Lecture 18

Image Processing
Fast Fourier Transform

Today’s lecture
• Image Representation
• Transform based filtration with the FFT
• High performance implementation with the Fast Fourier Transform

Announcements
• Assignment #6
  – Implement sample sort
  – Due in class on Thursday 6/5

Image representation
• A 2D array of picture elements or pixels
Discretization

- Each pixel takes on discrete values
- In our case, 256 gray scale quantization levels
- Color and higher levels of quantization

Image enhancement

- Image data may contain artifacts due to errors or noise in the data collection process
- The display process can also introduce artifacts that obscure the data contained in the image
- We may sometimes eliminate artifacts through image enhancement
- Devise a mapping $F()$ that maps the image $I(x,y)$ onto the enhanced image $I'(x,y)$

Simple image enhancement

- Thresholding
  where $I(x,y) > 0.5 \text{ then } I(x,y) := 1.0 \text{ else } I(x,y) := 0$
- Edge detection
  Takes differences between neighboring pixels

Convolution

- In a convolution, we update each image pixel $I$ as a weighted sum of other pixels within a window $w$
  $$S_{i=0:m-1} S_{j=0:n-1} I[i,j] \times w[m-i,n-j]$$
- What does this remind us of?
- We use the symbol $\circ$ to denote convolution
  $$O[m,n] = I[m,n] \circ w[m,n]$$
A smoothing window

- Consider a 3 x 3 window
- Values normalized so that their sum is 1
- Increased smoothing as we enlarge the window

Image enhancement in the frequency domain

- For certain kinds of enhancements, the convolution window can be easily derived
- We say that such enhancements work in the spatial domain of the problem, since they are expressed in terms of physical locations of pixel values
- But sometimes it is convenient to work in the frequency domain

Backround on Fourier series

- It can be shown that any continuous function \( x(t) \) defined over \( T_1 \leq t \leq T_2 \) can be expressed as the following infinite sum

\[
x(t) = \sum_{k=-\infty}^{\infty} z_k \exp(jk\omega_0 t)
\]

where \( j = \sqrt{-1} \), the \( z_k \) are complex numbers \( x + jy \)

\[
\omega_0 = \frac{2\pi (T_2 - T_1)}{T_2 - T_1}
\]

- Another representation for \( \exp(jk\omega_0 t) \) os \( \cos k\omega_0 t + j\sin k\omega_0 t \)
- We call the infinite sum a Fourier series

Deriving a Fourier Series

- The coefficients \( z_k \) are called the complex amplitude of the \( k \)th component
- They may be obtained by the following formula

\[
z_k = \frac{1}{T} \int_{T}^{T} x(t) \exp(-jk\omega_0 t) dt
\]

- For example, the square wave can be expressed as a sum of odd harmonics:

\[
A_1 \sin(x) + A_3 \sin(3x) + A_5 \sin(5x) + \ldots
\]
Fourier Series approximation of a square wave

\[ \sum_{k \text{ odd}} \frac{\sin(kx)}{k} \]

Discrete Fourier transform

- For a discrete function defined at uniformly space sampling points we have the Discrete Fourier Transform (DFT)
- Given a function \( f[] \) defined at \( N \) points the DFT is \( F[] \)

\[ F[u] = \frac{1}{N} \sum_{x=0}^{N-1} [f(x) e^{-j2\pi ux/N}] \]

\[ f(x) = \sum_{u=0}^{N-1} [F(u) e^{j2\pi ux/N}] \]

Fourier transform

- For a continuous signal defined from \(+ \infty \) to \(-\infty \) there is a continuous Fourier spectrum
- This is the Fourier transform

\[ X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \]

- Whereas the harmonics corresponding to the \( \omega_k \) have frequencies that are discrete multiples of \( \omega_0 \), the harmonics in the Fourier spectrum have all frequencies between \(+ \infty \) and \(-\infty \)
- We can recover the original function with the inverse transform

\[ x(t) = \sum_{k=0}^{N-1} [X[k] e^{j2\pi ku/N}] \]
Where does the 1/N scale factor come from?

- We take the FFT of \( F^*[u] \), we get \( f^*[x]/N \)
  \[
  f^*[x]/N = \frac{1}{N} \sum_{u=0}^{N-1} F^*[u] e^{-j2\pi ux/N}
  \]
  \[
  F[u] = \frac{1}{N} \sum_{x=0}^{N-1} f[x] e^{j2\pi ux/N}
  \]
- Multiplying by \( N \) and taking the complex conjugate gives us \( f[x] \)
  \[
  f[x] = N \text{ conj}(FFT(F^*[u]))
  \]
  \[
  = \sum_{x=0}^{N-1} F[u] e^{j2\pi ux/N}
  \]

Shifting

- The FT is centered about the origin
- But the DFT is centered about \( N/2 \)
- We need to correct with a circular shift operation
- Or, multiply by \((-1)^k\) prior to taking the transform

Using the Discrete Fourier transform

- Consider the spectrum of the square wave

2D Discrete Fourier Transform

- Given a function \( f[] \) defined at \( M \times N \) points the DFT is \( F[ ] \)
  \[
  F(u,v) = (MN)^{-1/2} \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} [f(x,y) e^{j2\pi (ux/M+vy/N)}]
  \]
  \[
  f(x,y) = (MN)^{-1/2} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} [F(u,v) e^{-j2\pi (ux/M+vy/N)}]
  \]
- We apply shift correction in both directions
A 2D Discrete Fourier Transform

1D Spatial Frequencies

2D Spatial Frequencies
Spatial frequency

real space Fourier space

Convolutions with the FFT

• It turns out that we can use the FFT to carry out convolutions
• We express a convolution in terms of an output image $O$, an input image $I$, and a convolution kernel $H$:
  $$O[m,n] = I[m,n] \ast H[m,n]$$
• In the frequency domain we associate italicized identifiers:
  $$O \leftrightarrow \text{FFT}(O) \quad I \leftrightarrow \text{FFT}(I) \quad H \leftrightarrow \text{FFT}(H)$$
• The convolution is expressed as
  $$0 = I \ast H$$
• Thus $O = \text{FFT}^{-1}(I \ast H)$
• I.e. transform to the frequency domain, multiply by the kernel, and take the back transform

Smoothing kernel

• 5x5 neighborhood of ones(5)

More on convolution

• Using transforms, we can gain an intuitive understanding of filter behavior
• When is the convolution method faster?
Implementing the 2D FFT

• The 2D FFT is **separable**
• We may express it in terms of 1D FFTs
• We take **row transforms**
  – Apply the 1D FFT algorithm to each row
• Next take **column transforms**
  – Apply the 1D FFT algorithm to each column

1D Decomposition

• Take row transforms
• All data is on the processor
• Transpose the data using all-to-all communication
• Data is on processor again
• We can simply transform the rows
• Avoid another transpose if the filter is transposed, too

Parallel implementation

• There are two ways to decompose the data
  – 1D: \([\text{Block}, \ast]\) (equivalently \([\ast, \text{Block}]\))
  – 2D: \([\text{Block}, \text{Block}]\)
• In either case, some communication is required

Transposition

• Each processor sends a chunk of data to every other processor

\[
\begin{array}{ccc}
\text{P0} & \text{P1} & \text{P2} \\
\text{P3} & \text{P0} & \text{P1} \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{P0} & \text{P1} & \text{P2} \\
\text{P3} & \text{P0} & \text{P1} \\
\end{array}
\]
What is the running time?

- Transforms cost $20N^2/P \lg N$
  - $N/P$ row and column transforms @ $10N \lg N$
- All-to-all communication
  - All send $P-1$ sets of $(N/P)^2$ complex numbers
  - If $(N/P)^2 \gg n_{1/2}$, we can ignore startup
  - Cost is $(P-1)/P (N^2/P) = (P-1)(N^2/P^2)$

What is the running time?

- Distributed 1D Transforms
  - $2N$ $N$-element transforms @ $10N \lg N / P$
  - These run in parallel in each block column and block row of processors
- Communication
  - Each processor communicates with $\sqrt{P}-1$ other processors
  - This occurs twice, once in each phase
  - Analysis …

2D Blocked layout

- Each processor computes a bundle of $N/P$ 1D transforms distributed over $\sqrt{P}$ processors
- Transforms are taken by block row and then by block column

Computing a distributed 1D FFT

- 1D serial algorithm is an iterative computation with $\lg N$ iterations
- At each iteration $i$, apply a butterfly operation to combine pairs of elements spaced $2^i$ units apart
- “Twiddle” factors
Block Layout of 1D FFT
- Using a block layout (N/P contiguous elements/processor)
- Communication in first $\lg P$ steps
- No communication in last $\lg N/P$ steps

FFT With Transpose
- Start with a cyclic layout
- No communication for first $\lg (P)$ steps
- Transpose
- Last $\lg (N/P)$ steps require no comm
- All communication is in the transpose

Cyclic Layout
- Interlaced decomposition
- No communication in first $\lg (N/P)$ steps
- Communication in last $\lg (P)$ steps

The Transpose
- Analogous to transposing an array
- View as a 2D array of N/P by P
What is the running time of the 2D FFT?

- Computation: ?
  - Simultaneous transforms by block row and block column
- All-to-all communication
  - Restricted to block columns and block rows of processors
  - Since we are computing $N^2 / \sqrt{P}$ transforms at once our messages are likely to be long
  - Cost: ?

A few details

- The data end up in bit-reversed order
  - Thus, element 17 = 010001 → 100010 = 34
- We use libraries to compute the on-processor FFTs
  - MKLIB, etc.
- Other algorithms
  - FFTW “Fastest FFT in the West” www.fftw.org
    http://www.cs.ucsd.edu/users/carter/Papers/pact00.ps