CS601: Software Development for Scientific Computing

Autumn 2022

Week7: Motifs – Sparse Matrices (contd.), Fourier Transforms, Intermediate C++ (object orientation)

Last week..

- Matrix Multiplication
  - ijk variants and recursive matmul
- Efficiency considerations
  - Storage (e.g. cache-oblivious data storage using Z-ordering)
  - Communication cost (data movement cost)
  - Special hardware (FMA, Vector units)
- Motif: Sparse Matrices
  - Triangular Matmul (as an e.g. that exploits structure to accelerate computation)
  - Storage scheme for sparse matrices (e.g. CSR)
  - Banded matrices (y=y+Ax with banded matrix and optimized storage)

y=y+Ax with *Separable* Matrices

Refer to (Section 1 only):

https://www.math.uci.edu/~chenlong/MathPKU/FMMsimple.pdf

Matrix Algebra and Efficient Computation

- **Pic source: the Parallel Computing Laboratory at U.C. Berkeley: A Research Agenda Based on the Berkeley View (2008)**

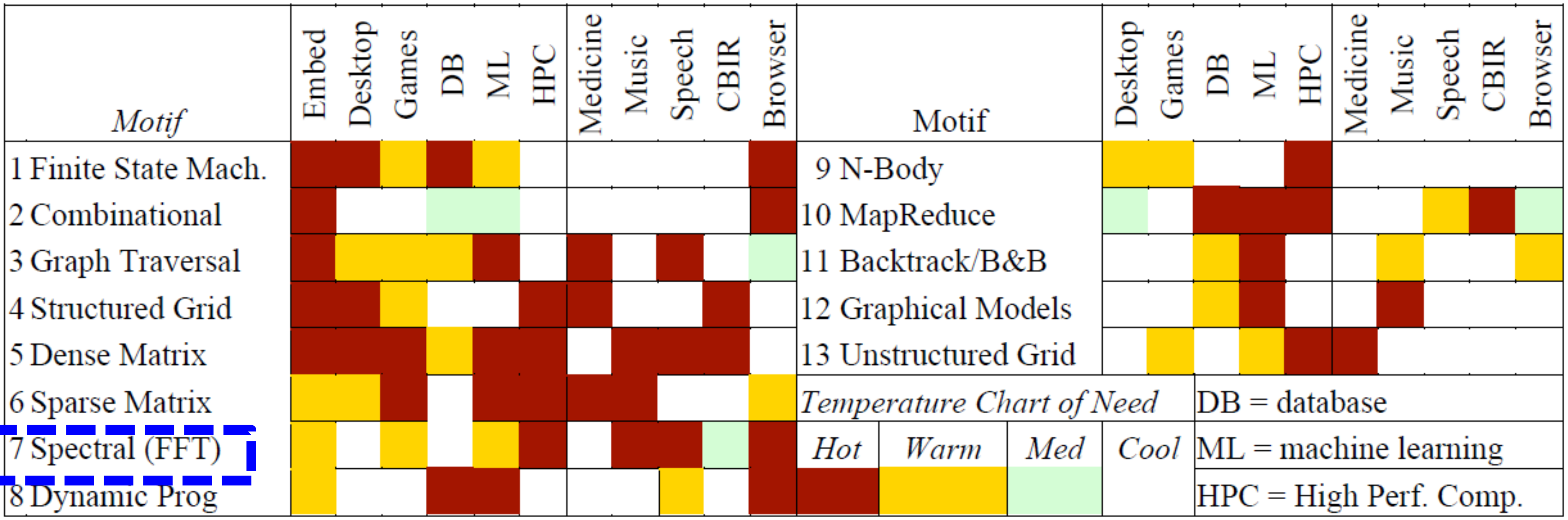


Figure 4. Temperature Chart of the 13 Motifs. It shows their importance to each of the original six application areas and then how important each one is to the five compelling applications of Section 3.1. More details on the motifs can be found in (Asanovic, Bodik et al. 2006).

⇨ Seen earlier

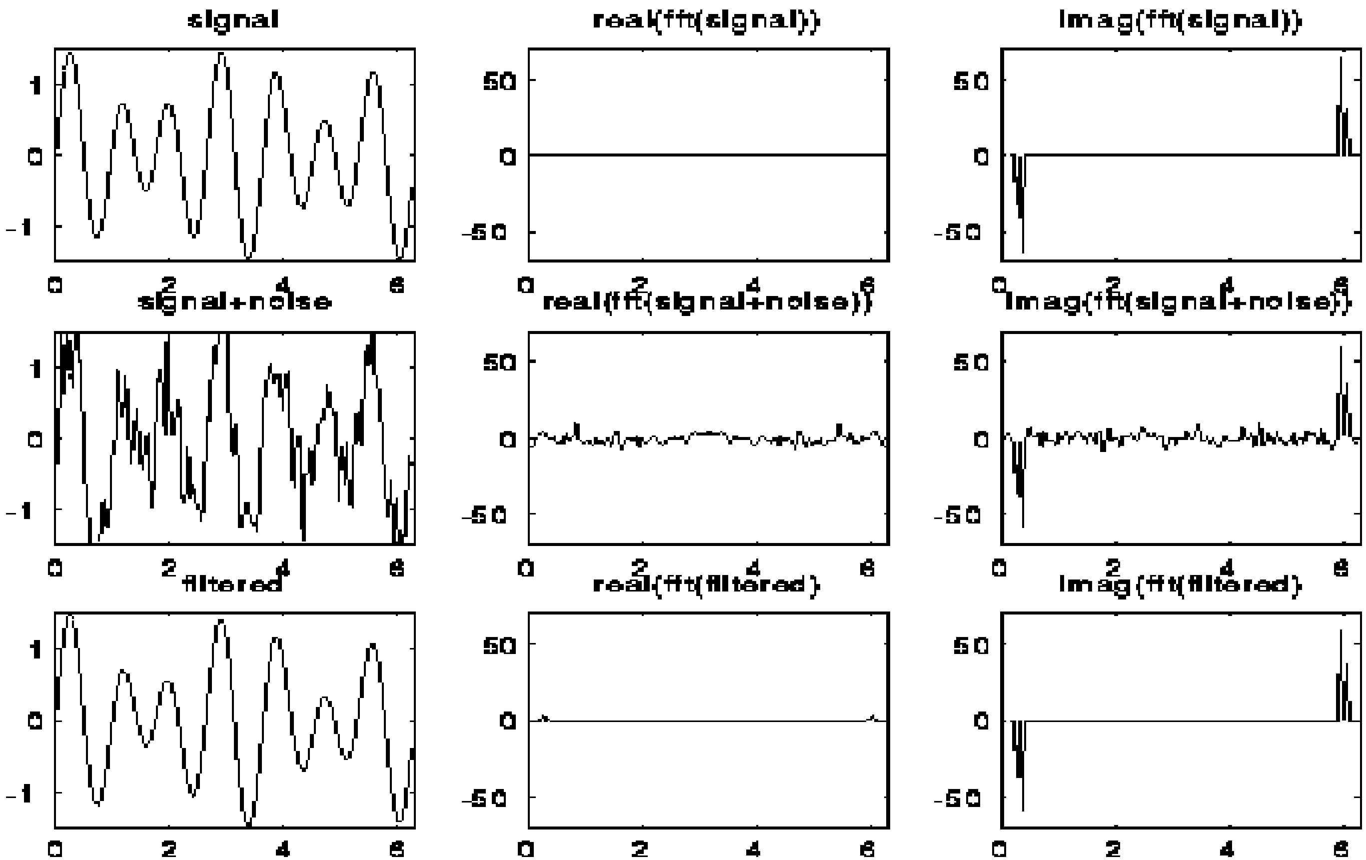
Next..

Faster y=Ax: Discrete Fourier Transforms (DFT)

- Very widely used
  - Image compression (jpeg)
  - Signal processing
  - Solving Poisson's Equation
- Represent A with F, a *Fourier Matrix* that has the following (remarkable) properties:
  - F^{-1} is easy to compute
  - Multiplications by F and F^{-1} is fast. (need to do Fx=y and x= F^{-1} y quickly)
- F has complex numbers in its entries.
  - Every entry is a power of a single number w such that $w^n=1$
  - Any entry of a Fourier matrix can be written using $f_{ij} = w^{ij}$ (row and col indices start from 0)

Using the 1D FFT for filtering

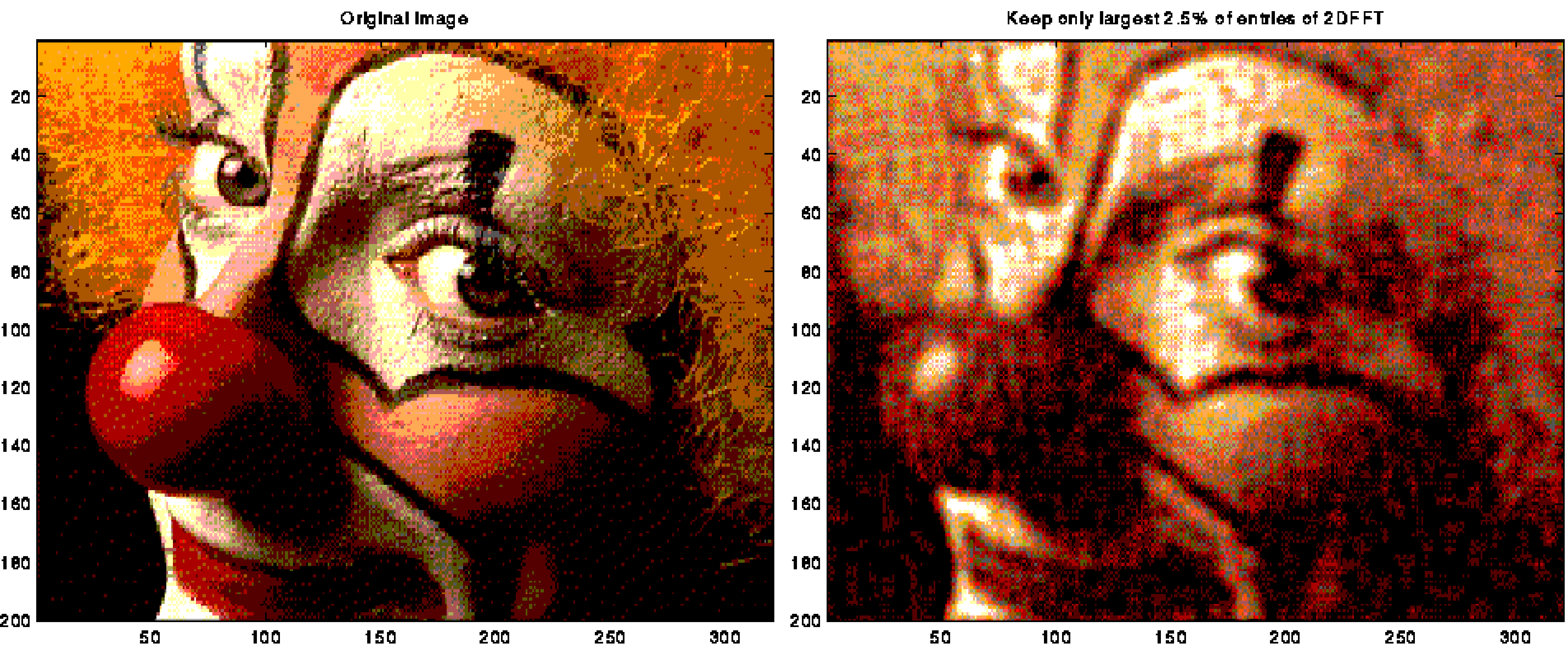
- Signal = sin(7t) + .5 sin(5t) at 128 points
- Noise = random number bounded by .75
- Filter by zeroing out FFT components < .25



Slide courtesy: James Demmel (CS267, Spring19) www.cs.berkeley.edu/~demmel

Using the 2D FFT for image compression

- Image = 200x320 matrix of values
- Compress by keeping largest 2.5% of FFT components
- Similar idea used by jpeg



Examples: Fourier Matrix

- 4x4: $\boldsymbol{F_4} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & i & i^2 & i^3 \\ 1 & i^2 & i^4 & i^6 \\ 1 & i^3 & i^6 & i^9 \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & 1 & w^2 \\ 1 & w^3 & w^2 & w^1 \end{bmatrix}, i = \sqrt{-1}$
  - Here, $w = i$ (also denoted as w_4=i). w^4 = 1 => i is a root.
- 8x8: $\boldsymbol{F_8} =$

  Here, $w = \frac{1+i}{\sqrt{2}}$
  (= sqrt of i)

$$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & w^8 & w^{10} & w^{12} & w^{14} \\ 1 & w^3 & w^6 & w^9 & w^{12} & w^{15} & w^{18} & w^{21} \\ 1 & w^4 & w^8 & w^{12} & w^{16} & w^{20} & w^{24} & w^{28} \\ 1 & w^5 & w^{10} & w^{15} & w^{20} & w^{25} & w^{30} & w^{35} \\ 1 & w^6 & w^{12} & w^{18} & w^{24} & w^{30} & w^{36} & w^{42} \\ 1 & w^7 & w^{14} & w^{21} & w^{28} & w^{35} & w^{42} & w^{49} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix}$$

Example: Faster y=Fx

$$
\begin{array}{l}
\begin{array}{cccccccc} \text{Column: } 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{array} \\
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\
1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\
1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\
1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\
1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\
1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\
1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1
\end{bmatrix}
\end{array}
=
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & \omega^2 & \omega^4 & \omega^6 & \omega & \omega^3 & \omega^5 & \omega^7 \\
1 & \omega^4 & 1 & \omega^4 & \omega^2 & \omega^6 & \omega^2 & \omega^6 \\
1 & \omega^6 & \omega^4 & \omega^2 & \omega^3 & \omega & \omega^7 & \omega^5 \\
1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\
1 & \omega^2 & \omega^4 & \omega^6 & -\omega & -\omega^3 & -\omega^5 & -\omega^7 \\
1 & \omega^4 & 1 & \omega^4 & -\omega^2 & -\omega^6 & -\omega^2 & -\omega^6 \\
1 & \omega^6 & \omega^4 & \omega^2 & -\omega^3 & -\omega & -\omega^7 & -\omega^5
\end{bmatrix}
$$

(Writing columns 1,3,5,7 first and then columns 2,4,6,8)

Example: Faster y=Fx

- $$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix} = \left[\begin{array}{cccc|cccc} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & \omega^2 & \omega^4 & \omega^6 & \omega & \omega^3 & \omega^5 & \omega^7 \\ 1 & \omega^4 & 1 & \omega^4 & \omega^2 & \omega^6 & \omega^2 & \omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & \omega^3 & \omega & \omega^7 & \omega^5 \\ \hline 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 \\ 1 & \omega^2 & \omega^4 & \omega^6 & -\omega & -\omega^3 & -\omega^5 & -\omega^7 \\ 1 & \omega^4 & 1 & \omega^4 & -\omega^2 & -\omega^6 & -\omega^2 & -\omega^6 \\ 1 & \omega^6 & \omega^4 & \omega^2 & -\omega^3 & -\omega & -\omega^7 & -\omega^5 \end{array}\right]$$

$\Uparrow$

(Partitioning into 4 matrix blocks of size 4x4.)

Recall: $F_4 = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 \\ 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w^9 \end{bmatrix}$, where $w = i = w_4$

Note: in F_8, w $= \frac{1+i}{\sqrt{2}} = w_8$

therefore, $w_8^2 = w_4$

Example: Faster y=Fx

- $$\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & w & w^2 & w^3 & w^4 & w^5 & w^6 & w^7 \\ 1 & w^2 & w^4 & w^6 & 1 & w^2 & w^4 & w^6 \\ 1 & w^3 & w^6 & w & w^4 & w^7 & w^2 & w^5 \\ 1 & w^4 & 1 & w^4 & 1 & w^4 & 1 & w^4 \\ 1 & w^5 & w^2 & w^7 & w^4 & w^1 & w^6 & w^3 \\ 1 & w^6 & w^4 & w^2 & 1 & w^6 & w^4 & w^2 \\ 1 & w^7 & w^6 & w^5 & w^4 & w^3 & w^2 & w^1 \end{bmatrix} = \begin{array}{c|c} \mathrm{F}_4 & \Omega_4 \mathrm{F}_4 \\ \hline \mathrm{F}_4 & -\Omega_4 \mathrm{F}_4 \end{array}$$

(because $w^2 = w_4$)

$$\Omega_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & w & 0 & 0 \\ 0 & 0 & w^2 & 0 \\ 0 & 0 & 0 & w^3 \end{bmatrix}$$ (note: $w = \frac{1+i}{\sqrt{2}} = w_8$)

- So, $F_8 = \begin{bmatrix} \mathrm{F}_4 & \Omega_4 \mathrm{F}_4 \\ \mathrm{F}_4 & -\Omega_4 \mathrm{F}_4 \end{bmatrix}$

FFT

We can obtain 8-point DFT from 4-point DFT. But how do we obtain the result of $y=F_8x$, from $y_{top} = \boxed{F_4x_{odd}}$ and $y_{bottom} = \boxed{F_4x_{even}}$?

$$\overset{F_8}{\left[\begin{array}{c|c} F_4 & \Omega_4 F_4 \\ \hline F_4 & -\Omega_4 F_4 \end{array}\right]} \overset{x}{\left[\begin{array}{c} x1 \\ x3 \\ x5 \\ x7 \\ \hline x2 \\ x4 \\ x6 \\ x8 \end{array}\right]} = \left[\begin{array}{c|c} I_4 & \Omega_4 \\ \hline I_4 & -\Omega_4 \end{array}\right] \left[\begin{array}{c} \boxed{F_4 \begin{bmatrix} x1 \\ x3 \\ x5 \\ x7 \end{bmatrix}} \\ \hline \boxed{F_4 \begin{bmatrix} x2 \\ x4 \\ x6 \\ x8 \end{bmatrix}} \end{array}\right] = \overset{y}{\left[\begin{array}{c} y1 \\ y3 \\ y5 \\ y7 \\ \hline y2 \\ y4 \\ y6 \\ y8 \end{array}\right]}$$

Note: can be done with 4 multiplications

$y1$ to $y4 = y_{top} + \Omega_4 * y_{bottom}$

$y5$ to $y8 = y_{top} - \Omega_4 * y_{bottom}$

$y_{top} = F_4x_{odd}$
$y_{bottom} = F_4x_{even}$

(x_{odd}= elements at odd numbered indices of vector x)

(x_{even}= elements at even numbered indices of vector x)

Divide-and-Conquer FFT (D&C FFT)

FFT(v, ϖ, m) … assume m is a power of 2

if $m = 1$ return $v[0]$

else

v_{even} = FFT(v[0:2:m-2], ϖ^2, $m/2$)

v_{odd} = FFT(v[1:2:m-1], ϖ^2, $m/2$)

ϖ-vec = $[\varpi^0, \varpi^1, \ldots \varpi^{(m/2-1)}]$ precomputed

return $[v_{even} + (\varpi\text{-vec} \;.\!*\; v_{odd}),$

$v_{even} - (\varpi\text{-vec} \;.\!*\; v_{odd})]$

- Matlab notation: ".*" means component-wise multiply.

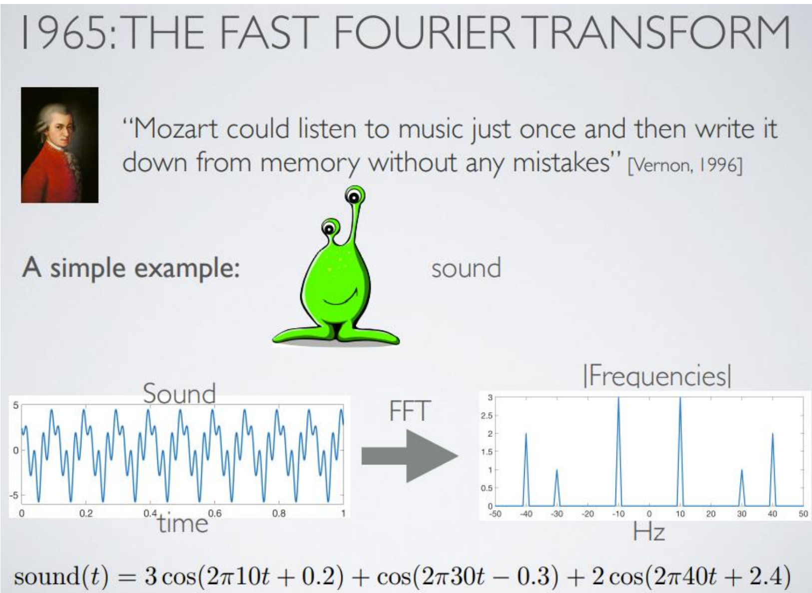
Cost: $T(m) = 2T(m/2)+O(m) = O(m \log m)$ operations.

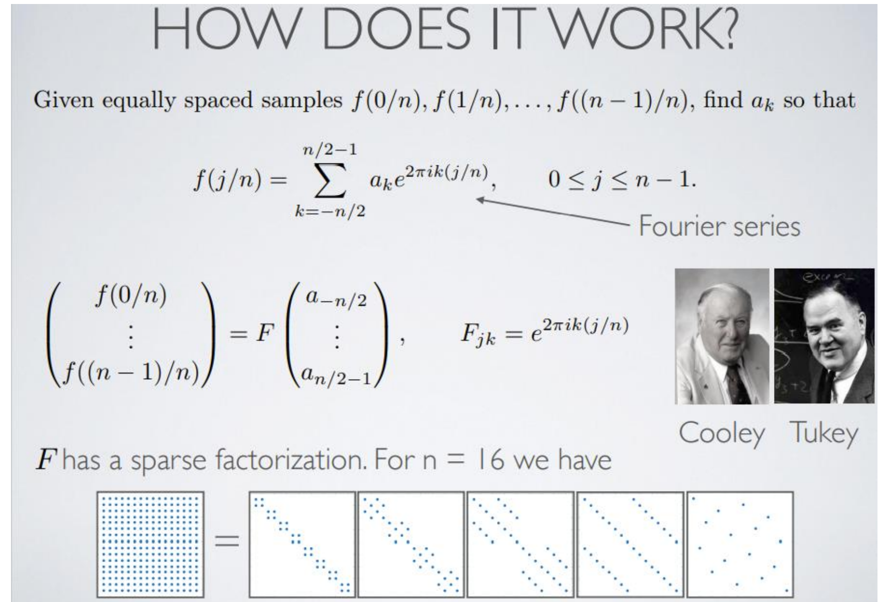
Popularized/published by Cooley-Tuckey in 1965.

Slide courtesy: James Demmel, www.cs.berkeley.edu/~demmel

FFT - Summary

- We will revisit FFT when solving Poisson’s equation
- 2-slide summary (**courtesy**: Alex Townsend, Cornell. Source)





- References:
  - Refer to Lecture 20 (Spring 2018) at https://inst.eecs.berkeley.edu/~cs267/archives.html
  - Section 1.4, Matrix Computations, 4th Ed, Golub and Van Loan
  - Section 3.5, Linear Algebra and Its Applications, 4th Ed, Gilbert Strang

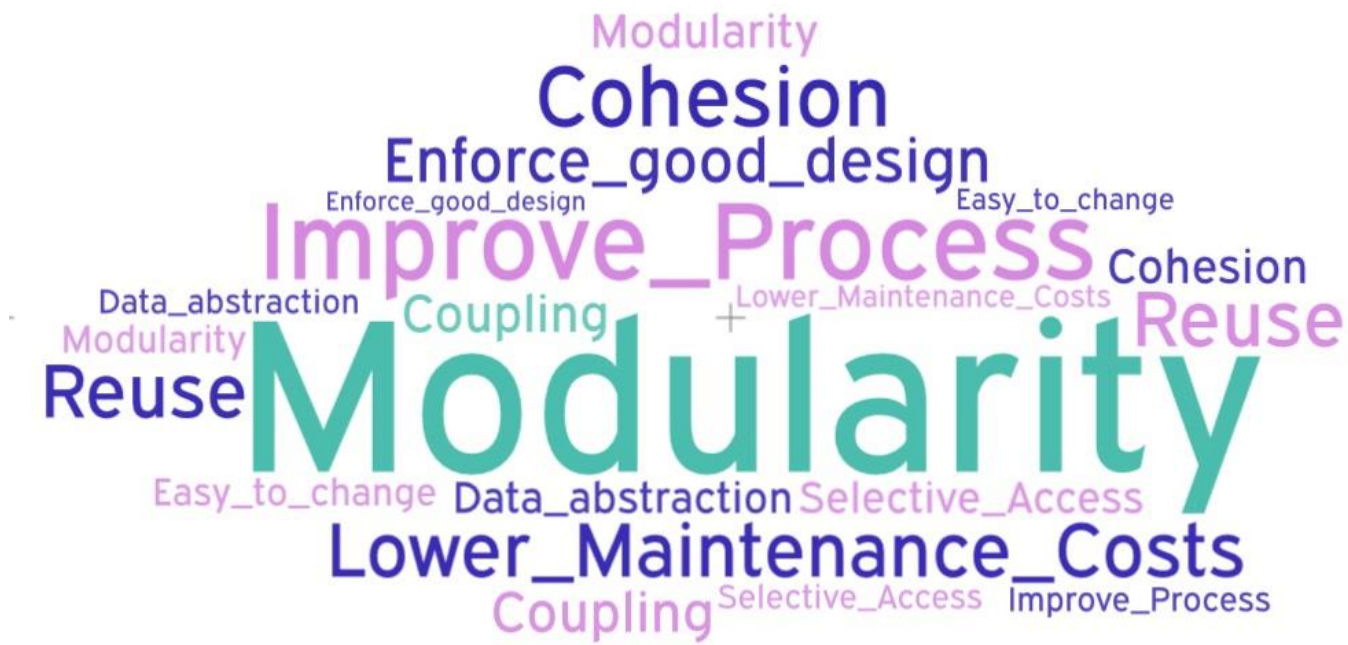
Object Orientation

- What does it mean to think in terms of object orientation?

  1. Give precedence to data over functions *(think: objects, attributes, methods)*
  2. Hide information under well-defined and stable interfaces *(think: encapsulation)*
  3. Enable incremental refinement and (re)use *(think: inheritance and polymorphism)*

Object Orientation: Why?

- Improve costs
- Improve development process and
- Enforce good design



Objects and Instances

- Object is a computational unit
  - Has a state and operations that operate on the state.
  - The state consists of a collection of *instance* variables or attributes.
  - Send a “message” to an object to invoke/execute an operation (*message-passing metaphor* in traditional OO thinking)
- An instance is a *specific version* of the object

Classes

- Template or blueprint for creating objects. Defines the shape of objects
  - Has *features* = attributes + operations
  - New objects created are *instances of the class*
  - E.g.

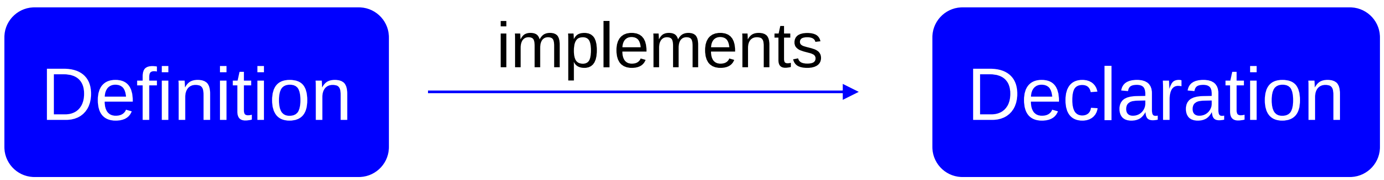
Class - lollypop mould

Objects - lollypops

Classes continued..

- Operations defined in a class are a prescription or service provided by the class to access the state of an object
- Why do we need classes?
  - To define user-defined types / invent new types and extend the language
  - Built-in or Primitive types of a language – int, char, float, string, bool etc. have implicitly defined operations:
    - E.g. cannot execute a *shift* operator on a negative integer
  - Composite types *(read: classes)* have operations that are implicit as well as those that are explicitly defined.

Classes declaration vs. definition



Implementation of functions in a .cpp file

listing of functions and attributes in a .h file

Classes: declaration

- *file* `Fruit.h`

Common terms for the state of an object:
“fields”, “attributes”, “property”, “data”
“characteristic”

```
#include<string>

class Fruit {
      string commonName;

public:
      Fruit(string name);
      string GetName();
};
```

Class Name

Attribute

Constructor

Method

Common terms for operations:
“functions”, “behavior”, “message”,
“methods”, “responsibilities”

Trivia: Python doesn't support data hiding

Classes: access control

- `Public / Private / Protected`

```
class Fruit {
        string commonName; // private by default

public:
        Fruit(string name);
        string GetName();
};
```

- `Private:` methods-only (self) access
- `Public:` all access
- `Protected:` methods (self and *sub-class*) access

Classes: definition

- *file* `Fruit.cpp`

```
#include<Fruit.h>

//constructor definition: initialize all attributes
Fruit::Fruit(string name) {
      commonName = name;
}
//constructor definition can also be written as:
Fruit::Fruit(string name): commonName(name) { }

string Fruit::GetName() {
      return commonName;
}
```

Objects: creation and usage

- *file* `Fruit.cpp`

```
#include<Fruit.h>

Fruit::Fruit(string name): commonName(name) { }
string Fruit::GetName() { return commonName; }

int main() {
      Fruit obj1("Mango"); //calls constructor
      //following line prints "Mango"
      cout<<obj1.GetName()<<endl; //calls GetName
method
}
```

- *How is `obj1` destroyed?* – by calling *destructor*

Objects: Destructor

```
Fruit::~Fruit(){ } //default destructor implicitly
defined

int main() {
	Fruit obj1("Mango"); //statically allocated
object
	Fruit* obj2 = new Fruit("Apple"); //dynamic
object
	delete obj2; //calls obj2->~Fruit();
	//calls obj1.~Fruit()
}
```

- Statically allocated objects: Automatic
- Dynamically allocated objects: Explicit

Post-class Exercise - Encapsulation

- The earlier quiz at the beginning of the class was a Pre-class Exercise.
- Re-attempt the same Quiz.

Inheritance

- Create a brand-new class based on existing class

```
file Mango.h
#include<Fruit.h>
class Mango : public Fruit {
        string variety;
public:
        Mango(string name, string var) : Fruit(name),
variety(var){}
};
```

calling base-class constructor

- Fruit is a base type, Mango is a sub-type
- Sub-type inherits attributes and methods of its base type

Inheritance

file Fruit.h

```
#include<string>

class Fruit {
	string commonName;
public:
	Fruit(string name);
	string GetName();
};
```

file Mango.h

```
#include<Fruit.h>
class Mango : public Fruit {
	string variety;
public:
	Mango(string name, string var) :
Fruit(name), variety(var){}
};
```

file Fruit.cpp

```
...
                         commonName  variety
int main() {
	Mango item1("Mango", "Alphonso"); //create sub-class object
	cout<<item1.GetName()<<endl; //only commonName is printed!
	(variety is not included).
	Refer slide 41.
}
```

Method overriding

- Customizing methods of derived / sub- class

file Fruit.h

```
#include<string>

class Fruit {
        string
commonName;
public:
        Fruit(string
name);
        string GetName();
};
```

file Mango.h

```
#include<Fruit.h>
class Mango : public Fruit {
        string variety;
public:
        Mango(string name, string var) :
Fruit(name), variety(var){}
    string GetName();
};
```

method with the same name as in base class

Method overriding

```
file Fruit.h
#include<string>

class Fruit {
protected:
	string commonName;
public:
	Fruit(string name);
	string GetName();
};
```

```
file Mango.h
#include<Fruit.h>
class Mango : public Fruit {
	string variety;
public:
	Mango(string name, string var) : Fruit(name), variety(var){}
	string GetName() {	return commonName + “_” + variety; }
};
```

accessing base class attribute

Method overriding

file Fruit.h

```
#include<string>

class Fruit {
protected:
	string commonName;
public:
	Fruit(string name);
	string GetName();
};
```

file Mango.h

```
#include<Fruit.h>
class Mango : public Fruit {
	string variety;
public:
	Mango(string name, string var) : Fruit(name), variety(var){}
	string GetName() {	return commonName + "_" + variety; }
};
```

file Fruit.cpp

```
...
int main() {
	Mango item1("Mango", "Alphonso"); //create sub-class object
	cout<<item1.GetName()<<endl;	//prints "Mango_Alphonso"
}
```

Polymorphism

- Ability of one type to appear and be used as another type
- E.g. type `Mango` used as type `Fruit`

```
file Fruit.cpp
...
int main() {
//create a sub-class object and initialize it to a pointer of
//type base-class
       Fruit* item1 = new Mango("Mango", "Alphonso");
       cout<<item1->GetName()<<endl;  //prints "Mango" !
       ...
}
```

Trivia: Java treats all functions as virtual

Polymorphism

- Declare overridden functions as `virtual` in base class
- Invoke those functions using pointers

```
file Fruit.h
#include<string>

class Fruit {
protected:
        string commonName;
public:
        Fruit(string name);
        virtual string GetName();
};
```

```
file Mango.h
#include<Fruit.h>
class Mango : public Fruit {
        string variety;
public:
        Mango(string name, string
var) : Fruit(name), variety(var){}
string GetName() {    return
commonName + "_" + variety; }
};
```

```
Fruit* item1 = new Mango("Mango", "Alphonso");
cout<<item1->GetName()<<endl; //prints "Mango_Alphonso"
```

Polymorphism and Destructors

- declare base class destructors as `virtual` if using base class in a polymorphic way

```
file Fruit.h
#include<string>

class Fruit {
protected:
	string commonName;
public:
	Fruit(string name);
	virtual string GetName();
	virtual ~Fruit();
};
```

```
...
Fruit* item1 = new Mango("Mango", "Alphonso");
...
delete item1; //calls Mango::~Mango() first and then Fruit::~Fruit()
```

Post-class Exercise - Inheritance

- The earlier quiz at the beginning of the class was a Pre-class Exercise.
- Re-attempt the same Quiz.