

# How to Write Fast Code

18-645, spring 2008

16<sup>th</sup> Lecture, Mar. 17<sup>th</sup>

**Instructor:** Markus Püschel

**TAs:** Srinivas Chellappa (Vas) and Frédéric de Mesmay (Fred)

# Today

- **Guide to benchmarking and making nice plots**
- **Starting on transforms**
  
- **Rough plan for the next lectures**
  - Next “homework” is working on project
  - Transforms and filters (same as: correlation, interpolation, stencil, polynomial multiplication)
  - Another round of one-on-one meetings
  - Shared memory parallelization, other functionality, advanced topics
  - Discuss project presentations

# Benchmarking

- **Before you start**
  
- **Type 1: Evaluation of the performance of your code**  
(no external competitor)
  
- **Type 2: Comparisons against other code**  
(you want to show your code is better)
  
- **Presenting your results (plots)**
  - In writing
  - Talking
  - Making nice plots

# Before You Start

- **Verify your code!**
  - And that very carefully
  - It is utterly embarrassing to publish or present meaningless results



# Evaluating Your Own Code

## ■ Measure

- Runtime
- Performance (floating point cost by analysis or instrumenting your code)
- Percentage of peak

## ■ Make sure you use your compiler properly

- Optimization flags (e.g., try -O2, -O3, specify platform if possible)
- For compiler vectorization and written vector code see vector lecture

# Comparison Against Other Code

- **Be fair!** 
  - Make sure the comparison is apples to apples
    - Your code computes exactly the same
    - Same interface (e.g., order of input array, data structures)
  - Compile other code properly (maybe specific flags are specified)
  - Use the same timing method
  - Always do a sanity check: compare to published results etc.
  - Apply obvious, easy optimizations also to the competitor code!  
(but say so when you report)
- **Compare against the fastest available code**
- **Report performance if possible**
  - But use same op count for computing (so it's inverse runtime)
  - Shows efficiency of code besides who is better
  - Yields higher is better plots (psychologically more intuitive)

# How to Present Results in Writing

## ■ Specify machine

- processor type, frequency
- relevant caches and their sizes
- operating system

## ■ Specify compilation

- compiler incl. version
- flags

## ■ Explain timing method

## ■ Plot

- **Has to be very readable** (colors, lines, fonts, etc.)
- **Discuss interesting aspects of plots and extract a main message**
- Choose proper type of plot: **message** as visible as possible

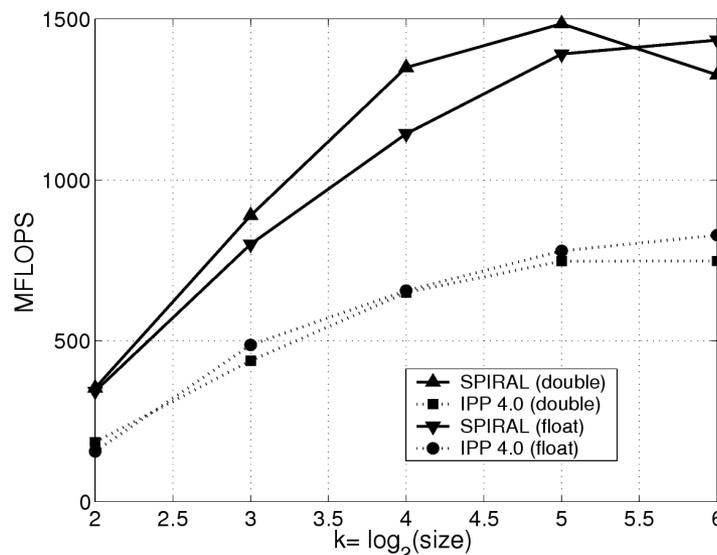
# How to Present Results Talking

- Briefly explain the experiment
- Explain x- and y-axis
- Say, e.g., “higher is better” if appropriate
- Give an example: this line/point means that ....
- Discuss plot and extract a message in the end

Performance of the  
discrete cosine transform:

## Message:

- Spiral code is 2x faster
- reaches up to 50% of peak



**Platform:**

P4 (HT), 3GHz,  
8KB L1, 512KB L2,  
WinXP

**Compiler:**

icc 8.0

**Compiler flags:**

/QxKW /G7 /O3

# Plots: The Basics

## ■ Very readable

- Title, x-label, y-label need to be there
- Fonts large enough
- Enough contrast line to background (e.g., no yellow on white please)
- Enough difference between lines
- Proper number format (where appropriate)
  - **No:** 13.254687; **yes:** 13.25
  - **No:** 2.0345e-05 s; **yes:** 20.3  $\mu$ s
  - **No:** 100000 B; **maybe:** 100,000 B; **yes:** 100 KB

## ■ Clearly shows the message

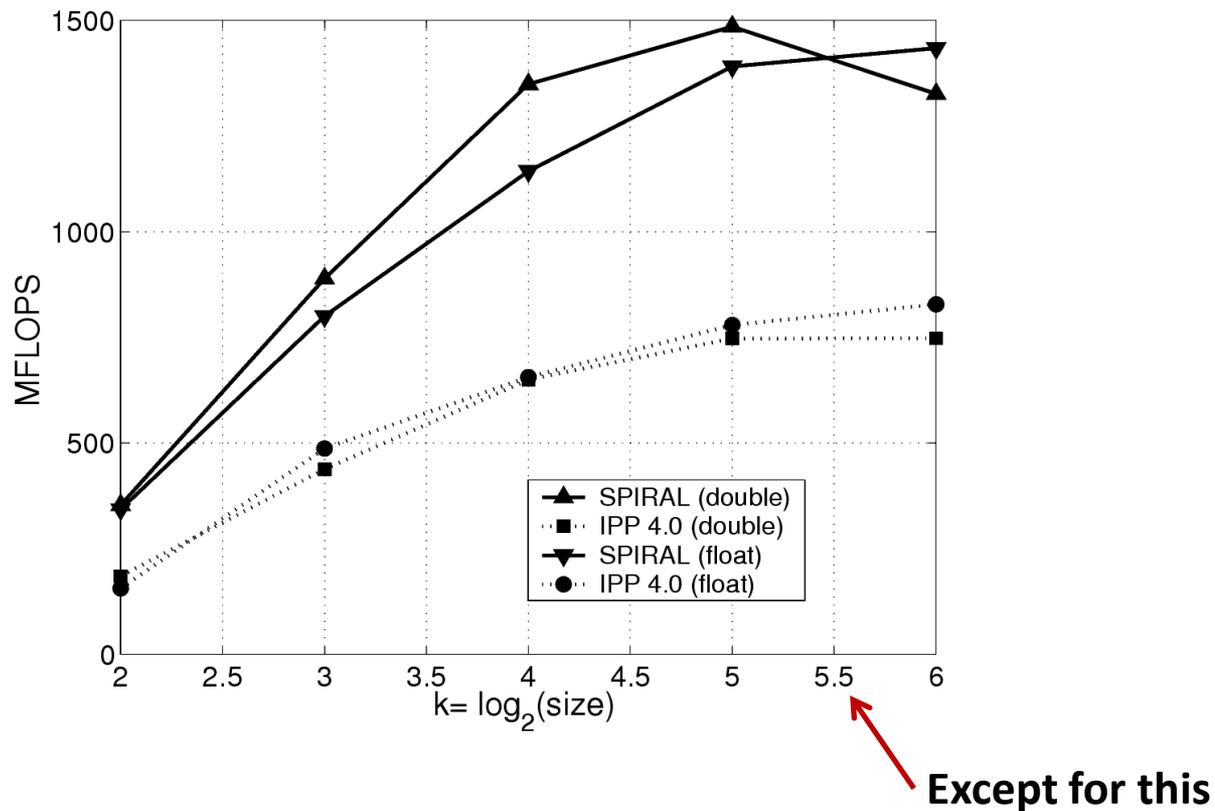
- Proper type of plot (line, bars, properly ordered)
- All the above
- Check it: you know the message; does it jump in your face?

## ■ Beautiful

- Tough, but all the above makes it more beautiful, more later

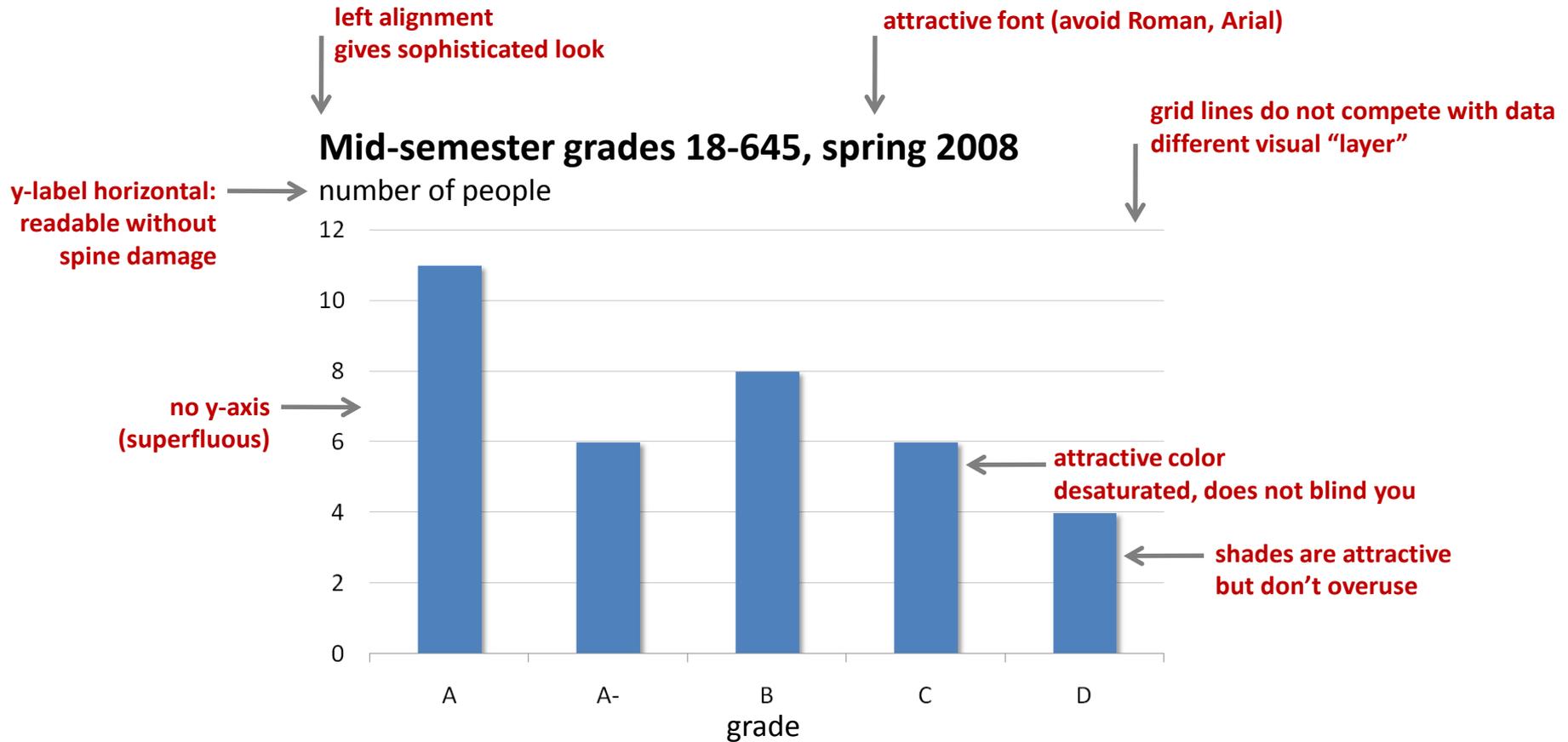
# Example: Mediocre Plot

*Well, 3 years ago I thought it is a good one 😊*

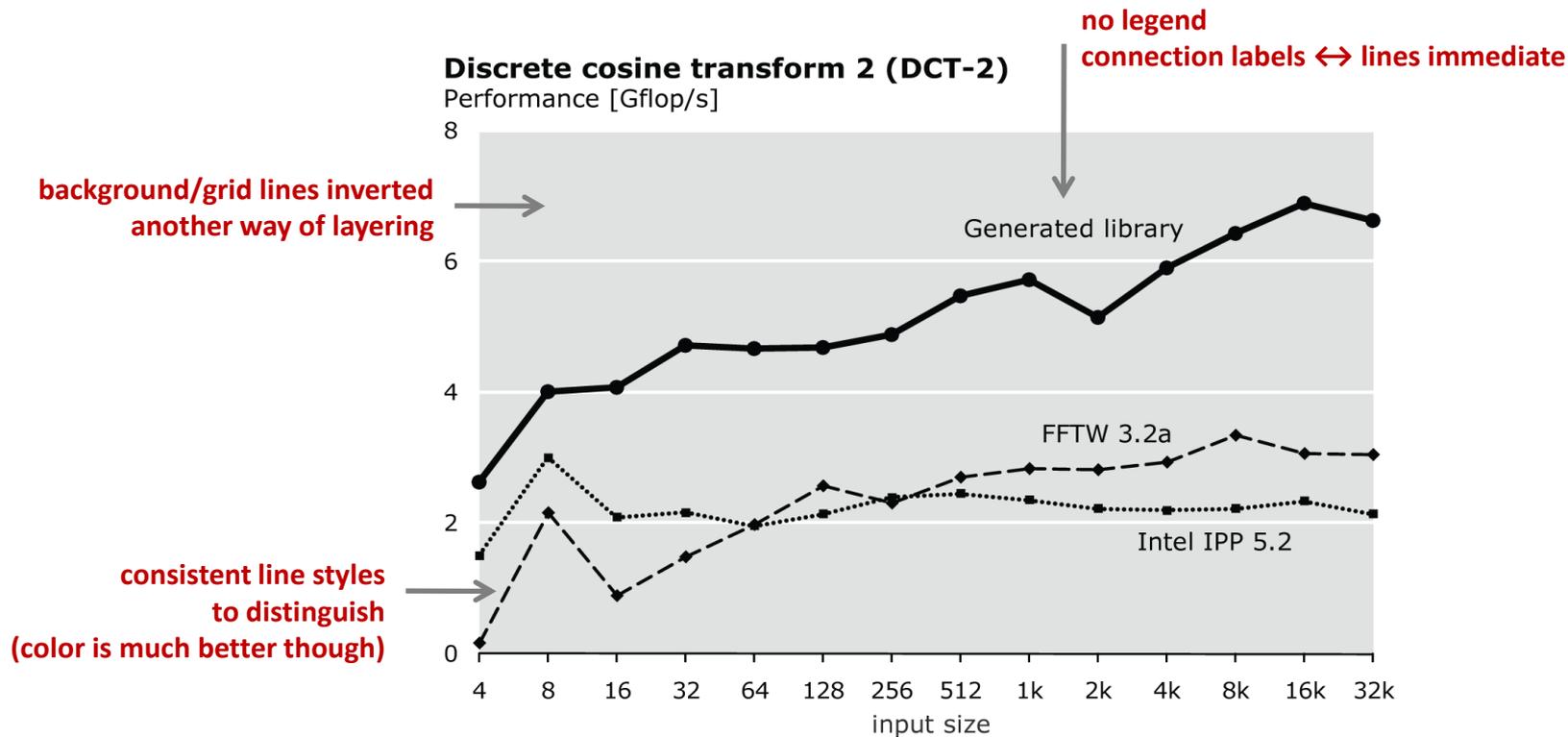


*How do we make it better?*

# Example I: Good Plot

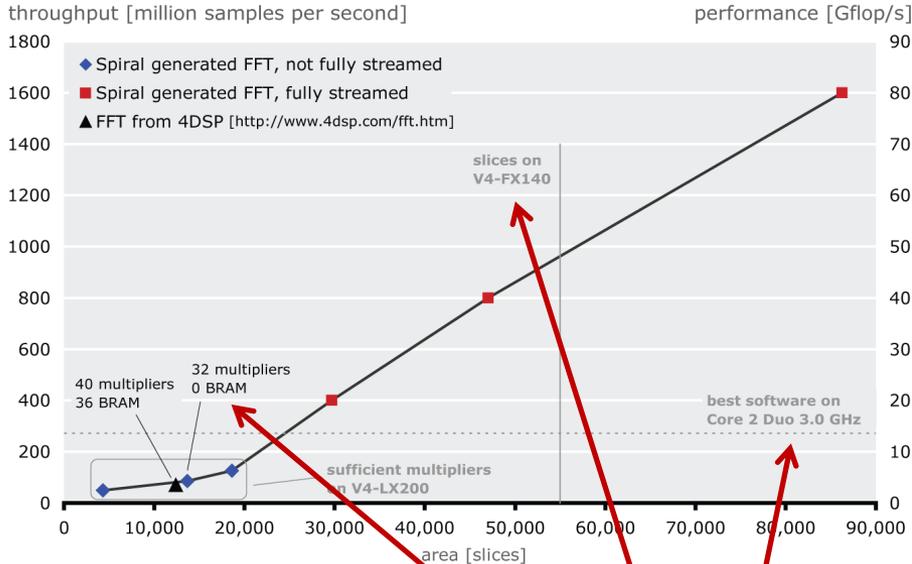


# Example II: Good Plot

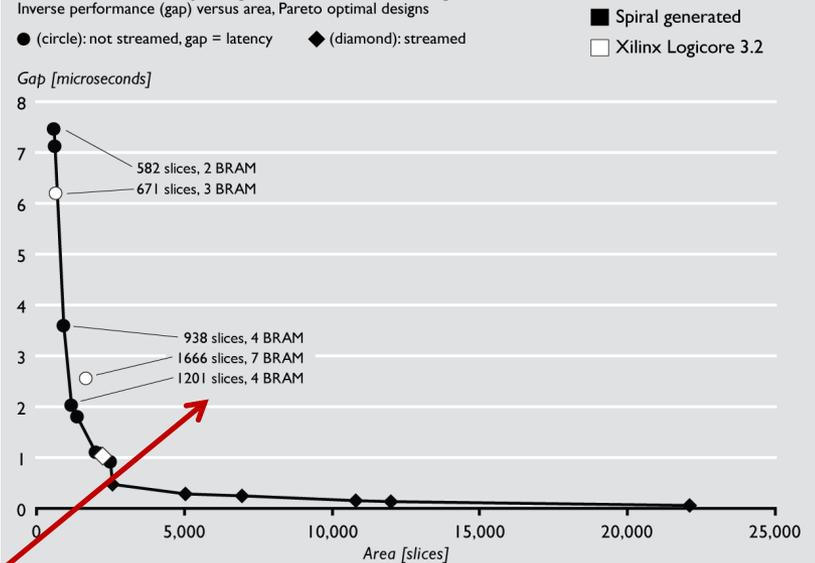


# Example III: Good Plots

**DFT 256 (single precision floating point) on Xilinx Virtex-4 platforms**



**DFT 256 IP Core: Spiral generated vs. Xilinx LogiCore**



- additional information can be packed into a plot
- but use different visual layers
- and make sure it is readable
- good for print publications or web (reader has time to study)

# Good Plots: Advanced Principles

## ■ No Roman or other serif font, avoid Arial if possible

- Calibri (Office 2007)
- Myriad
- Verdana
- Gill Sans

**M** serif

**M** sans serif

## ■ Layering

- Grid lines, axes, etc. should not compete with data lines for attention
- More care necessary when more information is packed into plot
- Good example for layering: maps

## ■ Alignment

- Title, horizontal y-label: left (general design principle)
- x-label, vertical y-label: center

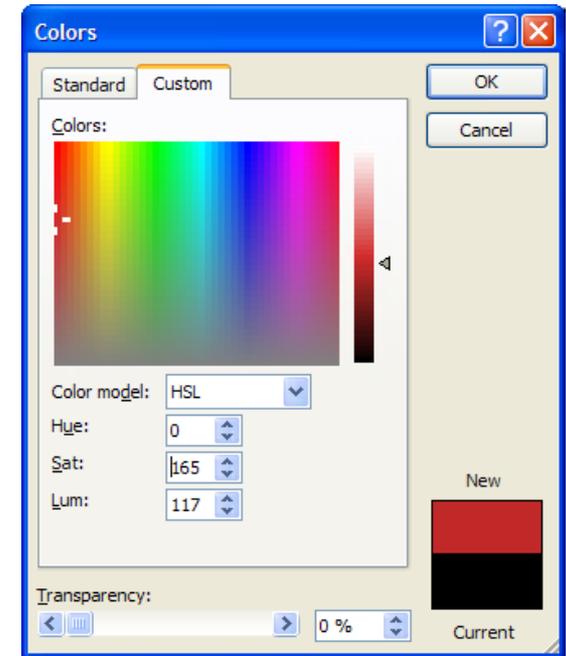
# Good Plots: Advanced Principles

## ■ Colors

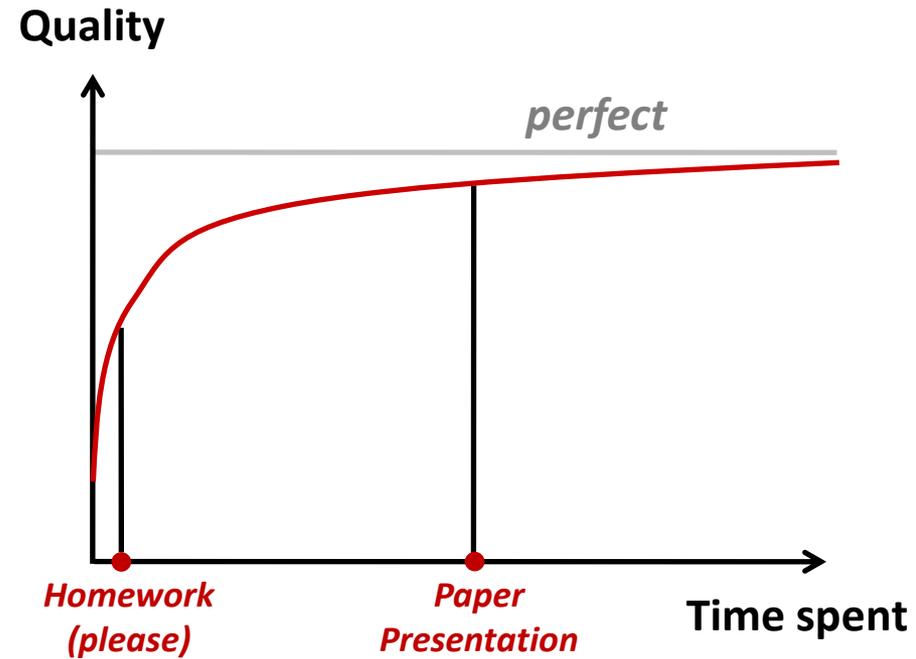
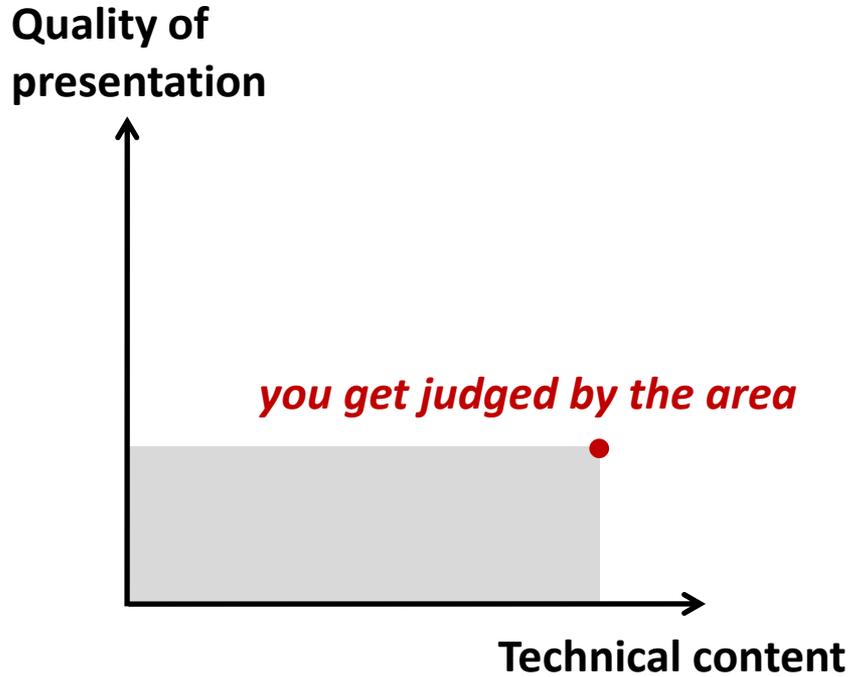
- Use them, except for most print publication
- Don't use **fully saturated colors** →
- Use **somewhat desaturated colors** →

## ■ Get rid of chart junk

- Maximize:  
(ink used on data)/(ink used on the rest)



# Keep in Mind



# Tools and More Information

## ■ Software for making plots

- Matlab (plots by default ugly, but totally configurable, scriptable)
- Excel (2003: by default ugly but a little clicking ....., get Office 2007!)
- Gnuplot (totally configurable, scriptable, only for linux really)
- For highest quality I use: Excel to get it roughly right, then copy-paste into Adobe Illustrator for fine-tuning (everything editable)

## ■ How to learn more

- Look how good magazines do it (Economist, National Geographic, NY Times, ...)
- Edward Tufte:
  - [Visual display of quantitative information](#)
  - [Beautiful evidence](#)
- See also: [Guide to making nice tables](#)

# Transforms

# The Protagonists: Linear Transforms

- **Mathematically: Change of basis**

Two “schools” of representation

$$y = \begin{pmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{pmatrix}$$

$$y_k = \sum_{\ell=0}^{n-1} t_{k,\ell} x_\ell$$

$$x = \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix}$$

**Summation formula**

$$y = Tx \quad T = [t_{k,\ell}]$$

**Matrix-vector product**

- Used in signal processing, scientific computing, ...
- Example: Discrete Fourier transform (DFT)

$$\text{DFT}_n = [e^{-2k\ell\pi i/n}]_{0 \leq k, \ell < n}$$

# Transforms: Examples

- More than 30 transforms in the literature

$$\text{DFT}_n = [e^{-2kl\pi i/n}]_{0 \leq k, l < n}$$

$$\text{RDFT}_n = [r_{kl}]_{0 \leq k, l < n}, \quad r_{kl} = \begin{cases} \cos \frac{2\pi k l}{n}, & k \leq \lfloor \frac{n}{2} \rfloor \\ -\sin \frac{2\pi k l}{n}, & k > \lfloor \frac{n}{2} \rfloor \end{cases}$$

$$\text{DHT} = [\cos(2kl\pi/n) + \sin(2kl\pi/n)]_{0 \leq k, l < n}$$

$$\text{WHT}_n = \begin{bmatrix} \text{WHT}_{n/2} & \text{WHT}_{n/2} \\ \text{WHT}_{n/2} & -\text{WHT}_{n/2} \end{bmatrix}, \quad \text{WHT}_2 = \text{DFT}_2$$

$$\text{IMDCT}_n = [\cos((2k+1)(2l+1+n)\pi/4n)]_{0 \leq k < 2n, 0 \leq l < n}$$

$$\text{DCT-2}_n = [\cos(k(2l+1)\pi/2n)]_{0 \leq k, l < n}$$

$$\text{DCT-3}_n = \text{DCT-2}_n^T \quad (\text{transpose})$$

$$\text{DCT-4}_n = [\cos((2k+1)(2l+1)\pi/4n)]_{0 \leq k, l < n}$$

universal tool

MPEG

JPEG

# Fast Transform Algorithms

- Reduce runtime from  $O(n^2)$  to  $O(n \log(n))$
- > 200 publications on transform algorithms
- Example: Cooley-Tukey fast Fourier transform (FFT)

Again two schools:

$$y_{n_2 j_1 + j_2} = \sum_{k_1=0}^{n_1-1} \left( \omega_n^{j_2 k_1} \right) \left( \sum_{k_2=0}^{n_2-1} x_{n_1 k_2 + k_1} \omega_{n_2}^{j_2 k_2} \right) \omega_{n_1}^{j_1 k_1}$$

**sequence of summations**

$$\text{DFT}_n = L_{n_2}^n (I_{n_1} \otimes \text{DFT}_{n_2}) T_{n_1}^n (\text{DFT}_{n_1} \otimes I_{n_2})$$

**matrix factorization**

# DCT, type III

## II. THE ODD-FACTOR ALGORITHM

The length- $N$  IDCT of input sequence  $X(k)$  is defined by

$$x(n) = \sum_{k=0}^{N-1} X(k) \cos \frac{\pi(2n+1)k}{2N} \quad 0 \leq n \leq N-1 \quad (1)$$

where sequence length  $N$  is an arbitrarily composite integer expressed by

$$N = 2^m \times q = 2^m \times \prod_{i=1}^{\infty} (2i+1)^{r_i} \quad (2)$$

## Algorithm derivation



mutually prime). The IDCT can be decomposed into

$$x\left(qn + \frac{q-1}{2}\right) = \sum_{k=0}^{N-1} X(k) \cos \frac{\pi(2n+1)k}{2(N/q)} \quad (3)$$

$$x(qn+m) = \sum_{k=0}^{N-1} X(k) \cos \frac{\pi[q(2n+1) - (q-1-2m)k]}{2N} \quad (4)$$

$$x(qn+q-m-1) = \sum_{k=0}^{N-1} X(k) \cdot \cos \frac{\pi[q(2n+1) + (q-1-2m)k]}{2N} \quad (5)$$

where for (3)-(5),  $n = 0$  to  $N/q - 1$  and  $m = 0$  to  $(q-3)/2$ . Equation (3) can be rewritten into

$$x\left(qn + \frac{q-1}{2}\right) = \sum_{k=1}^{N/q-1} \left\{ \sum_{i=1}^{(q-1)/2} X\left(\frac{2iN}{q} + k\right) \cdot \cos \frac{\pi(2n+1)(2iN/q+k)}{2(N/q)} + \sum_{k=1}^{(q-1)/2} X\left(\frac{2iN}{q} - k\right) \cdot \cos \frac{\pi(2n+1)(2iN/q-k)}{2(N/q)} \right\}$$

$$\begin{aligned} & + \sum_{k=1}^{N/q-1} X(k) \cos \frac{\pi(2n+1)k}{2(N/q)} \\ & + \sum_{i=0}^{(q-1)/2} X\left(\frac{2iN}{q}\right) \cos \frac{\pi(2n+1)(2iN/q)}{2(N/q)} \\ = & \sum_{k=1}^{N/q-1} \left\{ X(k) + \sum_{i=1}^{(q-1)/2} (-1)^i \left[ X\left(\frac{2iN}{q} + k\right) \right. \right. \\ & \left. \left. + X\left(\frac{2iN}{q} - k\right) \right] \right\} \cos \pi(2n+1)k \\ & + \sum_{i=0}^{(q-1)/2} (-1)^i X\left(\frac{2iN}{q}\right) \\ = & \sum_{k=0}^{N/q-1} U(k) \cos \frac{\pi(2n+1)k}{2(N/q)} \end{aligned}$$

It is noted that input  $x[(2i+1)N/q]$ , defining  $S_i(k) = X(2iN/q+k) + X(2iN/q-k) - X(2iN/q+k)$ , which have

$$U(k) = \begin{cases} X(k) + \sum_{i=1}^{(q-1)/2} (-1)^i S_i(k) \\ \sum_{i=0}^{(q-1)/2} (-1)^i X\left(\frac{2iN}{q}\right) \end{cases}$$

Therefore, (6) can be computed by a length- $N/q$  IDCT. In fact, using (4) and (5), we form two new sequences defined by

$$F(n, m) = \frac{x(qn+m) + x(qn+q-m-1)}{2} = \sum_{k=0}^{N-1} X(k) \cos \frac{\pi(q-1-2m)k}{2N} \cdot \cos \frac{\pi(2n+1)k}{2(N/q)} \quad (8)$$

$$G(n, m) = \frac{x(qn+m) - x(qn+q-m-1)}{2} = \sum_{k=0}^{N-1} X(k) \sin \frac{\pi(q-1-2m)k}{2N} \cdot \sin \frac{\pi(2n+1)k}{2(N/q)}$$

If we define  $\alpha = \pi(q-1-2m)$  for decomposed into

$$F(n, m) = \sum_{k=1}^{N/q-1} \left\{ \sum_{i=1}^{(q-1)/2} X\left(\frac{2iN}{q} + k\right) \cdot \cos \frac{\alpha(2iN/q+k)}{2N} \cos \frac{\pi(2n+1)(2iN/q+k)}{2(N/q)} + \sum_{i=1}^{(q-1)/2} X\left(\frac{2iN}{q} - k\right) \cdot \cos \frac{\alpha(2iN/q-k)}{2N} \cos \frac{\pi(2n+1)(2iN/q-k)}{2(N/q)} \right\} + \sum_{k=1}^{N/q-1} X(k) \cos \frac{\alpha k}{2N} \cos \frac{\pi(2n+1)k}{2(N/q)}$$

$$\begin{aligned} & + \sum_{i=0}^{(q-1)/2} X\left(\frac{2iN}{q}\right) \cos \frac{i\alpha}{q} \\ & \cdot \cos \frac{\pi(2n+1)(2iN/q)}{2(N/q)} \\ = & \sum_{k=1}^{N/q-1} \sum_{i=1}^{(q-1)/2} (-1)^i \left\{ S_i(k) \cos \frac{\pi\alpha_i}{q} \cos \frac{\alpha k}{2N} \right. \\ & \left. - T_i(k) \sin \frac{\alpha_i}{q} \sin \frac{\alpha k}{2N} \right\} \cos \frac{\pi(2n+1)k}{2(N/q)} \end{aligned}$$

**Fast implementation of this algorithm: next homework**

**Just kidding 😊**

$$G(n, m) = (-1)^n \sum_{k=0}^{N/q-1} W(N/q-k, m) \cos \frac{\pi(2n+1)k}{2(N/q)} \quad (14)$$

which is a length- $N/q$  IDCT. The final IDCT outputs can be obtained by (6) and

$$\begin{aligned} x(qn+m) &= F(n, m) + G(n, m) \\ x(qn+q-m-1) &= F(n, m) - G(n, m) \end{aligned} \quad (15)$$

**Typical derivation (> 200 such papers)**

**Recent research at CMU:**

- Simplifies derivation
- Explains origin

$$W(k, m) = \begin{cases} + \left\{ \sum_{i=1}^{(q-1)/2} (-1)^i T_i(k) \sin \frac{\alpha_i}{q} \right\} \cos \frac{\alpha k}{2N} & k = 1, \dots, N/q-1 \\ \sum_{i=1}^{(q-1)/2} (-1)^i X\left[\frac{2i-1}{q}\right] \sin \frac{\alpha(2i-1)}{2q} & k = N/q. \end{cases} \quad (13)$$

sequence length that is a power of odd integers. Therefore, the odd-factor algorithm is general and particularly suited to sequence length containing any possible combination of odd factors. Fig. 1 shows an example for  $N = 27$ . In principle, the proposed odd-factor algorithm is the reverse process of the FDCT algorithm presented in [12].

For a composite sequence length containing both odd and even factors, the radix-2 and the odd-factor algorithms can be jointly used. In principle, the decomposition process can be carried out in many ways. However, a lower count of operations is obtained if the decomposition process starts with the ascending order of the factors of  $N$ . This is because the number of butterfly operations, the growth rate with  $N$  is proportional to the number of arithmetic operations, which shows the growth rate of arithmetic operations is proportional to the number of arithmetic operations. The growth rate of the smallest algorithm before reaching the radix-2 algorithm is the smallest. In summary, the decomposition process of an arbitrarily long sequence is reached; the odd-factor

There are many ways of using the twiddle factors in (11) and (13). One multiplication is needed if the product of two twiddle factors in these equations is precalculated. Hence, the decomposition costs are as follows.

- (1)  $(q-1)(N/q-0.5)$  additions are needed for (7).
- (2)  $(q-1)(2N/q-1)$  additions are needed for  $m = 0$  in (11), and  $(q-1)N/q$  additions are for each value of  $m = 1, \dots, (q-3)/2$ . Furthermore,  $(N-q) + 0.5(q-1)$  multiplications are

# Discrete Fourier Transform (DFT)

- Blackboard