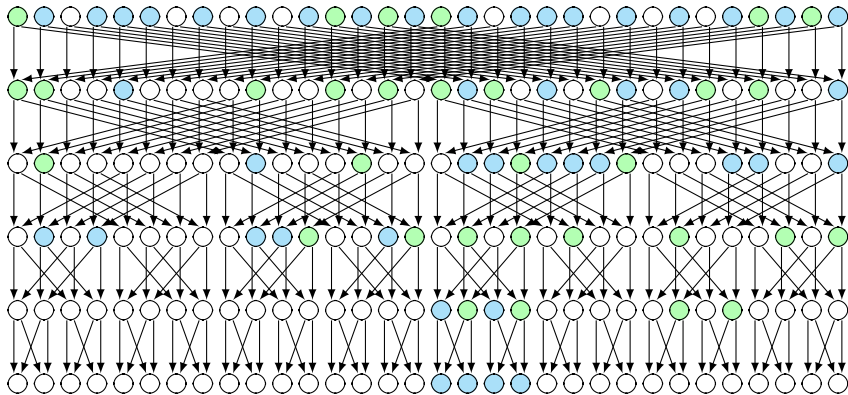
# Yet Another 32-Point FFT requiring 456 FLOPs



# 32-Point FFT requiring 490 FLOPs



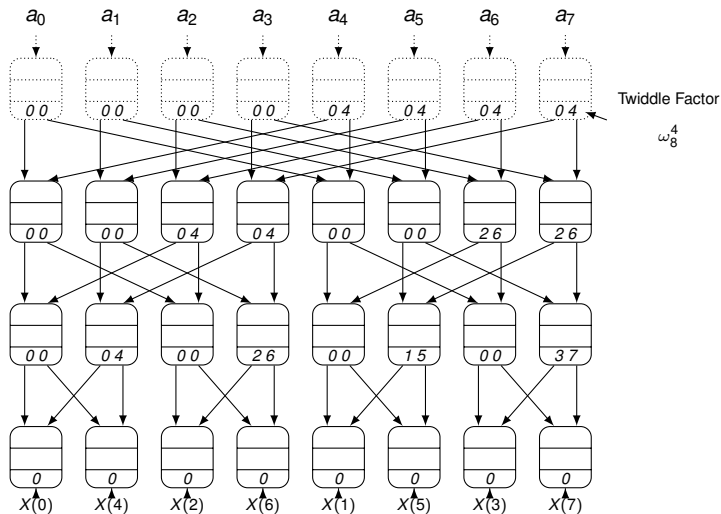
# 32-Point FFT requiring 688 FLOPs



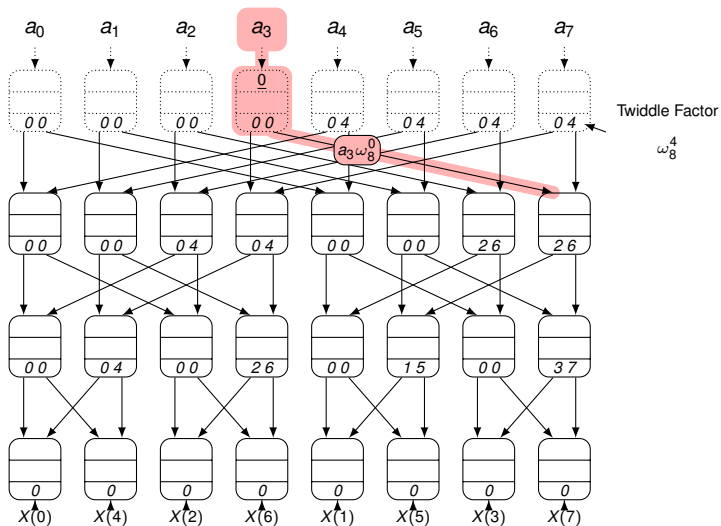
# Outline

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms**
- 3 Searching a Family of FFT Algorithms
- 4 Results and Conclusions

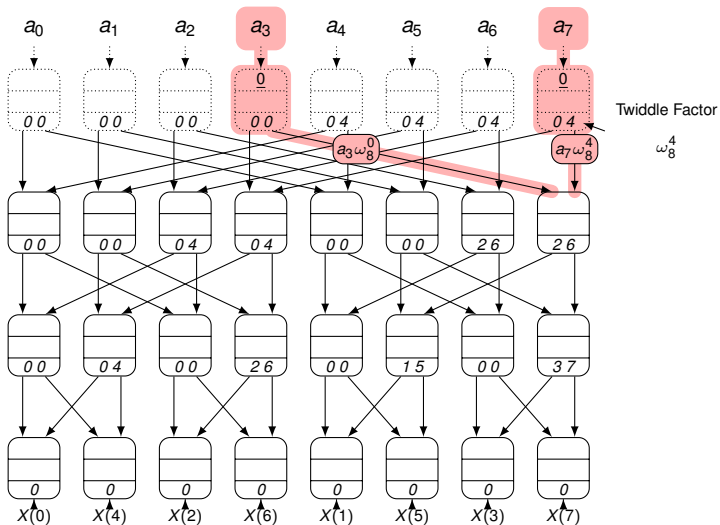
# Weight Stride Invariant



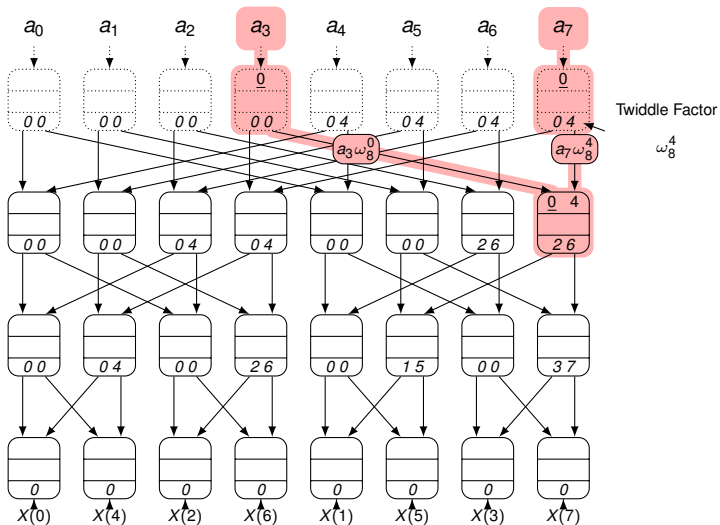
# Weight Stride Invariant



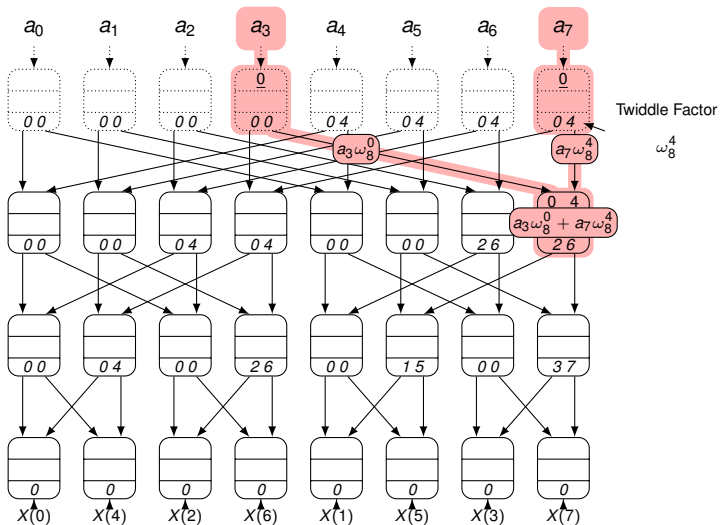
# Weight Stride Invariant



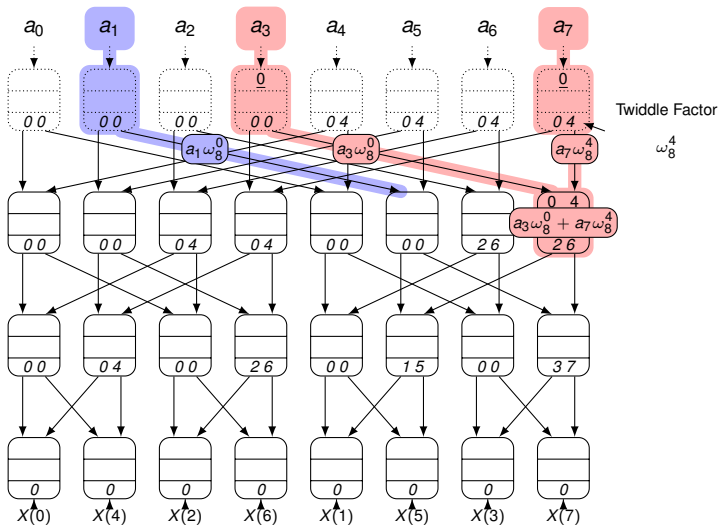
# Weight Stride Invariant



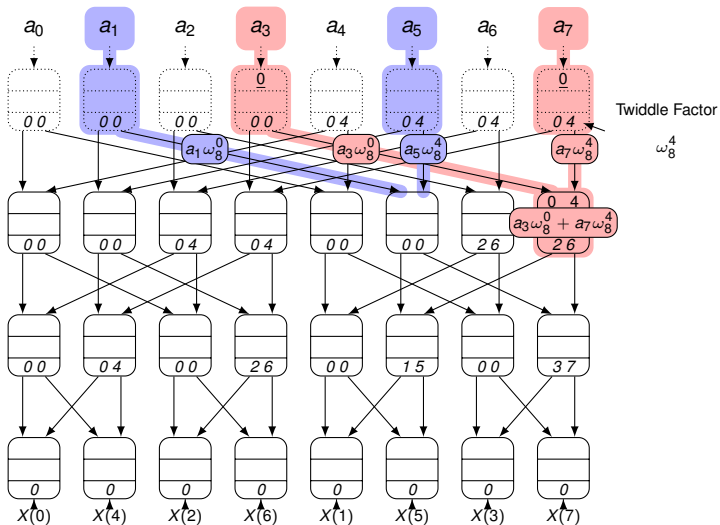
# Weight Stride Invariant



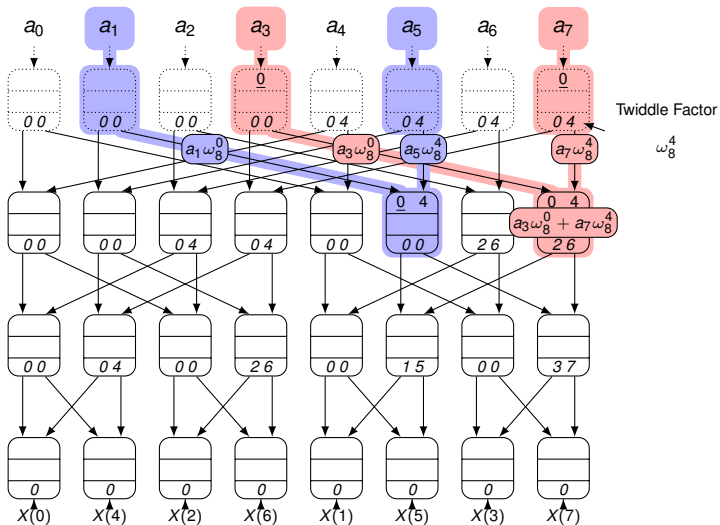
# Weight Stride Invariant



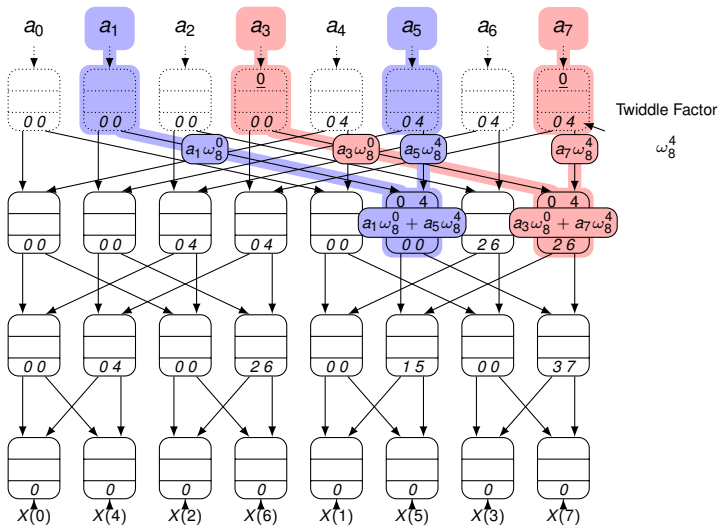
# Weight Stride Invariant



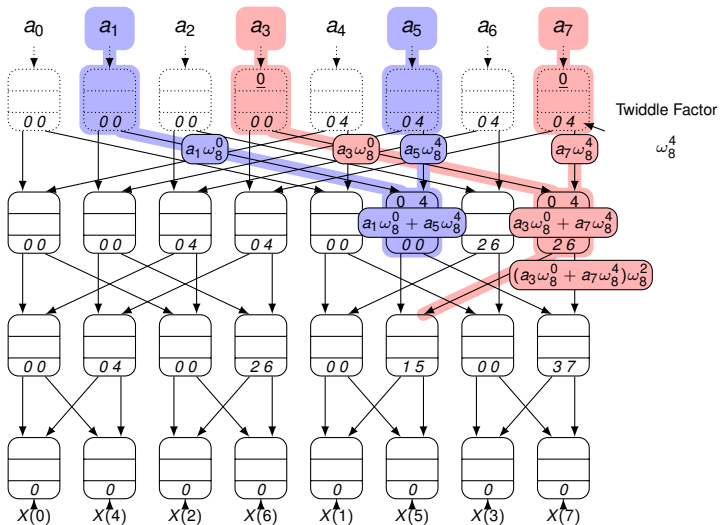
# Weight Stride Invariant



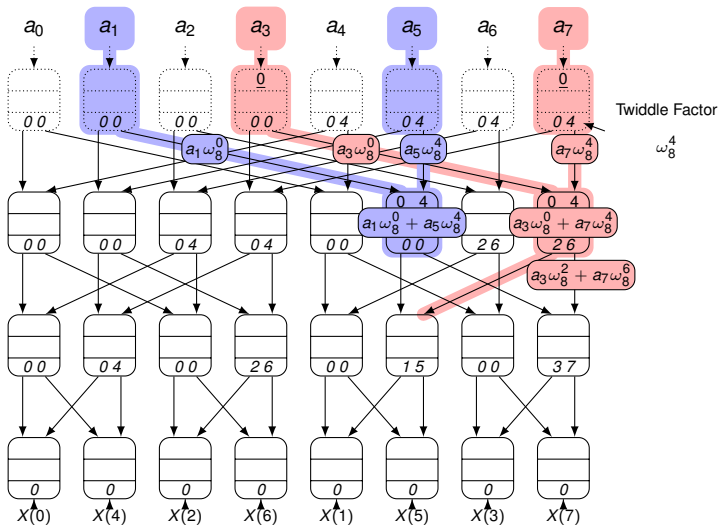
# Weight Stride Invariant



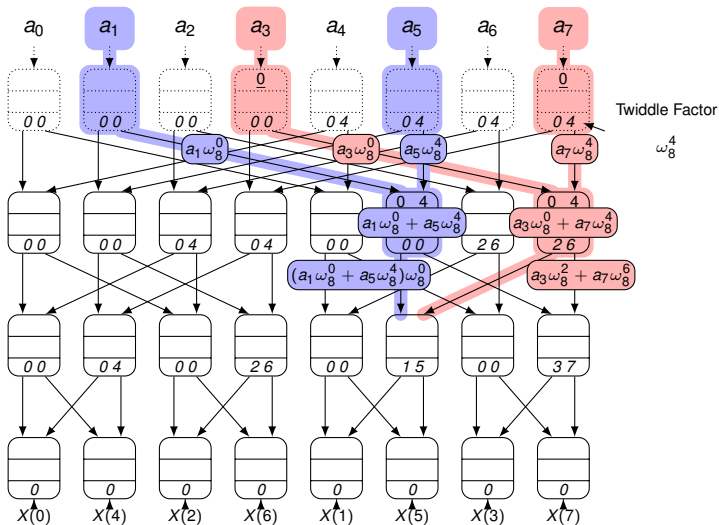
# Weight Stride Invariant



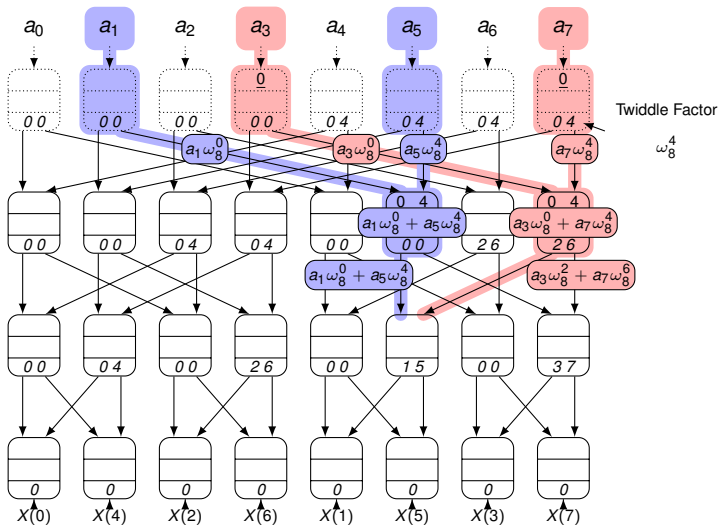
# Weight Stride Invariant



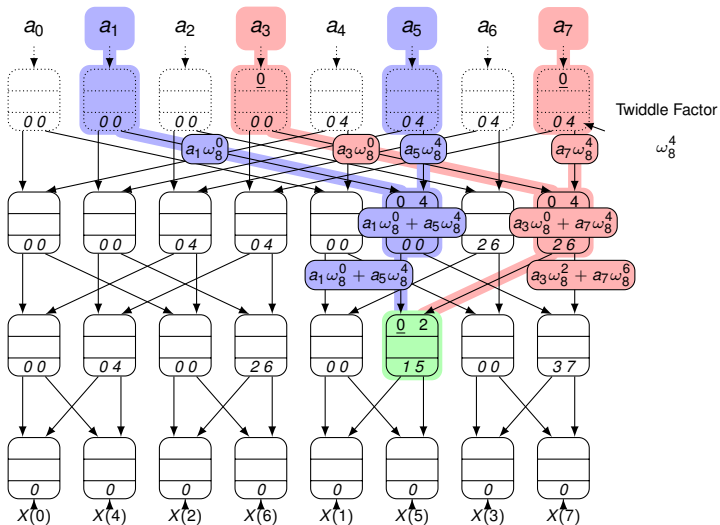
# Weight Stride Invariant



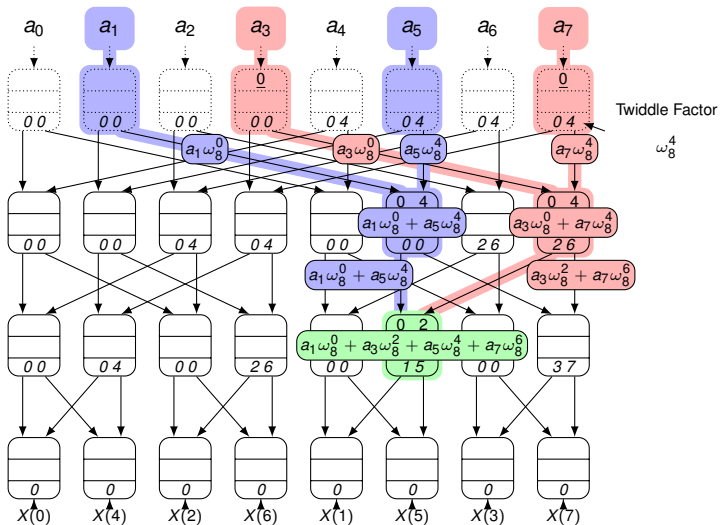
# Weight Stride Invariant



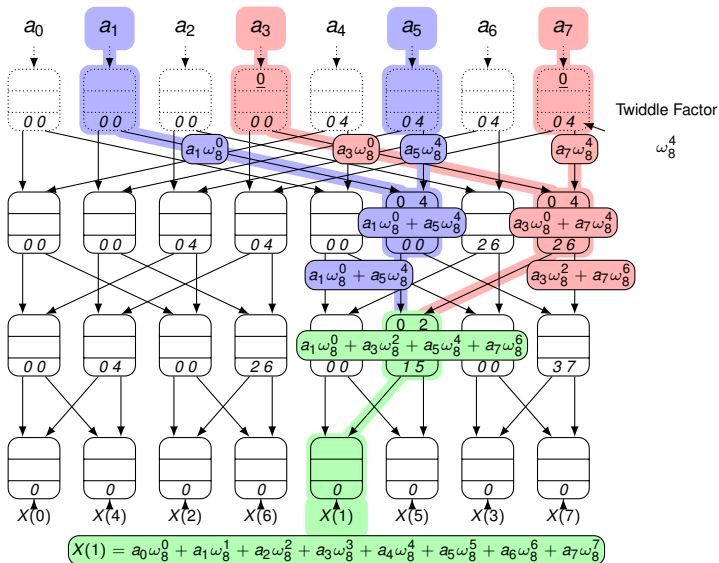
# Weight Stride Invariant



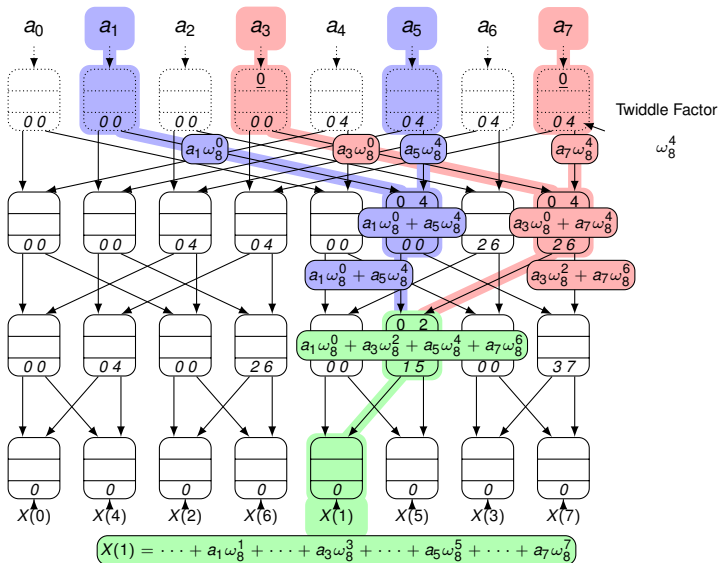
# Weight Stride Invariant



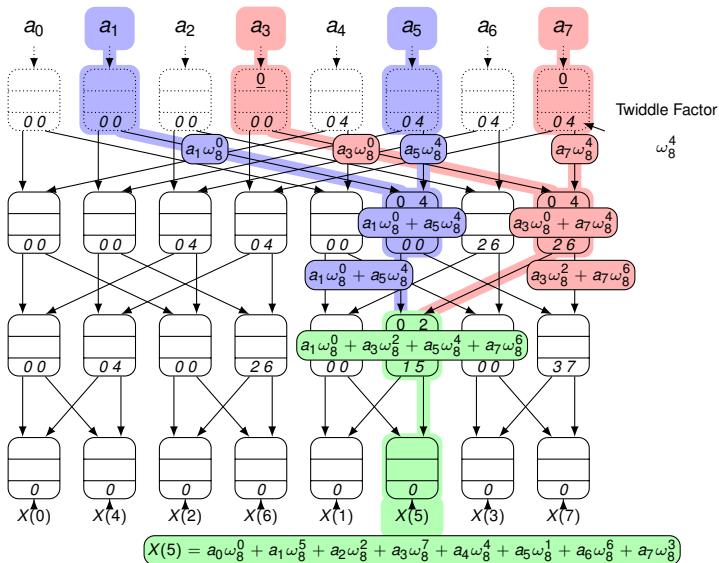
# Weight Stride Invariant



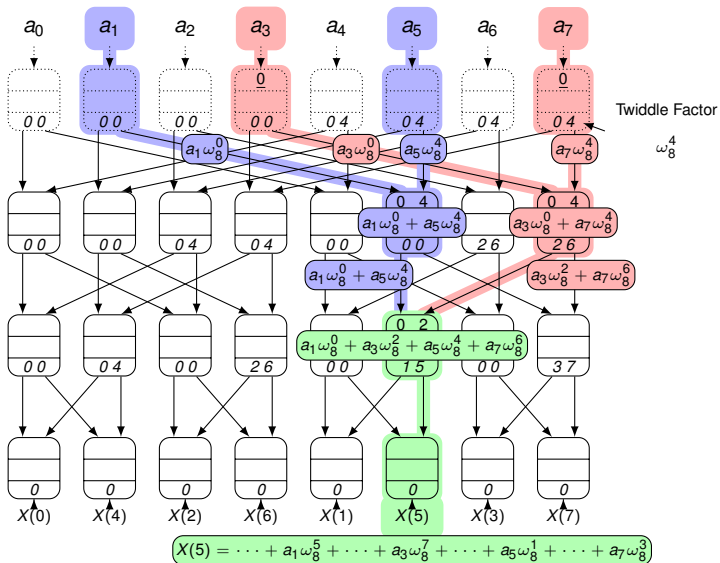
# Weight Stride Invariant



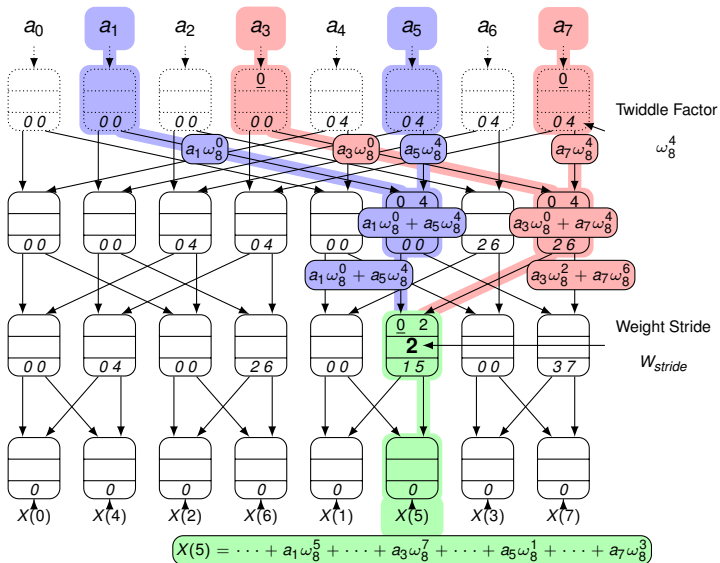
# Weight Stride Invariant



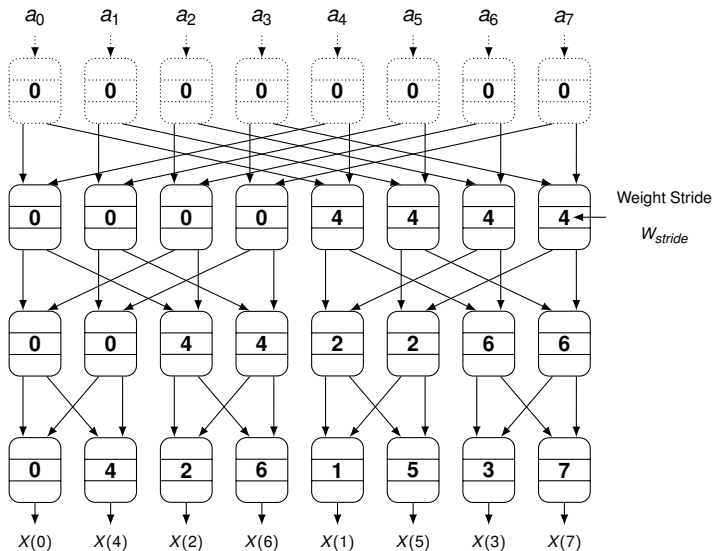
# Weight Stride Invariant



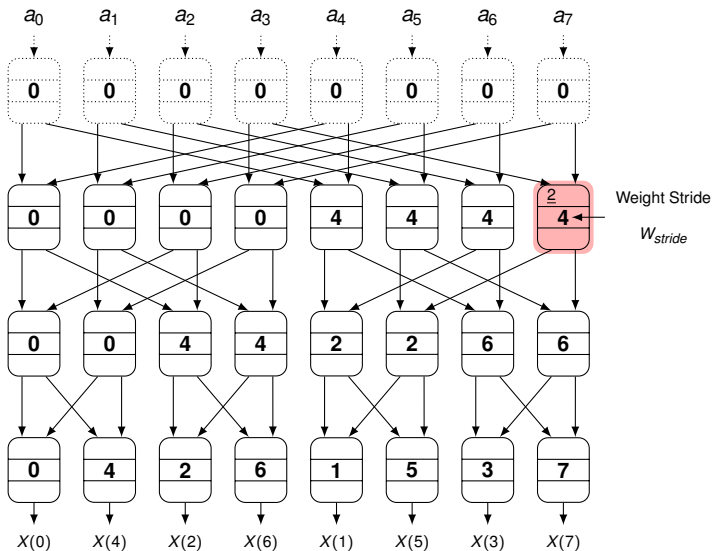
# Weight Stride Invariant



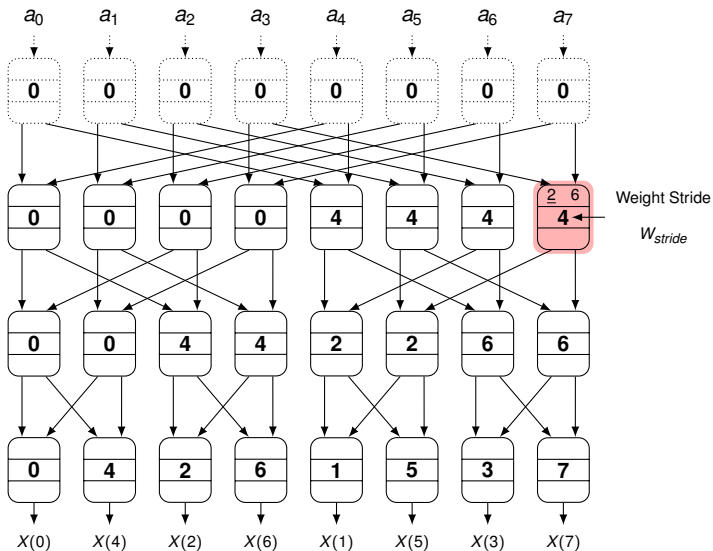
# A Random Family Member



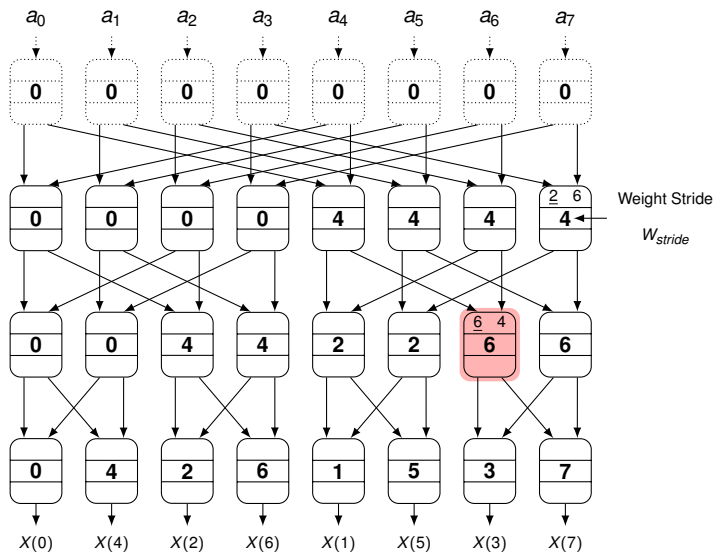
# A Random Family Member



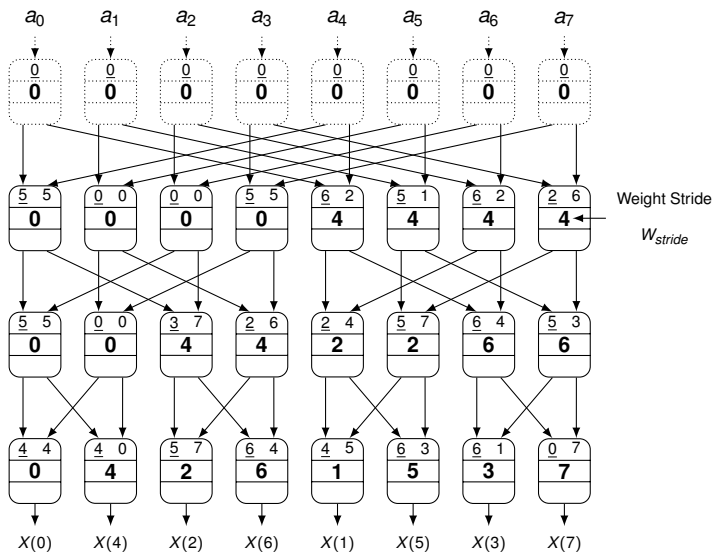
# A Random Family Member



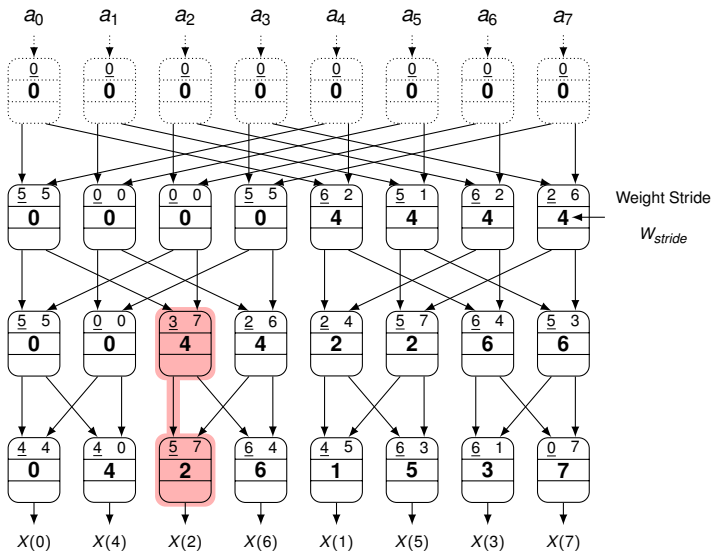
# A Random Family Member



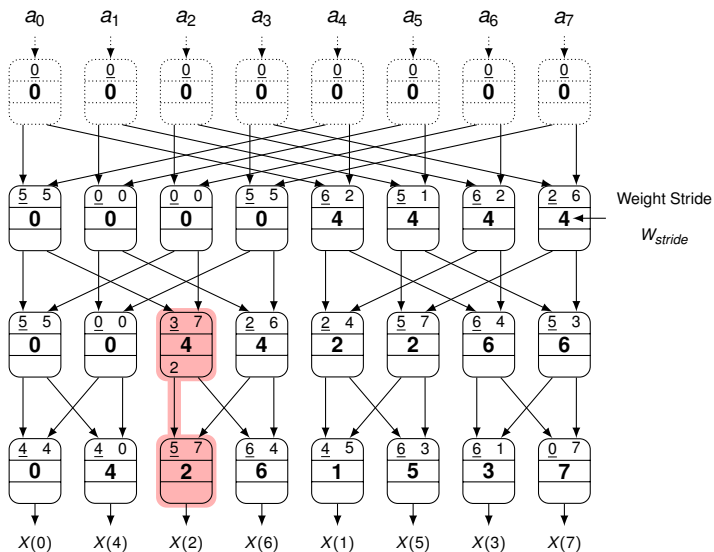
# A Random Family Member



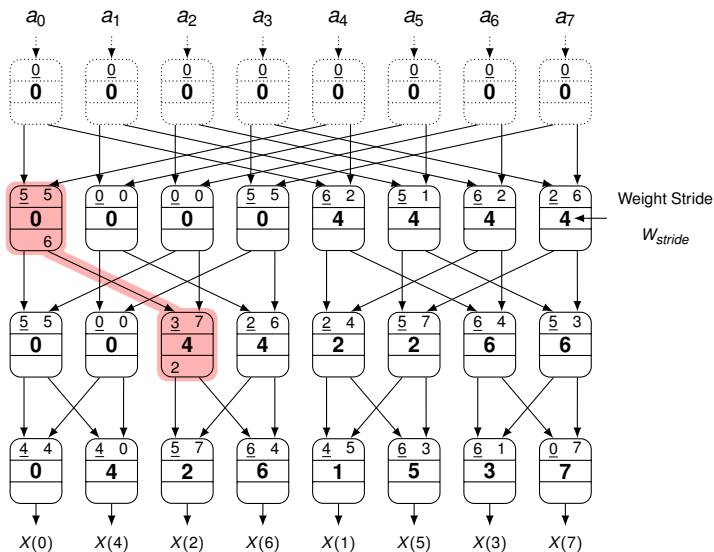
# A Random Family Member



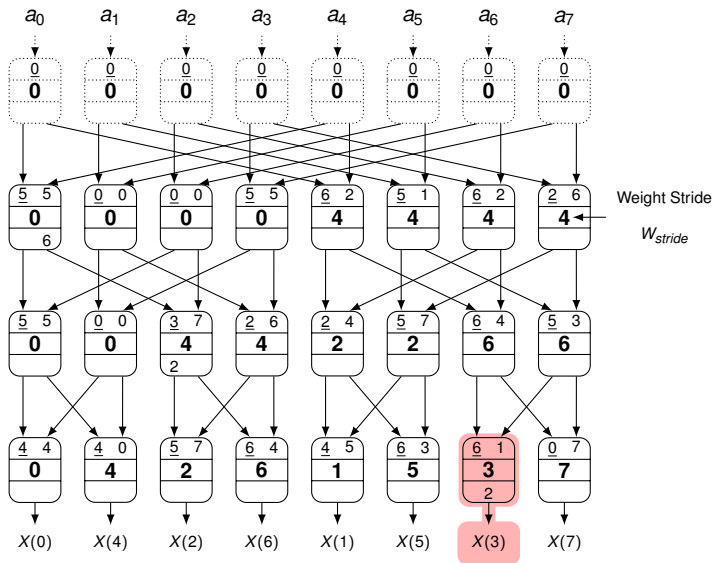
# A Random Family Member



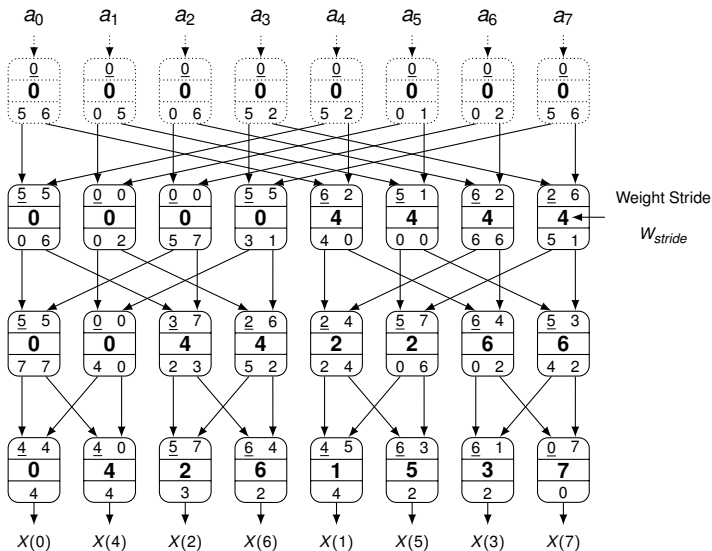
# A Random Family Member



# A Random Family Member



# A Random Family Member



# How to Generate a Random Member FFT Algorithm

**Input:** Size- $n$  flowgraph with labeled invariants

**Output:** Size- $n$  flowgraph with twiddle factors assigned

**foreach**  $nd \in \text{flowgraph}$  **do**

**if**  $nd.stride \neq n$  **then**

$nd.W_{base} \leftarrow \text{rand}() \pmod n$

$nd.rW_{base} \leftarrow nd.W_{base} + nd.W_{stride} \pmod n$

**else**

$nd.W_{base} \leftarrow 0$

**foreach**  $nd \in \text{flowgraph}$  **do**

**if**  $nd.stride \neq n$  **then**

$nd.lp.tfp \leftarrow nd.W_{base} - nd.lp.W_{base} \pmod n$

$nd.rp.tfp \leftarrow nd.rW_{base} - nd.rp.W_{base} \pmod n$

**if**  $nd.stride = 1$  **then**

$nd.tfp \leftarrow 0 - nd.W_{base} \pmod n$

# Outline

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms
- 3 Searching a Family of FFT Algorithms**
- 4 Results and Conclusions

# Searching a Family of FFT Algorithms

- All family members are not equally desirable
  - Some require fewer FLOPs
  - Others may require less communication of twiddle factors
  - **Need a way to search and find desirable members**
- How many family members are there?
  - $2^{n \log_2 n \log_2 n}$
  - For a 256-point FFT:  $2^{16384}$
  - Only 1 in  $2^{18432}$  chance of guessing correct twiddle factors
  - Estimated atoms in the universe is  $2^{264}$
  - Fastest supercomputer performs  $2^{144}$  FLOPS

# Searching a Family of FFT Algorithms

- All family members are not equally desirable
  - Some require fewer FLOPs
  - Others may require less communication of twiddle factors
  - **Need a way to search and find desirable members**
- How many family members are there?
  - $2^{n \log_2 n \log_2 n}$
  - For a 256-point FFT:  $2^{16384}$
  - Only 1 in  $2^{18432}$  chance of guessing correct twiddle factors
  - Estimated atoms in the universe is  $2^{264}$
  - Fastest supercomputer performs  $2^{144}$  FLOPS

# A First SMT Formulation

- Directly cast “Random Member Algorithm” as SMT
- Must also calculate FLOP count directly in SMT model
  - Psuedo-Boolean constraint
  - Nearly balanced add tree in implementation
  - ITE tree did not work well
  - Did not implement sorter-based techniques
- Size-32 455 FLOP search UNSAT in 30 seconds
- Time-out of 24 hours reached for size-64 1159 FLOP search

# A First SMT Formulation

- Directly cast “Random Member Algorithm” as SMT
- Must also calculate FLOP count directly in SMT model
  - Psuedo-Boolean constraint
  - Nearly balanced add tree in implementation
  - ITE tree did not work well
  - Did not implement sorter-based techniques
- Size-32 455 FLOP search UNSAT in 30 seconds
- Time-out of 24 hours reached for size-64 1159 FLOP search

# A First SMT Formulation

- Directly cast “Random Member Algorithm” as SMT
- Must also calculate FLOP count directly in SMT model
  - Psuedo-Boolean constraint
  - Nearly balanced add tree in implementation
  - ITE tree did not work well
  - Did not implement sorter-based techniques
- Size-32 455 FLOP search UNSAT in 30 seconds
- Time-out of 24 hours reached for size-64 1159 FLOP search

# Sample SMT 1.2 Code

```
1  (benchmark example1
2  :logic QF_BV
3  ...
4  :extrafuns ((Wb_2_1_6 BitVec[4]))
5  :extrafuns ((Wb_2_1_14 BitVec[4]))
6  ...
7  :formula
8  ...
9  (let (?Wb_16_14_0 bv0[4])
10 ...
11 (let (?rWb_2_1_6 (bvadd Wb_2_1_6 bv6[4]))
12 ...
13 (let (?lftp_4_1_12 (bvsb Wb_2_1_6 ?Wb_4_1_12))
14 (let (?lftp_4_3_12 (bvsb ?rWb_2_1_6 ?Wb_4_3_12))
15 ...
16 (flet ($c0_4_1_12 (= (extract[1:0] ?lftp_4_1_12) bv0[2]))
17 (flet ($c4_4_1_12 (and (= (extract[0:0] ?lftp_4_1_12) bv0[1]) (not $c0_4_1_12)))
18 (flet ($c6_4_1_12 (not (= (extract[0:0] ?lftp_4_1_12) bv0[1])))
19 (let (?cost_4_1_12 (ite $c6_4_1_12 bv6[4] (ite $c4_4_1_12 bv4[4] bv0[4])))
20 ...
21 (let (?totalcost (bvadd ?cost_2_2_1 (bvadd ?cost_4_1_12 ?cost_4_3_12)) ...)
22 (flet ($maxcost (bvule ?totalcost bv22[4]))
23 $maxcost
24 )...)
```

# What is Required for Larger Problems?

- A size-32 naïve formulation can be solved easily
  - Interesting results happen at size-256 and larger
- To solve larger problems:
  - Exclude cost symmetries
  - Share twiddle factors
  - Partition
  - Exclude local symmetries
- Will only present partitioning

# What is Required for Larger Problems?

- A size-32 naïve formulation can be solved easily
  - Interesting results happen at size-256 and larger
- To solve larger problems:
  - Exclude cost symmetries
  - Share twiddle factors
  - Partition
  - Exclude local symmetries
- Will only present partitioning

# What is Required for Larger Problems?

- A size-32 naïve formulation can be solved easily
  - Interesting results happen at size-256 and larger
- To solve larger problems:
  - Exclude cost symmetries
  - Share twiddle factors
  - Partition
  - Exclude local symmetries
- Will only present partitioning

# What is Required for Larger Problems?

- A size-32 naïve formulation can be solved easily
  - Interesting results happen at size-256 and larger
- To solve larger problems:
  - Exclude cost symmetries
  - Share twiddle factors
    - Partition
    - Exclude local symmetries
- Will only present partitioning

# What is Required for Larger Problems?

- A size-32 naïve formulation can be solved easily
  - Interesting results happen at size-256 and larger
- To solve larger problems:
  - Exclude cost symmetries
  - Share twiddle factors
  - Partition
  - Exclude local symmetries
- Will only present partitioning

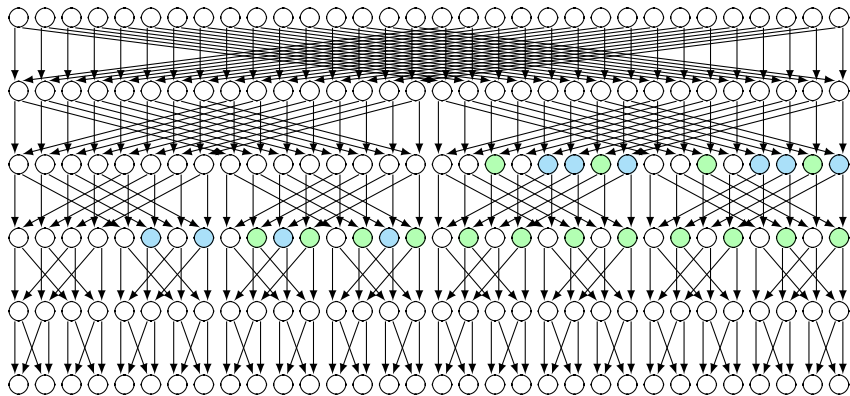
# What is Required for Larger Problems?

- A size-32 naïve formulation can be solved easily
  - Interesting results happen at size-256 and larger
- To solve larger problems:
  - Exclude cost symmetries
  - Share twiddle factors
  - Partition
  - Exclude local symmetries
- Will only present partitioning

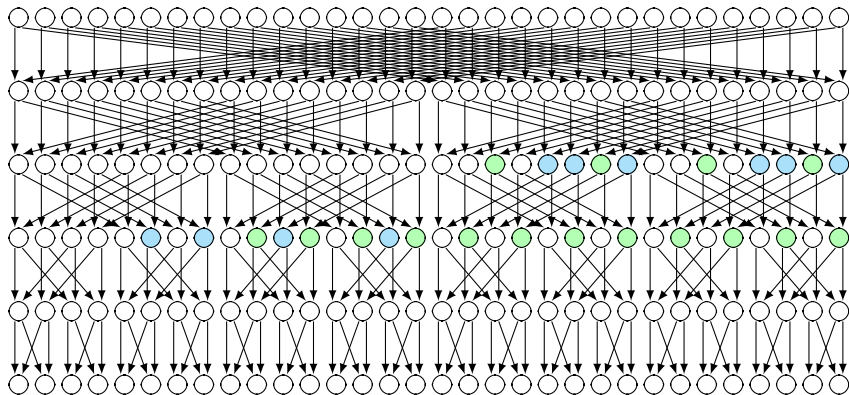
# What is Required for Larger Problems?

- A size-32 naïve formulation can be solved easily
  - Interesting results happen at size-256 and larger
- To solve larger problems:
  - Exclude cost symmetries
  - Share twiddle factors
  - Partition
  - Exclude local symmetries
- Will only present partitioning

# Partitioning

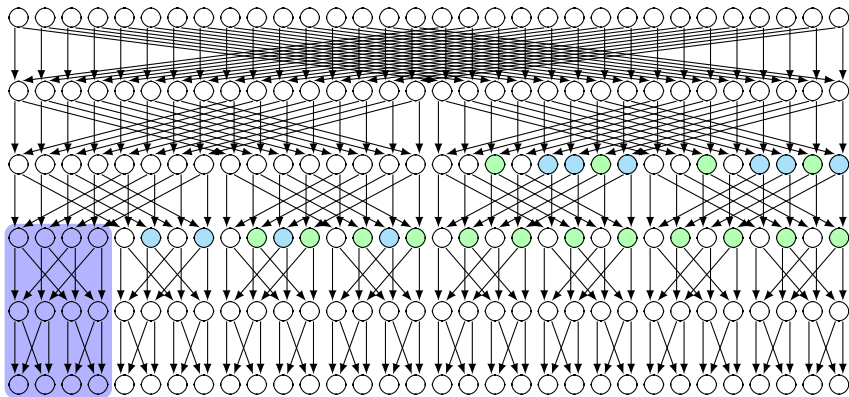


# Partitioning



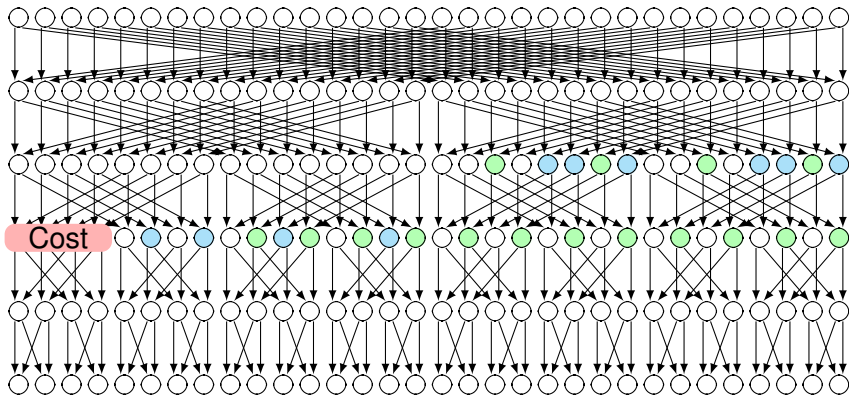
Terminal Weights Known

# Partitioning



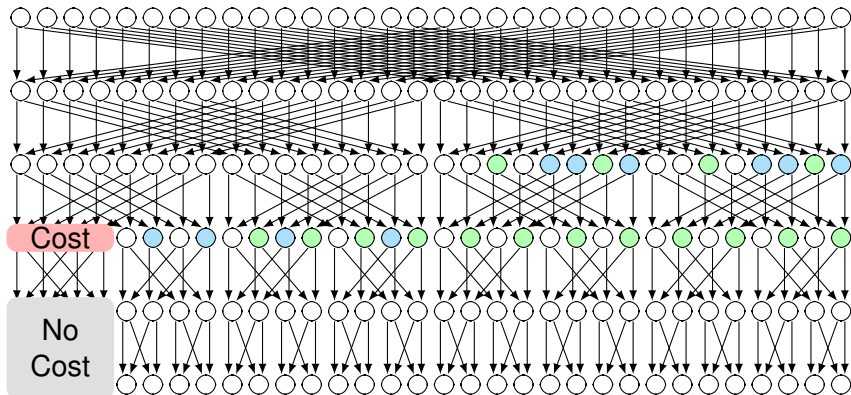
Terminal Weights Known

# Partitioning



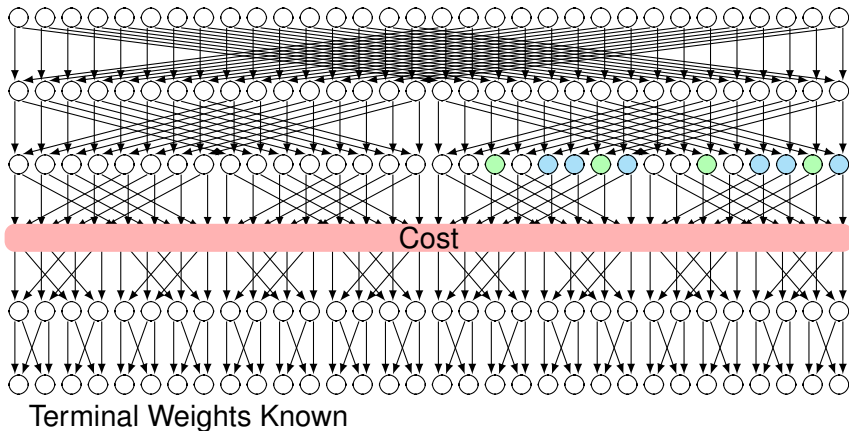
Terminal Weights Known

# Partitioning

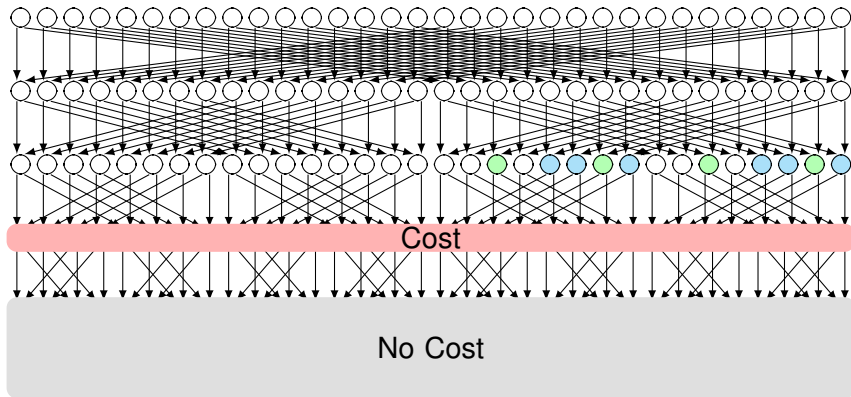


Terminal Weights Known

# Partitioning



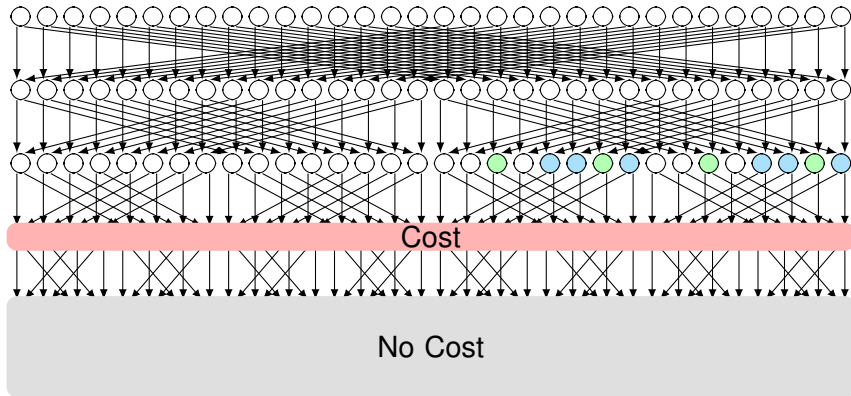
# Partitioning



Terminal Weights Known

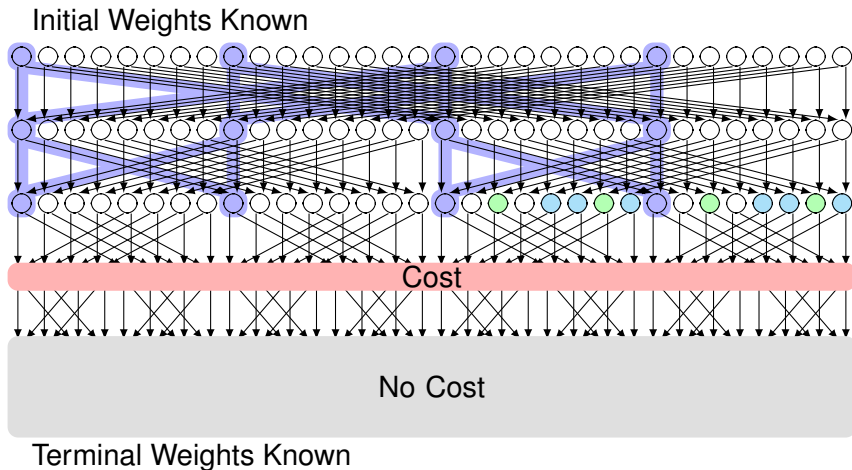
# Partitioning

Initial Weights Known



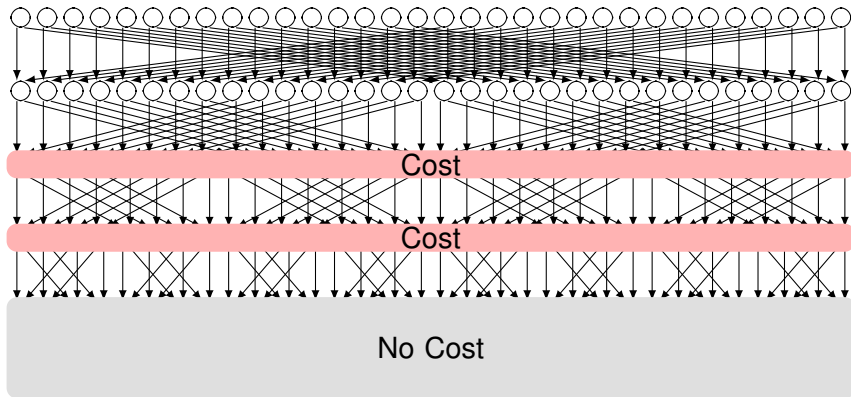
Terminal Weights Known

# Partitioning



# Partitioning

Initial Weights Known



Terminal Weights Known

# Partitioning

Initial Weights Known

No Cost

Cost

Cost

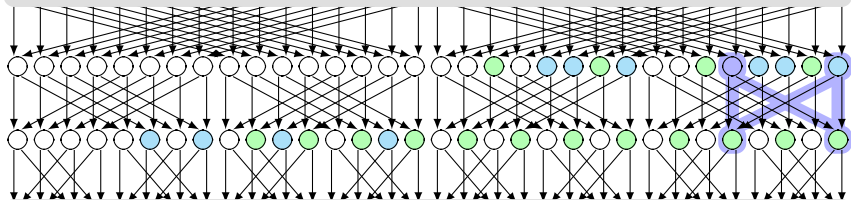
No Cost

Terminal Weights Known

# Partitioning

Initial Weights Known

No Cost



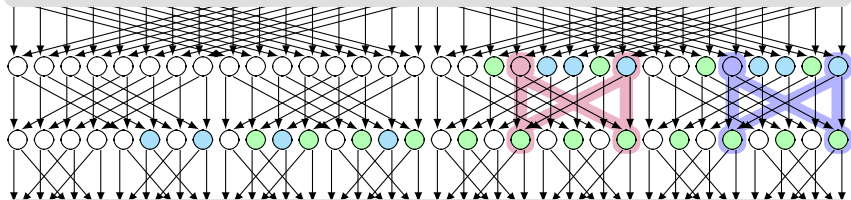
No Cost

Terminal Weights Known

# Partitioning

Initial Weights Known

No Cost



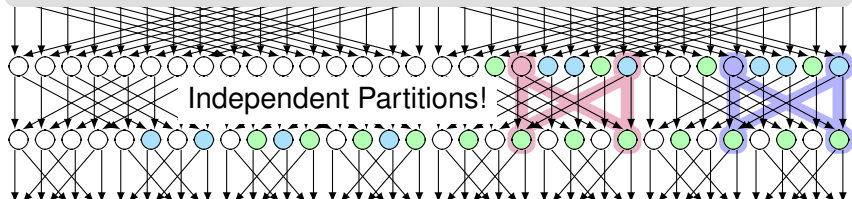
No Cost

Terminal Weights Known

# Partitioning

Initial Weights Known

No Cost



No Cost

Terminal Weights Known

- Hardest partitions for size-256 are 8 size-16
- Solution space for size-16 flowgraph is  $2^{192}$
- Size-16 best FLOP count minus one UNSAT in 5 seconds
- Size-256 6616 FLOP search SAT for all partitions in 8 seconds
- Size-256 6615 FLOP search UNSAT for all partitions in 50 seconds

# Partitioning

- Hardest partitions for size-256 are 8 size-16
- Solution space for size-16 flowgraph is  $2^{192}$
- Size-16 best FLOP count minus one UNSAT in 5 seconds
- Size-256 6616 FLOP search SAT for all partitions in 8 seconds
- Size-256 6615 FLOP search UNSAT for all partitions in 50 seconds

- Hardest partitions for size-256 are 8 size-16
- Solution space for size-16 flowgraph is  $2^{192}$
- Size-16 best FLOP count minus one UNSAT in 5 seconds
- Size-256 6616 FLOP search SAT for all partitions in 8 seconds
- Size-256 6615 FLOP search UNSAT for all partitions in 50 seconds

- Hardest partitions for size-256 are 8 size-16
- Solution space for size-16 flowgraph is  $2^{192}$
- Size-16 best FLOP count minus one UNSAT in 5 seconds
- Size-256 6616 FLOP search SAT for all partitions in 8 seconds
- Size-256 6615 FLOP search UNSAT for all partitions in 50 seconds

- Hardest partitions for size-256 are 8 size-16
- Solution space for size-16 flowgraph is  $2^{192}$
- Size-16 best FLOP count minus one UNSAT in 5 seconds
- Size-256 6616 FLOP search SAT for all partitions in 8 seconds
- Size-256 6615 FLOP search UNSAT for all partitions in 50 seconds

# Outline

- 1 The Fast Fourier Transform
- 2 Generating a Family of FFT Algorithms
- 3 Searching a Family of FFT Algorithms
- 4 Results and Conclusions**

# Results

FFT Size	Tangent	Split-Radix	SMT Search			
	$ \omega_n^*  = *$	$ \omega_n^*  = 1$	$ \omega_n^*  = 1$			
	FLOPs	FLOPs	Satisfiable		Unsatisfiable	
	FLOPs	FLOPs	FLOPs	time(s)	FLOPs	time(s)
32	456	456	456	$1.4 \times 10^{-1}$	455	$1.5 \times 10^{-1}$
64	1152	1160	1160	$3.1 \times 10^{-1}$	1159	$3.3 \times 10^{-1}$
128	2792	2824	2824	$9.3 \times 10^{-1}$	2823	$1.1 \times 10^0$
256	6552	6664	6616	$8.3 \times 10^0$	6615	$5.0 \times 10^1$
512	15048	15368	15128	$3.9 \times 10^4$	15127?	$>1 \times 10^6$

- Found FFT algorithms with restricted set of twiddle factors
- Boolector was our SMT solver of choice
- Found no computation advantage to using SMT over SAT
- Found specification advantage with SMT

# Results

FFT Size	Tangent	Split-Radix	SMT Search			
	$ \omega_n^*  = *$	$ \omega_n^*  = 1$	$ \omega_n^*  = 1$			
	FLOPs	FLOPs	Satisfiable		Unsatisfiable	
	FLOPs	FLOPs	FLOPs	time(s)	FLOPs	time(s)
32	456	456	456	$1.4 \times 10^{-1}$	455	$1.5 \times 10^{-1}$
64	1152	1160	1160	$3.1 \times 10^{-1}$	1159	$3.3 \times 10^{-1}$
128	2792	2824	2824	$9.3 \times 10^{-1}$	2823	$1.1 \times 10^0$
256	6552	6664	6616	$8.3 \times 10^0$	6615	$5.0 \times 10^1$
512	15048	15368	15128	$3.9 \times 10^4$	15127?	$>1 \times 10^6$

- Found FFT algorithms with restricted set of twiddle factors
- Boolector was our SMT solver of choice
- Found no computation advantage to using SMT over SAT
- Found specification advantage with SMT

# Results

FFT Size	Tangent	Split-Radix	SMT Search			
	$ \omega_n^*  = *$	$ \omega_n^*  = 1$	$ \omega_n^*  = 1$			
	FLOPs	FLOPs	Satisfiable		Unsatisfiable	
	FLOPs	FLOPs	FLOPs	time(s)	FLOPs	time(s)
32	456	456	456	$1.4 \times 10^{-1}$	455	$1.5 \times 10^{-1}$
64	1152	1160	1160	$3.1 \times 10^{-1}$	1159	$3.3 \times 10^{-1}$
128	2792	2824	2824	$9.3 \times 10^{-1}$	2823	$1.1 \times 10^0$
256	6552	6664	6616	$8.3 \times 10^0$	6615	$5.0 \times 10^1$
512	15048	15368	15128	$3.9 \times 10^4$	15127?	$>1 \times 10^6$

- Found FFT algorithms with restricted set of twiddle factors
- Boolector was our SMT solver of choice
- Found no computation advantage to using SMT over SAT
- Found specification advantage with SMT

# Results

FFT Size	Tangent	Split-Radix	SMT Search			
	$ \omega_n^*  = *$	$ \omega_n^*  = 1$	$ \omega_n^*  = 1$			
	FLOPs	FLOPs	Satisfiable		Unsatisfiable	
	FLOPs	FLOPs	FLOPs	time(s)	FLOPs	time(s)
32	456	456	456	$1.4 \times 10^{-1}$	455	$1.5 \times 10^{-1}$
64	1152	1160	1160	$3.1 \times 10^{-1}$	1159	$3.3 \times 10^{-1}$
128	2792	2824	2824	$9.3 \times 10^{-1}$	2823	$1.1 \times 10^0$
256	6552	6664	6616	$8.3 \times 10^0$	6615	$5.0 \times 10^1$
512	15048	15368	15128	$3.9 \times 10^4$	15127?	$>1 \times 10^6$

- Found FFT algorithms with restricted set of twiddle factors
- Boolector was our SMT solver of choice
- Found no computation advantage to using SMT over SAT
- Found specification advantage with SMT

# Results

FFT Size	Tangent	Split-Radix	SMT Search			
	$ \omega_n^*  = *$	$ \omega_n^*  = 1$	$ \omega_n^*  = 1$			
	FLOPs	FLOPs	Satisfiable		Unsatisfiable	
	FLOPs	FLOPs	FLOPs	time(s)	FLOPs	time(s)
32	456	456	456	$1.4 \times 10^{-1}$	455	$1.5 \times 10^{-1}$
64	1152	1160	1160	$3.1 \times 10^{-1}$	1159	$3.3 \times 10^{-1}$
128	2792	2824	2824	$9.3 \times 10^{-1}$	2823	$1.1 \times 10^0$
256	6552	6664	6616	$8.3 \times 10^0$	6615	$5.0 \times 10^1$
512	15048	15368	15128	$3.9 \times 10^4$	15127?	$>1 \times 10^6$

- Found FFT algorithms with restricted set of twiddle factors
- Boolector was our SMT solver of choice
- Found no computation advantage to using SMT over SAT
- Found specification advantage with SMT

# Conclusions

- SMT proof of the lowest operation count for classes of Fourier transforms
  - Flowgraph structure of common FFTs
  - All complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Next Steps
  - Impose regularity to develop traditional algorithm
  - Develop other search objectives
  - Relax graph structure
  - Relax constraint that twiddle factors must have modulus one
  - Apply to matrix multiplication
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)

# Conclusions

- SMT proof of the lowest operation count for classes of Fourier transforms
  - Flowgraph structure of common FFTs
  - All complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Next Steps
  - Impose regularity to develop traditional algorithm
  - Develop other search objectives
  - Relax graph structure
  - Relax constraint that twiddle factors must have modulus one
  - Apply to matrix multiplication
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)

# Conclusions

- SMT proof of the lowest operation count for classes of Fourier transforms
  - Flowgraph structure of common FFTs
  - All complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Next Steps
  - Impose regularity to develop traditional algorithm
  - Develop other search objectives
  - Relax graph structure
  - Relax constraint that twiddle factors must have modulus one
  - Apply to matrix multiplication
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)

# Conclusions

- SMT proof of the lowest operation count for classes of Fourier transforms
  - Flowgraph structure of common FFTs
  - All complex multiplication by  $n^{\text{th}}$  roots of unity
- Found new FFTs with lower FLOP count than split-radix
  - Undiscovered in past 40 years despite intense study
- Next Steps
  - Impose regularity to develop traditional algorithm
  - Develop other search objectives
  - Relax graph structure
  - Relax constraint that twiddle factors must have modulus one
  - Apply to matrix multiplication
- Full paper to appear in JSAT
  - Preprint at [www.arXiv.org](http://www.arXiv.org)
  - Preprint, slides and code at [www.softerhardware.com/fft](http://www.softerhardware.com/fft)