Parallel Programming

in C with MPI and OpenMP

Michael J. Quinn

Chapter 15

The Fast Fourier Transform

Outline

- Fourier analysis
- Discrete Fourier transform
- Fast Fourier transform
- Parallel implementation

Discrete Fourier Transform

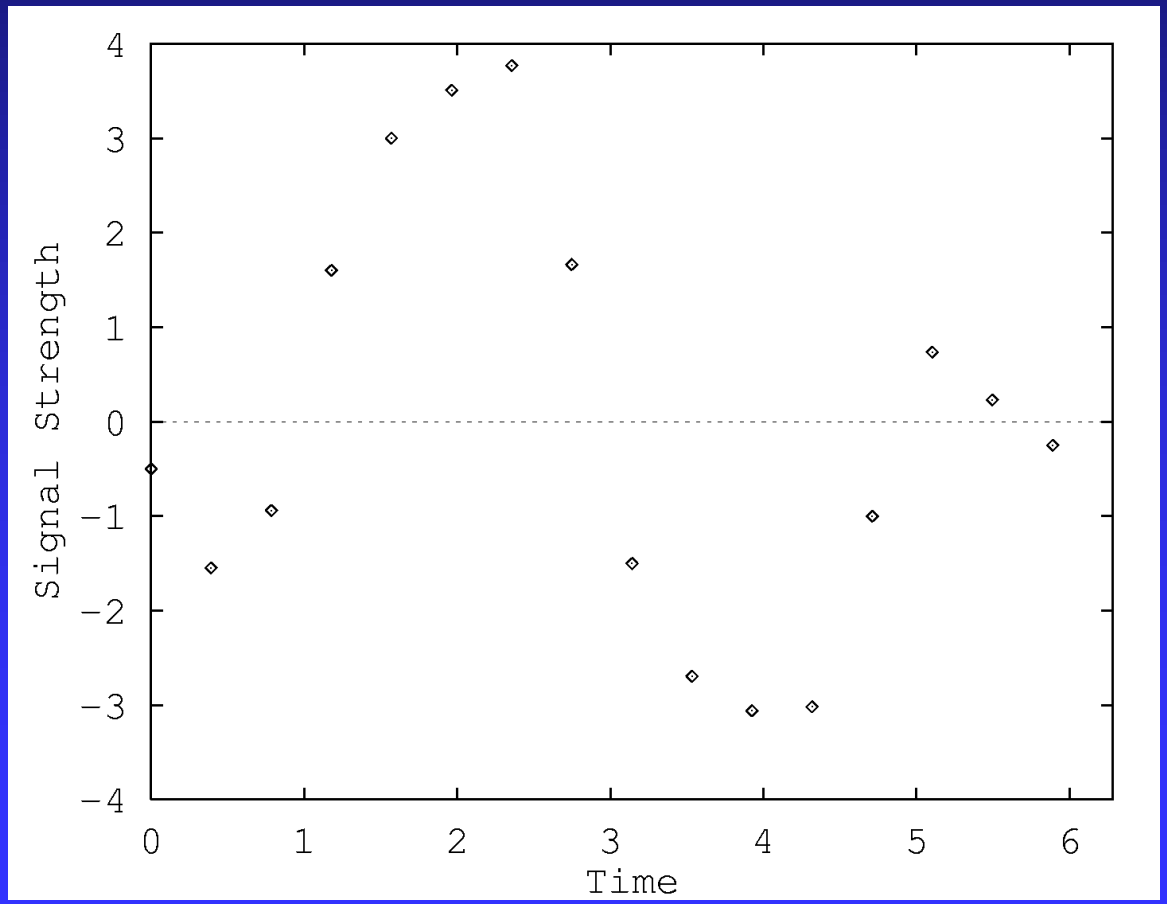
- Many applications in science, engineering
- Examples
  - Voice recognition
  - Image processing
- Straightforward implementation: $\Theta(n^2)$
- Fast Fourier transform: $\Theta(n \log n)$

Fourier Analysis

- Fourier analysis: Represent continuous functions by potentially infinite series of sine and cosine functions
- Discrete Fourier transform: Map a sequence over time to another sequence over frequency
  - Signal strength as a function of time $\Rightarrow$
  - Fourier coefficients as a function of frequency

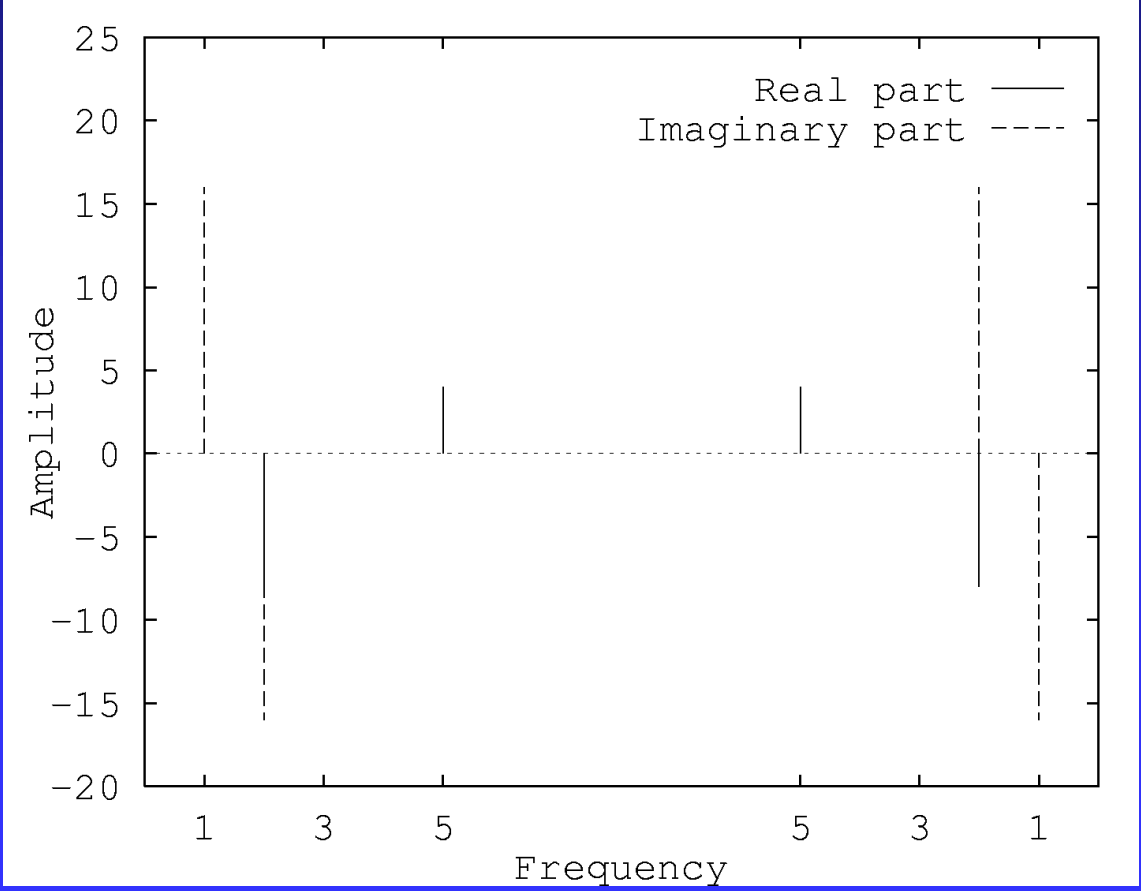
DFT Example (1/4)

16 data points representing signal strength over time



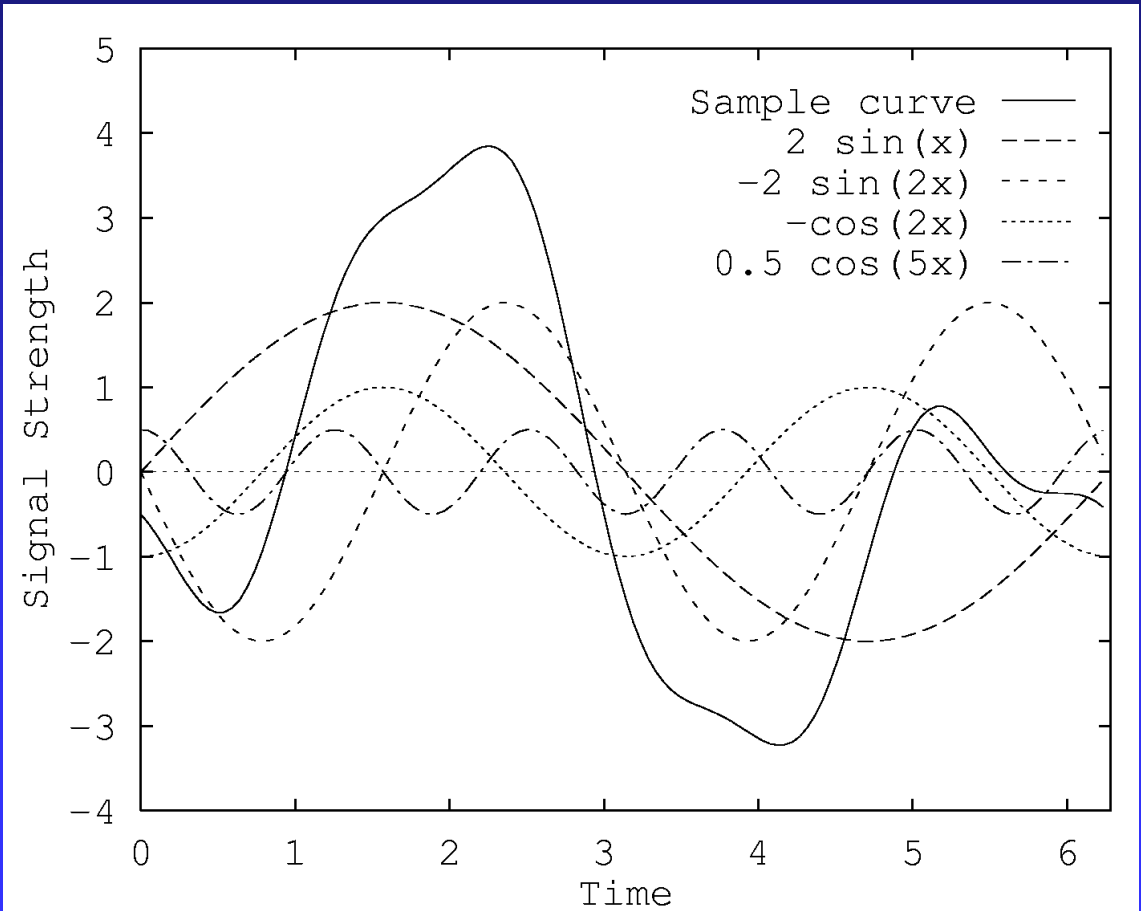
DFT Example (2/4)

DFT yields amplitudes and frequencies of sine/cosine functions



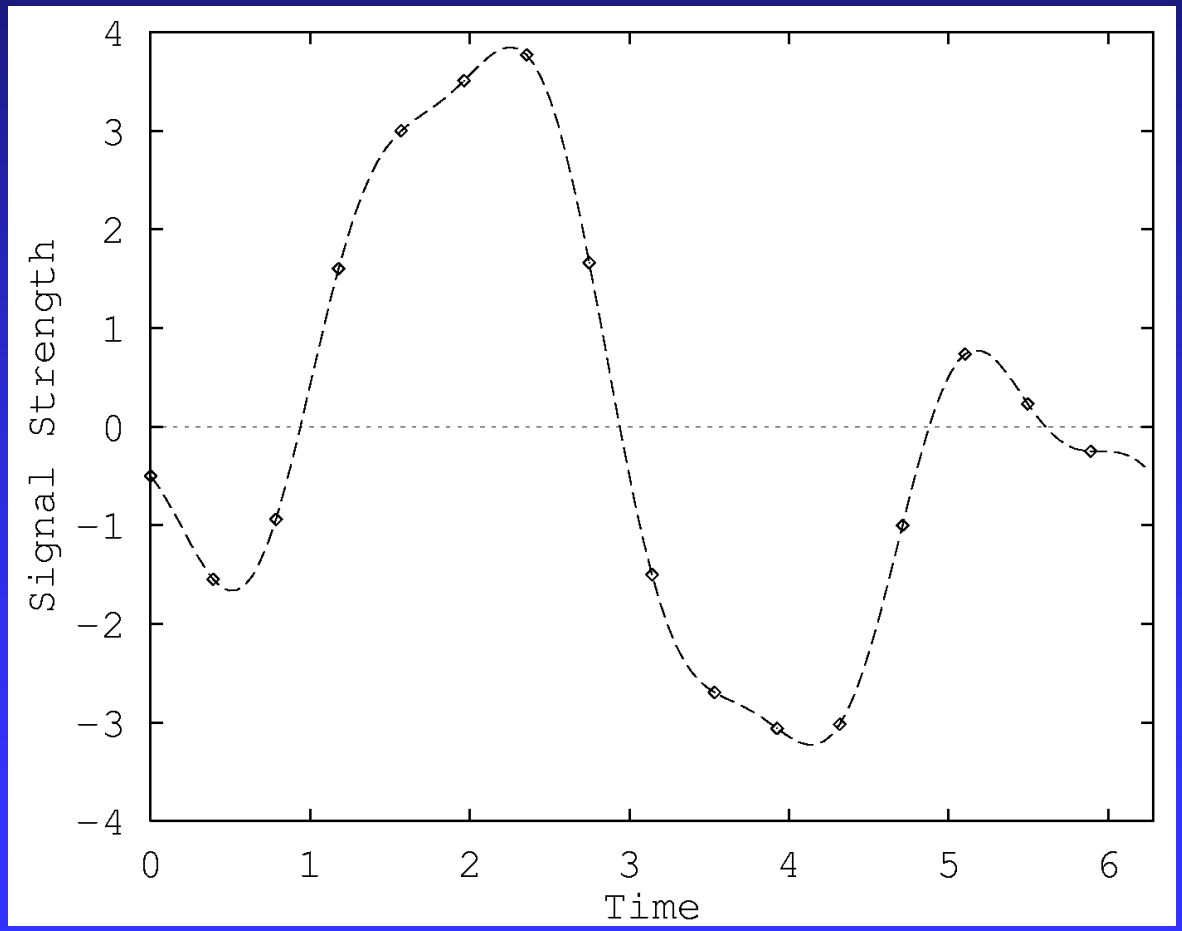
DFT Example (3/4)

Plot of four constituent sine/cosine functions and their sum



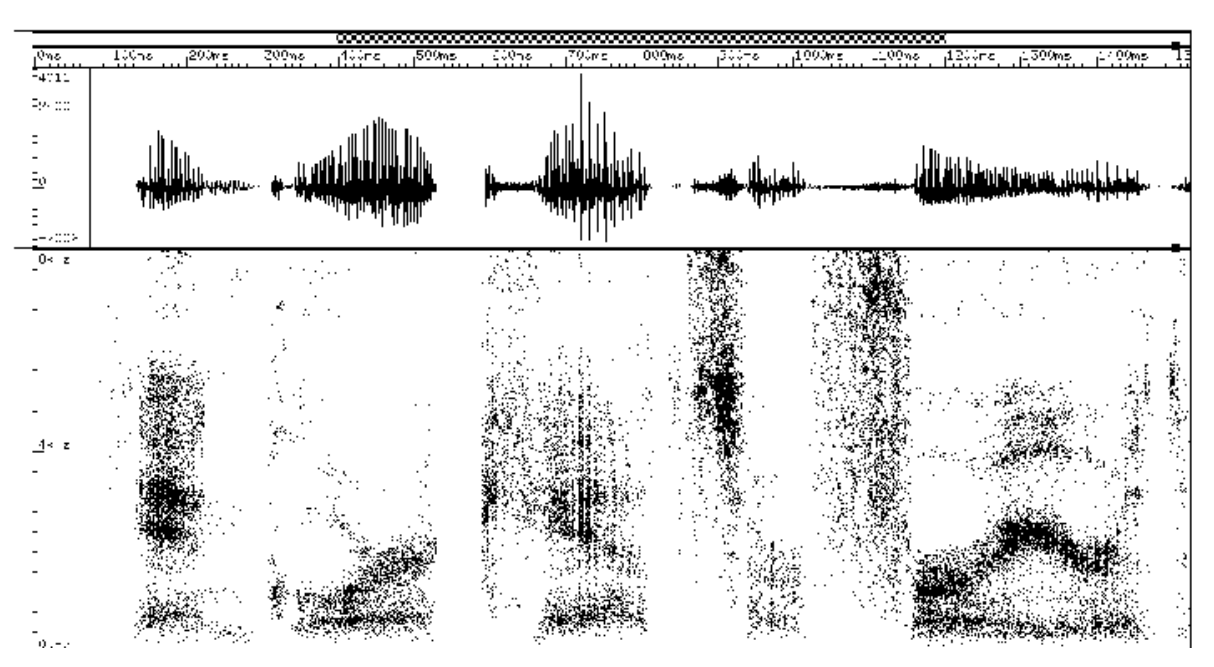
DFT Example (4/4)

Continuous function and original 16 samples.



DFT of Speech Sample

"An gorra cats are furrier..."



Signal

Frequency and amplitude

Figure courtesy Ron Cole and Yeshwant Muthusamy of the Oregon Graduate Institute

Computing DFT

- Matrix-vector product $F_n x$
  - x is input vector (signal samples)
  - $f_{i,j} = \omega_n^{ij}$ for $0 \leq i, j < n$ and ω_n is primitive nth root of unity

Example 1

- Compute DFT of vector (2, 3)
- ω_2, the primitive square root of unity, is -1

$$\begin{pmatrix} \omega_2^{0\times 0} & \omega_2^{0\times 1} \\ \omega_2^{1\times 0} & \omega_2^{1\times 1} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} = \begin{pmatrix} 5 \\ -1 \end{pmatrix}$$

Example 2

- Compute DFT of vector (1, 2, 4, 3)
- The primitive 4th root of unity is i

$$\begin{pmatrix} \omega_4^0 & \omega_4^0 & \omega_4^0 & \omega_4^0 \\ \omega_4^0 & \omega_4^1 & \omega_4^2 & \omega_4^3 \\ \omega_4^0 & \omega_4^2 & \omega_4^4 & \omega_4^6 \\ \omega_4^0 & \omega_4^3 & \omega_4^6 & \omega_4^9 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & i & -1 & -i \\ 1 & -1 & 1 & -1 \\ 1 & -i & -1 & i \end{pmatrix} \begin{pmatrix} 1 \\ 2 \\ 4 \\ 3 \end{pmatrix} = \begin{pmatrix} 10 \\ -3-i \\ 0 \\ -3+i \end{pmatrix}$$

Fast Fourier Transform

- An $\Theta(n \log n)$ algorithm to perform DFT
- Based on divide-and-conquer strategy
- Suppose we want to compute $f(x)$

$$f(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_{n-1} x^{n-1}$$

- We define two new functions, $f^{[0]}$ and $f^{[1]}$

$$f^{[0]} = a_0 + a_2 x + a_4 x^2 + \ldots + a_{n-2} x^{n/2-1}$$

$$f^{[1]} = a_1 + a_3 x + a_5 x^2 + \ldots + a_{n-1} x^{n/2-1}$$

FFT (continued)

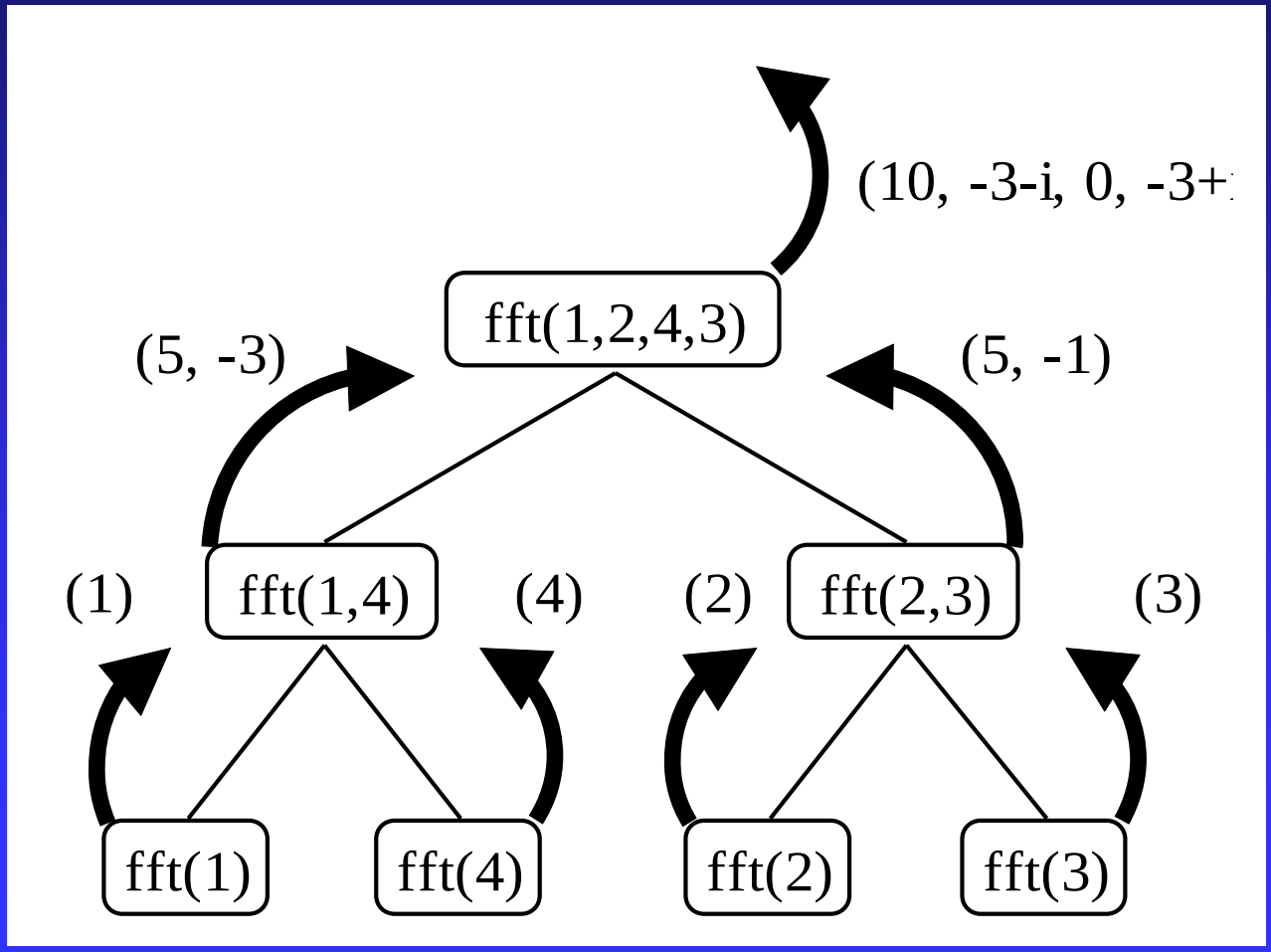
- Note: $f(x) = f^{[0]}(x^2) + x\, f^{[1]}(x^2)$
- Problem of evaluating $f(x)$ at n values of ω reduces to
  - Evaluating $f^{[0]}(x)$ and $f^{[1]}(x)$ at $n/2$ values of ω
  - Performing $f^{[0]}(x^2) + x\, f^{[1]}(x^2)$
- Leads to recursive algorithm with time complexity $\Theta(n \log n)$

Iterative Implementation Preferable

- Well-written iterative version performs fewer index computations than recursive version
- Iterative version evaluates key common sub-expression only once
- Easier to derive parallel FFT algorithm when sequential algorithm in iterative form

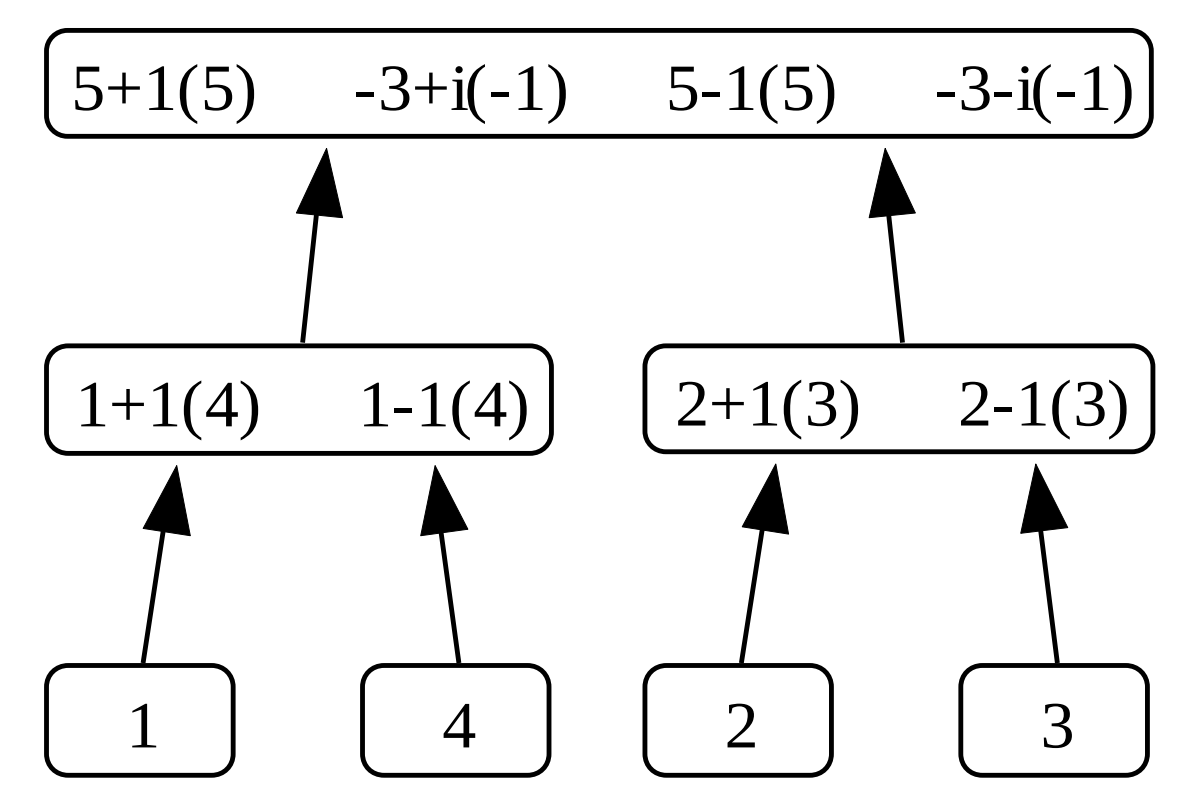
Recursive ⇒ Iterative (1/3)

Recursive implementation of FFT



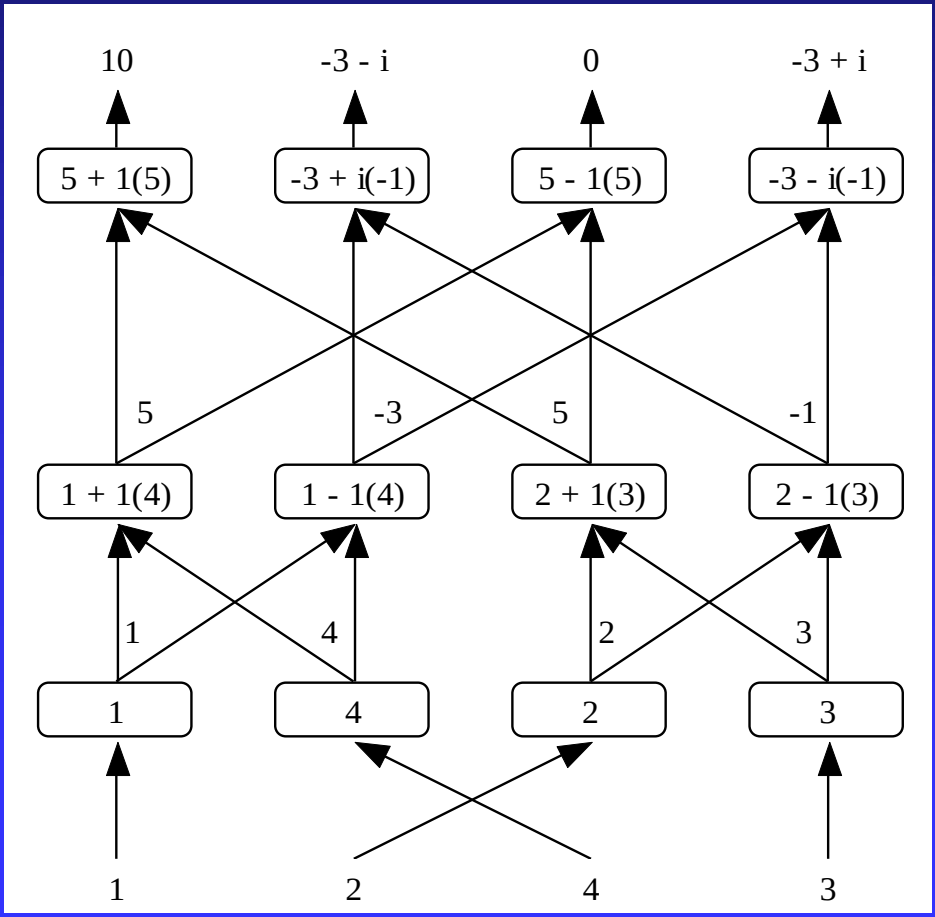
Recursive ⇒ Iterative (2/3)

Determining which computations are performed for each function invocation



Recursive ⇒ Iterative (3/3)

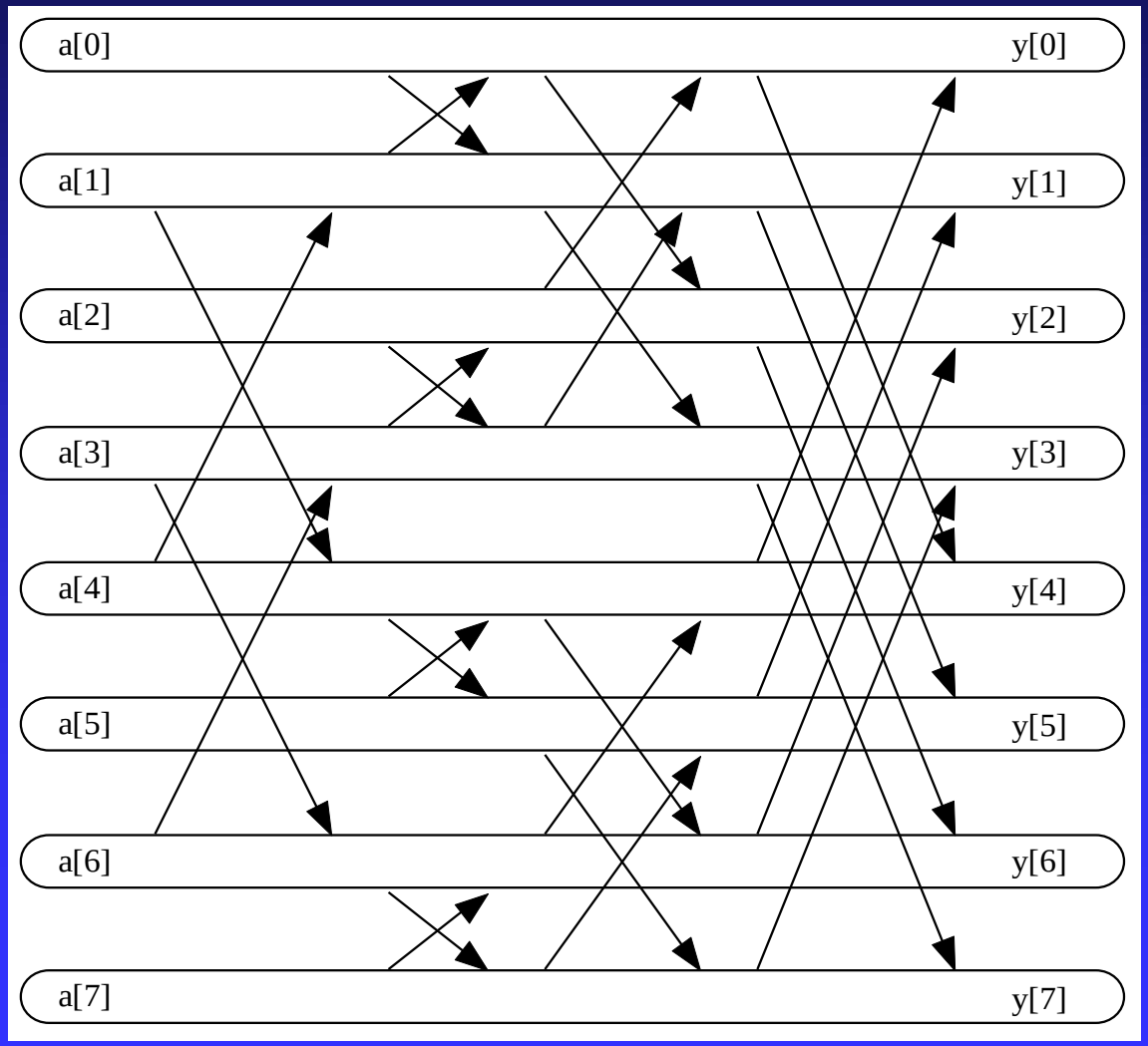
Tracking the flow of data values (input vector at bottom)



Parallel Program Design

- Domain decomposition
  - Associate primitive task with each element of input vector *a* and corresponding element of output vector *y*
- Add channels to handle communications between tasks

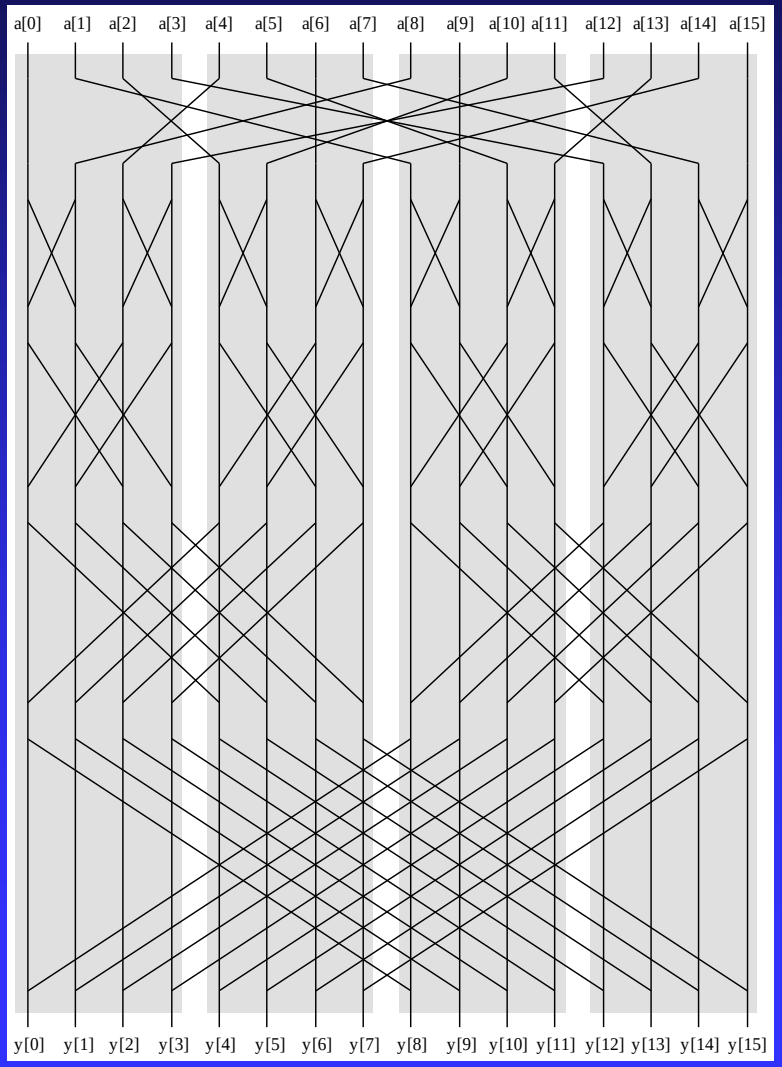
FFT Task/Channel Graph



Agglomeration and Mapping

- Agglomerate primitive tasks associated with contiguous elements of vector
- Map one agglomerated task to each process

After Agglomeration, Mapping



Input

Output

Phases of Parallel FFT Algorithm

- Phase 1: Processes permute *a*'s (all-to-all communication)
- Phase 2:
  - First log *n* – log *p* iterations of FFT
  - No message passing is required
- Phase 3:
  - Final log *p* iterations
  - Processes organized as logical hypercube
  - In each iteration every process swaps values with partner across a hypercube dimension

Complexity Analysis

- Each process performs equal share of computation: $\Theta(n \log n / p)$
- All-to-all communication: $\Theta(n \log p / p)$
- Sub-vector swaps during last log p iterations: $\Theta(n \log p / p)$

Isoefficiency Analysis

- Sequential time complexity: $\Theta(n \log n)$
- Parallel overhead: $\Theta(n \log p)$
- Isoefficiency relation:
  $n \log n \geq C\, n \log p \Rightarrow \log n \geq C \log p \Rightarrow n \geq p^C$

$$M(p^C)/p = p^C/p = p^{C-1}$$

- Scalability depends C, a function of the ratio between computing speed and communication speed.

Summary

- Discrete Fourier transform used in many scientific and engineering applications
- Fast Fourier transform important because it implements DFT in time $\Theta(n \log n)$
- Developed parallel implementation of FFT
- Why isn't scalability better?
  - $\Theta(n \log n)$ sequential algorithm
  - Parallel version requires all-to-all data exchange