LECTURE 12: FREQUENCY DOMAIN ANALYSIS

- Objectives:
  - Understand the Fourier Transform
  - Introduce the Discrete Cosine Transform
  - Understand frequency domain filterbanks
  - Justify the use of oversampling

This lecture combines material from the course textbook:

> X. Huang, A. Acero, and H.W. Hon, *Spoken Language Processing - A Guide to Theory, Algorithm, and System Development*, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN: 0-13-022616-5, 2001.

and information found in most standard DSP textbooks, including:

> J.G. Proakis and D.G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN: 0-13-373762-4, 1996 (third edition).

Return to Main

Introduction:

01: Organization
(html, pdf)

Speech Signals:

02: Production
(html, pdf)

03: Digital Models
(html, pdf)

04: Perception
(html, pdf)

05: Masking
(html, pdf)

06: Phonetics and Phonology
(html, pdf)

07: Syntax and Semantics
(html, pdf)

Signal Processing:

08: Sampling
(html, pdf)

09: Resampling
(html, pdf)

10: Acoustic Transducers
(html, pdf)

11: Temporal Analysis
(html, pdf)

12: Frequency Domain Analysis
(html, pdf)

13: Cepstral Analysis
(html, pdf)

14: **Exam No. 1**
(html, pdf)

15: Linear Prediction
(html, pdf)

16: LP-Based Representations
(html, pdf)

Parameterization:

17: Differentiation
(html, pdf)

18: Principal Components
(html, pdf)

ECE 8463: FUNDAMENTALS OF SPEECH RECOGNITION

Professor Joseph Picone
Department of Electrical and Computer Engineering
Mississippi State University

email: picone@isip.msstate.edu
phone/fax: 601-325-3149; office: 413 Simrall
URL: http://www.isip.msstate.edu/resources/courses/ece_8463

Modern speech understanding systems merge interdisciplinary technologies from Signal Processing, Pattern Recognition, Natural Language, and Linguistics into a unified statistical framework. These systems, which have applications in a wide range of signal processing problems, represent a revolution in Digital Signal Processing (DSP). Once a field dominated by vector-oriented processors and linear algebra-based mathematics, the current generation of DSP-based systems rely on sophisticated statistical models implemented using a complex software paradigm. Such systems are now capable of understanding continuous speech input for vocabularies of hundreds of thousands of words in operational environments.

In this course, we will explore the core components of modern statistically-based speech recognition systems. We will view speech recognition problem in terms of three tasks: signal modeling, network searching, and language understanding. We will conclude our discussion with an overview of state-of-the-art systems, and a review of available resources to support further research and technology development.

Tar files containing a compilation of all the notes are available. However, these files are large and will require a substantial amount of time to download. A tar file of the html version of the notes is available here. These were generated using wget:

```
wget -np -k -m http://www.isip.msstate.edu/publications/courses/ece_8463/lectures/current
```

A pdf file containing the entire set of lecture notes is available here. These were generated using Adobe Acrobat.

Questions or comments about the material presented here can be directed to help@isip.msstate.edu.

19: Linear Discriminant Analysis (html, pdf)

LECTURE 12: FREQUENCY DOMAIN ANALYSIS

- Objectives:
  - Understand the Fourier Transform
  - Introduce the Discrete Cosine Transform
  - Understand frequency domain filterbanks
  - Justify the use of oversampling

This lecture combines material from the course textbook:

> X. Huang, A. Acero, and H.W. Hon, *Spoken Language Processing - A Guide to Theory, Algorithm, and System Development*, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN: 0-13-022616-5, 2001.

and information found in most standard DSP textbooks, including:

> J.G. Proakis and D.G. Manolakis, *Digital Signal Processing: Principles, Algorithms, and Applications*, Prentice Hall, Upper Saddle River, New Jersey, USA, ISBN: 0-13-373762-4, 1996 (third edition).

Z-TRANSFORM

The z-transform of a discrete-time signal is defined as:

$$X(z) \equiv \sum_{n=-\infty}^{\infty} x(n) z^{-n} \qquad x(n) = \frac{1}{2\pi j} \oint_C X(z) z^{n-1} dz$$

Its properties include:

Property	Time-Domain	z-Domain
Notation	$x(n)$	$X(z)$
	$x_1(n)$	$X_1(z)$
	$x_2(n)$	$X_2(z)$
Linearity and Superposition	$a_1 x_1(n) + a_2 x_2(n)$	$a_1 X_1(z) + a_2 X_2(z)$
Time-Shifting	$x(n-k)$	$z^{-k} X(z)$
Scaling in the z-domain	$a^n x(n)$	$X(a^{-1} z)$
Time reversal	$x(-n)$	$X(z^{-1})$
Conjugation	$x^*(n)$	$X^*(z^*)$
Real part	$Re[x(n)]$	$\frac{1}{2}[X(z) + X^*(z^*)]$
Imag part	$Imag[x(n)]$	$\frac{1}{2j}[X(z) - X^*(z^*)]$

We typically assume the signal is time-limited, and compute the z-transform using a finite sum:

$$X(z) \equiv \sum_{n=0}^{N-1} x(n) z^{-n}$$

Note that the process of truncating a signal using a finite sum is essentially a windowing process, and hence, frequency domain aliasing is introduced.

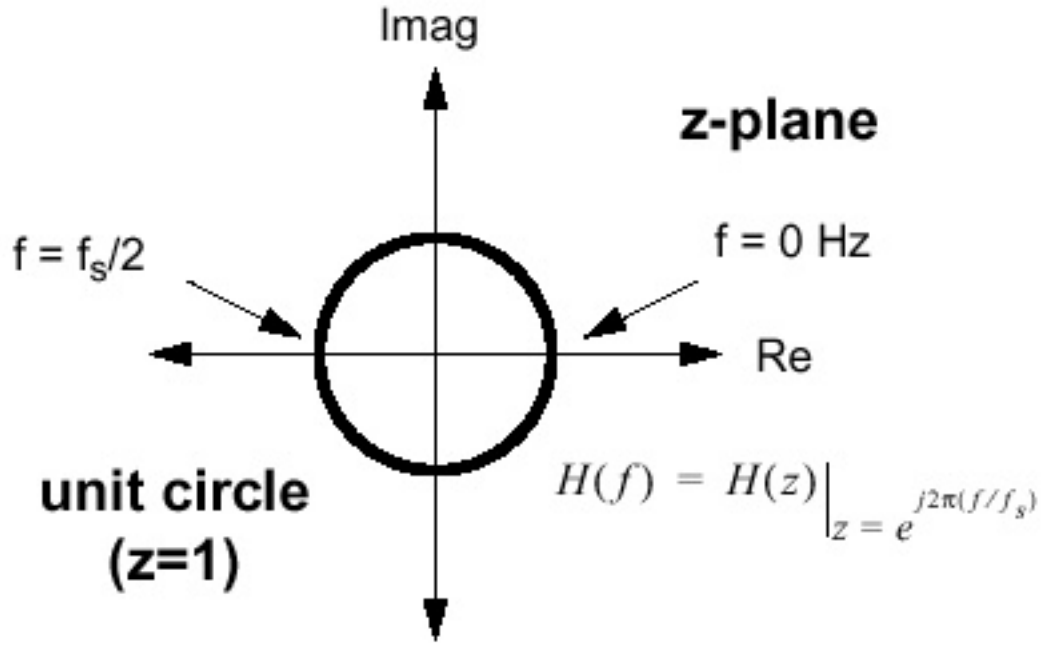
For a more detailed discussion of the z-transform, see DSP notes.

DISCRETE FOURIER TRANSFORM

The Fourier transform of x(n) can be computed from the z-transform as:

$$X(\omega) = X(z)\Big|_{z = e^{j\omega}} = \sum_{n=0}^{N-1} x(n)e^{-j\omega n}$$

The Fourier transform may be viewed as the time-limited (finit) z-transform evaluated around the unit circle:



The Discrete Fourier Transform (DFT) is defined as a sampled version of the (continuous) Fourier transform shown above:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi kn/N}, \qquad k = 0, 1, 2, \ldots, N-1$$

The inverse Discrete Fourier Transform (IDFT) is given by:

$$x(n) = \sum_{n=0}^{N-1} X(k)e^{j2\pi kn/N}, \qquad n = 0, 1, 2, \ldots, N-1$$

The DFT obeys the same properties one would expect for any **linear** transform (linearity, superposition, duality, etc.).

Note that these are not the only transforms used in speech processing (wavelets, Wigner distributions, fractals, etc.).

FAST FOURIER TRANSFORMS

The Fast Fourier Transform (FFT) is nothing more than a computationally efficient version of the Discrete Fourier Transform (DFT):

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi kn/N}, \qquad k = 0, 1, 2, \ldots, N-1$$

or,

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn}, \qquad k = 0, 1, 2, \ldots, N-1$$

where $W_N = e^{-j2\pi/N}$ and $W_N^{kn} = e^{-j2\pi kn/N}$.

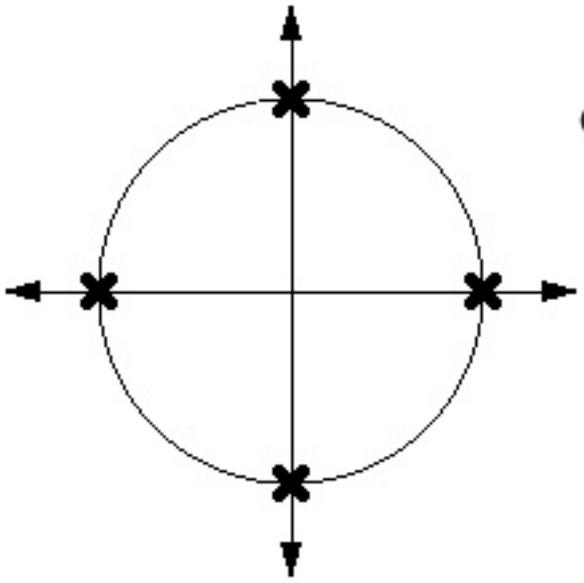
Note that W_N^{kn} are just samples on the unit circle:

For example, $W_4^{(3)(2)} = e^{(-j2\pi/4)(3)(2)} = e^{-j3\pi} = e^{-j\pi} = -1$.

N = 4

k = 0, 1, 2, 3

n = 0, 1, 2, 3



Only four unique values!

We note two important symmetry properties of W_N^{kn}:

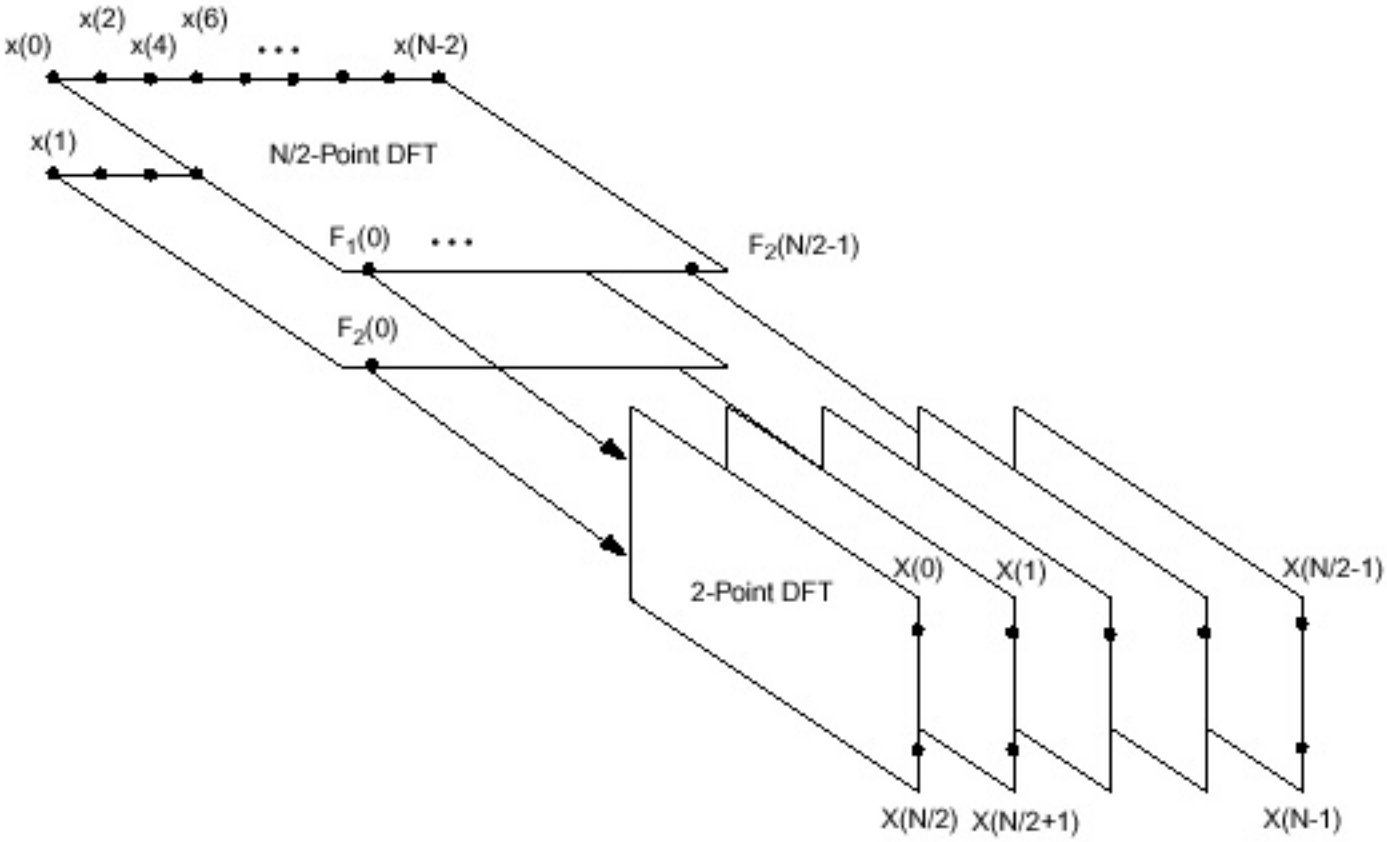
$W_N^{k+N/2} = -W_N^k$ (symmetry about the imaginary axis)

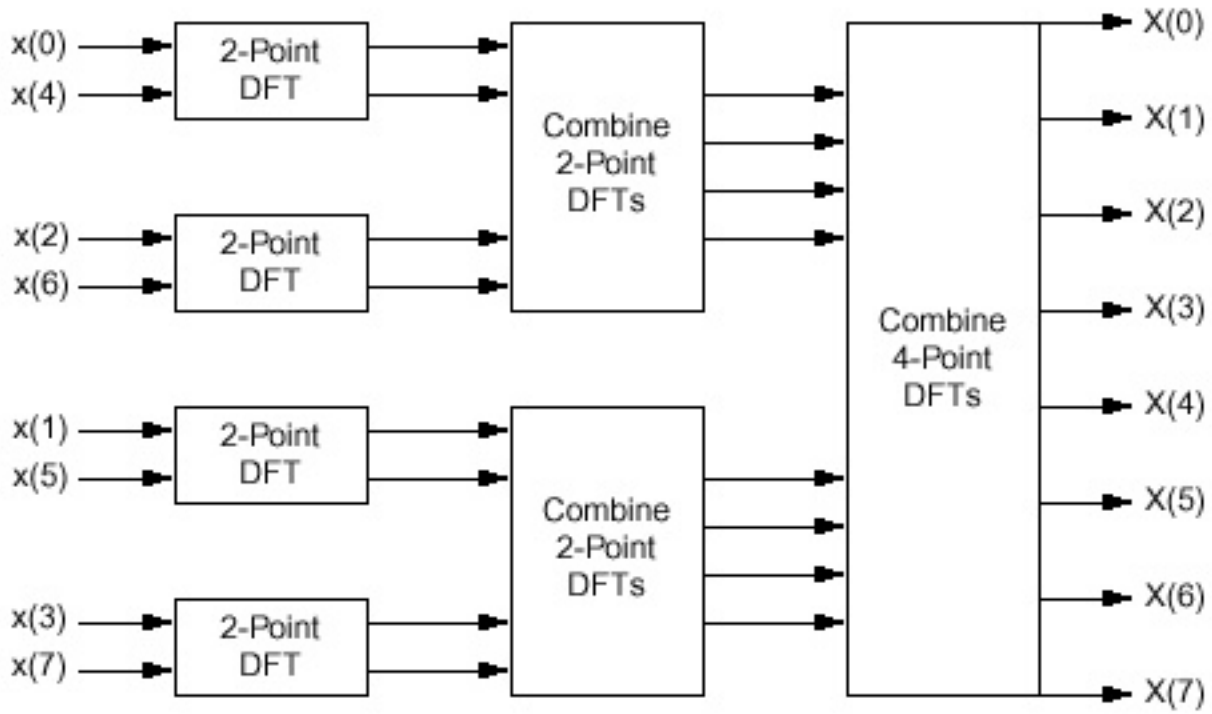
$W_N^{k+N} = W_N^k$ (periodicity)

This symmetry allows the number of computations for a DFT to be reduced significantly.

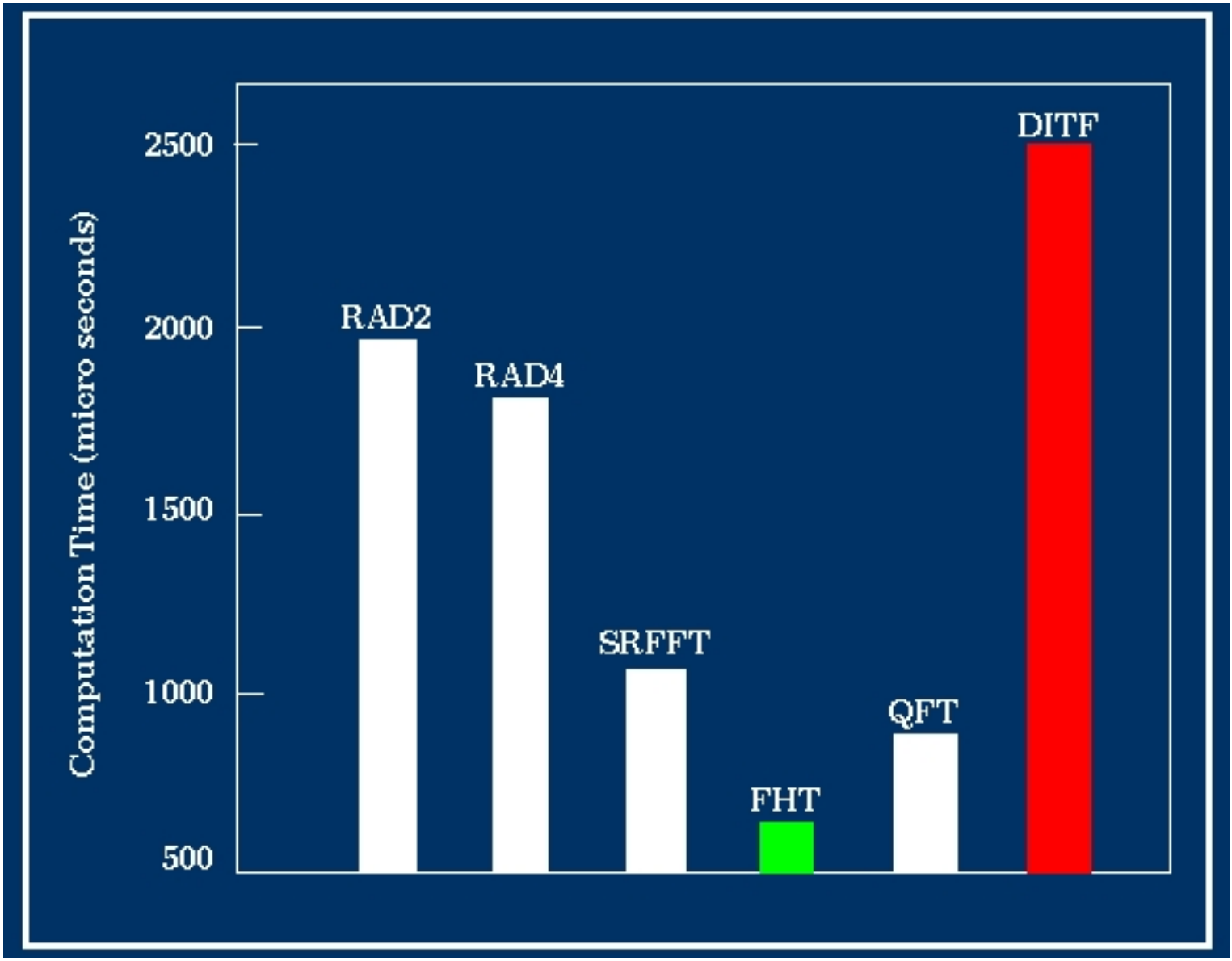
The most common approach to achieving this efficiency is to use a decimation-in-time strategy that benefits from the non-

linear computational complexity of the transform:





The Radix-2 and Radix-4 algorithms are extremely popular due to their computational efficiency and relatively simple implementations:



A definitive work on the computational complexity of FFT algorithms, including benchmarks and software, can be found at parallel FFTs.

DISCRETE COSINE TRANSFORMS

The Discrete Cosine Transform (DCT) is simply a computationally efficient version of the DFT for signals that are real and even (x(n) = x(N-n)): reduces to:

$$X(k) = \sum_{n=0}^{N-1} x(n)\cos(2\pi kn/N) \qquad 0 \le k < N$$

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)\cos(2\pi kn/N) \qquad 0 \le n < N$$

The DCT-II, which is one of two common implementations of the DCT used in speech processing, is defined as:

$$C(k) = \sum_{n=0}^{N-1} x(n)\cos(\pi k(n+1/2)/N) \qquad 0 \le k < N$$

$$x(n) = \frac{1}{N}\left\{ C(0) + 2\sum_{k=1}^{N-1} C(k)\cos(\pi k(n+1/2)/N) \right\} \qquad 0 \le n < N$$

The DFT and DCT are related by the following equations:

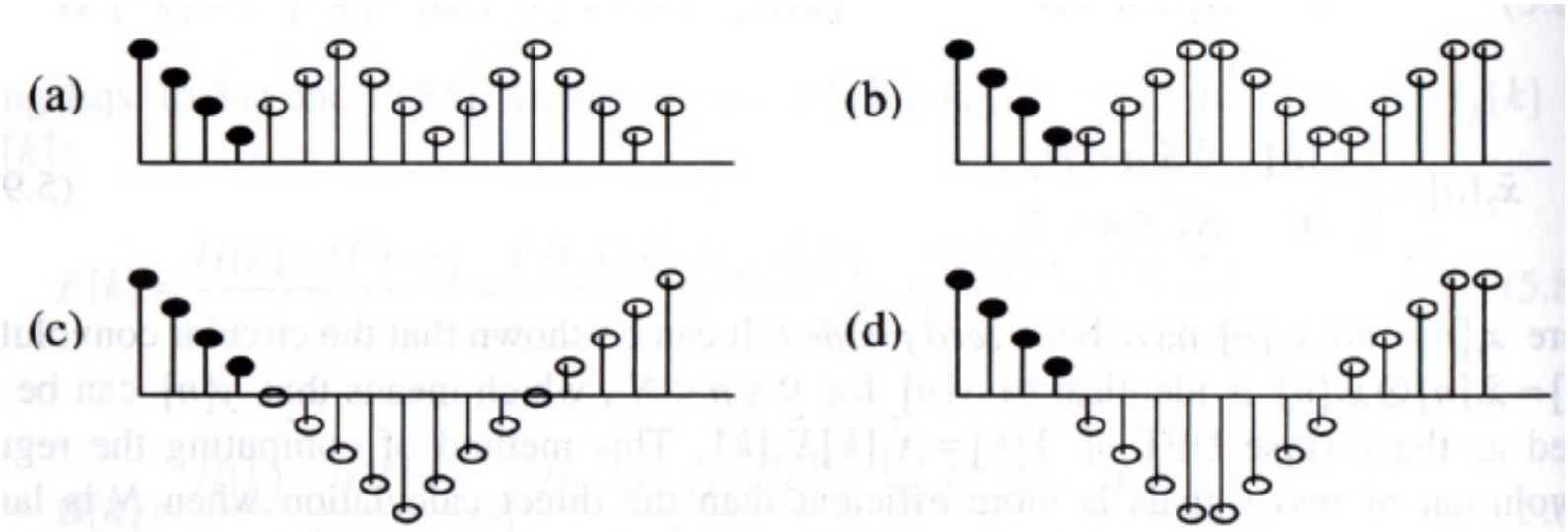
$$X(k) = 2e^{j\pi k/2N} C(k) \qquad 0 \le k < N$$

$$X(2N-k) = 2e^{-j\pi k/2N} C(k) \qquad 0 \le k < N$$

The forward DCT is used in a speech recognition front end to convert samples of the log magnitude spectrum to cepstral coefficients.

PERIODIC EXTENSION AND THE DCT

There are four common ways to extend a real signal to make it both periodic and have even symmetry:



which correspond to DCT types I, II, III, and IV respectively.

Types II and III are most commonly used in speech processing because they tend to offer the most *energy compaction* resulting in representing the signal in the fewest number of coefficients.

NONLINEAR FREQUENCY WARPING:
BARK AND MEL SCALES

- **Critical Bandwidths**: correspond to approximately 1.5 mm spacings along the basilar membrane, suggesting a set of 24 bandpass filters.
- **Critical Band**: can be related to a bandpass filter whose frequency response corresponds to the tuning curves of an auditory neurons. A frequency range over which two sounds will sound like they are fusing into one.
- **Bark Scale**:

$$Bark = 13\,\mathrm{atan}\left(\frac{0.76f}{1000}\right) + 3.5\,\mathrm{atan}\left(\frac{f^2}{(7500)^2}\right)$$

- **Mel Scale**:

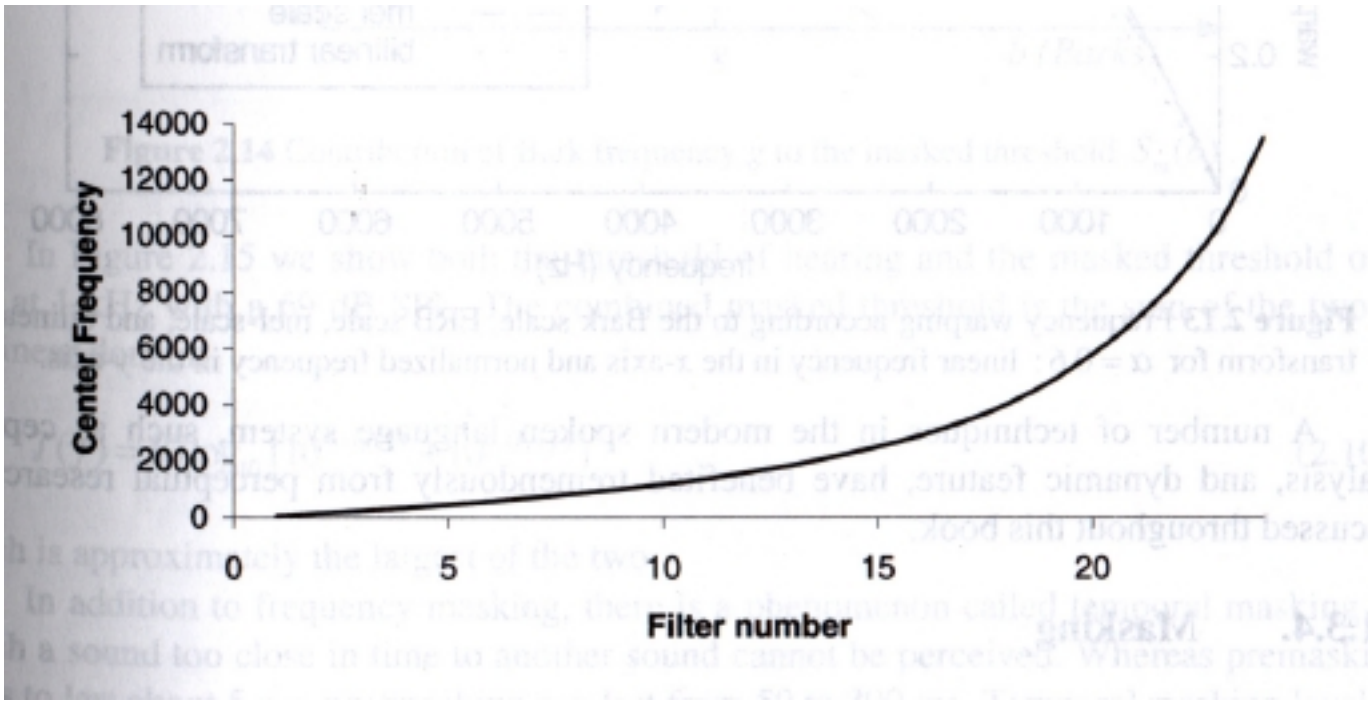
$$mel\ frequency = 2595\ \log10\,(1 + f/700.0)$$

- **Comparison**: filter bank implementations for a typical speech recognizer.

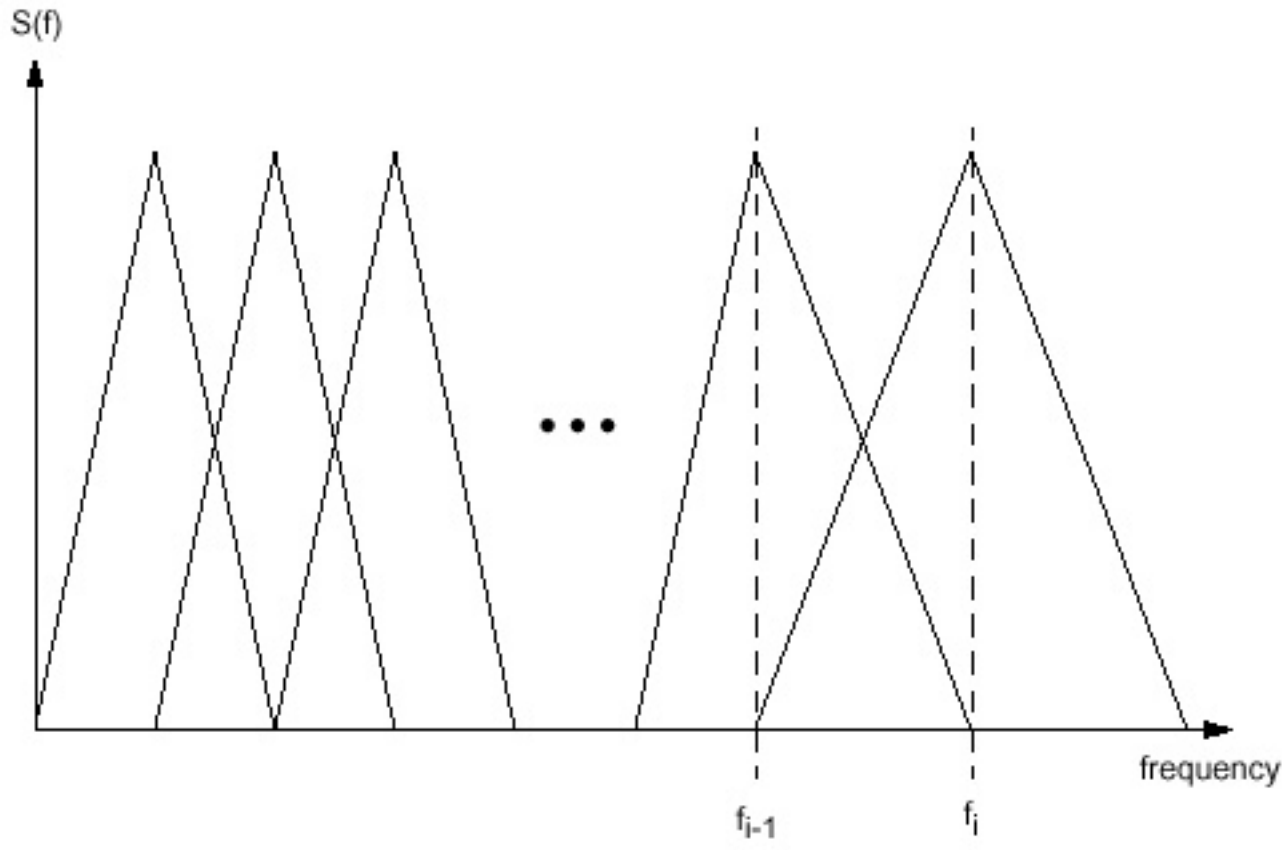
Index	Bark Scale		Mel Scale	
	Center Freq. (Hz)	BW (Hz)	Center Freq. (Hz)	BW (Hz)
1	50	100	100	100
2	150	100	200	100
3	250	100	300	100
4	350	100	400	100
5	450	110	500	100
6	570	120	600	100
7	700	140	700	100
8	840	150	800	100
9	1000	160	900	100
10	1170	190	1000	124
11	1370	210	1149	160
12	1600	240	1320	184
13	1850	280	1516	211
14	2150	320	1741	242
15	2500	380	2000	278
16	2900	450	2297	320
17	3400	550	2639	367
18	4000	700	3031	422
19	4800	900	3482	484
20	5800	1100	4000	556
21	7000	1300	4595	639
22	8500	1800	5278	734
23	10500	2500	6063	843
24	13500	3500	6964	969

24	13500	3500	6964	969

- **Nonlinear Frequency Warping**: The Bark scale implies a nonlinear frequency mapping of frequency.



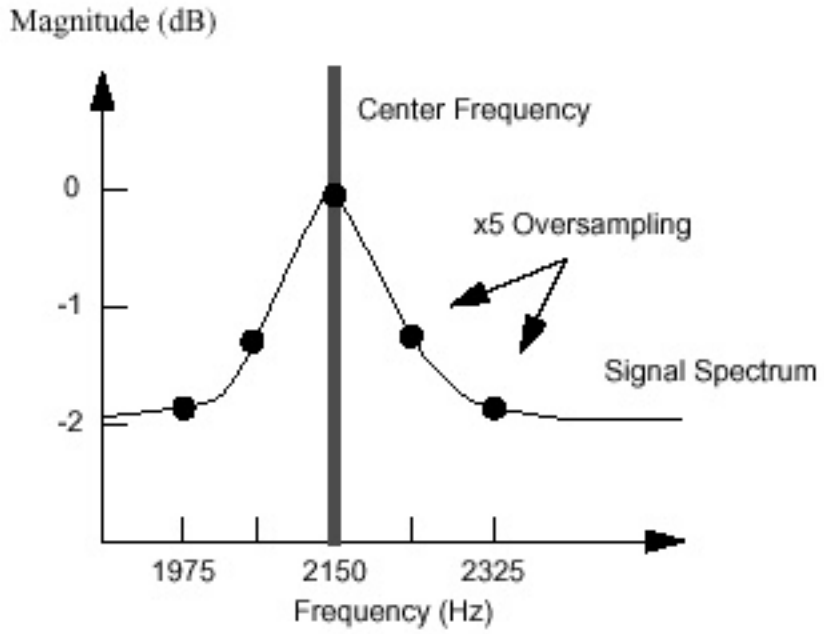
A DIGITAL FILTERBANK



- Note that an FFT yields frequency samples at $(k/N)f_s$.
- Oversampling provides a smoother estimate of the envelope of the spectrum.
- Other efficient techniques exist for different frequency scales (e.g., bilinear transform).

OVERSAMPLING IMPROVES PERFORMANCE

The spectrum is oversampled to avoid biased estimates and to reduce variation in the measurements due to quantization of the frequency scale (for example, formants with narrow bandwidths):



For example, consider the parameters of a typical front end:

$$\text{sample frequency} = 8 \text{ kHz}$$

$$\text{frame duration} = 10 \text{ msec}$$

$$\text{window duration} = 25 \text{ msec (200 points)}$$

$$\text{FFT length} = 256 \text{ points}$$

$$\text{max frequency} = \text{sample frequency} / 2 = 4 \text{ kHz}$$

$$\text{max mel frequency} = \text{max frequency in mel} = 2146.1 \text{ mel}$$

$$\text{number of mel frequency scale bins} = 24 \text{ bins}$$

$$\text{mel frequency resolution} = \text{max mel frequency} / (24 + 1) = 85.84 \text{ mel}$$

$$\text{center frequency} = i * 85.84 \text{ mel}$$

This approach generates the table shown below:

Bin #	Continuous Frequency			Discrete Frequency
	Start (Hz/Mel)	Center (Hz/Mel)	Stop (Hz/Mel)	Range (Index)
1	0.0 0.0	55.4 85.8	115.2 171.7	0 - 3
2	55.4 85.8	115.2 171.7	179.7 257.5	2 - 5
3	115.2 171.7	179.7 257.5	249.3 343.4	4 - 7
4	179.7 257.5	249.3 343.4	324.5 429.2	6 - 10
5	249.3 343.4	324.5 429.2	405.5 515.1	8 - 12
6	324.5 429.2	405.5 515.1	493.0 600.9	11 - 15
7	405.5 515.1	493.0 600.9	587.5 686.7	13 - 18
8	493.0 600.9	587.5 686.7	689.4 772.6	16 - 22
9	587.5 686.7	689.4 772.6	799.3 858.4	19 - 25
10	689.4 772.6	799.3 858.4	918.0 944.3	23 - 29
11	799.3 858.4	918.0 944.3	1046.1 1030.1	26 - 33
12	918.0 944.3	1046.1 1030.1	1184.2 1116.0	30 - 37
13	1046.1 1030.1	1184.2 1116.0	1333.4 1201.8	34 - 42
14	1184.2 1116.0	1333.4 1201.8	1494.3 1287.6	38 - 47
15	1333.4 1201.8	1494.3 1287.6	1668.0 1373.5	43 - 53
16	1494.3 1287.6	1668.0 1373.5	1855.4 1459.3	48 - 59
17	1668.0 1373.5	1855.4 1459.3	2057.6 1545.2	54 - 65
18	1855.4 1459.3	2057.6 1545.2	2275.9 1631.0	60 - 72
19	2057.6 1545.2	2275.9 1631.0	2511.4 1716.9	66 - 80

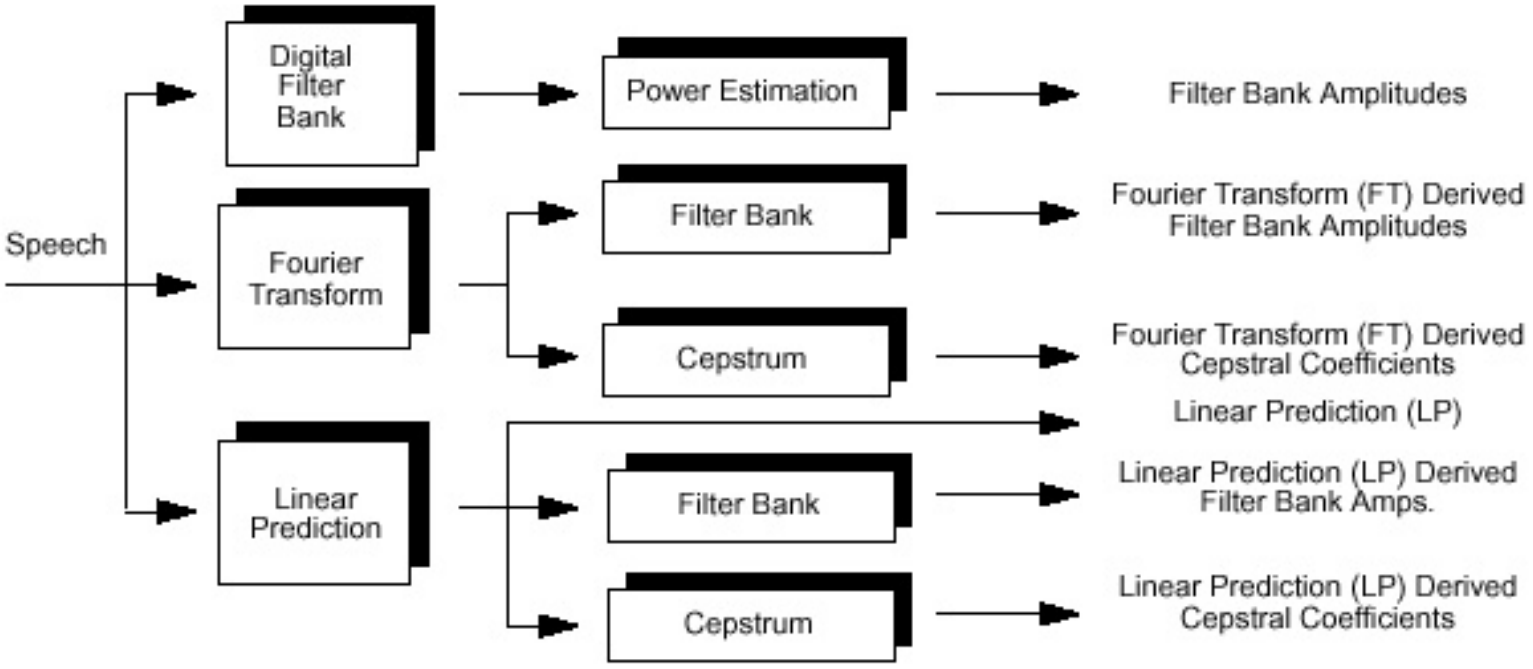
20	2275.9 1631.0	2511.4 1716.9	2765.6 1802.7	73 - 88
21	2511.4 1716.9	2765.6 1802.7	3039.9 1888.5	81 - 97
22	2765.6 1802.7	3039.9 1888.5	3335.9 1974.4	89 - 106
23	3039.9 1888.5	3335.9 1974.4	3655.3 2060.2	98 - 116
24	3335.9 1974.4	3655.3 2060.2	4000.0 2146.1	107 - 127

Finally, these 24 points are used to compute a forward DCT (extended to be a 48-point periodic and even sequence). The first 12 coefficients are retained.

The forward DCT is used because of its energy compaction property (a property shared by many orthogonal transforms). This transform allows us to approximate the data with fewer coefficients, since the coefficients are more concentrated at lower indices. Hence, we truncate the representation to 12 coefficients and retain most of the important information, as well as ensure that the coefficients are orthogonal to one another.

ALTERNATIVE METHODS FOR
FREQUENCY DOMAIN ANALYSIS

We have now established two different ways to perform a filterbank analysis of the speech signal (temporal and spectral):



The most popular front ends are those that use cepstral coefficients dervied from the Fourier transform. Why?

IS PHASE IMPORTANT IN SPEECH RECOGNITION?

An FIR filter composed of all zeros that are inside the unit circle is minimum phase. There are many realizations of a system with a given magnitude response; one is a minimum phase realization, one is a maximum-phase realization, others are in-between. Any non-minimum phase pole-zero system can be decomposed into:

$$H(z) = H_{min}(z)H_{ap}(z)$$

It can be shown that of all the possible realizations of |H(f)|, the minimum-phase version is the most compact in time. Define:

$$E(n) = \sum_{k=0}^{n} |h(k)|^2$$

Then, $E_{min}(n) >= E(n)$ for all n and all possible realizations of |H(f)|.

Why is minimum phase such an important concept in speech processing?

We prefer systems that are invertible:

$$H(z)\, H^{-1}(z) = 1$$

We would like both systems to be stable. The inverse of a non-minimum phase system is not stable.

We end with a very simple question: is phase important in speech recognition?

EXPLORATIONS INTO SPECTRUM ANALYSIS

"All applets on our web site now require the java plugin to run. This is necessary so we can bring state-of-the-art features to you which are not currently supported by browser vendors. You can find the plugin at http://java.sun.com/products/plugin. For additional information or suggestions please contact help@isip.msstate.edu"
No JDK 1.2 support for APPLET!!

- Source Code: Download the source code for this applet.
- Tutorial: Learn how to use this applet.

All applets on our web site now require a Java plug-in. This is necessary so we can bring state-of-the-art Java features to you which are not currently supported by browser vendors such as Netscape. You can find the appropriate plug-in at http://java.sun.com/products/plugin. We have generated a list of steps necessary for installing the plug-in in a Unix environment. For additional information or help with your installation please contact help@isip.msstate.edu.

FFT Algorithms

This directory contains software generated as part of a joint project involving ISIP and High Performance Computing Laboratory (HPCL), which does pioneering research in Message Passing Interface (MPI) standards. The main goals of this project were to test the performance portability using object oriented concepts.

FFT's were a natural choice to test these concepts; the large number of algorithms developed over the years for its efficient computation give us a wide range of implementation choices. The ultimate goal of this part of the project was a capability for coarse-grain poly-algorithmic selection. For this reason we needed to analyze and benchmark the algorithms based on a large set of criteria.

- So what were the algorithms benchmarked?
- Traditional benchmarking has focused on computation speed or number of computations. In our work we deviate from this mainly because advances in CPU speeds and compilers in past decade or so have made the whole process of computation highly system dependent. The criteria used are therefore quite comprehensive.
- The software has a simple command-line interface
- The algorithms were compared for computation speed by running multiple iterations on a Pentium Pro 200MHz processor and compiled using GCC. Since memory usage and computation times are well correlated in any algorithm implementation it is worth seeing the relationship. As expected, in most cases there exists an inverse relationship.
- One of the most important results of the work is the bare-bone comparison of the algorithms in terms of various mathematical operation counts. We have analyzed integer and floating point operations separately since computation times for each of these operations differ significantly on many CPUs.
- Another very significant result is the performance comparison of various contemporary widely used CPUs. In many cases results may be affected dramatically by the compilers in use. We calibrated the effect of GCC and MSVC++ on two algorithms FHT and SRFFT.
- There were some general conclusions from this work as well as some algorithm level conclusions which are valuable and can be used effectively in the choice of FFT algorithms.
- You can find all this material in the Master's presentation on this work by Aravind Ganapathiraju, which sums it all in great detail.
- We also have a detailed report on this work.
- Source code generated from this work is available for public. Enjoy!!!!!

Introduction

FFTW is a C subroutine library for computing the Discrete Fourier Transform (DFT) in one or more dimensions, of both real and complex data, and of arbitrary input size. We believe that FFTW, which is free software, should become the FFT library of choice for most applications. Our benchmarks, performed on on a variety of platforms, show that FFTW's performance is typically superior to that of other publicly available FFT software. Moreover, FFTW's performance is *portable*: the program will perform well on most architectures without modification.

It is difficult to summarize in a few words all the complexities that arise when testing many programs, and there is no "best" or "fastest" program. However, FFTW appears to be the fastest program most of the time for in-order transforms, especially in the multi-dimensional and real-complex cases (Kasparov is the best chess player in the world even though he loses some games). Hence the name, "FFTW," which stands for the somewhat whimsical title of "Fastest Fourier Transform in the West." Please visit the benchFFT home page for a more extensive survey of the results.

The FFTW package was developed at MIT by Matteo Frigo and Steven G. Johnson.

Features

FFTW 2.1.3 is the latest official version of FFTW (refer to the release notes to find out what is new). Subscribe to the fftw-announce mailing list to receive announcements of future updates. Here is a list of some of FFTW's more interesting features:

- Speed.
- Both one-dimensional and multi-dimensional transforms.
- Arbitrary-size transforms. (However, sizes with small prime factors are best. FFTW now uses an O(N lg N) algorithm even for prime sizes, at least for complex transforms.)
- Efficient handling of multiple, strided transforms. (This lets you do things like transform multiple arrays at once, transform one dimension of a multi-dimensional array, or transform one field of a multi-component array.)
- Real-to-complex transforms. (Completely rewritten in version 2.0; the interface has changed slightly.)
- Parallel transforms: parallelized code for platforms with Cilk or for SMP machines with some flavor of threads (e.g. POSIX). An MPI version for distributed-memory transforms is also available.
- Works on any platform with a C compiler. Documentation in HTML and other formats.

- Callable from Fortran (via wrapper routines included with FFTW).
- FFTW is Free software, released under the GNU General Public License (GPL). (Non-free licenses may also be purchased from MIT, for users who do not want their programs protected by the GPL. Contact us for details.) (Also see the FAQ.)

FFTW Wins Wilkinson Prize

FFTW received the 1999 J. H. Wilkinson Prize for Numerical Software, which is awarded every four years to the software that "best addresses all phases of the preparation of high quality numerical software."

Wilkinson was a seminal figure in modern numerical analysis as well as a key proponent of the notion of reusable, common libraries for scientific computing, and we are especially honored to receive this award in his memory.

Documentation

For answers to some common questions, read the FFTW FAQ.

You can read the FFTW 2.1.3 manual online. (Also available, in the FFTW package, are PostScript, LaTeX, and TeXinfo versions of the documentation.) For general questions about Fourier transforms, see our links to FFT-related resources.

We benchmarked FFTW against every public-domain FFT we could get our hands on, in both one and three dimensions, on a variety of platforms. You can view the results from this benchmark, or download it to run on your own machine and compiler, at the benchFFT web page.

Three papers about FFTW are available online. The most current general paper on FFTW was published in the 1998 ICASSP conference proceedings (vol. 3, pp. 1381-1384) with the title "FFTW: An Adaptive Software Architecture for the FFT" (also in Postscript), by M. Frigo and S. G. Johnson. If you wish to cite FFTW, we suggest referencing this ICASSP paper. An earlier (and somewhat out-of-date) technical report is "The Fastest Fourier Transform in the West," MIT-LCS-TR-728 (September 1997) (also in Postscript.). The paper "A Fast Fourier Transform Compiler," by Matteo Frigo, appears in the Proceedings of the 1999 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI '99), Atlanta, Georgia, May 1999. This paper describes the guts of the FFTW codelet generator. (Also in Postscript. The slides from the talk are also available.) You might also be interested in "Cache-Oblivious Algorithms," by M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran (FOCS '99).

The slides from the 7/28/98 talk "The Fastest Fourier Transform in the West," by M. Frigo, are also available, along with the slides from a shorter 1/14/98 talk on the same subject by S. G. Johnson.

By popular demand, we now have our very own Y2K Statement.

Downloading

Version 2.1.3 of FFTW may be downloaded from this site. Feel free to post FFTW on your own site, but be sure to tell us so that we can link to your page and notify you of updates to the software.

Acknowledgements

We are grateful for the support of many people and companies, including Sun, Intel, the GNU project, and the Linux community. Please see the acknowledgements section of our manual for a more complete list of those who helped us. We are especially thankful to all of our users for their continuing support, feedback, and interest in the development of FFTW.

Related Links

We have put together a list of links to other interesting sites with FFT-related code or information. This should be helpful if you want to know more about Fourier transforms or if for some reason FFTW doesn't satisfy your needs.

Feedback

If you have comments, questions, or suggestions regarding FFTW, don't hesitate to email us at fftw@fftw.org. We support encrypted/signed email. Use our public keys.