Fast Fourier Transform

Overview

- What does it do and why do we use it?
- Implementation in C using MPI
- Performance

Mechanics and Usage

Why do we care?

Fourier Transform

- In essence, it allows us to move from one domain to another
- Creates what is called a “Fourier Series” from an expression
- This series is representative of the original expression and can be reversed
- For our case, we will use it go decompose complex signals into parts for analysis

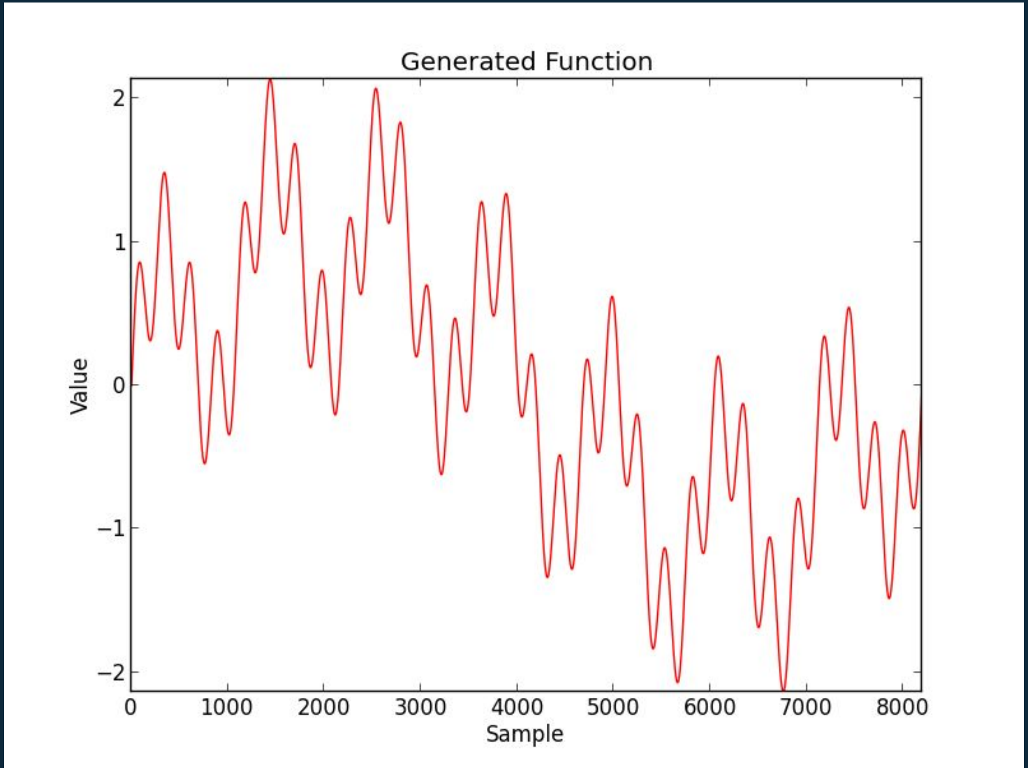
Discretization

- In the real world, we do not have expressions to represent signals
- Given a set of samples, we can run operations on them to achieve the same sort of results.
- This process is of **$O(n^2)$** in a naive DFT

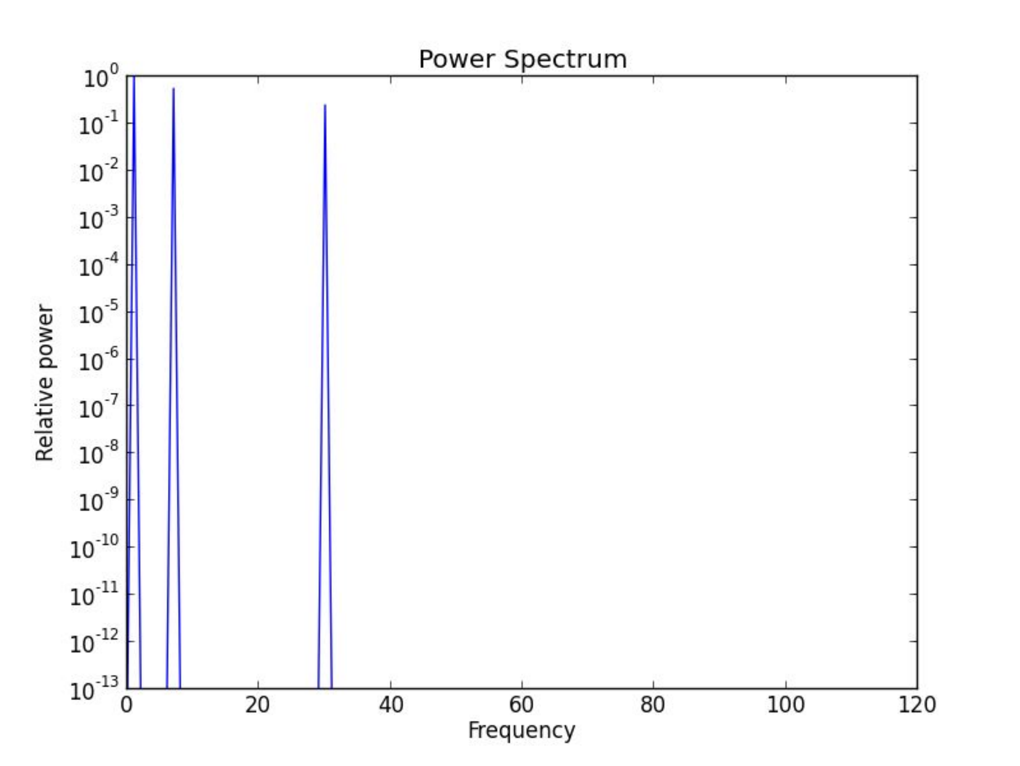
$$X_k = \frac{1}{N} \sum_{n=0}^{N-1} x_n e^{i2\pi k \frac{n}{N}}$$

To find the energy at a particular frequency, spin your signal around a circle at that frequency and average a bunch of points along that path

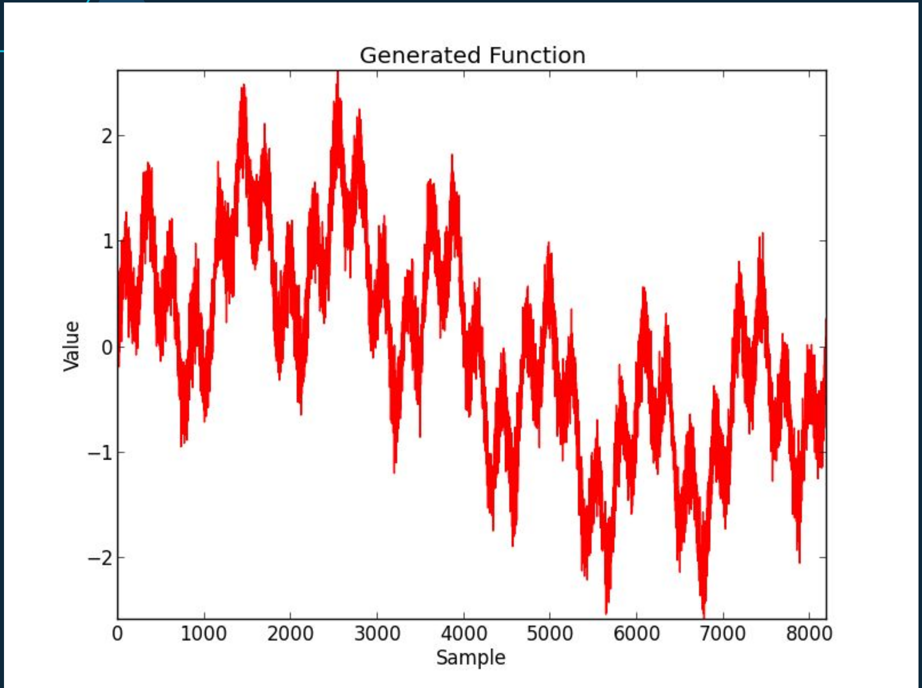
Time Domain Signal

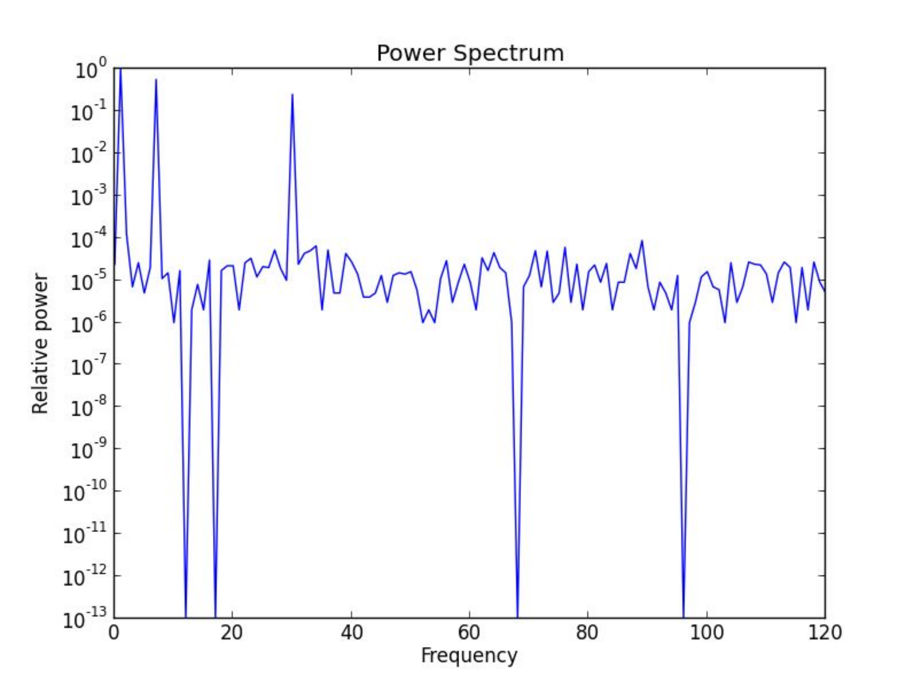


Frequency Domain Signal



Noise Resistance





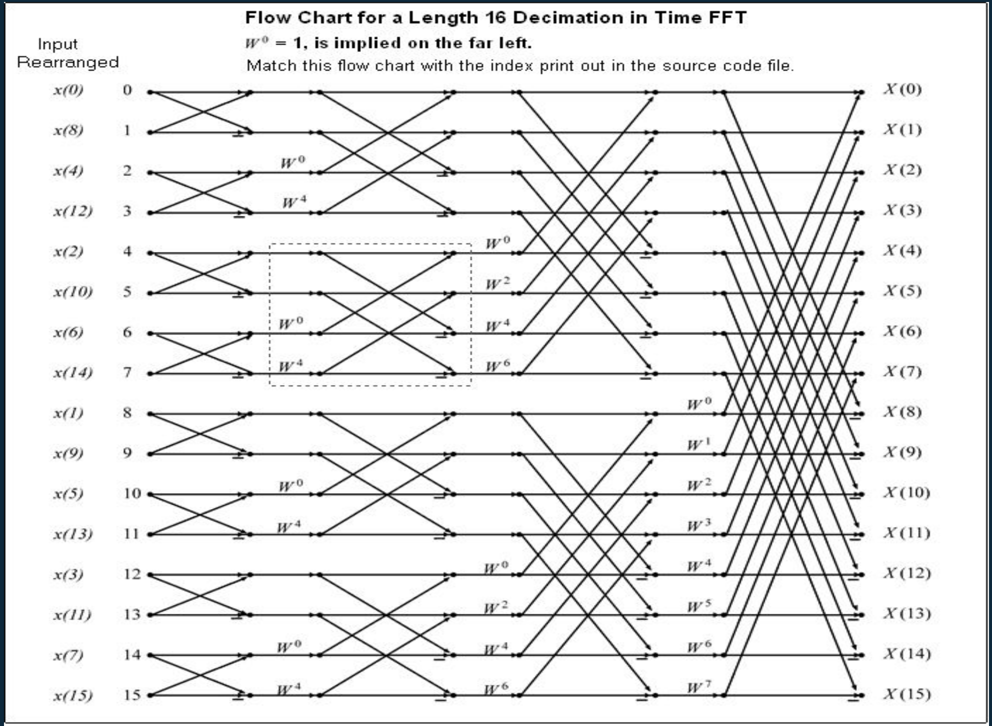
Fast Fourier Transform

- Most commonly used FFT is the Cooley - Tukey Algorithm
- Process is expressed much more succinctly and requires fewer operations
- Is inherently recursive, which can be tricky with large problem sizes
- Reduces work to **O(n * logn)** for 1D FFT
- A 2D FFT can be achieved by applying the operation over all rows, then all columns

Implementation

Serial, OpenMP, and MPI

Butterfly Operations



Burrus, C. Fast Fourier Transforms, Connexions Web site. http://cnx.org/content/col10550/1.22/.

Recursive Algorithm

$X_{0,\ldots,N-1} \leftarrow$ **ditfft2**(x, N, s): *DFT of* $(x_0, x_s, x_{2s}, \ldots, x_{(N-1)s})$:

  if $N = 1$ then

    $X_0 \leftarrow x_0$ *trivial size-1 DFT base case*

  else

    $X_{0,\ldots,N/2-1} \leftarrow$ **ditfft2**$(x, N/2, 2s)$ *DFT of* $(x_0, x_{2s}, x_{4s}, \ldots)$

    $X_{N/2,\ldots,N-1} \leftarrow$ **ditfft2**$(x+\mathrm{s}, N/2, 2s)$ *DFT of* $(x_s, x_{s+2s}, x_{s+4s}, \ldots)$

    for $k = 0$ to $N/2-1$ *combine DFTs of two halves*

      $\mathrm{t} \leftarrow X_k$

      $X_k \leftarrow \mathrm{t} + \exp(-2\pi i\, k/N)\, X_{k+N/2}$

      $X_{k+N/2} \leftarrow \mathrm{t} - \exp(-2\pi i\, k/N)\, X_{k+N/2}$

    endfor

  endif

General Ideas

- We can make our operations easier by sorting in bit reverse order
- This way, we can use an iterative approach, which solves several problems regarding stack and memory
- Requires problem size of 2^K nodes
- Allows for more parallelizable functions later on

OpenMP

- I quickly discovered that this problem benefited very little from parallel optimizations in 1D
- I turned my focus to the 2D problem
- Decomposed problem into “Stripes” of multiple rows of whole
- Ran FFT over each item in chunk, transposed information and repeated
- Still did not see much of a speed up in the OpenMP version

MPI

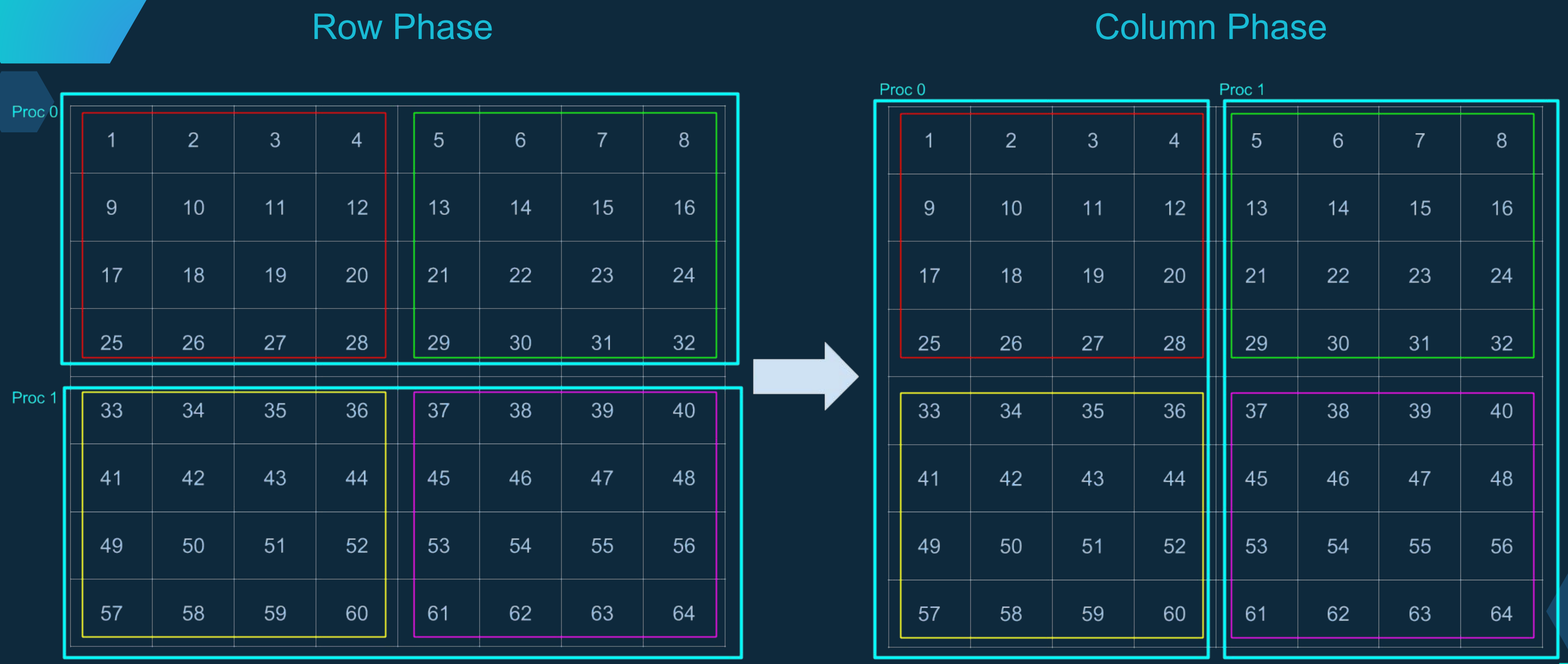
- Approach was very similar to OpenMP version
- Communication was the most important element of my changes
- Made all operations run based on 1D complex arrays (no complicated structures)
- Special operations were needed to package messages for distribution

```
(0.000000 0.000000j) (1.000000 0.000000j) (2.000000 0.000000j) (3.000000 0.000000j) (4.000000 0.000000j) (5.000000 0.000000j) (6.000000 0.000000j) (7.000000 0.000000j)
(8.000000 0.000000j) (9.000000 0.000000j) (10.000000 0.000000j) (11.000000 0.000000j) (12.000000 0.000000j) (13.000000 0.000000j) (14.000000 0.000000j) (15.000000 0.000000j)
(16.000000 0.000000j) (17.000000 0.000000j) (18.000000 0.000000j) (19.000000 0.000000j) (20.000000 0.000000j) (21.000000 0.000000j) (22.000000 0.000000j) (23.000000 0.000000j)
(24.000000 0.000000j) (25.000000 0.000000j) (26.000000 0.000000j) (27.000000 0.000000j) (28.000000 0.000000j) (29.000000 0.000000j) (30.000000 0.000000j) (31.000000 0.000000j)
(32.000000 0.000000j) (33.000000 0.000000j) (34.000000 0.000000j) (35.000000 0.000000j) (36.000000 0.000000j) (37.000000 0.000000j) (38.000000 0.000000j) (39.000000 0.000000j)
(40.000000 0.000000j) (41.000000 0.000000j) (42.000000 0.000000j) (43.000000 0.000000j) (44.000000 0.000000j) (45.000000 0.000000j) (46.000000 0.000000j) (47.000000 0.000000j)
(48.000000 0.000000j) (49.000000 0.000000j) (50.000000 0.000000j) (51.000000 0.000000j) (52.000000 0.000000j) (53.000000 0.000000j) (54.000000 0.000000j) (55.000000 0.000000j)
(56.000000 0.000000j) (57.000000 0.000000j) (58.000000 0.000000j) (59.000000 0.000000j) (60.000000 0.000000j) (61.000000 0.000000j) (62.000000 0.000000j) (63.000000 0.000000j)
```

```
(224.000000 0.000000j) (232.000000 0.000000j) (240.000000 0.000000j) (248.000000 0.000000j) (256.000000 0.000000j) (264.000000 0.000000j) (272.000000 0.000000j) (280.000000 0.000000
j)
(-32.000000 77.254834j) (-32.000000 77.254834j) (-32.000000 77.254834j) (-32.000000 77.254834j) (-32.000000 77.254834j) (-32.000000 77.254834j) (-32.000000 77.254834j) (-32.000000 7
7.254834j)
(-32.000000 32.000000j) (-32.000000 32.000000j) (-32.000000 32.000000j) (-32.000000 32.000000j) (-32.000000 32.000000j) (-32.000000 32.000000j) (-32.000000 32.000000j) (-32.000000 3
2.000000j)
(-32.000000 13.254834j) (-32.000000 13.254834j) (-32.000000 13.254834j) (-32.000000 13.254834j) (-32.000000 13.254834j) (-32.000000 13.254834j) (-32.000000 13.254834j) (-32.000000 1
3.254834j)
(-32.000000 0.000000j) (-32.000000 0.000000j) (-32.000000 0.000000j) (-32.000000 0.000000j) (-32.000000 0.000000j) (-32.000000 0.000000j) (-32.000000 0.000000j) (-32.000000 0.000000
j)
(-32.000000 -13.254834j) (-32.000000 -13.254834j) (-32.000000 -13.254834j) (-32.000000 -13.254834j) (-32.000000 -13.254834j) (-32.000000 -13.254834j) (-32.000000 -13.254834j) (-32.0
00000 -13.254834j)
(-32.000000 -32.000000j) (-32.000000 -32.000000j) (-32.000000 -32.000000j) (-32.000000 -32.000000j) (-32.000000 -32.000000j) (-32.000000 -32.000000j) (-32.000000 -32.000000j) (-32.0
00000 -32.000000j)
(-32.000000 -77.254834j) (-32.000000 -77.254834j) (-32.000000 -77.254834j) (-32.000000 -77.254834j) (-32.000000 -77.254834j) (-32.000000 -77.254834j) (-32.000000 -77.254834j) (-32.0
00000 -77.254834j)
```

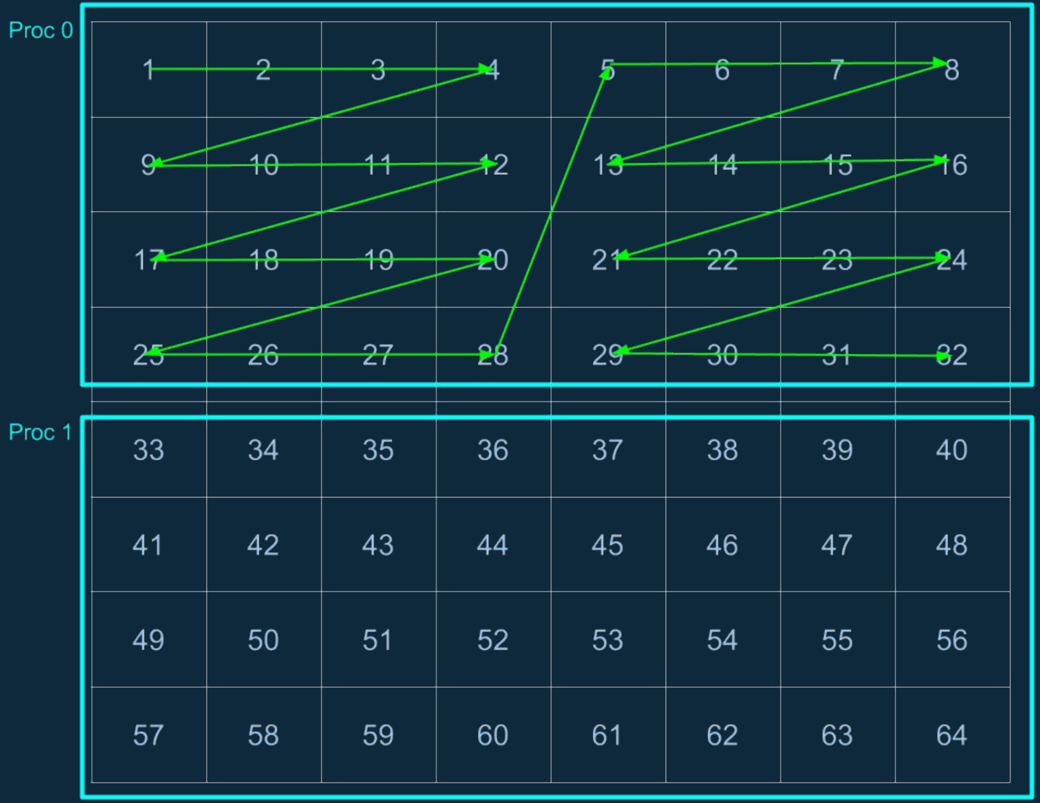
```
(0.000000 0.000000j) (1.000000 0.000000j) (2.000000 0.000000j) (3.000000 0.000000j) (4.000000 0.000000j) (5.000000 0.000000j) (6.000000 0.000000j) (7.000000 0.000000j)
(8.000000 -0.000000j) (9.000000 -0.000000j) (10.000000 -0.000000j) (11.000000 -0.000000j) (12.000000 -0.000000j) (13.000000 -0.000000j) (14.000000 -0.000000j) (15.000000 -0.000000j)

(16.000000 0.000000j) (17.000000 0.000000j) (18.000000 0.000000j) (19.000000 0.000000j) (20.000000 0.000000j) (21.000000 0.000000j) (22.000000 0.000000j) (23.000000 0.000000j)
(24.000000 0.000000j) (25.000000 0.000000j) (26.000000 0.000000j) (27.000000 0.000000j) (28.000000 0.000000j) (29.000000 0.000000j) (30.000000 0.000000j) (31.000000 0.000000j)
(32.000000 0.000000j) (33.000000 0.000000j) (34.000000 0.000000j) (35.000000 0.000000j) (36.000000 0.000000j) (37.000000 0.000000j) (38.000000 0.000000j) (39.000000 0.000000j)
(40.000000 -0.000000j) (41.000000 -0.000000j) (42.000000 -0.000000j) (43.000000 -0.000000j) (44.000000 -0.000000j) (45.000000 -0.000000j) (46.000000 -0.000000j) (47.000000 -0.000000
j)
(48.000000 -0.000000j) (49.000000 -0.000000j) (50.000000 -0.000000j) (51.000000 -0.000000j) (52.000000 -0.000000j) (53.000000 -0.000000j) (54.000000 -0.000000j) (55.000000 -0.000000
j)
(56.000000 -0.000000j) (57.000000 -0.000000j) (58.000000 -0.000000j) (59.000000 -0.000000j) (60.000000 -0.000000j) (61.000000 -0.000000j) (62.000000 -0.000000j) (63.000000 -0.000000
j)
```


Row Phase
Proc 0
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
Proc 1
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64
Column Phase
Proc 0
Proc 1
1 2 3 4 5 6 7 8
9 10 11 12 13 14 15 16
17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32
33 34 35 36 37 38 39 40
41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56
57 58 59 60 61 62 63 64

Data read pattern for transfer into send buffer

All to all would cut data along jump from 28 to 5 in this case

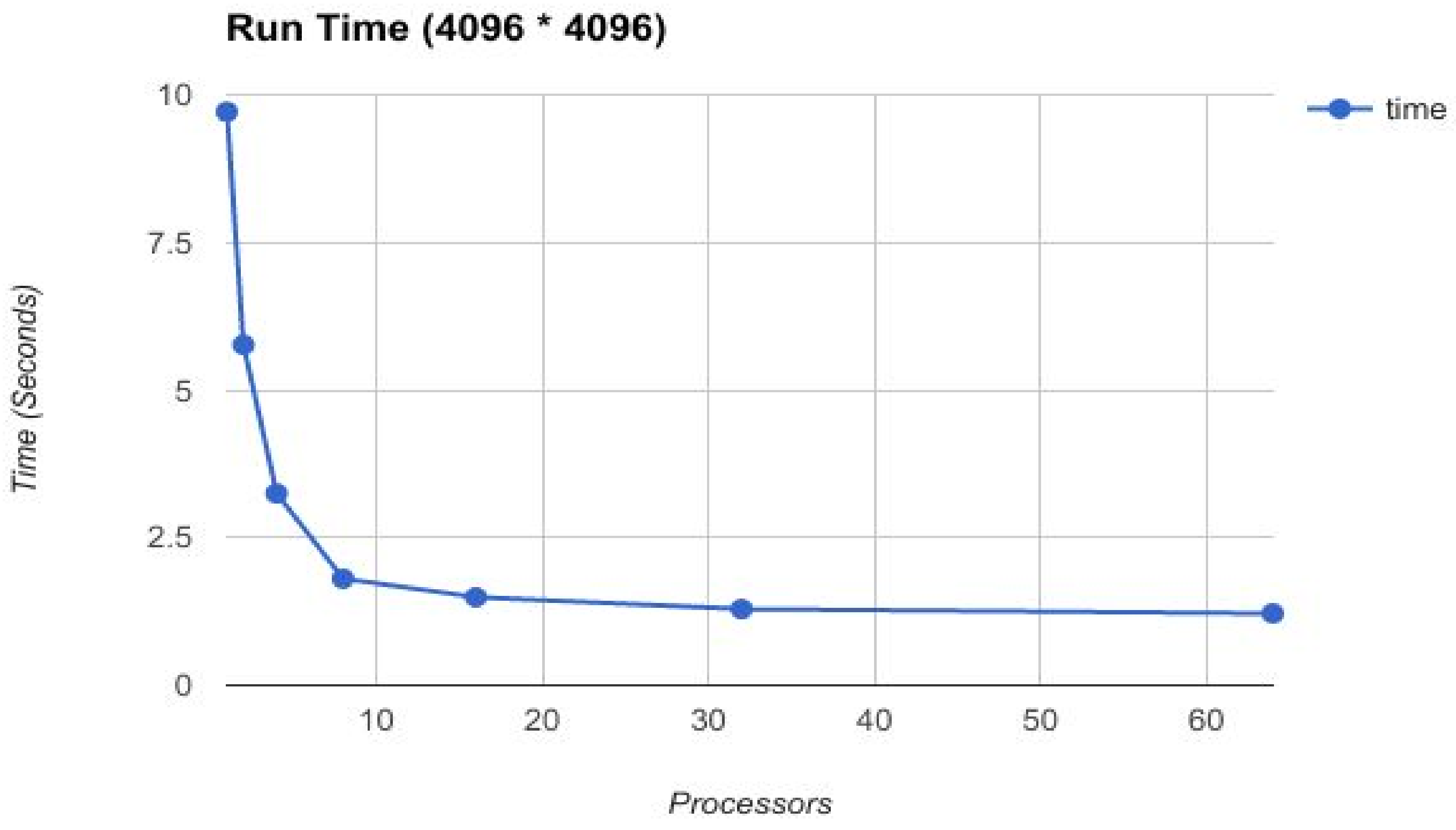


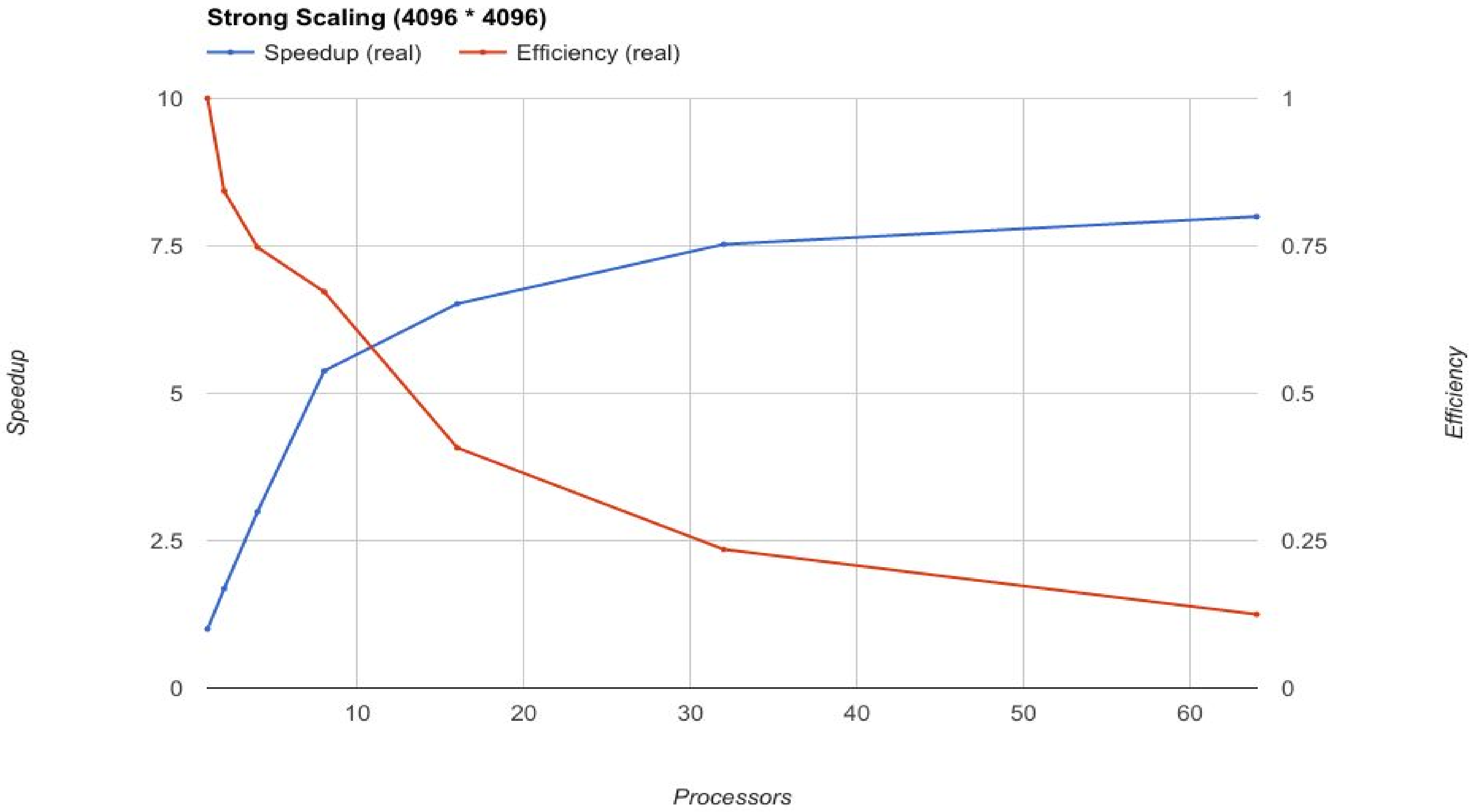
Parallel Performance

Large problems, fast times

Maneframe Strong Scaling

N	N * N	Processors	Time	Speedup (real)	Efficiency (real)
4096	16777216	1	9.712320089	1	1
4096	16777216	2	5.76313591	1.685249184	0.842624592
4096	16777216	4	3.248805046	2.989505357	0.7473763393
4096	16777216	8	1.806185007	5.377256511	0.6721570639
4096	16777216	16	1.490156889	6.517649357	0.4073530848
4096	16777216	32	1.290680885	7.524958493	0.2351549529
4096	16777216	64	1.214707851	7.99560164	0.1249312756


Run Time (4096 * 4096)
10
7.5
5
2.5
0
Time (Seconds)
10
20
30
40
50
60
Processors
time


Strong Scaling (4096 * 4096)
Speedup (real)
Efficiency (real)
Speedup
Efficiency
Processors
10
7.5
5
2.5
0
1
0.75
0.5
0.25
0
10
20
30
40
50
60


Weak Scaling Efficiency (Work per Proc = 65536)
Efficiency
1
0.75
0.5
0.25
0
10
20
30
40
50
60
Processors
Efficiency

Notes on Performance

- Although uncertain, I believe that network topology has a great deal of impact
- Weak scaling was poor, this is due to the fact that communication scales with size
- Due to limitations of size, only powers of two processors can be tested
- Was already very fast in serial, so all speedup is dependent on communications

Final Thoughts

Considerations and Future Work

Moving Forward

- Remove dependance on square matrix inputs
- Add real time system functionality, make callable from other applications
- Interface with image libraries for image signal analysis
- Run tests on higher numbers of procs, larger files
- Look into ways of distributing file I/O for faster load and dump operations

Thank You!

References

[1]"An Interactive Guide To The Fourier Transform – BetterExplained", *Betterexplained.com*, 2017. [Online]. Available: https://betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/. [Accessed: 08- May- 2017].

[2]"Butterfly Diagram", *Iowahills.com*, 2017. [Online]. Available: http://www.iowahills.com/Example%20Code/FFT%20Butterfly%20Diagram.png. [Accessed: 11- May- 2017].

[3]"Implementing FFTs in Practice", *Cnx.org*, 2017. [Online]. Available: http://cnx.org/contents/ulXtQbN7@15/Implementing-FFTs-in-Practice. [Accessed: 11- May- 2017].