Discrete Fourier Transform

- Let $i = \sqrt{-1}$ and index matrices and vectors from 0.
- The DFT of a vector $x$ of dimension $n$ is:

$$y_k = \sum_{j=0}^{n-1} \omega_n^{kj} x_j$$

In matrix form $y = Fx$, where $F$ is the $n \times n$ matrix defined as:

$$F[j,k] = \omega^{(j,k)}_n$$

$\omega_n$ is:

$$\omega_n = e^{-2\pi i / n} = \cos(2\pi/n) - i\sin(2\pi/n)$$

$\omega_n$ is the $n^{th}$ root of unity: $(\omega_n)^n = 1$
Fast Fourier Transform

• Generally attributed to Cooley and Turkey (1965), but FFT algorithm in Gauss notes (1805)

• Several different algorithms available:
  Decimation in time
  Decimation in frequency
  Prime factor algorithm
  Bluestein approach
  
  ..........

• It reduces the computational complexity from $O(n^2)$ to $O(n \log n)$. For $n=10^6$, if FFT=1sec, DFT=24h!!

References:
Van Loan “Computational Frameworks for the FFT”, SIAM
Briggs and Henson “The DFT”, SIAM
Discrete Fourier Transform

\[ F_1 = \begin{bmatrix} 1 \end{bmatrix} \]

\[ F_2 = \begin{bmatrix} 1 & 1 \\ 1 & \omega_2 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \]

\[ F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & \omega^4 & \omega^6 \\ 1 & \omega^3 & \omega^6 & \omega^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & \omega & \omega^2 & \omega^3 \\ 1 & \omega^2 & 1 & \omega \\ 1 & \omega^3 & \omega^2 & \omega \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & -i & -1 & i \\ 1 & -1 & 1 & -1 \\ 1 & i & -1 & -i \end{bmatrix} \]

\[ \omega \] is called twiddle factor
Fast Fourier Transform

How to compute the DFT in $O(n \log n)$ operations?

Establish a connection between $F(n)$ and $F(n/2)$. Repetition of this process is the heart of the radix-2 FFT.

$$\begin{bmatrix}
1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{bmatrix} \quad \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -i & -1 & i \\
1 & -1 & 1 & -1 \\
1 & i & -1 & -i
\end{bmatrix} \quad \begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -1 & -i & i \\
1 & 1 & -1 & -1 \\
1 & -1 & i & -i
\end{bmatrix}$$

Defining

$$\Omega_2 = \begin{bmatrix} 1 & 0 \\ 0 & -i \end{bmatrix} = \text{diag}(1, \omega_4)$$

and using

$$F_2 = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$F_4 \cdot \Pi_4 = \begin{bmatrix} F_2 & \Omega_2 F_2 \\ F_2 & -\Omega_2 F_2 \end{bmatrix}$$
Radix-2 splitting

When \( n=2m \)

\[
\Omega_m = \text{diag}(1, \omega_n, \ldots, \omega_n^{m-1})
\]

\[
F_n \cdot \Pi_n = \begin{bmatrix} F_m & \Omega_m F_m \\ F_m & -\Omega_m F_m \end{bmatrix} = \begin{bmatrix} I_m & \Omega_m \\ I_m & -\Omega_m \end{bmatrix} (I_2 \otimes F_m)
\]

The splitting can be generalized to \( n=pm \) (see Van Loan)

Several different algorithms can be recast in this framework
An iterative method

\[ \text{FFT(0,1,2,3,…,15)} = \text{FFT(xxxx)} \]

\[ \text{FFT(0,2,…,14)} = \text{FFT(xxx0)} \]

\[ \text{FFT(1,3,…,15)} = \text{FFT(xxx1)} \]

\[ \text{FFT(xx00)} \]

\[ \text{FFT(xx10)} \]

\[ \text{FFT(xx11)} \]

\[ \text{FFT(xx01)} \]

\[ \text{FFT(xx11)} \]

\[ \text{FFT(x100)} \]

\[ \text{FFT(x010)} \]

\[ \text{FFT(x110)} \]

\[ \text{FFT(x001)} \]

\[ \text{FFT(x101)} \]

\[ \text{FFT(x011)} \]

\[ \text{FFT(x111)} \]

- The call tree of the d&c FFT algorithm is a complete binary tree of log m levels
- Practical algorithms are iterative, going across each level in the tree starting at the bottom (at the leaves level, we have scalar 1-point DFT \( F_1 x_k = x_k \))
- Algorithm overwrites \( v[i] \) by \((F^*v)[\text{bitrevers}e(i)]\)
Data dependencies in 1D FFT:

- **Butterfly pattern**
Block layout for FFT

• Using a block layout: (m/p) contiguous elements per processor

• No communication in the last log m/p steps

• Each step requires fine-grained communications in first log p steps
Cyclic layout for FFT

- Using a cyclic layout:
  1 element per processor, wrapped

- No communication in the first $\log m/p$ steps

- Communication in last $\log p$ steps
FFT with transpose

• If we start with a cyclic layout for first $\log(p)$ steps, there is no communication

• Then transpose the vector for last $\log(m/p)$ steps

• All communication is in the transpose
FFT with transpose

- Analogous to transposing an array
- View as a 2D array of $n/p$ by $p$

### Block Layout

<table>
<thead>
<tr>
<th>Processor</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>4</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>5</td>
<td>9</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>6</td>
<td>10</td>
<td>14</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>7</td>
<td>11</td>
<td>15</td>
</tr>
</tbody>
</table>

### Cyclic Layout

<table>
<thead>
<tr>
<th>Processor</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>
Higher dimension FFT

- FFTs on 2 or 3 dimensions are defined as 1D FFTs on vectors in all dimensions. E.g., a 2D FFT does 1D FFTs on all rows and then all columns.

- There are 3 obvious possibilities for the 2D FFT:
  1. 2D blocked layout for matrix, using 1D algorithms for each row and column
  2. Block row layout for matrix, using serial 1D FFTs on rows, followed by a transpose, then more serial 1D FFTs
  3. Block row layout for matrix, using serial 1D FFTs on rows, followed by parallel 1D FFTs on columns

- For a 3D FFT the options are similar
  2 phases done with serial FFTs, followed by a transpose for 3rd can overlap communication with 2nd phase in practice
FFT libraries

• Do not write your own fft library, use available vendor or freeware libraries

• FFTW is a fast implementation (both serial and parallel):
  • Fast (similar concept to ATLAS, autotuning)
  • Callable from C and Fortran
  • Parallel transforms use Cilk on SMP and MPI on distributed memory

http://www.fftw.org
Implement the transpose in N-1 steps:

- At the $i^{th}$ step, the processor $j$ exchange data with the processor number $\text{XOR}(j-1,\text{step})$
- $\text{XOR}(k,l)$ is the logical exclusive OR operation applied to the binary representation of the integer $k$ and $l$.

```fortran
np=4
do istep=1,np-1
   do j=1,np
      idest=xor(j-1,istep)
      print *,"Step:“,istep,” Processor:“,j,”Destination“,idest
   end do
end do
```
Direct exchange transpose

Step 1

Step 2

Step 3

Proc 1 | Proc 2 | Proc 3 | Proc 4
-----|-------|-------|-------
 2   |  1   |  4   |  3   
 3   |  4   |  1   |  2   
 4   |  3   |  2   |  1   

Step 1

Step 2

Step 3
Application of FFT

- Numerical integration
- Spectral methods for the solution of PDE
- Fast Poisson solvers
- Image processing
- Digital filtering
Image compression

Image = 200x320 matrix of values
Compress by keeping largest 2.5% of FFT components
Fast Poisson Solver

\[ \nabla^2 \phi = f \]

\[ \frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} = f \]

\[ -k_x^2 \phi + \frac{\partial^2 \tilde{\phi}}{\partial y^2} = \tilde{f} \]

\[ -k_x^2 \bar{\phi} - k_y^2 \bar{\phi} = \bar{f} \implies \bar{\phi} = \frac{\bar{f}}{-\left(k_x^2 + k_y^2\right)} \]
Fast Poisson Solver

- Compute Fourier coefficient $f_{ik}$ of right hand side
  - Apply 2D FFT to values of $f(i,k)$ on grid
- Compute Fourier coefficients $\phi_{ik}$ of solution
  - Divide each transformed $f(i,k)$ by function of wave number $(i,k)$
- Compute solution $\phi(x,y)$ from Fourier coefficients
  - Apply 2D inverse FFT to values of $f(i,k)$

You can apply FFT in one direction and use FD in the other