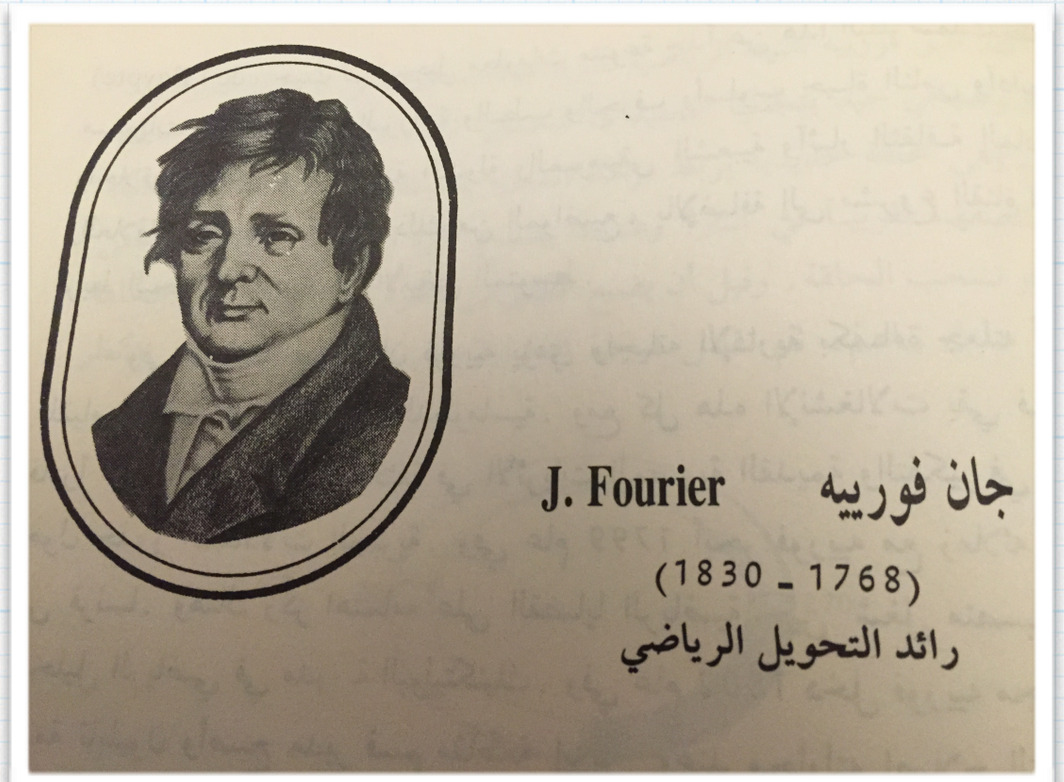
Fast Fourier Transforms (FFT)

Samar Aseeri, PhD
Computational Scientist
King Abdullah University of Science and Technology - Building 1 -Office: 0128
samar.aseeri@kaust.edu.sa

www.fft.report

"Fourier" the Scientist

* Mathematical Physicist.
* Father of Mathematical Transforms
* His major work, "The Analytic Theory of Heat", changed the way scientists think about functions and successfully stated the equations governing the heat transfer in solids.
* In 1807, he invented a technique to solve this equation: Fourier Transform.
* He applied this technique to explain many heat transfer problems.



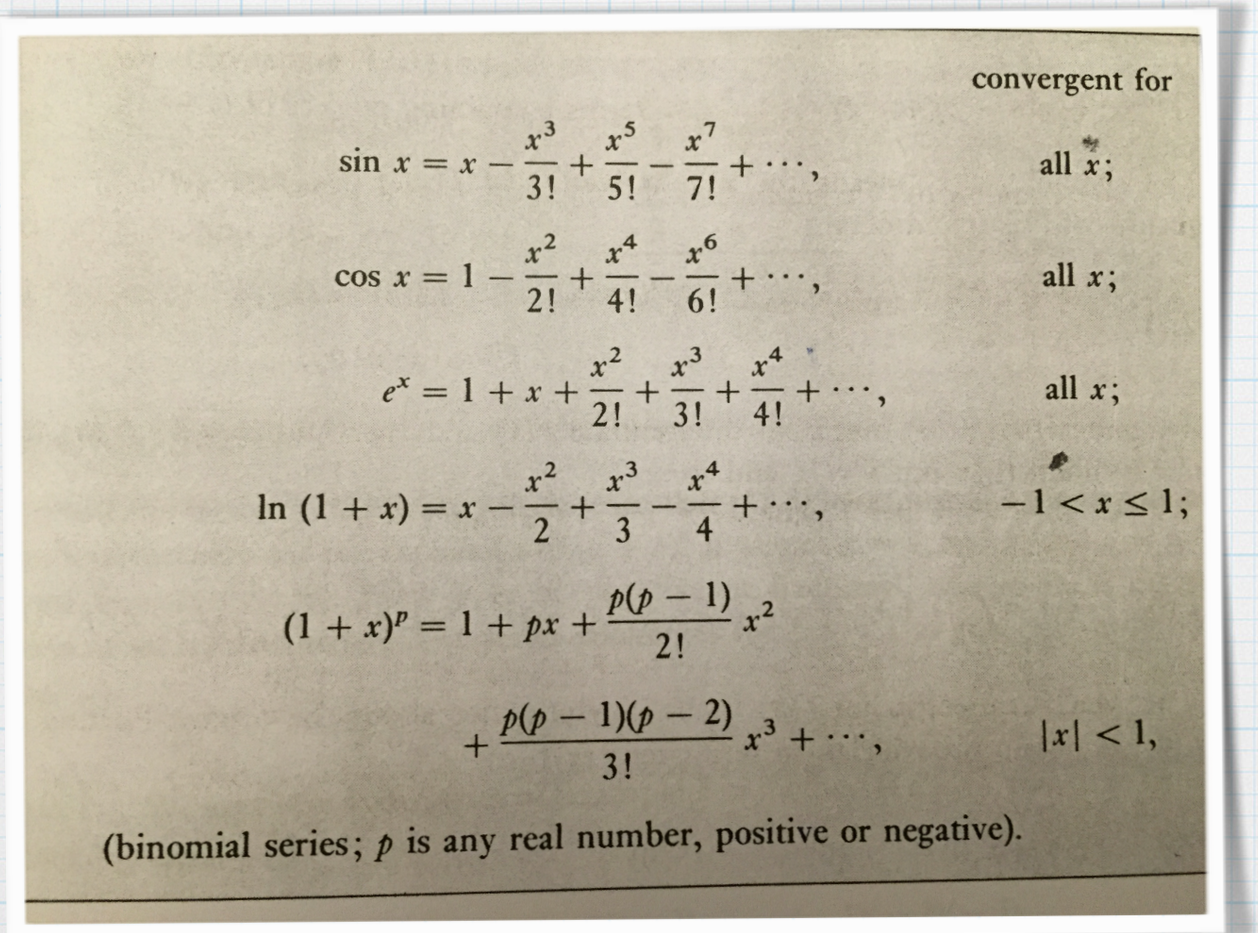
* Prior to Fourier's work, no solution to the heat equation was known in the general case.
* Fourier series is a way to represent any periodic function as an infinite sum of sines and cosines.

Power Series

- We use power series to approximate complicated functions.
- A power series is a series whose terms are functions of x.
- This series usually arises as the Taylor series of some known function.

$$f(x) = \sum_{n=0}^{\infty} \frac{f^n(0)}{n!} x^n$$

- These are called power series, because the terms are multiples of power x. Examples;



convergent for

$$\sin x = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots, \qquad \text{all } x;$$

$$\cos x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \cdots, \qquad \text{all } x;$$

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots, \qquad \text{all } x;$$

$$\ln(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \cdots, \qquad -1 < x \leq 1;$$

$$(1 + x)^p = 1 + px + \frac{p(p-1)}{2!} x^2 + \frac{p(p-1)(p-2)}{3!} x^3 + \cdots, \qquad |x| < 1,$$

(binomial series; p is any real number, positive or negative).

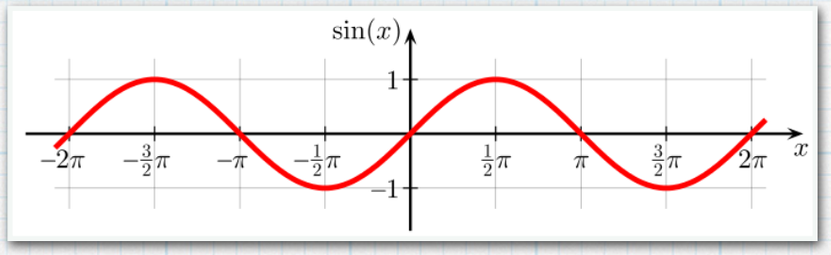
- Other power series to consider:
  - Fourier series whose terms involve sines and cosines. Also, Legendre, Bessel, etc...
  - Legendre and Bessel in which the terms may be polynomials of the functions.

Fourier Series

* Problems involving vibrations or oscillations occur frequently in physics and engineering.
* Examples like vibrating tuning fork, a pendulum and water waves.
* Other examples as the heat conduction, electric and magnetic fields and light does not appear to have anything oscillatory but turns out to involve the sines and cosines which are used in describing both simple harmonic motion and wave motion.
* In many problems, series called Fourier series, whose terms are sines and cosines, are more useful than power series.

Fourier Series

* A Periodic function is a function that repeats its values in regular intervals or periods.



* Fourier series for a periodic function $f(x)$ of period 2π

$$f(x) = \tfrac{1}{2}a_0 + a_1 \cos x + a_2 \cos 2x + a_3 \cos 3x + \cdots + b_1 \sin x + b_2 \sin 2x + b_3 \sin 3x + \cdots,$$

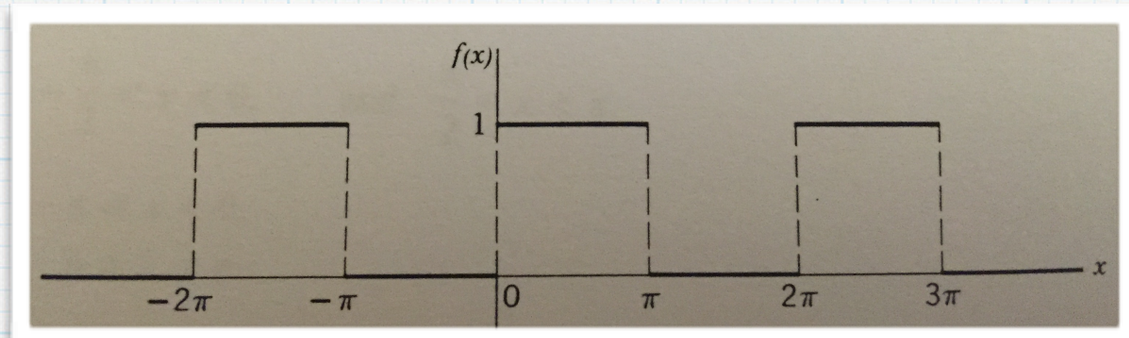
* Coefficients

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x) \cos nx \, dx.$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x) \sin nx \, dx.$$

Fourier Series

* Example: Expand in a Fourier series the sketched function $f(x)$



* Note $f(x)$ is a function of period 2π. In this problem, instead of a sketch, you might have been given

$$f(x) = \begin{cases} 0, & -\pi < x < 0, \\ 1, & 0 < x < \pi. \end{cases}$$

Fourier Series

* To find a_n we use;

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\cos nx\,dx = \frac{1}{\pi}\left[\int_{-\pi}^{0} 0\cdot\cos nx\,dx + \int_{0}^{\pi} 1\cdot\cos nx\,dx\right]$$

$$= \frac{1}{\pi}\int_{0}^{\pi}\cos nx\,dx = \begin{cases} \frac{1}{\pi}\cdot\frac{1}{n}\sin nx\Big|_{0}^{\pi} = 0 & \text{for } n \neq 0, \\ \frac{1}{\pi}\cdot\pi = 1 & \text{for } n = 0. \end{cases}$$

Thus $a_0 = 1$, and all other $a_n = 0$.

* To find b_n we use

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\sin nx\,dx = \frac{1}{\pi}\left[\int_{-\pi}^{0} 0\cdot\sin nx\,dx + \int_{0}^{\pi} 1\cdot\sin nx\,dx\right]$$

$$= \frac{1}{\pi}\int_{0}^{\pi}\sin nx\,dx = \frac{1}{\pi}\left[\frac{-\cos nx}{n}\right]_{0}^{\pi} = -\frac{1}{n\pi}\left[(-1)^n - 1\right]$$

$$= \begin{cases} 0 & \text{for even } n, \\ \frac{2}{n\pi} & \text{for odd } n. \end{cases}$$

* Putting these values for the coefficients into Fourier Series formula, we get

$$f(x) = \frac{1}{2} + \frac{2}{\pi}\left(\frac{\sin x}{1} + \frac{\sin 3x}{3} + \frac{\sin 5x}{5} + \cdots\right).$$

Fourier Series

* Dirichlet Conditions:

If $f(x)$ is periodic of period 2π, and if between $-\pi$ and π it is single-valued, has a finite number of maximum and minimum values, and a finite number of discontinuities, and if $\int_{-\pi}^{\pi} |f(x)|\,dx$ is finite, then the Fourier series (5.1) [with coefficients given by (5.9) and (5.10)] converges to $f(x)$ at all the points where $f(x)$ is continuous; at jumps the Fourier series converges to the midpoint of the jump. (This includes jumps that occur at $\pm\pi$ for the periodic function.)

* Complex Form of Fourier Series:

$$f(x) = c_0 + c_1 e^{ix} + c_{-1} e^{-ix} + c_2 e^{2ix} + c_{-2} e^{-2ix} + \cdots = \sum_{n=-\infty}^{n=+\infty} c_n e^{inx}$$

$$c_n = \frac{1}{2\pi} \int_{-\pi}^{\pi} f(x) e^{-inx}\,dx.$$

Integral Transforms

* If $f(t) = e^{-t}$, then the integral

$$\int_0^\infty f(t)t^p\,dt = \int_0^\infty t^p e^{-t}\,dt = F(p)$$

Is a function of p. Starting with a function of t, we have multiplied it by a function of p and t. The function $F(p)$ is called an integral transform of $f(t)$.

* Integral transforms are used in a variety of applications, for example, in solving ODE or PDE.
* There are many different kinds of integral transforms with different names, depending on what function of p and t we multiply by and what the range of integration is (the above example is called a Mellin transform).
* Laplace and Fourier transforms are the most widely used of all integral transforms.
* Laplace is considered a generalised Fourier transform.

Fourier Transforms

* This came as an answer to the question: is it possible to represent a function which is not periodic by something analogous to Fourier series?
* If you recall that an integral is a sum, it may not surprise you to learn that the Fourier series that is a sum of terms is replaced by a Fourier integral to cover non periodic cases.
* If we go back to the complex Fourier series formulas:

$$f(x) = \sum_{-\infty}^{\infty} c_n e^{in\pi x/l},$$

$$c_n = \frac{1}{2l} \int_{-l}^{l} f(x) e^{-in\pi x/l}\, dx.$$

and use substitutions to consider the case of a continuous range of frequencies. We get the corresponding following formulas for the Fourier transforms;

$$f(x) = \int_{-\infty}^{\infty} g(\alpha) e^{i\alpha x}\, d\alpha,$$

$$g(\alpha) = \frac{1}{2\pi} \int_{-\infty}^{\infty} f(x) e^{-i\alpha x}\, dx.$$

Fourier Transforms

* To represent odd functions, we use the Fourier sine transforms;

$$f_s(x) = \sqrt{\frac{2}{\pi}} \int_0^\infty g_s(\alpha) \sin \alpha x \, d\alpha,$$

$$g_s(\alpha) = \sqrt{\frac{2}{\pi}} \int_0^\infty f_s(x) \sin \alpha x \, dx.$$

* To represent even functions, we use the Fourier cosine transforms;

$$f_c(x) = \sqrt{\frac{2}{\pi}} \int_0^\infty g_c(\alpha) \cos \alpha x \, d\alpha,$$

$$g_c(\alpha) = \sqrt{\frac{2}{\pi}} \int_0^\infty f_c(x) \cos \alpha x \, dx.$$

Fourier Transforms

* Separation of variable (also known as the Fourier method) is a technique for solving PDE equations, in which algebra allows one to rewrite an equation so that each two variables occurs on a different side of the equation.

$$\begin{aligned}
\frac{dy}{dt} &= y \\
\frac{dy}{y} &= dt \\
\int \frac{dy}{y} &= \int dt \\
\ln y + a &= t + b \\
e^{\ln y + a} &= e^{t+b} \\
e^{\ln y}\, e^{a} &= e^{t} e^{b} \\
y &= \frac{e^{b}}{e^{a}} e^{t} \\
y(t) &= c e^{t}.
\end{aligned}$$

* If $u_t = u_{xx}$, suppose $U = X(x)\,T(t)$

$$\frac{\frac{dT}{dt}(t)}{T(t)} = \frac{\frac{d^2X}{dx^2}(x)}{X(x)} = -C,$$

* Solving each side separately with using linearity gives the solution;

$$U = \sum_n \alpha_n \exp(-C_n t)\sin(\sqrt{C_n}x) + \beta_n \exp(-C_n t)\cos(\sqrt{C_n}x)$$

Fourier Transforms

* Example: (Heat Equation)

$$k\frac{\partial^2 u}{\partial x^2} = \frac{\partial u}{\partial t}, \quad k > 0, \quad 0 < x < L, \quad t > 0,$$

$$u(0, t) = 0, \quad u(L, t) = 0, \quad t > 0,$$

$$u(x, 0) = f(x), \quad 0 < x < L.$$

$$\frac{X''}{X} = \frac{T'}{kT} = -\lambda^2$$

* Using the product U= X T and λ^2as a separation constant leads to

* Solution:

$$X = c_1 \sin\frac{n\pi}{L}x, \qquad n = 1, 2, 3, \ldots$$

$$T = c_3 e^{-k\lambda^2 t} = c_3 e^{-k(n^2\pi^2/L^2)t},$$

$$u_n = XT = A_n e^{-k(n^2\pi^2/L^2)t} \sin\frac{n\pi}{L}x$$

* Substituting t=0 gives the half-range expression of Fourier series. Therefore,

$$A_n = \frac{2}{L}\int_0^L f(x) \sin\frac{n\pi}{L}x\, dx.$$

$$\longrightarrow \quad u(x, t) = \frac{2}{L}\sum_{n=1}^{\infty}\left(\int_0^L f(x) \sin\frac{n\pi}{L}x\, dx\right) e^{-k(n^2\pi^2/L^2)t} \sin\frac{n\pi}{L}x.$$

Fourier Transforms

* In such problems we use separation of variables method (Fourier method) to solve the problem.
* When the # of derivatives is more than the given boundary conditions it is not possible to use the separation of variable methods. In this case Fourier and Laplace transforms are the way to go.
* When the domain is from -∞ -> ∞ we think of using the Fourier transform to solve the problem.
* If we have the following problem;

$$\frac{\partial^2 v}{\partial x^2} = \frac{\partial^2 v}{\partial t^2}, x = \pm\infty$$

$$v(x,0) = f(x), \quad \frac{\partial v(x,0)}{\partial t} = g(x)$$

and then define the complex Fourier as

$$\int_{-\infty}^{\infty} v(x,t)\, e^{ipt} dx = \bar{v}(p,t)$$

then the substitution in the problem equation will lead to the following solution;

$$v(x,t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} e^{ipt}\left[\bar{v}(p,0)\cos cpt + \frac{1}{cp}\frac{\partial \bar{v}(p,0)}{\partial t} sine\ cpt\right] dp$$

Fast Fourier Transform (FFT)

* In many application context the Fourier transform is approximated with a Discrete Fourier Transform (DFT).

* Back to the general solution of the first heat equation solved with Fourier method;

$$U = \sum_n \alpha_n \exp(-C_n t)\sin(\sqrt{C_n}x) + \beta_n \exp(-C_n t)\cos(\sqrt{C_n}x) = \sum_n \Upsilon_n exp(-C_n t)e^{i\sqrt{C_n}x}$$

* The Fast Fourier Transform allows one to find good approximations of the coefficients α_n, β_n when the solution is found at a finite number of equally spaced grid points.

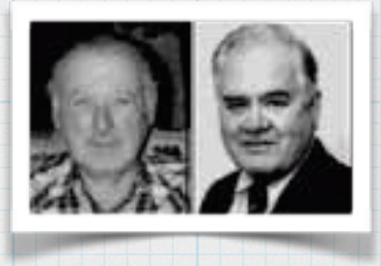
The Fast Fourier Transform (FFT) Algorithm

* Numerically efficient method to calculate DFT.

$$X(k)=\sum_{n=0}^{N-1} x(n)\, e^{\frac{-i2\pi nk}{N}}, k=0\ldots N-1$$

* It was originally developed by Gauss in 1805 but not recognised until more modern times.

* In 1965 Cooley+ Tukey published a paper on the Fast Fourier Transform and the efficient way to do it and of course the time was right because the use of computer was growing and there was a need for faster and faster data analysis.



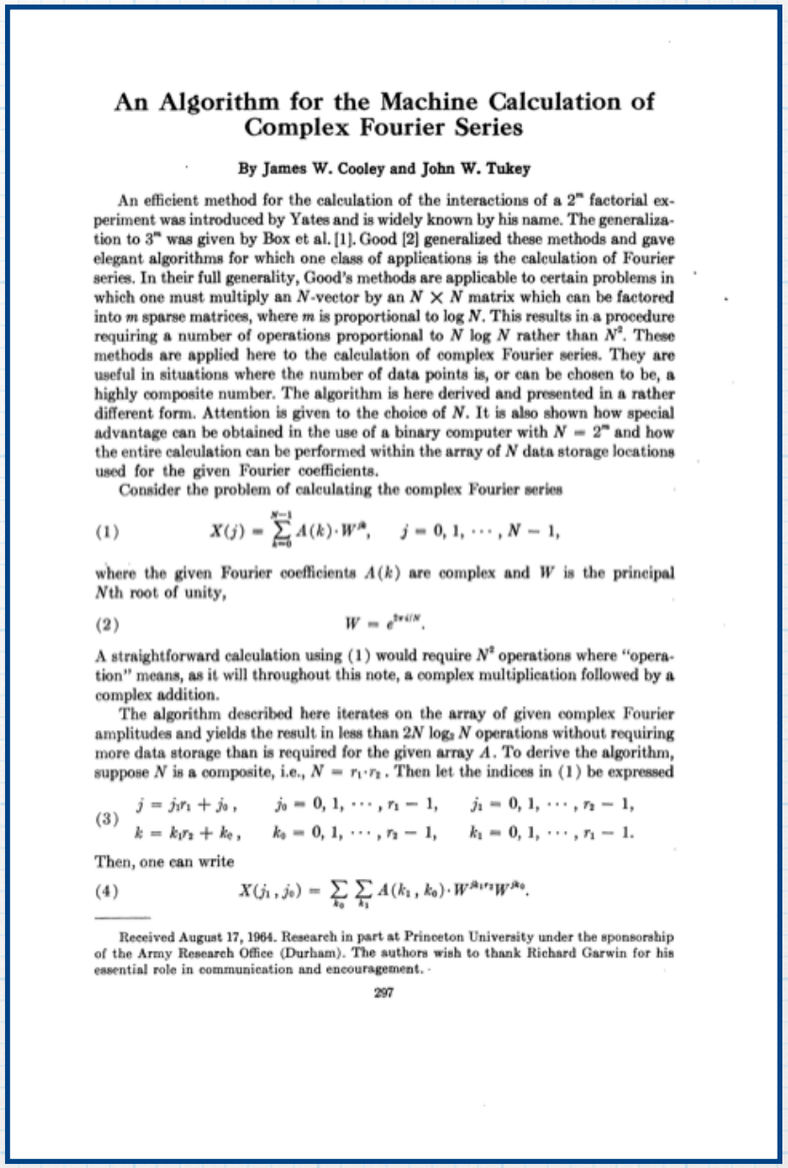
The Fast Fourier Transform (FFT) Algorithm

* Why do we need FFT if we have DFT? The reason is that DFT is computationally expensive.

* The Cooley-Tukey FFT reduced the DFT computations from $O(N^2)$ to $O(N \log_2 N)$.

* The difference shows when N gets large (see table)

N	1000	10^6	10^9
N^2	10^6	10^{12}	10^{18}
$N \log_2 N$	10^4	$20*10^6$	$30*10^9$



An Algorithm for the Machine Calculation of Complex Fourier Series

By James W. Cooley and John W. Tukey

An efficient method for the calculation of the interactions of a 2^m factorial experiment was introduced by Yates and is widely known by his name. The generalization to 3^m was given by Box et al. [1]. Good [2] generalized these methods and gave elegant algorithms for which one class of applications is the calculation of Fourier series. In their full generality, Good's methods are applicable to certain problems in which one must multiply an N-vector by an $N \times N$ matrix which can be factored into m sparse matrices, where m is proportional to $\log N$. This results in a procedure requiring a number of operations proportional to $N \log N$ rather than N^2. These methods are applied here to the calculation of complex Fourier series. They are useful in situations where the number of data points is, or can be chosen to be, a highly composite number. The algorithm is here derived and presented in a rather different form. Attention is given to the choice of N. It is also shown how special advantage can be obtained in the use of a binary computer with $N = 2^m$ and how the entire calculation can be performed within the array of N data storage locations used for the given Fourier coefficients.

Consider the problem of calculating the complex Fourier series

$$X(j) = \sum_{k=0}^{N-1} A(k) \cdot W^{jk}, \qquad j = 0, 1, \cdots, N-1, \tag{1}$$

where the given Fourier coefficients $A(k)$ are complex and W is the principal Nth root of unity,

$$W = e^{2\pi i/N}. \tag{2}$$

A straightforward calculation using (1) would require N^2 operations where "operation" means, as it will throughout this note, a complex multiplication followed by a complex addition.

The algorithm described here iterates on the array of given complex Fourier amplitudes and yields the result in less than $2N \log_2 N$ operations without requiring more data storage than is required for the given array A. To derive the algorithm, suppose N is a composite, i.e., $N = r_1 \cdot r_2$. Then let the indices in (1) be expressed

$$\begin{aligned} j &= j_1 r_1 + j_0, \qquad j_0 = 0, 1, \cdots, r_1 - 1, \qquad j_1 = 0, 1, \cdots, r_2 - 1, \\ k &= k_1 r_2 + k_0, \qquad k_0 = 0, 1, \cdots, r_2 - 1, \qquad k_1 = 0, 1, \cdots, r_1 - 1. \end{aligned} \tag{3}$$

Then, one can write

$$X(j_1, j_0) = \sum_{k_0} \sum_{k_1} A(k_1, k_0) \cdot W^{j_0 k_1 r_2} W^{j_0 k_0}. \tag{4}$$

Received August 17, 1964. Research in part at Princeton University under the sponsorship of the Army Research Office (Durham). The authors wish to thank Richard Garwin for his essential role in communication and encouragement.

Sequential Algorithm

* We will begin reminding our self with the DFT

$$X(k)=\sum_{n=0}^{N-1} x(n)\, e^{\frac{-i2\pi nk}{N}}, k=0 \dots N-1$$

for each k : N complex multiplications + N complex adds = N*N

* A more efficient way of calculating the DFT leads us to FFT.

* What is the Trick/Idea behind the Fast Fourier Transform?

* The idea is to divide the sequence X(n) into odd and even sequences.

X(2r) even sequence X(2r+1) odd sequence

If we put: $W_N = e^{-j(\frac{2\pi}{N})}$, we get

entries of Fourier matrix

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{nk} = \sum_{n=0}^{N/2-1} x(2r) W_N^{2rk} + \sum_{n=0}^{N/2-1} x(2r+1) W_N^{(2r+1)k}$$

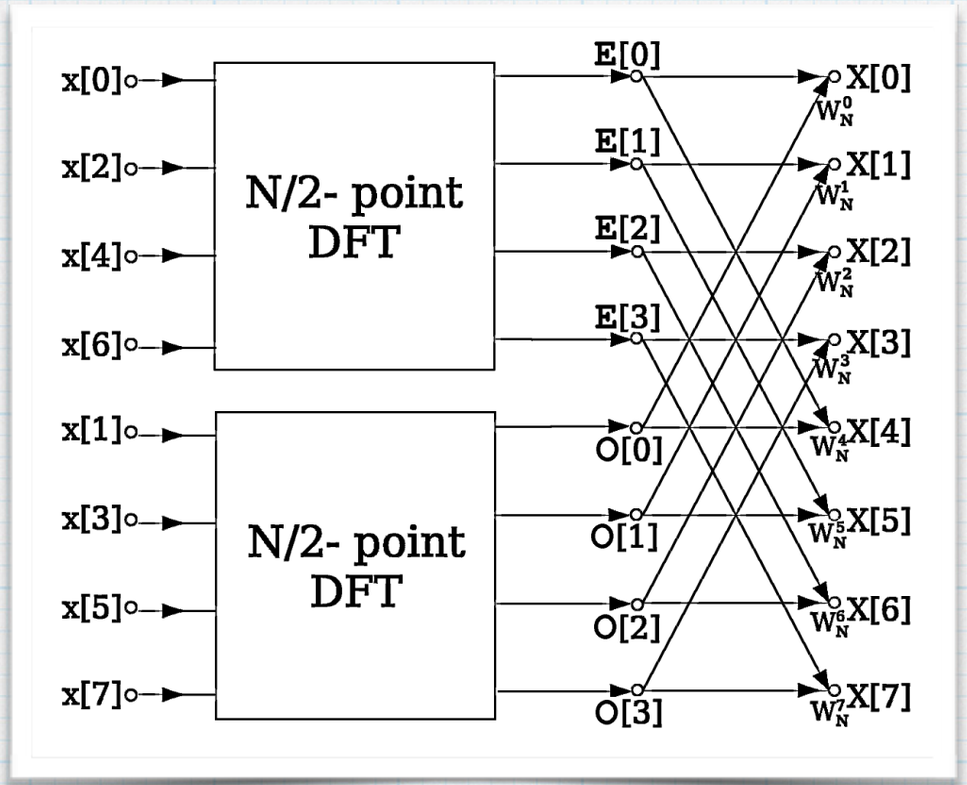
$$= \sum_{n=0}^{N/2-1} x(2r)(W_N^2)^{kr} + W_N^k \sum_{n=0}^{N/2-1} x(2r+1)(W_N^2)^{kr}$$

Sequential Algorithm

* $X(k)$ = DFT even $N/2$ samples + DFT odd $N/2$ samples
* Total operations: $\left(\frac{N}{2}\right)^2 + \left(\frac{N}{2}\right)^2 = \frac{N^2}{4} + \frac{N^2}{4} = \frac{2N^2}{4} = \frac{N^2}{2}$ multiplications!
* Since: $W_N^2 = e^{-j(\frac{2\pi}{N})2} = e^{-j\frac{2\pi}{N/2}} = W_{N/2}$, therefore:

$$X(k) = \sum_{n=0}^{N/2-1} x(2r) W_{N/2}^{kr} + W_N^k \sum_{n=0}^{N/2-1} x(2r+1) W_{N/2}^{kr}$$

* Example: N=8



Total Multiplications:

$(N/2)^2 . 2 + N \approx N^2/2 + N$

To Compute back:

$x[0] = E[0] + W_N^0 O[0]$

$x[1] = E[0] - W_N^0 O[0]$

Sequential Algorithm

- We started with N^2 so if did the full FFT we ruffle cut things by a factor of 2 using this approach.
- This splitting saved us some computation so why not continuing this process.
- The method consists of organising the problem so that the number of data points N being used can be easily factored, particularly into powers of 2.
- Keep splitting: each $N/2$ p -> 2 $N/4$ p
- How many times? $N/2$, $N/4$, ..., $N/2^{p-1}$, $N/2^p = 1$

Sequential Algorithm

What is the total # of multiplications in this FFT?

If $M(N) = N^2$ -> DFT

Then $M(N) = 2\,M(N/2) + N$ -> FFT

Substituting M(N) over and over again to observe a closed form:

> $M(N) = 2(2M(N/4) + N/2) + N$

$= 4M(N/4) + 2N$

> $M(N) = 4(M(N/8) + N/4) + 2N$

$= 8\,M(N/4) + N + 2N$

$= 8M(N/8) + 3N$

:

> $M(N) = 2^k M(N/2^k) + k\,N$

For trivial DFT $M(1) = 0$ then, if $N/2^k = 1 \Rightarrow N = 2^k \Rightarrow \log_2 N = k$

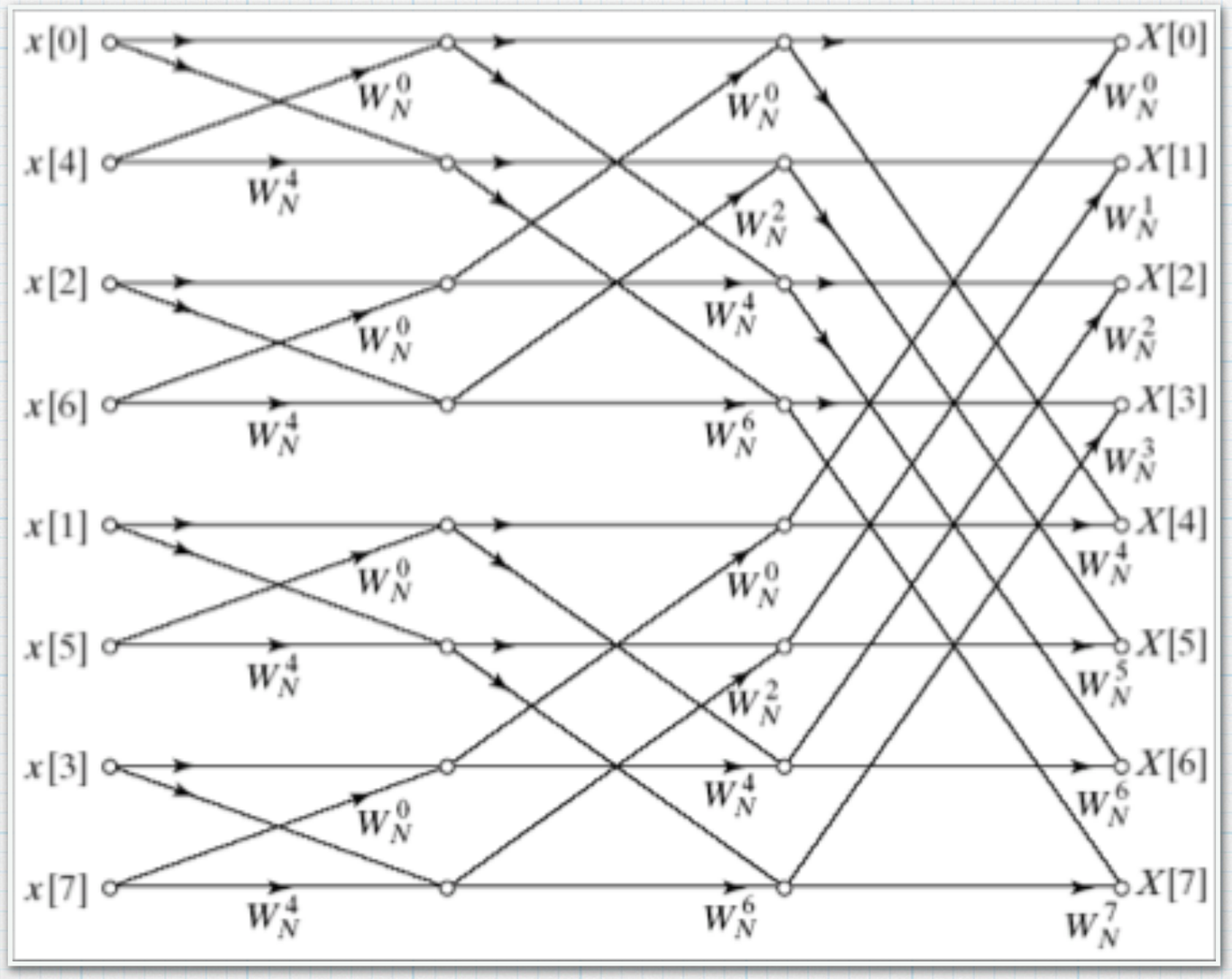
> $M(N) = N \log_2 N$

Sequential Algorithm

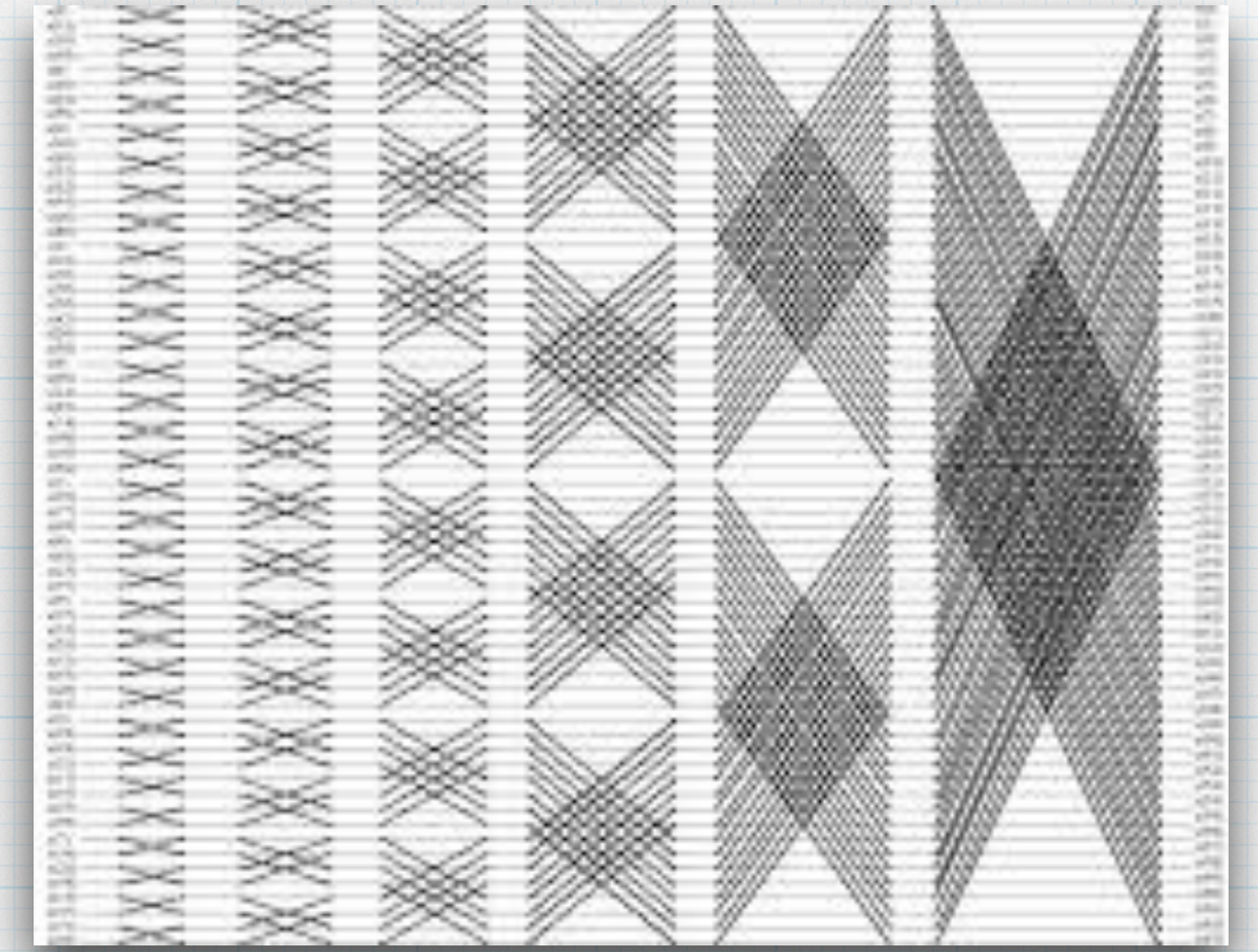
```
procedure  fft(x, X, n, w)
If n=1 then
  X[0] = x[0]
else
  for k=0 to (n/2)-1
    p[k] = x[2k]
    s[k] = x[2k+1]
  end
  fft(p, q, n/2 , w^2)
  fft(s, t, n/2 , w^2)
  for k = 0 to n-1
      X[k] = q[k mod (n/2)] + w^2t[k mod (n/2)]
  end
end
```

Sequential Algorithm

n=8



Algoritm Mapping

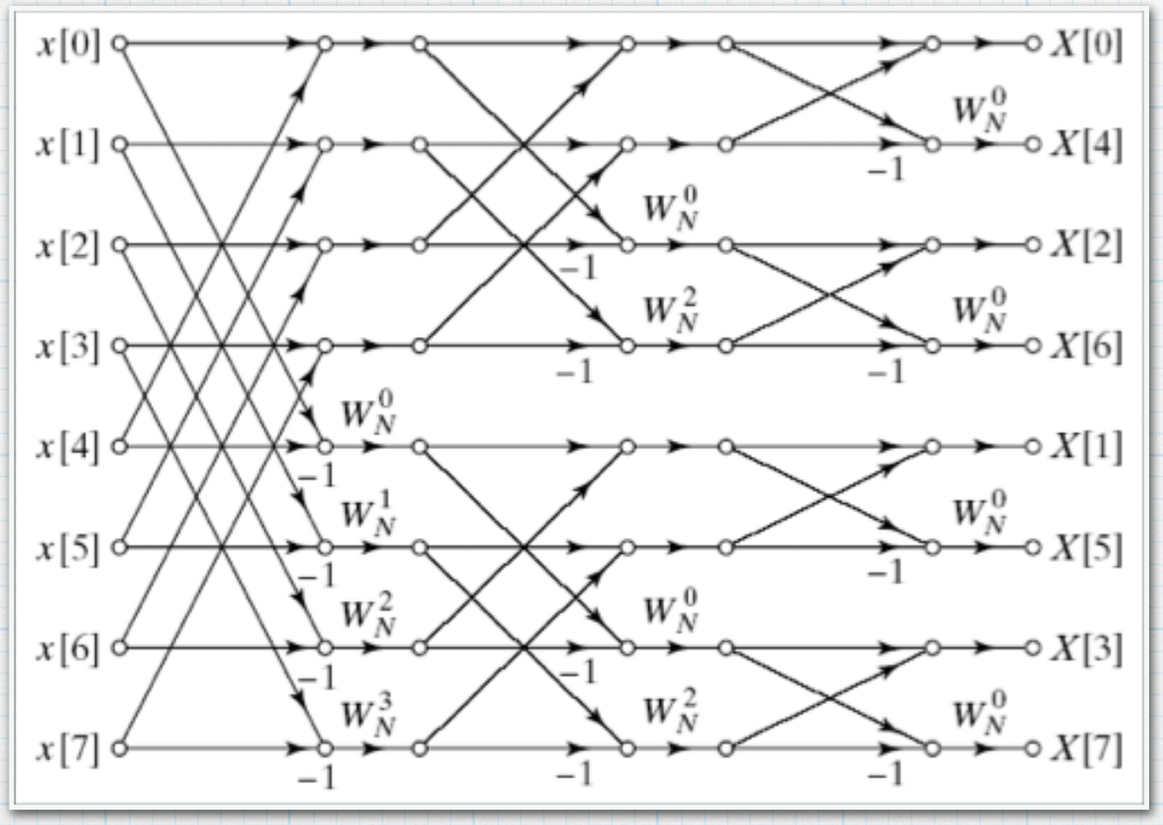


Stage= 1 2 3 4 5 6

- A Full FFT will be done in stages.
- Number of stages depends on the number of points N which a power of 2
- For the case $N=8=2^3$ the number of stages is 3.
- This mapping of the FFT algorithm is called the Butterfly diagram.

Iterative Sequential Algorithm

n=8



- We take the elements in pairs, compute the DFT of each pair, using one butterfly operation, and replace the pair with its DFT
- We take these n/2 DFT's in pairs and compute the DFT of the four vector elements
:
- We take 2 (n/2)-element DFT's and combine them using n/2 butterfly operations into the final n-element DFT

Iterative Sequential Algorithm

```
1.  procedure ITERATIVE_FFT(x, X, n)
2.  begin
3.    r := log n;
4.    for i:= 0 to n-1 do R[i] := x[i];
5.    for m:= 0 to r-1 do
6.      begin
7.       for i:= 0 to n-1 do S[i] := R[i];
8.       for i:= 0 to n-1 do
9.         begin
        /* Let (b0, b1, b2, ... br-1) be the binary representation of i */
10.           j := (b0 ... bm-1 0bm+1 .. br-1);
11.           k := (b0 ... bm-1 1 bm+1 .. br-1);
12.           R[i] := S[j] + S[k] x w^(bm bm-1..b0 0..0) ;
13.      endfor;
14.   endfor;
15.   for i:= 0 to n-1 do X[i] := R[i];
16. end ITERATIVE_FFT
```

Other FFT Algorithms

* Any algorithm that reduces operations for the DFT is called FFT. Therefore, the FFT is not just one algorithm and the simplest one was discovered by Cooley+ Tukey and it is the serial algorithm where we divide the even and odd terms to reduce the number of operations.
* Another class of FFTs subdivides the initial data set of length N not all the way down to the trivial transform of length 1
* There are also FFT algorithms for data sets of length N not a power of two.
* MPI Parallel FFT algorithm where all butterflies in a stage can be performed in parallel and then at the end of the stage, the results can be gathered. (not very efficient and creates a lot of overheads)
* 2D and 3D FFT algorithms

Parallel FFT Algorithms

There are two approaches for parallelizing:

1. Binary Exchange Algorithms: Where tasks exchange data at each stage of the serial algorithm.
2. Transpose algorithms: Where data are transposed using all-to-all personalised collective communication if the array is partitioned by columns each raw of data array is now stored in single task.

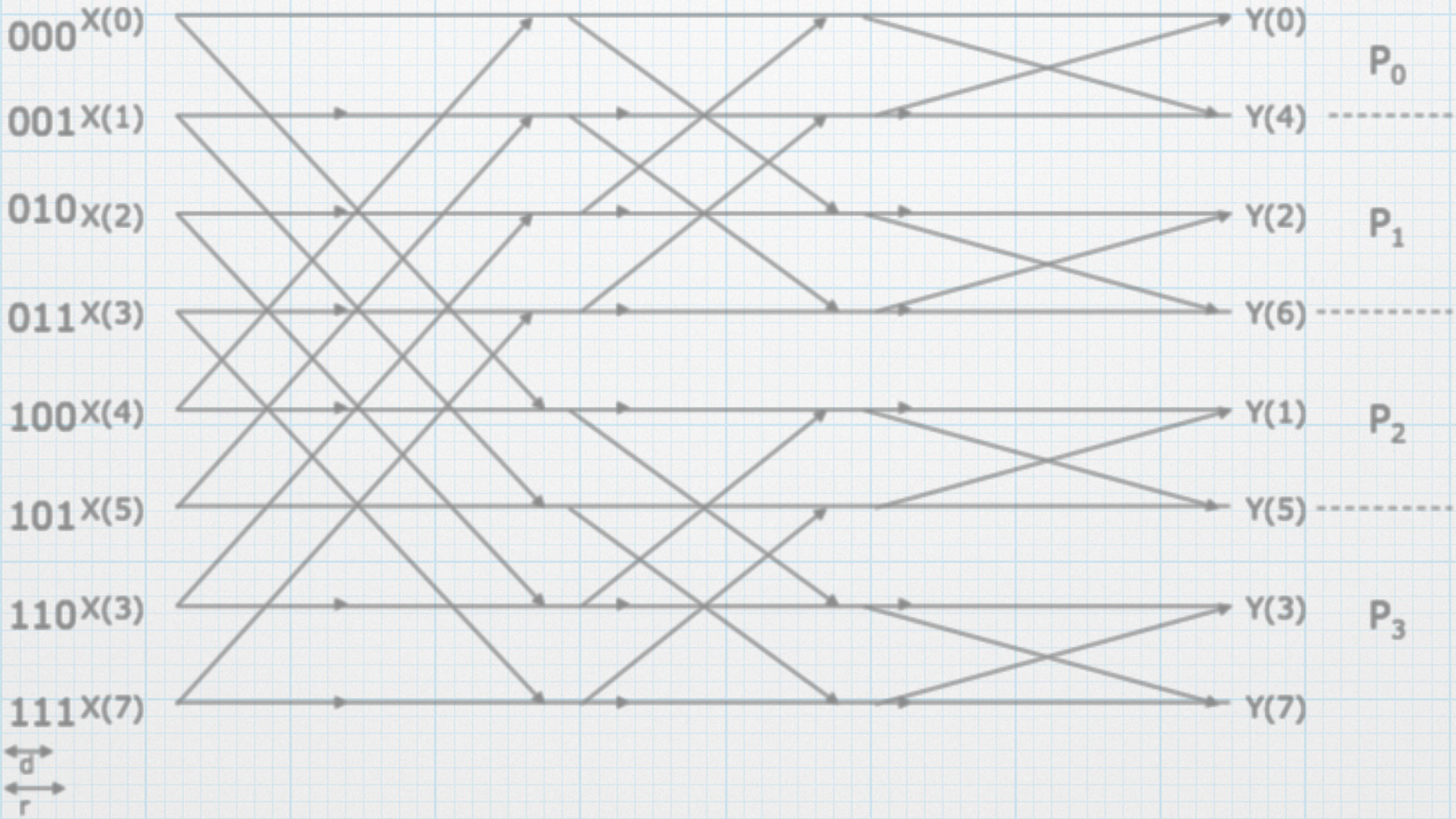
Binary Exchange FFT

* Data exchange takes place between all pairs of processors that differ by one bit.

* One element per processor is Easy.

* For Multiple elements per processor we Assign contiguous blocks to processors and we get same algorithm, just exchange blocks.

Binary Exchange FFT



Binary Exchange FFT

- d – number of bits for representing processes; r – number of bits representing the elements
- The d most significant bits of element i indicate the process that the element belongs to.
- Only the first d of the r iterations require communication
- In a given iteration, m, a process i communicates with only one other process obtained by flipping the (m+1)th most significant bit of i
- Total execution time - ? (n/P) long n + log P + (n/P) log P

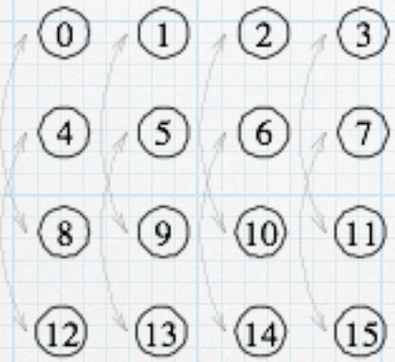
Binary Exchange FFT

* Big bandwidth requirement: Communication increases as n increases.
* Duplicated computations: Powers of ω cannot be pre-calculated and it is used at different times on different processors

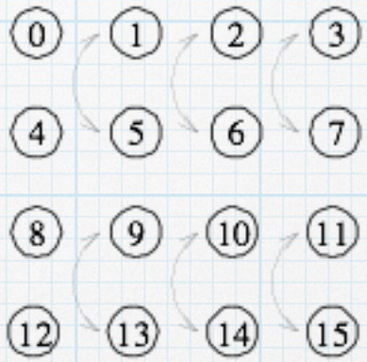
The Transpose FFT

* The data is arranged in a sqrt(n) x sqrt(n) two-dimensional square array
* Rather than do an exchange transpose the matrix halfway through algorithm

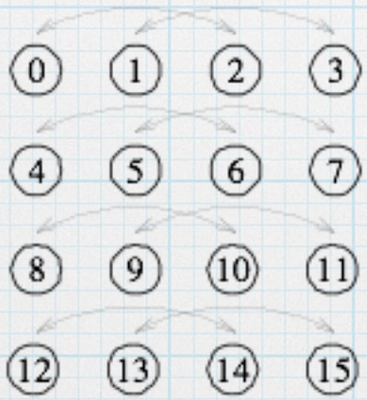
The Transpose FFT



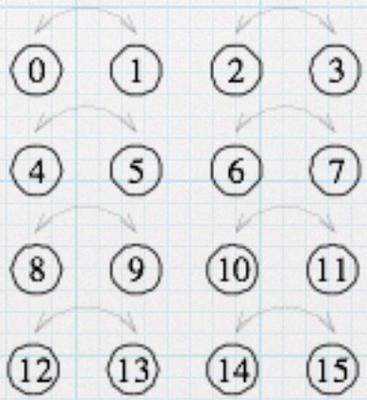
(a) Iteration m = 0



(b) Iteration m = 1



(c) Iteration m = 2



(d) Iteration m = 3

- Notice:
- First two iterations are columnwise
- Last two iterations are rowwise

The Transpose FFT

* p processes arranged along columns. Each process owns sqrt(n/p) columns.

* Each process does sqrt(n/p) FFTs of size sqrt(n) each.

* Parallel runtime - 2(sqrt(n)/p)* sqrt(n)*log(sqrt(n)) + (p-1)+ n/p(b)

Which Algorithm is Better?

* Binary exchange - small latency, large bandwidth
* 2D transpose - large latency, small bandwidth

* Transpose algorithm is easy to generalize to higher dimensions
* It depends on the architecture and amount of data

Fourier Techniques & Applications

* One of the major computational methods that uses FFTs is the so-called Spectral Methods.
* Spectral Methods are just one of the many ways to represent a function on a computer.
* Fourier series are particularly suited for the discretization of periodic domain.
* An efficient way to compute this is via fast Fourier transform (FFT) for the following reasons:
  * the FFT major relation;

$$\widehat{f^{(n)}} = (ik)^n \hat{f}$$

the Fourier transform of the n^{th} derivative.

  * the speed

$O(N^2) \rightarrow O(N \log N)$

Fourier Techniques & Applications

* The idea behind Fourier is the following Fourier transform represents functions in frequency space versus time domain or spatial domain.

$$F(k) = \int_{-\infty}^{\infty} e^{-ikx} f(x) dx \qquad f(x) = \frac{1}{2\pi} \int_{-\infty}^{\infty} e^{ikx} F(k) dk$$

* Unlike FDM where we chop off the domain into a certain number of points and to evaluate the solution we depend on neighbors so everything is local if you want to calculate a second derivative for instance each point cares only about its neighbors.

* For Spectral methods everything is represented in terms of Sines & Cosines which are global modes and lives on the whole domain and with this you get big accuracy gains.

* Spectral methods with Fourier bases limits you to a very small boundary condition set.

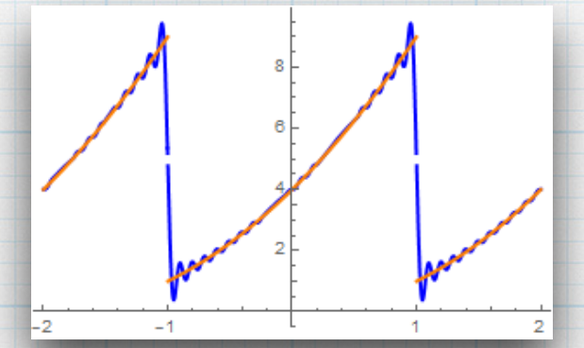
Fourier Techniques & Applications

* Spectral methods with Chebyshev basis allows a little bit more flexibility with how would you do with boundaries.

O(N log N)

* Accuracy & Speed makes one work hard to modify things to can still make use of these spectral routines.

* For non-periodic B.C. one of method to use is the Periodic Extensions for the function.



* Fourier said: I can take any function and represent it in terms of Sines and Cosines.

* Gibbs phenomenon is the behavior of Fourier series for periodic functions at discontinuity jumps.

Fourier Spectral Implementation

* Let's take a generic PDE $u_t = Lu + N(u)$, where L is some operator that has derivative terms as follows $L - a\partial^2 x + b\partial x + c$ and N is the rest (non linear terms & non constant coefficients).

* The Spectral technique is to fast Fourier transform everything; , where $\hat{u}_t = a(k)\hat{u} + \hat{N}$

ODE

$$\begin{aligned} Lu &= au_x x + bu_x + cu \\ &= -ak^2\hat{u} + ikb\hat{u} + c\hat{u} \\ &= (-ak^2 + ikb + c)\hat{u} \end{aligned}$$

* Now you are in the fast Fourier domain so you are not solving for time you are solving for the evolution in frequency domain.

Fourier Spectral Implementation

* Fast Fourier Transform in 2D

Fast Poisson Solver + Periodic BC

Fast Fourier Transform in x direction.

Fast Fourier Transform in y direction.

Inverse the Fast Fourier Transform to get solution

$$\nabla^2\psi = \omega$$

$$\psi_{xx} + \psi_{yy} = \omega$$

$$\widehat{\psi_{xx} + \psi_{yy}} = \hat{\omega}$$

$$\hat{\psi}_{xx} + \hat{\psi}_{yy} = \hat{\omega}$$

$$-k_x^2\hat{\psi} + \hat{\psi}_{yy} = \hat{\omega}$$

$$\widetilde{-k_x^2\hat{\psi} + \hat{\psi}_{yy}} = \tilde{\hat{\omega}}$$

$$-k_x^2\tilde{\hat{\psi}} - k_y^2\tilde{\hat{\psi}} = \tilde{\hat{\omega}}$$

$$\tilde{\hat{\psi}} = \frac{-\tilde{\hat{\omega}}}{k_x^2 + k_y^2}$$

Fourier Spectral Implementation

* There exists various implementation of the spectral method and the most common approach, namely is the **Galerkin** approach.

* There are many tools available for working with spectral methods like **chebfun** in Matlab and **shenfun** the Python module.

* The shenfun's purpose is to simplify the implementation of the spectral Gherkin method, to make is easily accessible to researchers, and to make it easier to solve the advanced PDEs on supercomputers. Found at (**github.com/spectralDNS/shenfun**).

* The Extreme Computing Research Center (ECRC) at KAUST, in collaboration with the University of Oslo, Norway, has developed a new efficient implementation of parallel FFT that is utilized by shenfun.

FFT Facts

* Operation cost: O(N log N)
* BC: Periodic
* Discretization: 2^n
* Accuracy: Beyond all algebraic orders

FFT Packages

* The existing FFT packages uses different algorithms.
* Practically speaking in any language you will be able to find an FFT package
* One of the most popular ones is written in C is called FFTW (the Fast Fourier Transform in the West) and most languages will just wrap over that and it is very fast and if you are going to use that all you need to know is that you have a function and you pass to the function your data and the number of bins you want to calculate and it spits out an array of the Fourier Transform where k the length of the array goes from zero up to the bins you chose. The output is two dimensions. We take the absolute value of the first half of the output array.
* The output doesn't correspond to physical numbers you have to plot it such that it is understandable.
* Can Change number of bins based on the speed of the computer your are using.
* Other available FFT packages: 2Decomp&FFT, AccFFT, P3FFT, PFFT, OpenFFT, CUFFT, FFT MKL. .

FFT Packages Common Features

- The 2D FFT is simply the 1D FFT applied first to each row and then to each column of an array
- Can compute:
  - For one or more dimensions.
  - For single and double precision where doubles store a much broader range of values, and with much more precision.
  - For real and complex data.
  - For even or odd terms i.e. the discrete Sine or Cosine Transforms.
  - In Parallel shared/distributed memory for parallel one- and multi-dimensional transforms of both real and complex data.

FFT Packages

ID	Package	Language	Description
1	FFTW	Written in C but can be used for C and Fortran codes	The fastest Fourier Transform in the West, is a popular open-source library for computing discrete Fourier transforms in one or more dimensions, of arbitrary input size, and of both real and complex data (as well as even/odd, i.e. the discrete cosine/sine transform or DCT/DST. It is written in C and supports both real and complex transformations in both single and double precision. Download: http://www.fftw.org/download.html
2	2deomp&fft	Written in Fortran and optimized C codes needs to be developed in order to use it.	The 2DECOMP&FFT library is a software framework in Fortran to build large-scale parallel applications. It is designed for applications using three-dimensional structured mesh and spatially implicit numerical algorithms. Download: http://www.2decomp.org/download.html
3	PFFT	Written in C and can be used for both C and Fortran codes	Presents fast Fourier transforms (FFTs) on massively parallel, distributed memory architectures based on the Message Passing Interface standard (MPI). This library offers great flexibility and portable performance since it proposes an algorithm that calculates pruned FFTs more efficiently on distributed memory architectures. Download: http://flash.uchicago.edu/~dubey/pfft/
4	P3DFFT		It is a library for large-scale computer simulations on parallel platforms. 3D FFT is an important algorithm for simulations in a wide range of fields, including studies of turbulence, climatology, astrophysics and material science. This project was initiated at San Diego Supercomputer Center (SDSC). Its approach had shown good scalability up to 524,288 cores. P3DFFT is written if Fortran90 and is optimized for parallel performance. It uses Message Passing Interface (MPI) for interprocessor communication, and starting from v.2.7.5 there is a multithreading option for hybrid MPI/OpenMP implementation. C interface is available,. The package depends on a serial FFT library such and FFTW or ESSL.
			Download: https://www.p3dfft.net/download
5	OpenFFT		OpenFFT is an open source parallel package for computing multi-dimensional Fast Fourier Transforms (3-D and 4-D FFTs) of both real and complex numbers of arbitrary input size. It adopts a communication-optimal domain decomposition method that is adaptive and capable of localizing data when transposing from one dimension to another for reducing the total volume of communication. It is written in C and MPI, with support for Fortran through the Fortran interface, and employs FFTW3 for computing 1-D FFTs. It is developed in Tokyo. Download: http://www.openmx-square.org/openfft/
6	AccFFT		AccFFT is a new massively parallel FFT library for CPU/GPU architectures. AccFFT is specifically designed with the goal of achieving maximum performance and scalability for both CPUs and GPUs. It uses a series of novel algorithms to reduce communication costs inherent in distributed FFTs. Download: https://github.com/amirgholami/accfft
7	FFTE		A package to compute Discrete Fourier Transforms of 1-, 2- and 3- dimensional sequences of length (2^p)*(3^q)*(5^r). Download: http://www.ffte.jp
8	FFT MKL		Intel has its own version of FFT code implemented as part of the Intel Math Kernel Library MKL, which is highly optimized for Intel architecture-based platforms.
9	cuFFT		The NVIDIA CUDA Fast Fourier Transform library. It provides a simple interface for computing FFTs up to 10x faster. By using hundreds of processor cores inside NVIDIA GPUs, cuFFT delivers the floating-point performance of a GPU without having to develop your own custom GPU FFT implementation. Download: https://developer.nvidia.com/cufft
10	Parallel FFT		http://www.sandia.gov/~sjplimp/docs/fft/README.html

FFTW packages

- FFTW adapt itself to your machines, your cache, the size of memory, the number of register, etc...
- FFTW doesn't use a fixed algorithm to calculate DFT. It choses the best algorithm for your machines.
- FFTW includes serial and parallel transforms for both shared and distributed memory system.
- FFTW supports both real and complex transforms as well as even/odd, i.e. the discrete cosine/sine transform or DCT/DST.
- FFTW supports both single and double precision and it compiles the double-precision libraries by default.
- FFTW Computation is split into two phases: Plan creation and Execution.
- Two major versions of FFTW are available: FFTW2 and FFTW3. These two versions are incompatible and their interfaces are different. FFTW2 is now considered obsolete and has not been updated since 1999.

FFTW3

* The fftw3 is used everywhere it is used in the background of matlab.
* The fftw3 supports Hybrid implementation MPI-OpenMP.
* www.fftw.org download from here. This library is also a standard component of linux so it is directly installed in a linux environment and you can use it and I'll just guide you through how this library is being used.

How to Use? Install

* In the webpage www.fftw.org can be found the source code of FFTW3. There it is explained how it can be installed but, in most Linux environments, the following works:

1. Download the source code fftw-X.X.X.tar.gz from ftp://ftp.fftw.org/pub/fftw/fftw-X.X.X.tar.gz

2. Decompress it:

   $ tar -xvvf fftw-X.X.X.tar.gz

3. Enter the directory fftw-X.X.X:

   $ cd fftw-X.X.X

4. Now, proceed to configure, compile and install the package:

   $ ./configure && make && make install

* If the above does not work, read carefully the documentation that is in www.fftw.org.

How to Use? Code Example

```
1   #include<fftw3.h>
    int main(void)
    {
2       int N;
3       fftw_complex *in, *out;
4       fftw_plan my_plan;

5       in = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N);
6       out = (fftw_complex*) fftw_malloc(sizeof(fftw_complex)*N);
7       my_plan = fftw_plan_dft_1d(N, in, out, FFTW_FORWARD, FFTW_ESTIMATE);

8       fftw_execute(my_plan);

10      fftw_destroy_plan(my_plan);
11      fftw_free(in);
12      fftw_free(out);

13      return 0;
    }
```

How to Use? Explain Code

* Line 1 includes the header file fftw3.h needed in order to use the package.
* Line 2 contains an integer N which has the dimension of the input and output data of the FFT
* Line 3 declares two pointers of type fftw_complex, in and out, which will contain the input and output of the FFT. Note that to allocate memory we use the function fftw_malloc instead of the sodlib.h function malloc,
* Line 4 declares a variable of type fftw_plan, a plan, which will store the type of FFT that we want to perform.
* Lines 5 and 6 allocates memory for the pointers in and out. Note that it must be specified that they are of type fftw_complex.

How to Use? Explain Code

* Line 7 declares the type of plan which we want to perform via the function fftw_plan_dft_1d which has the arguments

  1. int N: the dimension of the pointers in and out.
  2. fftw_complex *in: the pointer that stores the input data.
  3. fftw_complex *out: the pointer that stores the output data.
  4. int FFTW_FORWARD is an integer constant of the package that tells the function that the FFT to perform must be the forward one. To perform the backward one, we will introduce FFTW_BACKWARD.
  5. unsigned FFTW_ESTIMATE is a flag that tell to the function how well must be optimized, with respect to the computational time.

* Line 8 performs the FFT stored in my_plan.
* Lines 10, 11 and 12 deallocate the memory stored by the plan and the pointers. Note that for the pointers we use fftw_free and not the sodlib.h function free.

How to Use? Compile

* To compile this code with gcc we just type in the following in the command line

```
$ gcc Example.c -lfftw3 -c Example.exe
```

How to Use? On ShaheenII

System	Cray XC40 with 36 cabinets
Processor type	Intel Haswell 2.3GHz, 2 CPU sockets per node, 16 processors cores per CPU
Total Nodes	6174 nodes
Total Cores	197,568 cores
Memory	128 GB of memory per node, over 790 TB total memory
Interconnect	Cray Aries with Dragonfly topology
Scheduler	SLURM
Storage	Lustre parallel file system with 17.4 PB

How to Use? On ShaheenII

* Three programming environments are supported on ShaheenII as in the below table:

PrgEnv	Description	Real Compilers
PrgEnv-cray	Cray Compilation Environment	crayftn, craycc, crayCC
PrgEnv-intel	Intel Composer Suite	ifort, icc, icpc
PrgEnv-gnu	GNU Compiler Collection	gfortran, gcc, g++

* Use the compiler driver wrappers cc, CC, ftn to compile and link C, C++, and Fortran codes respectively.

* The wrappers are the same for all the programming environments.

* Refer to the ShaheenII Get Started Flyer for more information about usage. It can be found at https://www.hpc.kaust.edu.sa/sites/default/files/files/public/GetStartedFlyer.pdf

Flyer Page 1

جامعة الملك عبدالله للعلوم والتقنية
King Abdullah University of Science and Technology

SHAHEEN SUPERCOMPUTING LABORATORY

Shaheen II Get Started

Shaheen II Spec:

System	**Cray XC40 with 36 cabinets**
Processor type	Intel Haswell 2.3GHz, 2 CPU sockets per node, 16 processors cores per CPU
Total Nodes	6174 nodes
Total Cores	197,568 cores
Memory	128 GB of memory per node, over 790 TB total memory
Interconnect	Cray Aries with Dragonfly topology
Scheduler	SLURM
Storage	Lustre parallel file system with 17.4 PB

To login:

```
$ ssh <username>@shaheen.hpc.kaust.edu.sa
```

Login with "-X" or "-Y" to enable X11 forwarding.

To compile:

Three programming environments are supported. i.e. PrgEnv-cray (default), PrgEnv-intel, and PrgEnv-gnu. Use module swap to change PrgEnv, e.g.

```
$ module swap PrgEnv-cray PrgEnv-intel
```

Use the compiler driver wrappers cc, CC, ftn to compile and link C, C++, and Fortran codes, respectively. The wrappers are the same for all programming environments. For example

C	cc –c <any_other_flags> prog.c
C++	CC –c <any_other_flags> prog.cpp
Fortran	ftn –c <any_other_flags> prog.f90

Within a programming environment a user can switch between different compiler versions.

```
$ module swap cce cce/8.4.0
```

Scheduler and Queues

To run:

- SLURM is the batch scheduler. The following is a basic example of a batch script:

```
#!/bin/bash
#SBATCH --account=k##
#SBATCH --job-name=job_name
#SBATCH --output=job_name.out
#SBATCH --error=job_name.err
#SBATCH --nodes=4
#SBATCH --time=00:30:00

srun --ntasks=128 --hint=nomultithread --ntasks-per-node=32 --ntasks-per-socket=16 ./exe
```

- Hyperthreading is enabled by default and might improve the performance of some codes. Use "--hint=nomultithread" option in the execution line to disable it. For binding tasks to cores you can use "--cpu_bind=rank".
- Launch/Cancel jobs with:

```
$ sbatch job_script.sh
$ scancel job_id
```

Queues:

- Use "**sinfo**" for the queue status and "**squeue**" to observe your job status.
- **workq**: Default queue with a maximum 24 hours wall clock time.
- **72hours**: Queue with a maximum of 72 hours wall clock time. Mainly intended for pre/post processing and running one node job that cannot be check-pointed to finish within a stipulated 24 hours wall clock time. Add the following in your job script to use the 72 hours queue.

```
#!/bin/bash
#SBATCH --partition=72hours
#SBATCH --qos=72hours
```

Storage, Quotas, Allocations

To store:

- Storage Compute nodes can access only **/scratch** and **/project** directories. Jobs submitted from /home will fail.
- **/home/<username>**: Home directory, designed for development, quota of 200GB. Previous versions of files can be recovered from /home/<username>/.snapshot directory.
- **/scratch/<username>**: Temporary individual storage for data needed for execution. Files not accessed in the last 60 days will be deleted.
- **/scratch/project/k##**: Temporary storage for the project. Files not accessed in the last 60 days will be deleted.
- **/project/k##**: 20 TB per project. All files are copied to tape. Once a project has used 20 TB of disk storage, files will be automatically deleted from disk with a weighting based on date of last access. Stub files will remain on disk that link to the tape copy.
- **/scratch/tmp**: temporary folder that will be cleaned every 3 days.

To check:

- Your Group information, use the "**groups**" command
- Your allocation information use the "**sb**" command, e.g.

```
username@cdl3:~> sb k##
Project k##: MEMBER_INFO
PI: PI_NAME
Allocations    Core hours
-------------------------
2015-06-25      50000000
2014-09-02            50
-------------------------
Expiry on 2020-01-01
-------------------------
Allocated       50000050
```

Flyer Page II

Compiler Flags

- Following table displays some advanced flags

Feature	Cray	Intel	GNU
Recommended compiler optimization level	default (-O3)	default (-O2)	-O3 -ffast-math
Aggressive Optimization	-O3 -hfp3	-Ofast -fp-model fast=2	-Ofast -mavx -funroll-loops
Activate OpenMP directives and pragmas in the code	-homp (default)	-openmp	-fopenmp
Desactivate OpenMP	-hnoomp		
Read and write Fortran unformatted data files as big-endian	-h byteswapio	-convert big_endian	-fconvert=swap
Process Fortran source using fixed form specifications.	-f fixed	-fixed	-ffixed-form
Process Fortran source using free form specifications.	-f free	-free	-ffree-form
Show version number of the compiler.	-V	--version	--version
Zero fill all uninitialized variables	-h zero	not implemented	-finit-local-zero
Creates .mod files to hold Fortran90 module information for future compiles	-e m		
Specifies the directory to which *file*.mod files are written when the -e m option is specified	-J *dir_name*		
Listing compiler feedbacks , produces .lst files	-hlist=a	-opt-report3	-fdump-tree-all

- For more information on individual compilers

PrgEnv	C	C++	Fortran
PrgEnv-cray	man craycc	man crayCC	man crayftn
PrgEnv-intel	man icc	man icpc	man ifort
PrgEnv-gnu	man gcc	man g++	man gfortran

Software & Libraries

- Before requesting the installation of new packages or libraries, please check if the desired package is already installed on the system.
- To find the list of all the packages installed:

```
$ module avail
```

- To find a specific package:

```
$ module avail --long 2>&1 | grep xxxx
```

- To get information on the package usage:

```
$ module help xxxx
$ module show xxxx
```

- To display Cray Scientific Libraries execute the following line:

```
$ module avail -l
```

Here is a selection of libraries and applications already installed on Shaheen II:

- I/O Libraries
  - HDF5, NetCDF
- Numerical Libraries
  - LIBSCI, PETCS, FFTW, MKL, …
- Visualization Tools
  - Gnuplot, Paraview
- Debugging Tools
  - lgdb, atp, Allinea, Totalview, STAT
- Performance tools
  - Craypat, Allinea, PAPI
- Some Third party Software
  - VASP, CP2K, NAMD, LAMMPS, …

General Tips

- Currently, static linking is the default. To switch between different link types you can either set CRAYPE_LINK_TYPE to "static" or "dynamic" or pass the "-static" or "-dynamic" option to the linking wrapper (cc, CC or ftn).
- LIBSCI is the collection of numerical routines optimized for best performance on Cray systems. It gathers BLAS, LAPACK, SCALAPACK and is highly recommended to be used instead of your own versions.
- When calling libraries installed by Cray, such as LIBSCI, HDF5, NetCDF you do not need to add -I, -L and -l flags. Instead, you will have to remove these paths from your Makefiles.
- Default I/O striping is 1, optimal for many cases especially when every MPI process writes to its own file resulting in as many files as number of processes used.
- Increase the stripe count when multiple processes write to a single shared file as with MPI-IO and HDF5 or NetCDF. Use the following command with a maximum stripe count of 144:

```
$ lfs setstripe --count [stripe-count] filename/directory
```

To get an account:

- KAUST members should fill-in the Individual Access Application (IAA) and the Project Proposal (PP) forms. Forms are available at: http://hpc.kaust.edu.sa/account-applications

For more information:

- Please visit the user guide and training materials at: http://hpc.kaust.edu.sa/
- Please email any issues/forms to help@hpc.kaust.edu.sa so that the KSL staff can get back to you immediately.
- Follow us on Twitter: twitter.com/KAUST_HPC

Prepared By: S Aseeri(KSL), B. Hadri(KSL), A. Esposito(Cray), October 2015

How to Use? On ShaheenII

* Cray's main FFT library is FFTW from MIT with some additional optimizations for Cray hardware
* Usage is simple
  * Load the module
  * In the code, call the FFTW plan
* Cray's FFTW provides wisdom file for these system. You can use the wisdom file to skip the plan stage.
* In FFTW the wisdom mechanism is used for saving plans.
* When calling libraries installed by Cray, such as FFTW, LIBSCI, HDF5, NetCDF you do not need to add -I, -L and -l flags. Instead, you will have to remove these paths from your Makefiles.

How to Use? On ShaheenII

- $ ssh -X username@shaheen.hpc.kaust.edu.sa
- $ cd /scrach/username/FFTW
- wget http://www2.math.uu.se/~figueras/fftw_tutorial/examples/EXAMPLE2_transform.c
- cd EXAMPLE2_transform.c Example_2
- $ salloc (to start an interactive session)
- $ module avail -L (to lis all cary available libraries)
- $ module avail fftw (to list all fftw versions)
- $ module load fftw (to load the default version)
- $ cc Example_2.c -o Example_2 (to compile on cray)
- $ cc Example_2.c -Wall -lfftw3 -lm -o Example_2 (to compile on other platforms)
- $ srun Example_2 (to run the executable)

How to Use? On ShaheenII

* You can see all the used compile and link options using the wrapper option -craype-verbose
* $ cc -craype-verbose Example_2.c -o Example_2
* The default link type is static, on the login nodes as well as on the compute nodes. You can specify the link type using the -dynamic or -static compiler/linker option, e.g.:
  * $ cc -dynamic Example_2.c -o Example_2
* OR set the environment variable, e.g.:
  * $ CRAYPE_LINK_TYPE=dynamic

How to Use? On ShaheenII

* OpenMP is supported by all of the PrgEnvs.
* The CCE (PrgEnv-cray) recognizes and interprets OpenMP directives by default. If you have OpenMP directives in your application but do not want to use them, disable OpenMP recognition with **-hnoomp**.

PrgEnv	Enable OpenMP	Disable OpenMP
PrgEnv-cray	-homp	**-hnoomp**
PrgEnv-intel	**-openmp**	
PrgEnv-gnu	**-fopenmp**	

* Autothreading in NOT on by default;
  * **-hautothread** to turn on
  * Interacts with OMP directives

FFTW / Some Useful Instructions

- Including FFTW Lib:
  - For Serial Codes
    - C -> #include<fftw.h> & #include<fftwf.h> for single precision
    - Fortran -> include 'fftw3.f03' & include 'fftw3.ff03'
  - Fort Parallel Codes
    - C -> #include <fftw-mpi.h> & #include <fftwf-mpi.h> for single precision
    - Fortran -> include 'fftw3-mpi.f03' & include 'fftw3f-mpi.f03' // // //
- MPI Initialization:
  - C -> void fftw_mpi_init(void)
  - Fortran -> fftw_mpi_init()

Create Arrays

C:

* Fixed size array:
  ff_complex data=[n0][n1][n2]
* Dynamic array
  data = fftw_alloc_complex(n0*n1*n2)
* MPI dynamic arrays:
  fftw_complex *data
  ptrdiff_t alloc_local, local_no, local_no_start
  alloc_local= fftw_mpi_local_size_3d(n0, n1, n2, MPI_COMM_WORLD, &local_n0,&local_n0_start)
  data = fftw_alloc_complex(alloc_local)

FORTRAN:

* Fixed size array (simplest way):
  complex(C_DOUBLE_COMPLEX), dimension(n0,n1,n2) :: data
* Dynamic array (simplest way):
  complex(C_DOUBLE_COMPLEX), allocatable, dimension(:,:,:) :: data
  allocate (data(n0, n1, n2)
* Dynamic array (fastest method):
  complex(C_DOUBLE_COMPLEX), pointer :: data(:,:,:))
  type(C_PTR) :: cdata
  cdata - fftw_alloc_complex(n0*n1*n2)
  call c_f_pointer(cdata, data, [n0,n1,n2])
* MPI dynamic arrays:
  complex(C_DOUBLE_COMPLEX), pointer :: data(:,:,:)
  type(C_PTR) :: cdata
  integer(C_INTPTR_T) :: alloc_local, local_n2, local_n2_offset
  alloc_local = fftw_mpi_local_size_3d(n2, n1, n0, MPI_COMM_WORLD, local_n2, local_n2_offset)
  cdata = fftw_alloc_complex(alloc_local)
  call c_f_pointer(cdata, data, [n0,n1,local_n2])

Plan Creation (C2C)

FFTW_FORWARD
FFTW_BACKWARD

1D Complex to complex DFT:

* C :

fftw_plan = fftw_plan_dft_1d(int nx, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)

FFTW_FORWARD
FFTW_BACKWARD

* FORTRAN :

plan = fftw_plan_dft_1d(nz, in, out, dir, flags)

2D Complex to complex DFT:

* C :

fftw_plan = fftw_plan_dft_2d(int nx, int ny, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)
fftw_plan = fftw_mpi_plan_dft_2d(int nx, int ny, fftw_complex *in, fftw_ complex *out, MPI_COMM_WORLD, fftw_direction dir, int flags)

* FORTRAN :

plan = ftw_plan_dft_2d(ny, nx, in, out, dir, flags)
plan = ftw_mpi_plan_dft_2d(ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

3D Complex to complex DFT:

* C :

fftw_plan = fftw_plan_dft_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)
fftw_plan = fftw_mpi_plan_dft_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out, MPI_COMM_WORLD, fftw_direction dir, int flags)

* FORTRAN :

plan = ftw_plan_dft_3d(nz, ny, nx, in, out, dir, flags)
plan = ftw_mpi_plan_dft_3d(nz, ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

Plan Creation (R2C)

FFTW_FORWARD
FFTW_BACKWARD

1D Real to complex DFT:

* C :

fftw_plan = fftw_plan_dft_r2c_1d(int nx, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)

FFTW_FORWARD
FFTW_BACKWARD

* FORTRAN :

plan = fftw_plan_dft_r2c_1d(nz, in, out, dir, flags)

2D Real to complex DFT:

* C :

fftw_plan = fftw_plan_dft_r2c_2d(int nx, int ny, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)
fftw_plan = fftw_mpi_plan_dft_r2c_2d(int nx, int ny, fftw_complex *in, fftw_ complex *out, MPI_COMM_WORLD, fftw_direction dir, int flags)

* FORTRAN :

plan = ftw_plan_dft_r2c_2d(ny, nx, in, out, dir, flags)
plan = ftw_mpi_plan_dft_r2c_2d(ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

3D Real to complex DFT:

* C :

fftw_plan = fftw_plan_dft_r2c_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out, fftw_direction dir, unsigned flags)
fftw_plan = fftw_mpi_plan_dft_r2c_3d(int nx, int ny, int nz, fftw_complex *in, fftw_complex *out, MPI_COMM_WORLD, fftw_direction dir, int flags)

* FORTRAN :

plan = ftw_plan_dft_r2c_3d(nz, ny, nx, in, out, dir, flags)
plan = ftw_mpi_plan_dft_r2c_3d(nz, ny, nx, in, out, MPI_COMM_WORLD, dir, flags)

Plan Execution

Complexct to complexct DFT:
* C :
void fftw_execut_dft(fftw_plan plan, fftw_complex *in, fftw_complexct *out)
void fftw_mpi_execute_dft(fftw_plan plan, fftw_complexct *in, fftw_complex *out)

* FORTRAN :
fftw_execute_dft(plan, in, out)
fftw_mpi_execute_dft(plan, in, out)

Real to complexct DFT:
* C :
void fftw_execut_dft(fftw_plan plan, double *in, fftw_complexct *out)
void fftw_mpi_execute_dft(fftw_plan plan, double *in, fftw_complex *out)

* FORTRAN :
fftw_execute_dft(plan, in, out)
fftw_mpi_execute_dft(plan, in, out)

Finalizing FFTW

Destroying PLAN :
* C :
void fftw_destroy_plan(fftw_plan plan)
* FORTRAN :
fftw_destroy_plan(plan)

FFTW MPI cleanup :
* C :
void fftw_mpi_cleanup( )
* FORTRAN :
fftw_mpi_cleanup ( )

Deallocate data :
* C :
void fftw_free(fftw_complex data)
* FORTRAN :
fftw_free(data)

References

- مآثر العلماء في الرياضيات والفلك في الكهرباء والفيزياء في الطب والكيمياء، المهندس أسامة حوحو
- Note Books
- Numerical Recipes The Art of Scientific Computing, William H. Press, Saul A. Teukolsky, William T. Vetterling & Brian P. Flannery
- Numerical Analysis, Richard L. Burden & J. Douglas Faires
- Mathematical Methods in the Physical Sciences, Mary L. Boas
- http://www.ime.unicamp.br/~ms211-cursao/sites/default/files/material-didatico/machine-calculation-complex-fourier.pdf
- http://www.fftw.org
- http://www2.math.uu.se/~figueras/fftw_tutorial/fftw_tutorial.html

References

* Philipp Grandcl´ement, Introduction to spectral methods, 5 place J. Janssen, 92195 Meudon Cedex, France

* Philipp Schlatter, Spectral Methods, Computational Fluid Dynamics SG2212, Version 20100301

* M. Mortensen. Shenfun - Automating the Spectral Galerkin Method, In Bjørn Helge Skallerud and Helge Ingolf Andersson (ed.), MekIT'17 - Ninth national conference on Computational Mechanics. International Center for Numerical Methods in Engineering (CIMNE)., 2017, http://arxiv.org/abs/1708.03188.

* M. Gentleman and G. Sande, "Fast Fourier transforms-For fun and profit," 1966 Fall Joint Comput. Conf., AFIPS Proc., vol. 29. Washington, D. C.: Spartan, 1966, pp. 563-578.

Thanks!

* MS2 & MS13 at https://www.siam.org/meetings/pp18/
* Twitter: @SamarHpc