Parallel FFT in Julia

Review of FFT

- *Discrete Fourier Transform*, or *DFT*, of sequence $\boldsymbol{x} = [x_0, \ldots, x_{n-1}]^T$ is sequence $\boldsymbol{y} = [y_0, \ldots, y_{n-1}]^T$ given by

$$y_m = \sum_{k=0}^{n-1} x_k \, \omega_n^{mk}, \quad m = 0, 1, \ldots, n-1$$

or

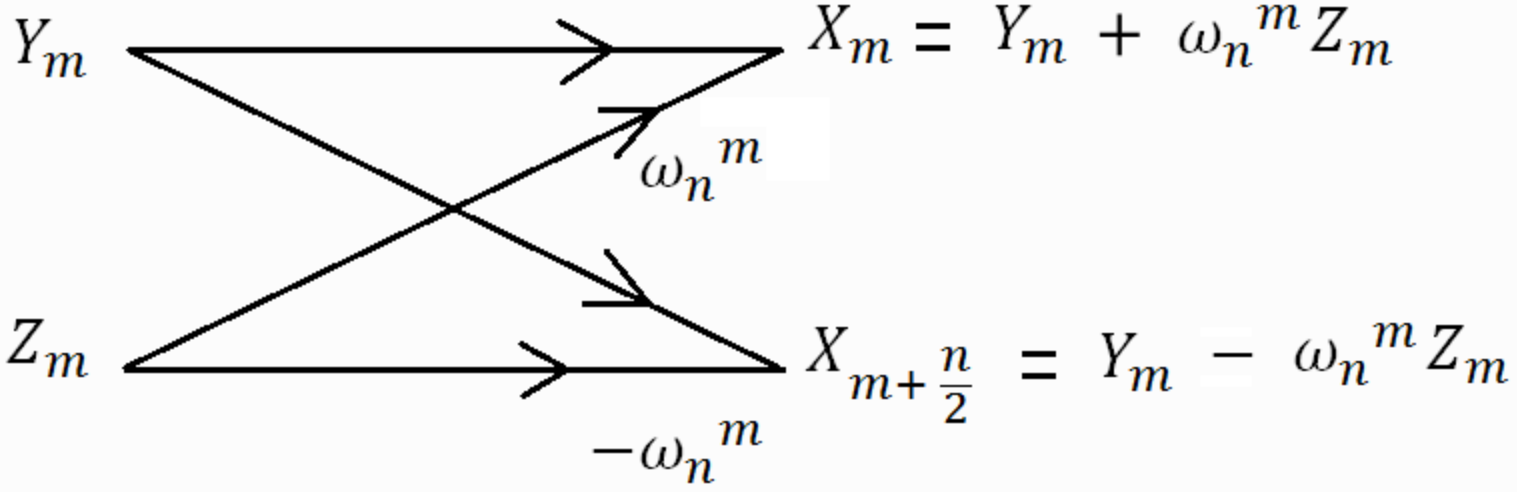
$$\boldsymbol{y} = \boldsymbol{F}_n \, \boldsymbol{x}$$

where entries of Fourier matrix $\boldsymbol{F}_n$ are given by

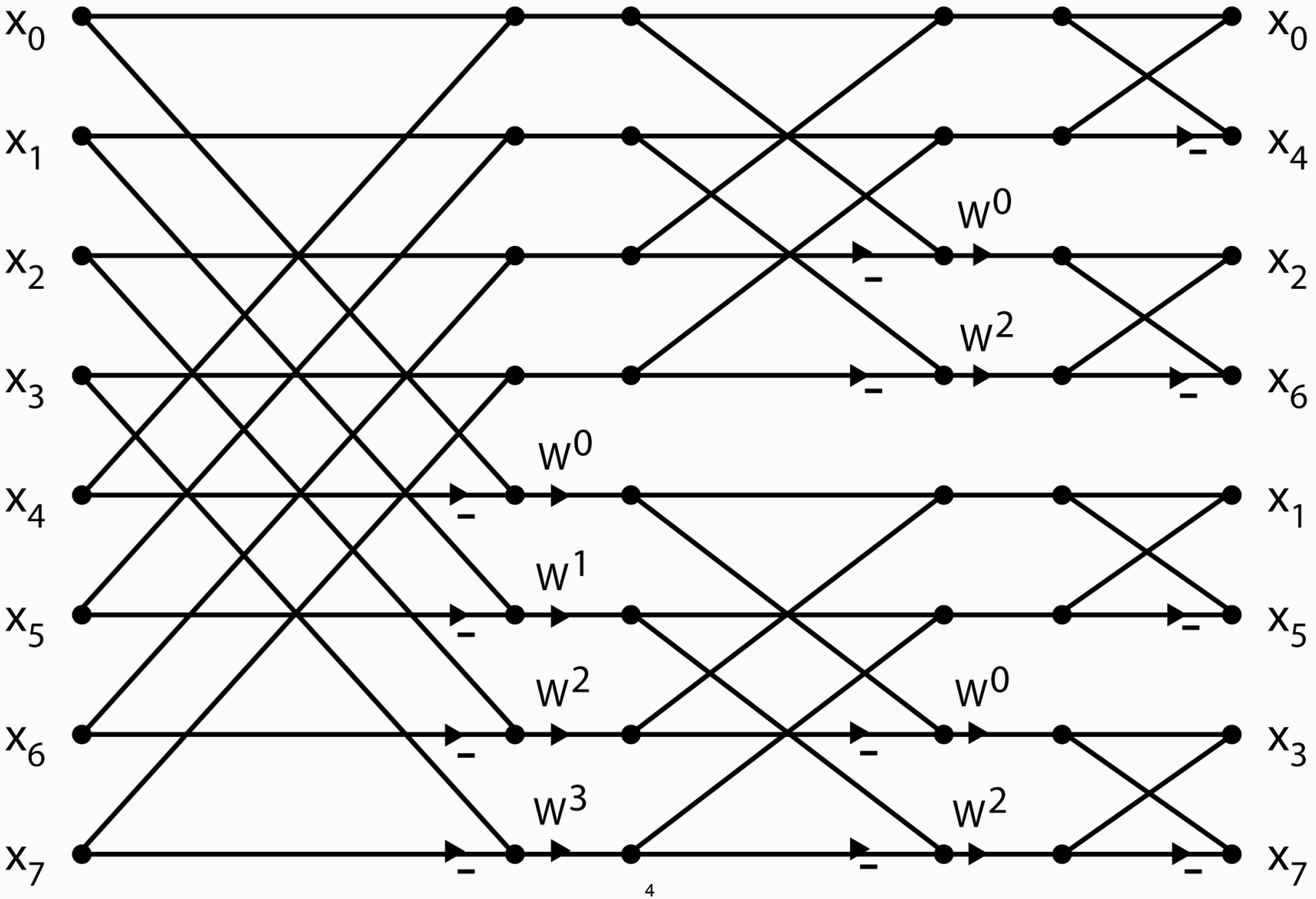
$$\{\boldsymbol{F}_n\}_{mk} = \omega_n^{mk}$$

Review of FFT (cont.)

$$y_m = \sum_{k=0}^{\frac{n}{2}-1} x_{2k}\, {\omega_n}^{mk}_{\frac{n}{2}} + {\omega_n}^m \sum_{k=0}^{\frac{n}{2}-1} x_{2k+1}\, {\omega_n}^{mk}_{\frac{n}{2}}$$



Review of FFT (cont.)



Sequential FFT Pseudocode

Recursive-FFT (array)

- arrayEven = even indexed elements of array
- arrayOdd = odd indexed elements of array
- Recursive-FFT (arrayEven)
- Recursive-FFT (arrayOdd)
- Combine results using Cooley-Tukey butterfly
- Bit reversal, could be done either before, after or in between

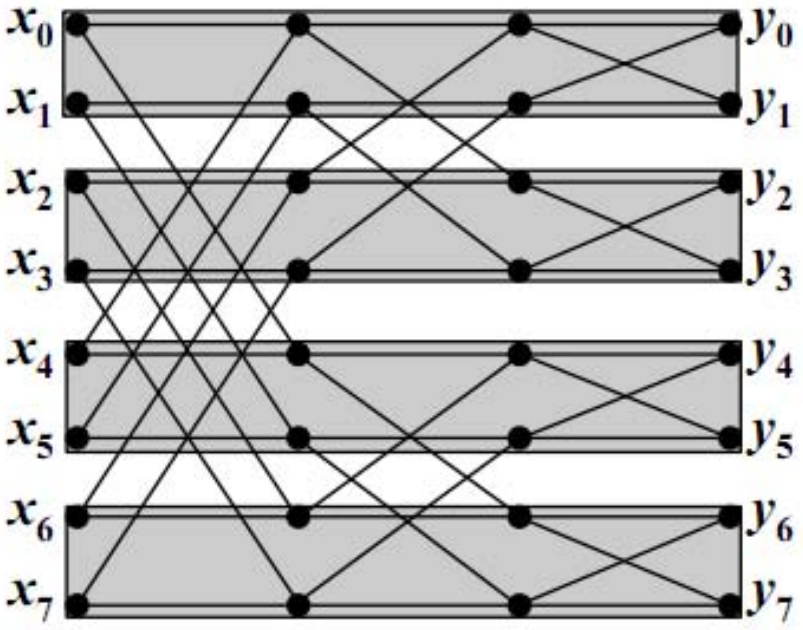
Parallel FFT Algorithms

- Binary Exchange Algorithm

- Transpose Algorithm

Binary Exchange Algorithm

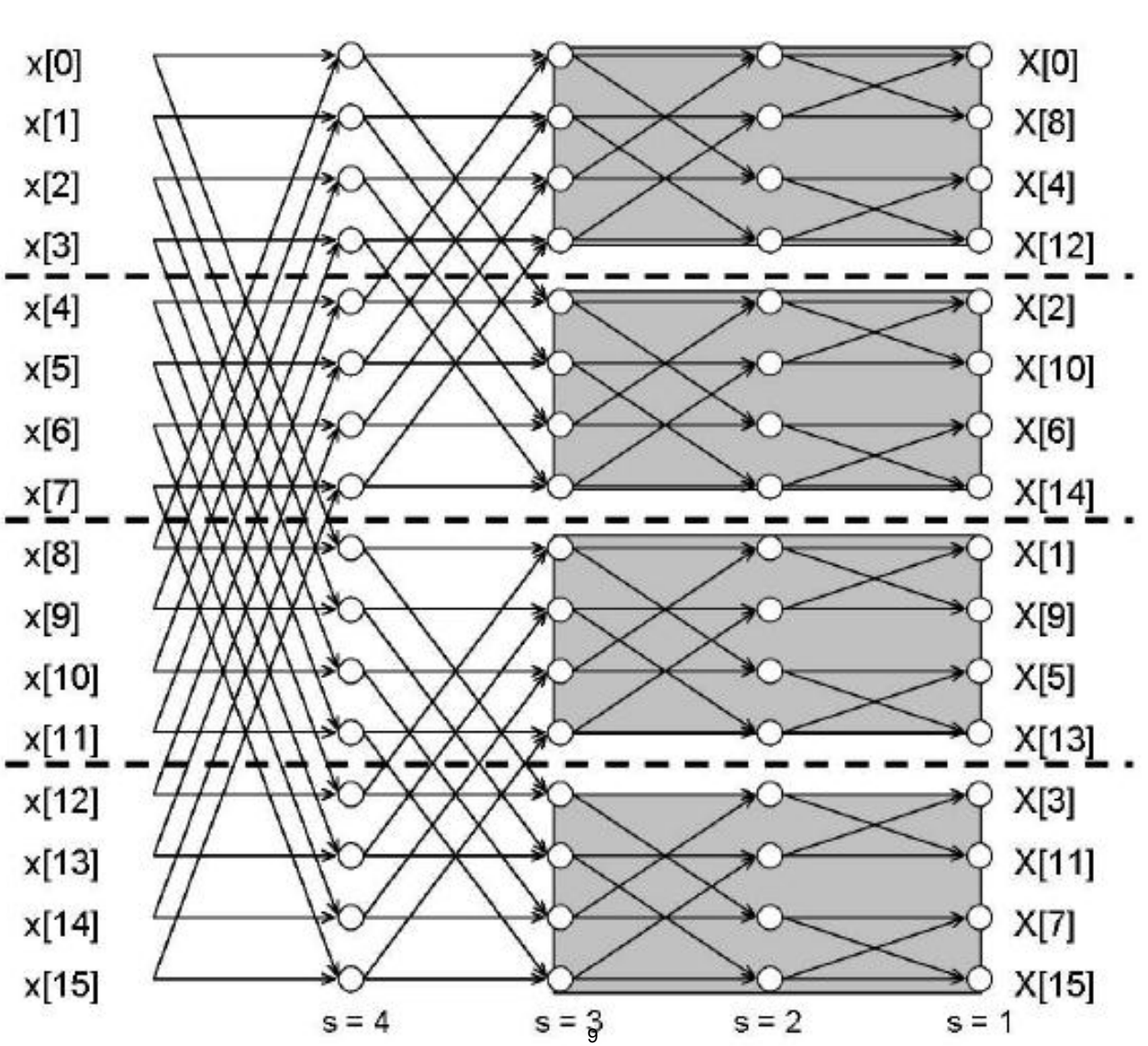
- To obtain smaller number of coarse-grain tasks, agglomerate sets of n/p components of input and output vectors x and y, where we assume p is also power of two



Binary Exchange Algorithm

- Thus, exchanges are required in binary exchange algorithm only for first $\log p$ stages, since data are local for remaining $\log(n/p)$ stages

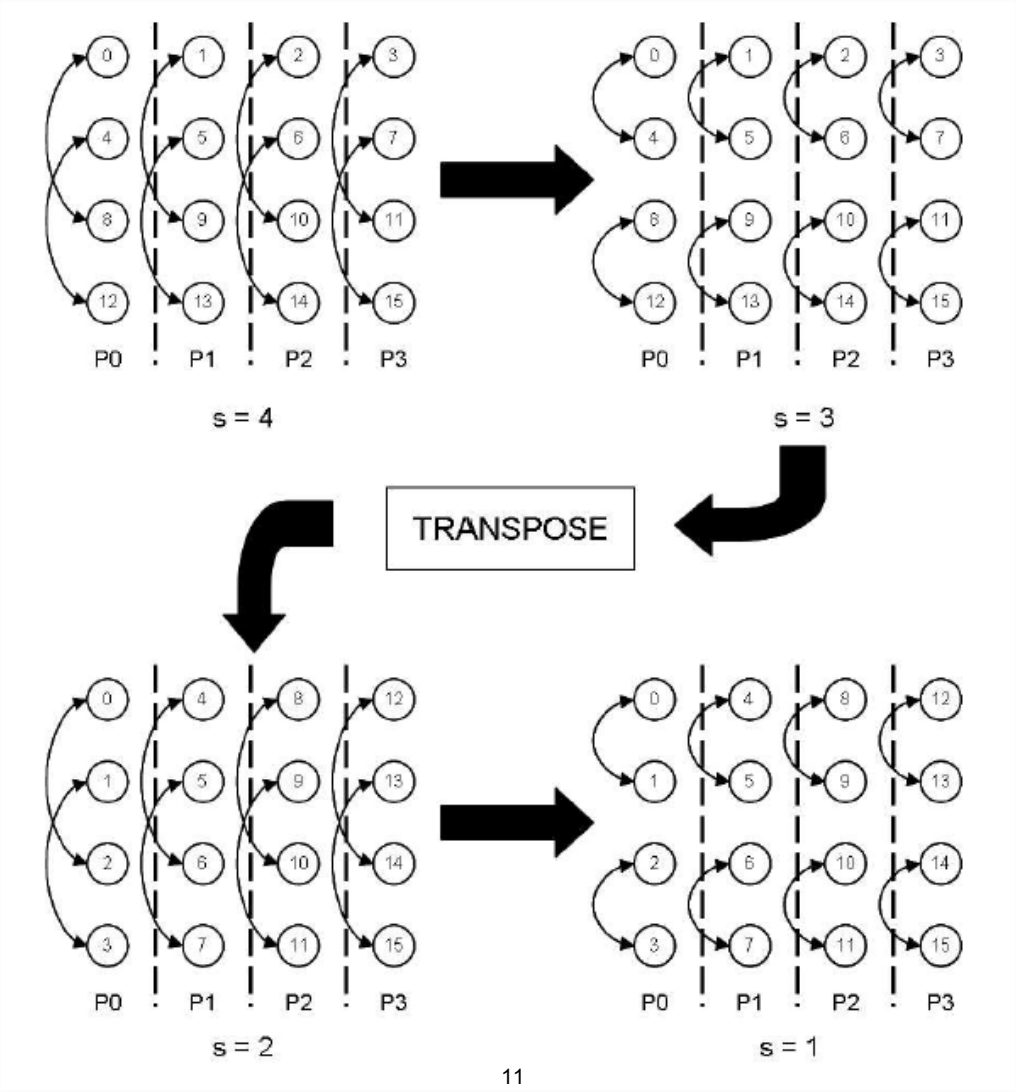
Binary Exchange Algorithm



Parallel Transpose Algorithm

- Input represented as a $\sqrt{n}$ x $\sqrt{n}$ matrix
- Local computation of sub-problems
- Transpose input
- Local computation of remaining problems

Parallel Transpose Algorithm



Julia implementations

- Input is represented as distributed arrays.
- Assumptions: N, P are powers of 2 for sake of simplicity
- More focus on minimizing communication overhead versus computational optimizations

Easy

Recursive-FFT (array)

- ……
- @spawn Recursive-FFT (arrayEven)
- @spawn Recursive-FFT (arrayOdd)
- ……

Same as FFTW parallel implementation for 1D input using Cilk.

Not so fast

Too much unnecessary overhead because of random spawning.

Better:

Recursive-FFT (array)

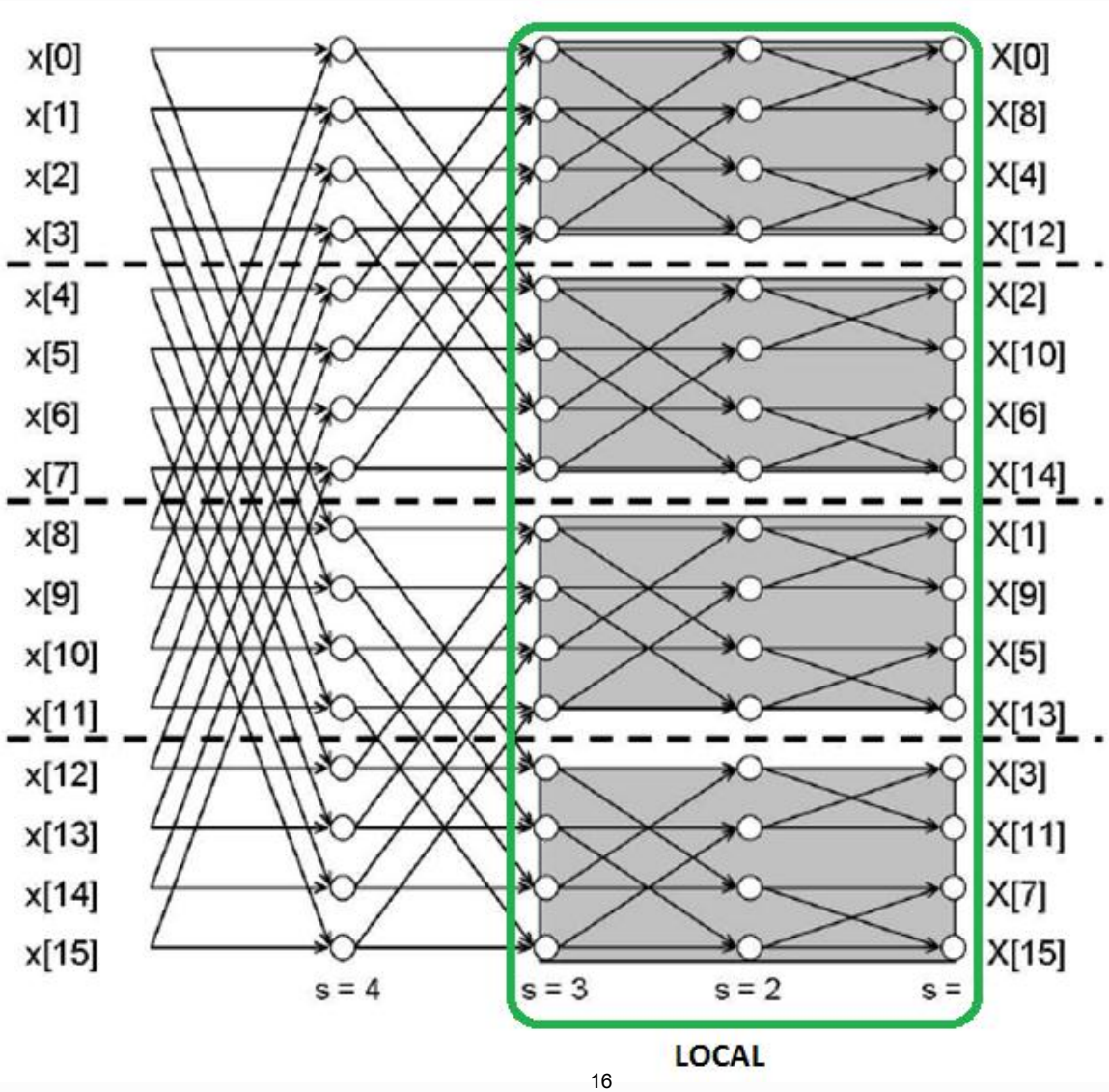
- ……..
- @spawnat owner Recursive-FFT (arrayEven)
- @spawnat owner Recursive-FFT (arrayOdd)
- ……..

Binary Exchange Implementation

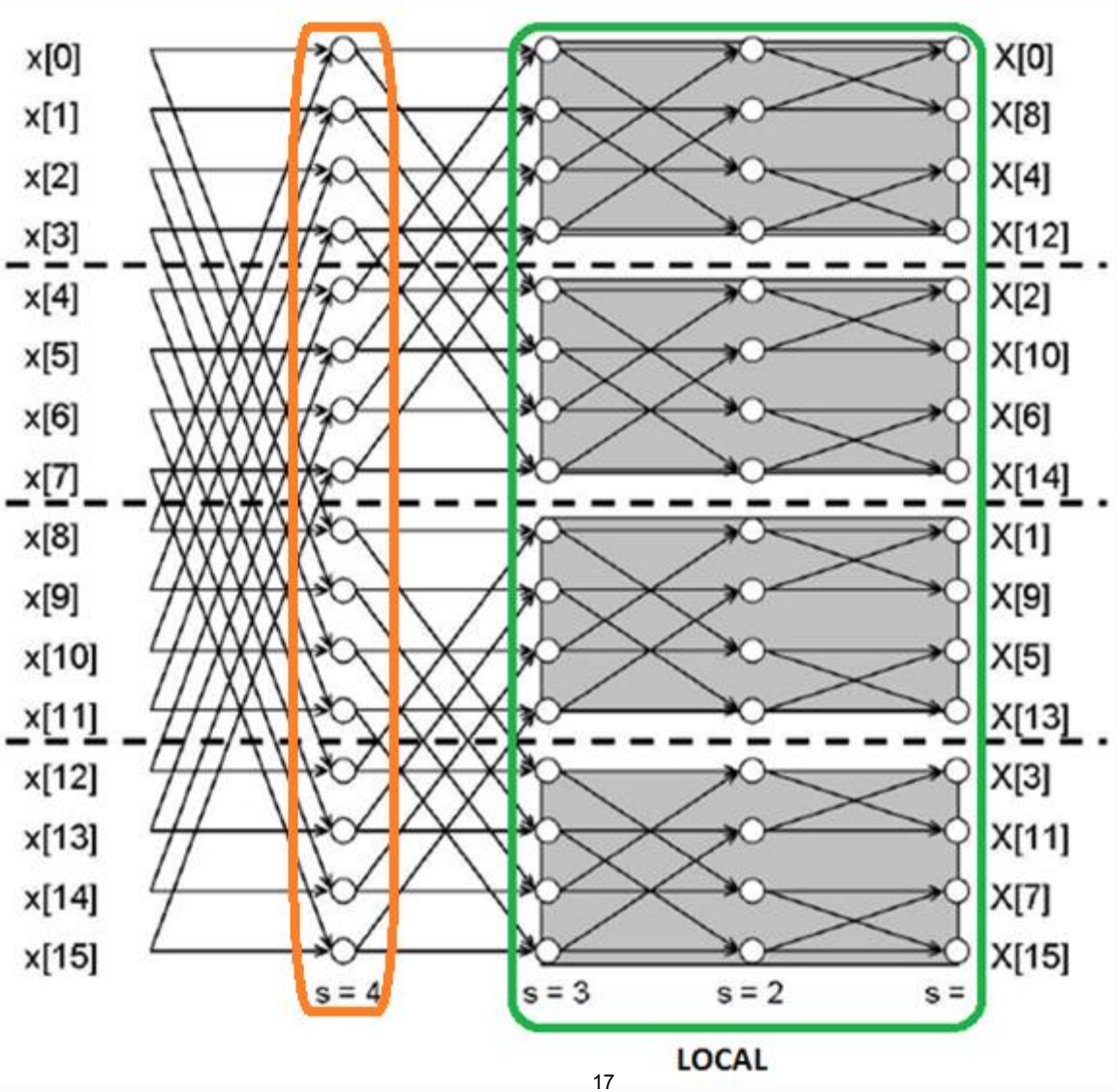
FFT_Parallel (array)

- Bit reverse input array and distribute
- @spawnat owner Recursive-FFT (first half)
- @spawnat owner Recursive-FFT (last half)
- Combine results

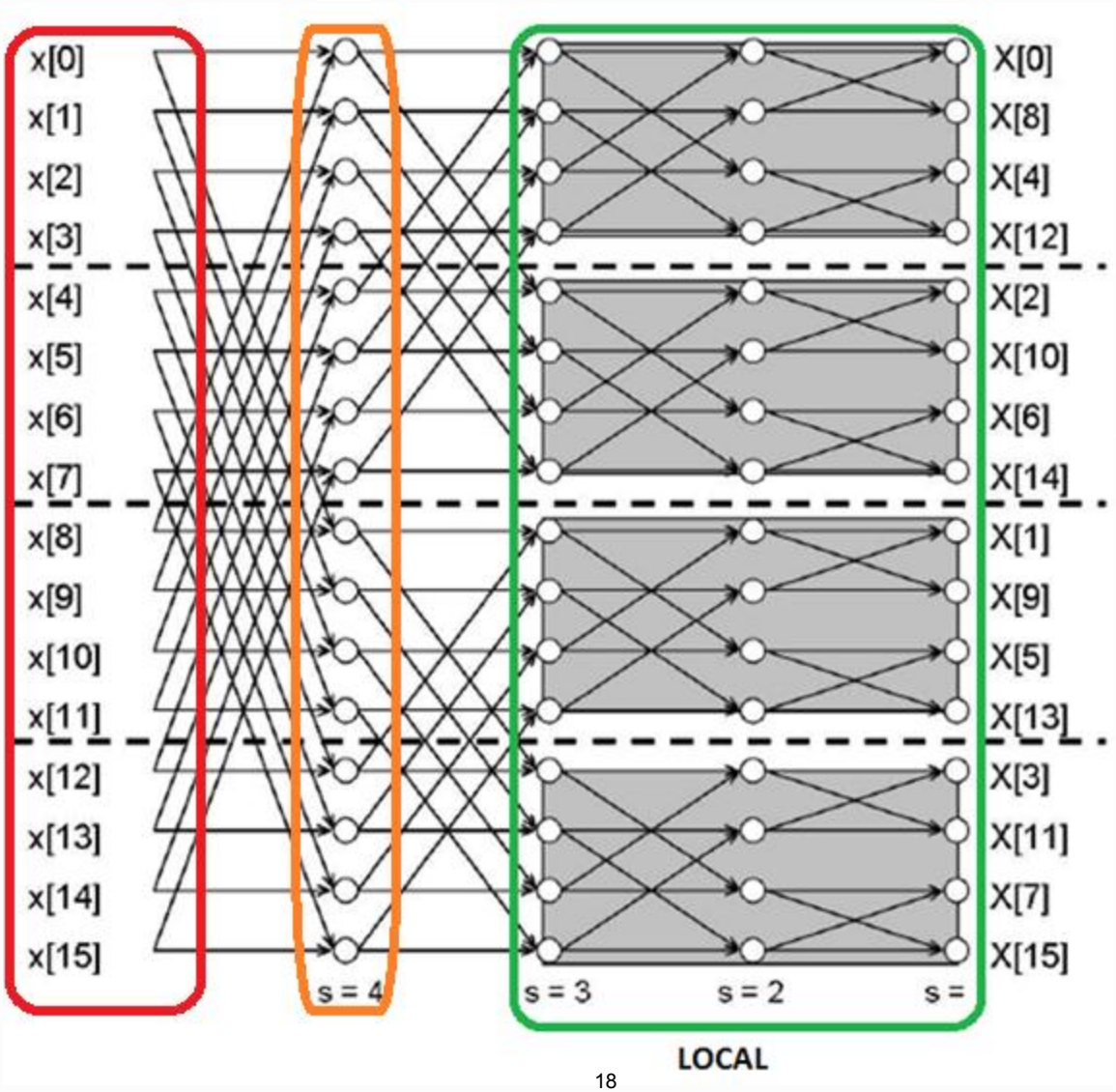
Binary Exchange Implementation



Binary Exchange Implementation



Binary Exchange Implementation



Binary Exchange Implementation

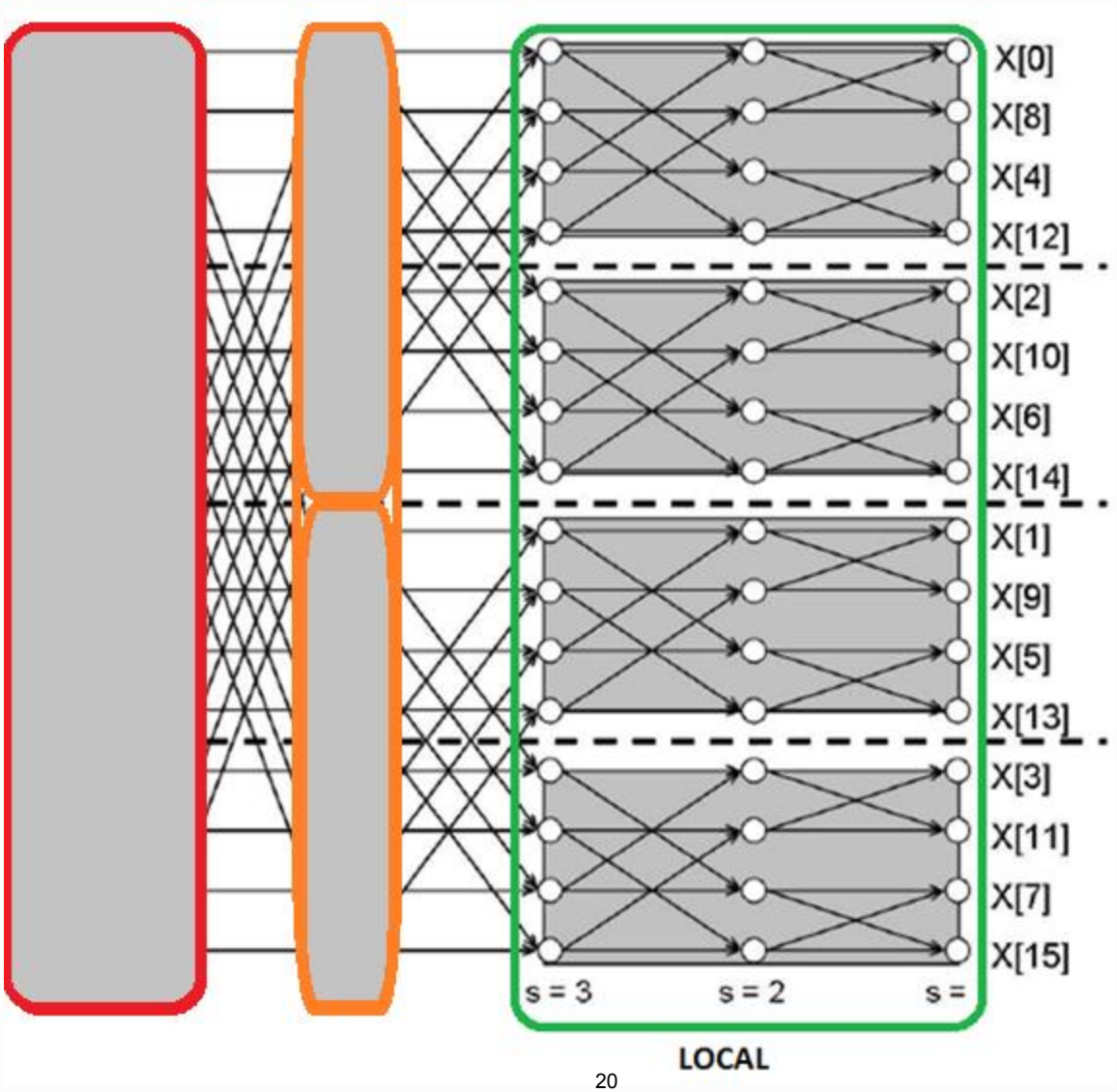
Problem:

- Single element DArray access is slow.
- The last $\log p$ stages incur too many duplicate array accesses

Solution:

- Redistribute the array so that at the first non-local stage, data will be moved all at once
- Final heavy load on last processor, but still cheaper than latency cost, especially with N large.

Binary Exchange Implementation



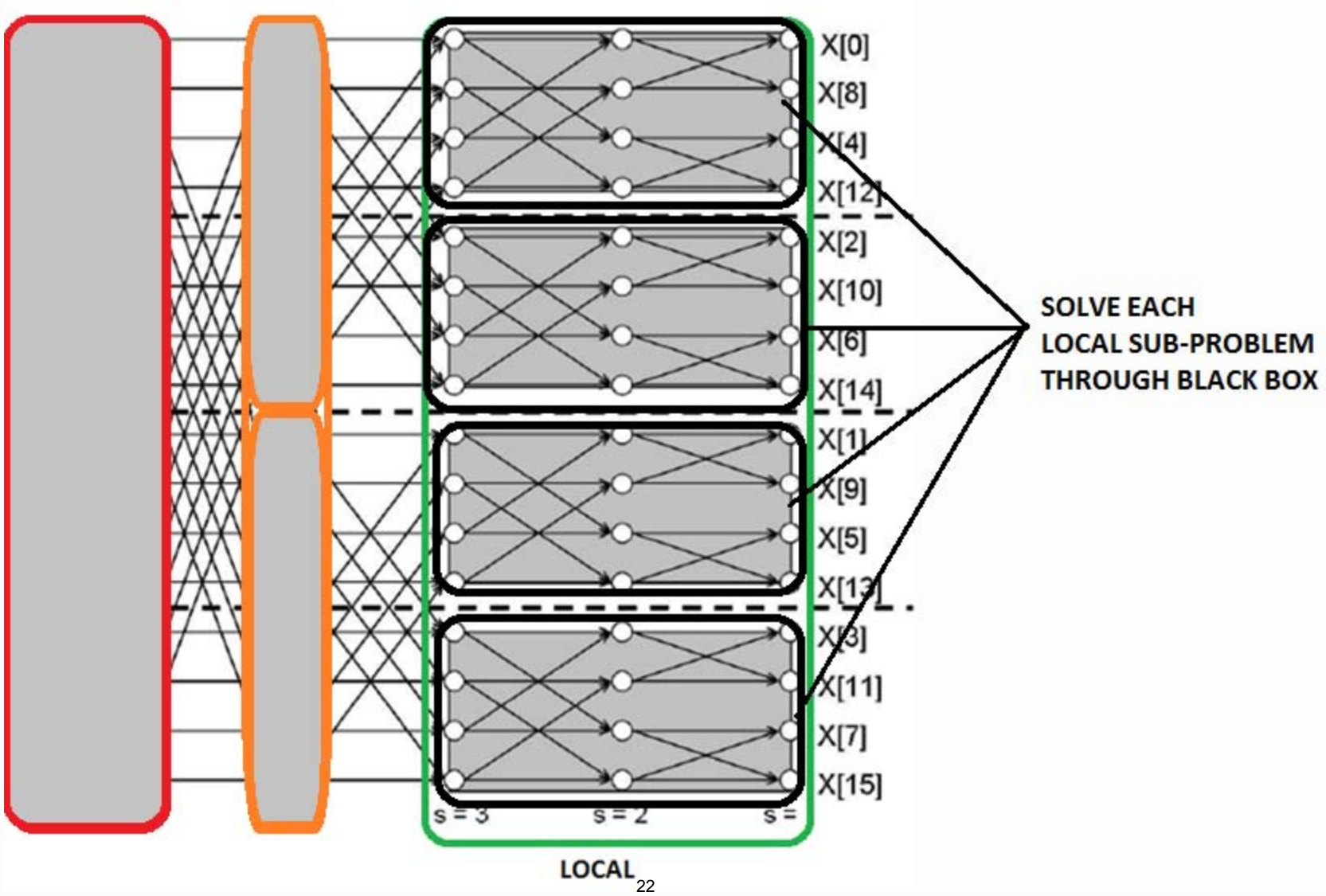
Alternate approach – Black box

- Initially similar to parallel transpose method: data is distributed so that each sub-problem is locally contained within one node

FFT_Parallel (array)

- Bit reverse input array and distribute equally
- for each processor
  - @spawnat proc FFT-Sequential (local array)
- Redistribute data and combine locally

Alternate approach



Alternate approach – Black box

Pros:

- Eliminates needs for redundant spawning which is expensive
- Can leverage black box packages such as FFTW for local sequential run, or black box combiners
  - Warning: Order of input to sub-problems is important

Note:

- Have not tried FFTW due to configuration issues

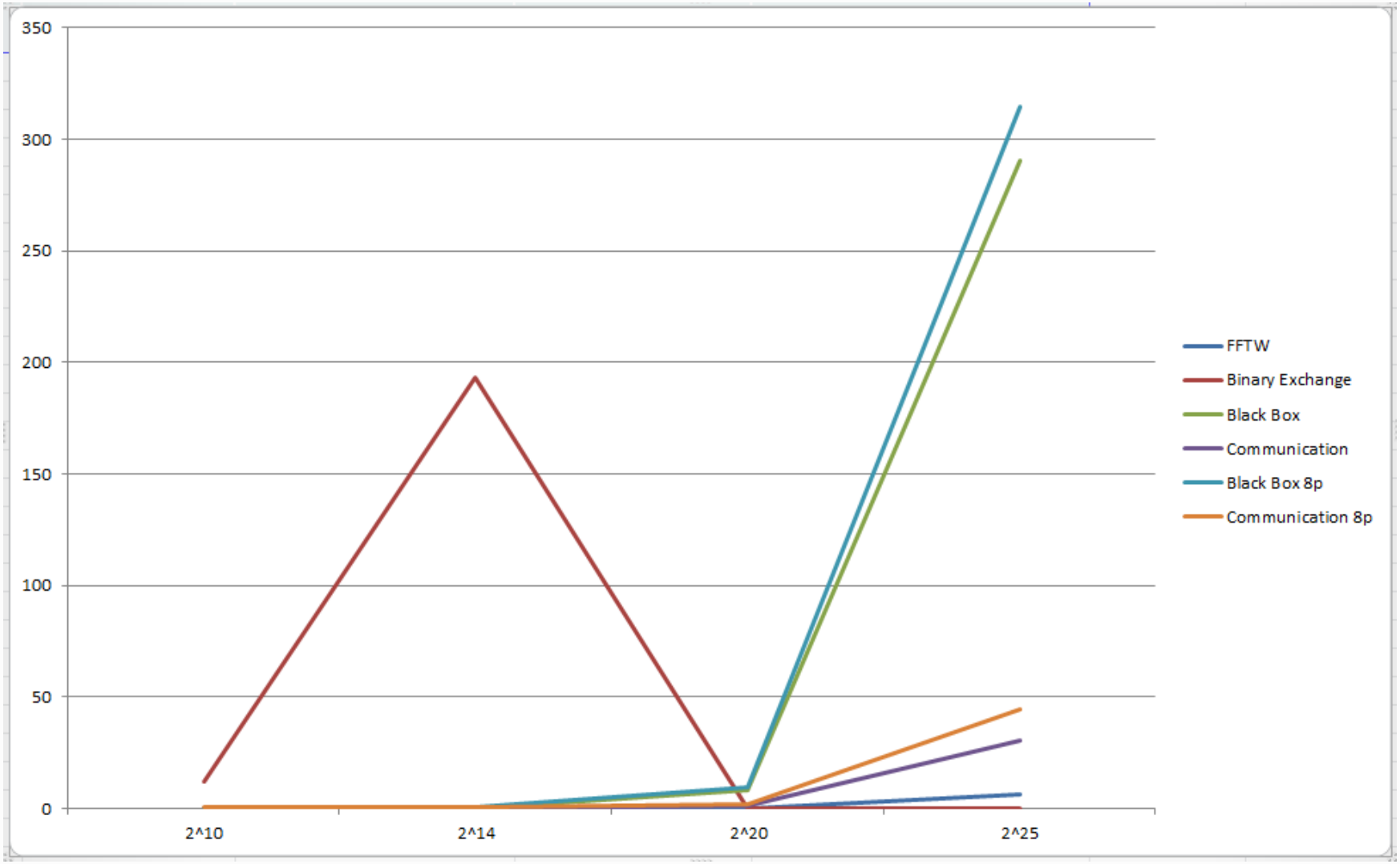
Benchmark Caveats

- Up to input size of 2^{25}, bigger data causes random memory error, broken pipe ⇒ unstable results.
- Binary Exchange implementation does not have the redistribution phase
- Black Box implementation uses in-house FFT solver instead of FFTW
- Communication cost still the bottleneck
- 4 processors

Benchmark Results

	FFTW	Binary Exchange	Black Box	Communication	Black Box 8p	Communication 8p
2^{10}	0.0002	11.938756	0.10948204	0.150836	0.37363982	0.4737592
2^{14}	0.0009	193.102961	0.18138098	0.1402709	0.59792995	0.416522
2^{20}	0.128	A year?	8.28874588	1.1951251	9.86574506	1.701771
2^{25}	6.3023	Apocalypse	290.634222	30.29218	314.73069	44.75283

Benchmark Results



Communication issues

Issue:

- Redistribution done separately at each stage of the last $\log p$ stages, total $\sum_{k=1}^{\log p} 2^{k-1}$ calls
- Could save overhead by eliminating these calls

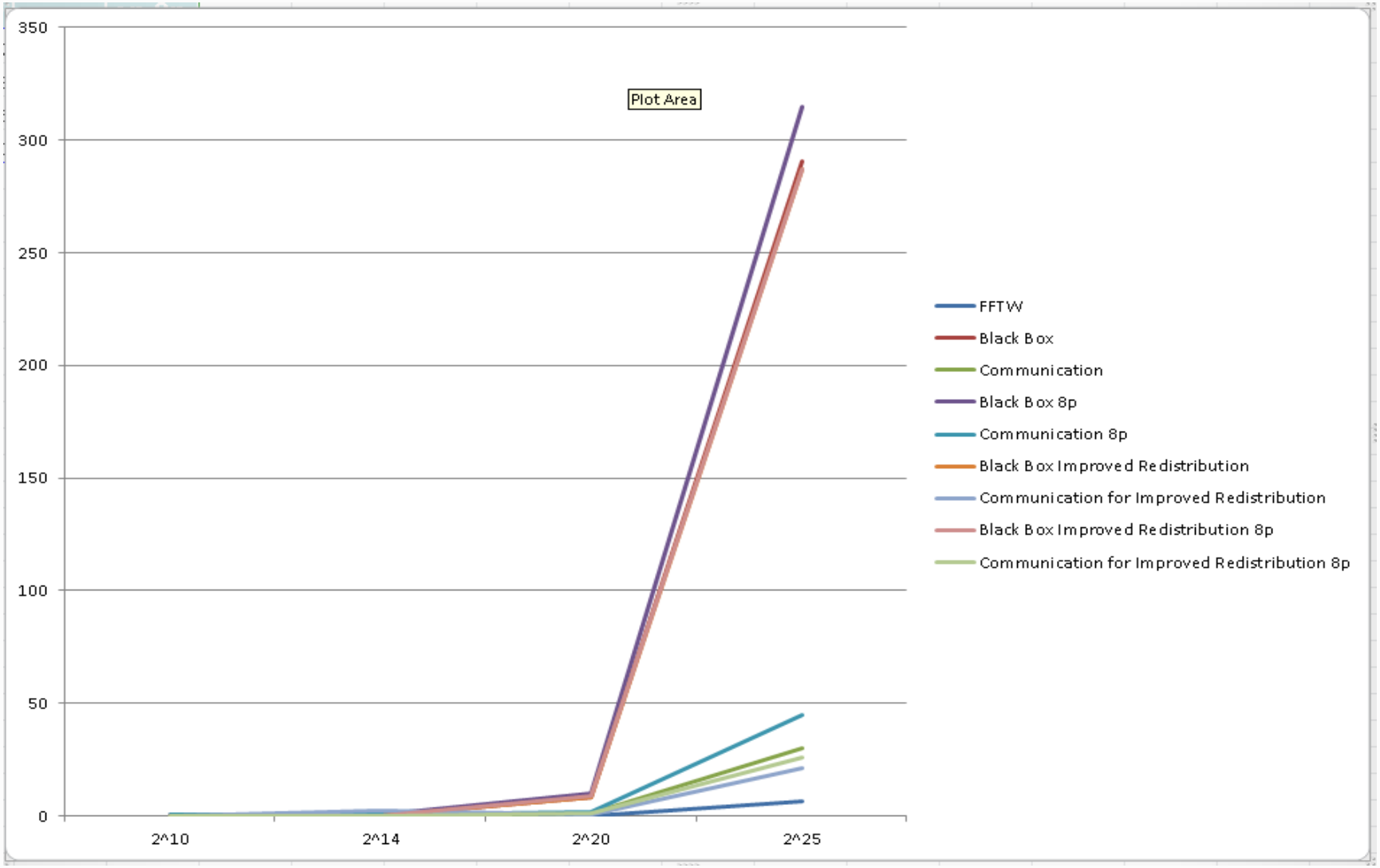
Attempted solution:

- Bundle into one redistribution phase ⇒ total $\log p$ calls

New Results

	FFTW	Black Box Improved Redistribution	Communication for Improved Redistribution	Black Box Improved Redistribution 8p	Communication for Improved Redistribution 8p
2^{10}	0.0002	0.10238204	0.08972	0.154734	0.145179
2^{14}	0.0009	0.247728	2.703539	0.284916	0.191254
2^{20}	0.128	8.389301	0.783675	8.717208	1.014697
2^{25}	6.3023	287.0166	21.3901	287.4002	26.08323

New Results


350
300
250
200
150
100
50
0
Plot Area
2^10
2^14
2^20
2^25
FFTW
Black Box
Communication
Black Box 8p
Communication 8p
Black Box Improved Redistribution
Communication for Improved Redistribution
Black Box Improved Redistribution 8p
Communication for Improved Redistribution 8p

More issues and considerations

- Communication cost: where and how. Better redistribution method.
- Leverage of sequential FFTW on black box problems
- A separate algorithm, better data distribution?

Questions

18.337J / 6.338J Parallel Computing

Fall 2011